NONBLOCKING COMMIT PROTOCOLS

MC714 Sistemas Distribuídos
Nonblocking Commit Protocols

Dale Skeen

“From a certain point onward there is no longer any turning back. That is the point that must be reached.”

Kafka

1. Introduction

Recently, considerable research interest has been focused on distributed data base systems (Eshelby, 1981, 1982; Gehrke, 1981; Graff, 1981), which are in various stages of implementation, including: Data Base Management System (DBMS), SYSTEM-9 (LTM97), and Ingres (SCG192). It is widely recognized that distributed transaction processing is vital to the usefulness of these systems. However, resilient protocols are hard to design and are expensive. Crash recovery algorithms are based on the notion that certain basic operations on the data are logically indivisible. These operations are called transactions.

This research was sponsored by the U.S. Air Force Office of Scientific Research Grant 78-3339, the U.S. Army Research Office Grant DAAG29-76-G-0045, and the Naval Electronics Systems Command Contract N00039-76-C-0613.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the date of initial publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

Transactions Management

By definition, a transaction on a distributed data base system is an atomic operation: either it executes to completion, or it is not executed at all. However, a transaction it rarely a physically atomic operation. Rather, forcing execution must be decomposed into a sequence of physical operations. This discrepancy between logical atomicity (as seen by the application) and physical atomicity poses a significant problem in the implementation of distributed systems. This problem is amplified when transaction atomicity must be preserved across multiple failures. Nonetheless, the approach requires that a notion of transaction atomicity exist as an entity (atomicity) must be supported and made resilient to failures.

Preserving transaction atomicity in the single site case is a well understood problem (LTM97, Graff). The processing of a single transaction is viewed as follows. At some time during its execution, a commit point is reached where the site decides to commit or abort the transaction. A commit is an unconditional guarantee to execute the transaction to completion, even in the event of multiple failures. Similarly, an abort is an unconditional guarantee that the transaction will not be executed. If a single failure occurs before the commit point is reached it is immediately upon recovering the site will abort the transaction. Commit and abort are

Abstract

- Nonblocking protocols allow operational sites to continue transaction processing even though site failures have occurred.
- Many applications require nonblocking protocols, whose properties are investigated in this paper.
- Necessary and sufficient conditions for a protocol to be nonblocking are presented and from these conditions a method for designing them is derived.
- Both a central site nonblocking protocol and a decentralized nonblocking protocol are presented.
Introduction

- A logically indivisible operation is said to be atomic.
- A transaction is an atomic operation on a database system.
  - Being atomic, a transaction either executes to completion or appears never to have happened at all.
- There is growing interest in distributed databases.
- Distributed crash recovery is essential to transaction-based distributed systems.
  - Resilient protocols are expensive and hard to design.
Transaction Management

- A transaction usually comprises a sequence of sub-operations on different objects.
  - For a flat transaction, all objects reside on the same server.
  - For a distributed transaction, such objects may reside on different servers.
- The discrepancy between logical atomicity and physical multiplicity poses a significant problem in the implementation of distributed systems.
  - This problem is amplified when transaction atomicity must be preserved across multiple failures.
The processing of a single transaction

- At some time during the execution of a transaction, a commit point is reached where the site decides to commit or to abort the transaction.
  - A commit is an unconditional guarantee to execute the transaction to completion, even in the event of multiple failures.
  - An abort is an unconditional guarantee to “back out” the transaction so that none of its effects persist.
- When a failure occurs before the commit point is reached, the site will abort the transaction immediately upon recovering.
- Commit and abort are irreversible.
Enforcing atomicity of distributed transactions

- It is assumed that each site has a local recovery strategy that provides atomicity at the local level.
- To guarantee transaction atomicity it is required that the sites, either unanimously abort or unanimously commit.
  - A mixed decision results in an inconsistent database.
- Protocols for preserving transaction atomicity are called commit protocols.
1-Phase Commit Protocol

- A transaction comes to an end when the client requests it to be committed or aborted.
  - To complete the transaction in an atomic manner, the coordinator communicates the client’s decision to all participants and waits until they have carried it out.

- 1PC is the simplest commit protocol.
  - However, it is inadequate because it does not allow an unilateral abort by a server.
    - In general, a server may not be able to commit its part of a transaction due to issues of concurrency control.
      - E.g. the resolution of a deadlock, when a locking scheme is adopted, or the failure of validation, under optimistic concurrency control.
2-Phase Commit Protocol

- 2PC is the simplest commit protocol that allows unilateral abort.
- 2PC also uses a designated site to coordinate the execution of the transaction at the other sites.
- In the first phase of 2PC the coordinator distributes the transaction to all sites and waits for each site to vote on whether to commit or abort the transaction.
- In the second phase, the coordinator collects all the votes and informs each site of the outcome.
- In the absence of failures, this protocol preserves atomicity.
2-Phase Commit Protocol

**SITE 1**
- Transaction is received.
  - “Start Xact” is sent.
- The vote is received.
  - If vote = “yes” and site 1 agrees, “commit” is sent, otherwise “abort” is sent.

**SITE 2**
- “Start Xact” is received.
  - Site votes:
    - “yes” to commit
    - “no” to abort.
  - Vote is sent to site 1.
- Either “commit” or “abort” is received and processed.
2PC Protocol under crash failures

**SITE 1**

Transaction is received. “Start Xact” is sent.

The votes are received. If votes = “yes” and site 1 agrees, “commit” is sent, otherwise “abort” is sent.

**SITE 2**

“Start Xact” is received. Site votes:

- “yes” to commit
- “no” to abort.

Vote is sent to site 1.

Either “commit” or “abort” is received and processed.

**SITE 3**

“Start Xact” is received. Site votes:

- “yes” to commit
- “no” to abort.

Vote is sent to site 1.

Either “commit” or “abort” is received and processed.
A termination protocol is used by working sites when crashes of other sites impair the execution of a commit protocol.

- The termination protocol must preserve transaction atomicity.
- A termination protocol can only be effective if the associated commit protocol is nonblocking.

A recovery protocol is invoked by a crashed site to resume transaction processing upon recovery.
Design assumptions

- Finite state automata are used to model commit protocols.
- Only two prevalent paradigms are considered:
  - Central site model
  - Fully decentralized model
- Protocols in both models have synchronization points.
- The necessary and sufficient conditions for a protocol to be nonblocking are derived.
- Blocking protocols are made nonblocking by adding “buffer states”.
- It is assumed that the underlying network:
  - Provides point-to-point communication and never fails
  - Can detect the failure of a site and reliably report it to an operational site
The formal model in brief

- Transaction execution at each site is modelled as a finite state automaton (FSA), with the network serving as a common input/output tape to all sites.
  - The states of the FSA for site $i$ are called the local states of site $i$.

- A state transition involves the site reading a (nonempty) string of messages addressed to it, writing a string of messages, and moving to the next local state.

- The change of local state is an instantaneous event, marking the end of the transition (and all associated activity).

- In the absence of a site failure, a state transition is atomic.

- State transitions at one site are asynchronous with respect to transitions at other sites.
The PSAs for the 2PC protocol

Site 1: Coordinator

\[ q_1 \rightarrow W_1 \]
\[ \text{request} \]
\[ xact_2, \ldots, xact_n \]

\( (\text{yes}_1), \text{yes}_2, \ldots, \text{yes}_n \)
\[ \text{commit}_2, \ldots, \text{commit}_n \]

\( (\text{no}_1) | \text{no}_2 | \ldots | \text{no}_n \)
\[ \text{abort}_2, \ldots, \text{abort}_n \]

\[ c_1 \rightarrow a_1 \]

Site \( i \) (\( i=2, \ldots, n \)): Slave

\[ q_i \rightarrow W_i \]
\[ xact_i \]
\[ \text{yes}_i \]
\[ \text{no}_i \]

\[ \text{abort}_i \]

\[ c_i \rightarrow a_i \]

\[ \text{commit}_i \]
Properties of the FSAs for distributed commit protocols

- They are nondeterministic
- Their final states are partitioned into two sets
  - Abort states
  - Commit states
- Committing and aborting are irreversible operations
- Their state diagrams are acyclic
- They have (at least) two phases
  - A phase occurs when all sites executing the protocol make a state transition
The definition of a global transaction state

- The global state of a distributed transaction is defined as:
  - A global state vector containing the local states of all FSA's and
  - The outstanding messages in the network.
- The global state defines the complete processing state of a transaction.
  - A global state is said to be inconsistent if it contains both a local commit state and a local abort state.
  - A protocol which maintains transaction atomicity can have no inconsistent global states.
- The graph of all global states reachable from a transaction’s initial global state is called the reachable state graph for that transaction.
Reachable state graph for the 2-site 2PC protocol
Comments on reachable state graphs

- A global state is a final state if all local states in the state vector are final states.
- A global state is a terminal state if it has no immediately reachable successors.
  - A terminal state that is not a final state is a deadlocked state.
- Assuming that the state of site $k$ is $s_k$, it is possible to derive from the global state graph the local states that may be concurrently occupied by other sites.
  - This set of states is called the concurrency set for state $s_k$.
- The reachable state graph grows exponentially with the number of sites, but, in practice, we seldom need to actually build it.
Committable States

- A local state is called **committable** if occupancy of that state by any site implies that all sites have voted *yes* on committing the transaction.

- A state that is not committable is called **noncommittable**.
  - To call “noncommittable” states “abortable” would be misleading.
    - A transaction that is not in a final commit state at any site can still be aborted.

- A site in a noncommittable state does not know whether all the other sites have voted to commit.

- A blocking protocol usually has only one committable state, while nonblocking protocols always have more than one.
Site failures and atomicity of local state transitions

- It cannot be assumed that local state transitions are atomic under site failures.
  - E.g. a site may only partially complete a transition before failing.
  - E.g. only part of the messages that should be sent during a transition are actually transmitted.

- Failures cause an exponential growth in the number of reachable global states.
  - However, it won’t be necessary to construct the (reachable) global state graph under failures.

- In subsequent slides, references to global state graphs will assume the absence of failures.
The Two Paradigms for Commit Protocols

- There are two generic classes of commit protocols:
  - “Central Site”
  - “(Fully) Decentralized”

- These classes represent two very distinct philosophies in commit protocols.
  - The next slides show an example of each class.
  - The examples were chosen because they are the simplest and most renowned protocols in these classes.
  - Both protocols are blocking, but they will be later extended to become nonblocking.
The Central Site Model

- This model uses one distinguished site (the coordinator) to direct transaction processing at all the other participating sites (the slaves).

- The properties of protocols in this class are:
  1. There is a single coordinator, which executes the coordinator protocol.
  2. All other participants (slaves) execute the slave protocol.
  3. A slave can communicate only with the coordinator.
  4. During each phase of the protocol the coordinator sends the same message to each slave and waits for a response from each one of them.

- Strengths of protocols in this class are:
  - They are cheap, simple and robust.

- Weaknesses of protocols in this class are:
  - They are vulnerable to coordinator failure.
Synchronicity within one state transition

- Property 4 assures that the sites progress through the protocol at about the same rate.
- Such a property can be defined as follows:
  - A protocol is said to be *synchronous within one state transition* if one site never leads another by more than one state transition during the execution of the protocol.
- The central site protocol (including both the coordinator and the slave protocols) is "*synchronous within one state transition*".
- This property will be used in constructing nonblocking central site commit protocols.
The fully decentralized model

- In this model, there are no distinguished sites and all of them execute the same protocol.
  - Every site communicates with every other site.
- Decentralized protocols are characterized by successive rounds of message interchanges.
- In this example, during a round of message interchange, each site will send the same message to every other site.
  - The sender then waits until it has received messages from all its peers before beginning the next round of message interchange.
- For simplicity, as part of a message interchange, sites will be assumed to send messages to themselves.
  - The mechanism by which the transaction is distributed to the sites will not be modelled (an “xact” message will be simply received).
The decentralized 2PC protocol

- All participating sites run the same protocol.
- Messages have two subscripts:
  - The first subscript refers to the sender
  - The second refers to the receiver
- In the first phase each site receives the “xact” message, decides whether to unilaterally abort, and sends that decision to each of its peers.
- In the second phase, each site collects all the decisions and moves to a final state.
- The decentralized 2PC protocol is synchronous within one state transition.
When a site failure occurs, the operational sites must reach a consensus on committing the transaction by examining only their local states.

Suppose that a single site remains operational. This site must be able to infer the progress of the other sites solely from its local state.

- The site will be able to safely abort the transaction if and only if the concurrency set for its local state does not contain a commit state.
- The site will be able to safely commit if its local state is “committable” and the concurrency set for its state does not contain an abort state.
The fundamental nonblocking theorem (2/3)

- A blocking situation arises whenever
  - The concurrency set for the local state contains both a commit and an abort state.
  - The site is in a “noncommittable” state and the concurrency set for that state contains a commit state.
    - In this case, the site cannot commit because it cannot infer that all sites have voted yes on committing and it cannot abort because another site may have committed before crashing.

- Notice that both 2PC protocols can block for either reason.
The fundamental nonblocking theorem (3/3)

- A protocol is nonblocking if and only if, in every participating site, it satisfies both of the following conditions:
  1. There exists no local state such that its concurrency set contains both an abort and a commit state,
  2. There exists no noncommittable state whose concurrency set contains a commit state.

- The necessity of the conditions stated in the theorem is demonstrated by the single operational site case.

- To prove sufficiency, it must be shown that it is always possible to terminate the protocol, in a consistent state, at all operational sites.
A corollary to the fundamental nonblocking theorem

- A commit protocol is nonblocking with respect to \( k-1 \) site failures \((2 < k \leq \text{the number of participating sites})\) if and only if there is a subset of \( k \) sites that obeys both conditions of the fundamental nonblocking theorem.
  - A protocol with \( k \) sites obeying the fundamental theorem will be nonblocking as long as one site remains operational.

- The fundamental nonblocking theorem provides a way to check whether a protocol is nonblocking.
  - However, it does not provide a methodology for constructing non-blocking protocols.
The similarity between 2PC protocols

- Structural equivalence
- Synchronicity within one state transition
Concurrency sets in the canonical 2PC protocol

- The canonical 2PC protocol is synchronous within one state transition.
- The concurrency set for a given state in 2PC can only contain states that are adjacent to the given state and the given state itself.
  - $CS(q) = \{q, w, a\}$
  - $CS(w) = \{q, w, a, c\}$
  - $CS(a) = \{q, w, a\}$
  - $CS(c) = \{w, c\}$
Blocking in the canonical 2PC protocol

- The 2PC canonical protocol is blocking because
  - A site in state $w$ may not be able to decide on committing or aborting without communicating with all other sites.
  - This happens because 2PC violates the constraints of the following lemma.

- Lemma
  - A protocol which is synchronous within one state transition is nonblocking if and only if
    - It contains no local state adjacent to both a commit and an abort state, and
    - It contains no noncommittable state adjacent to a commit state.

CS($q$) = \{q, w, a\}
CS($w$) = \{q, w, a, c\}
CS($a$) = \{q, w, a\}
CS($c$) = \{w, c\}
Making the canonical 2PC protocol nonblocking

- With the introduction of a “buffer state” \(p\) (prepare to commit) between states \(w\) and \(c\), the canonical 2PC protocol satisfies the two constraints of the lemma and becomes nonblocking.

Lemma
- A protocol which is synchronous within one state transition is nonblocking if and only if
  - It contains no local state adjacent to both a commit and an abort state, and
  - It contains no noncommittable state adjacent to a commit state.
A nonblocking central site 3PC protocol

Site 1: Coordinator

q1
request
xact2, ... xactn

w1
(no1) | no2 | ... | no

p1
ack2, ... ackn
commit2, ... commitn
c1

Site i (i=2, ...n): Slave

qi
xacti

wi
yesi
aborti
noi

pi
prepare
acki

ai
commiti

ci

A nonblocking decentralized 3PC protocol

Site i (i=1, ...n)

- \( xact_i \)
  - yes\(_{i1} \) ... yes\(_{in} \)
  - no\(_{i1} \) ... no\(_{in} \)

\( q_i \)

\( w_i \)

- yes\(_{1i} \) ... yes\(_{ni} \)
  - prepare\(_{i1} \) ... prepare\(_{in} \)

\( p_i \)

\( c_i \)

- prepare\(_{1i} \) ... prepare\(_{ni} \)
  - no\(_{1i} \) | ... | no\(_{ni} \)
Termination Protocols

- Termination protocols are used when a commit protocol cannot proceed due to site failures.
  - E.g. when the coordinator fails in a central site protocol or when any site fails in a decentralized protocol.

- The purpose of the termination protocol is to terminate the transaction at all operational sites in a consistent manner.
  - This requires that the current state of at least one operational site obeys the conditions of the fundamental nonblocking theorem.
  - As subsequent site failures may occur during the termination protocol, in the worst case, all of the operational sites must obey the fundamental nonblocking theorem.

- A termination protocol should successfully terminate the transaction as long as one site executing a nonblocking commit protocol remains operational.
Central Site Termination Protocol

- When a coordinator crash is detected, a backup coordinator will be selected from the set of operational sites.
- Any distributed election mechanism can be used to choose the backup coordinator.
- Once the backup coordinator is chosen, it will direct the remaining sites toward a commit or an abort, based only on its local state, according to the decision rules presented next.
Decision Rule For Backup Coordinators

- If the concurrency set for the current state of the backup coordinator contains a commit state, then the transaction is committed. Otherwise, it is aborted.

- To apply the decision rule, the coordinator executes the following 2-phase protocol
  1. It issues a message to all sites to make a transition to its local state and waits for an acknowledgement from each site.
  2. It issues a commit or abort message to each site.

- Phase 1 of the backup protocol is required because the backup coordinator may fail.
  - It can be omitted if the backup coordinator is initially in a commit or abort state.
Termination protocol for the canonical 3PC

- Assume that $s$ was the local state of the backup coordinator at the time the termination protocol was initiated.
- The backup protocol would ask all sites to move to state $s$.
- Then it would ask them to
  - Commit if $s \in \{p, c\}$
  - Abort if $s \in \{q, w, a\}$
Main points of the paper

- Characterization and illustration of the most common commit classes
  - Central site
  - Decentralized
- Discussion of blocking in 2-phase protocols
- Fundamental nonblocking theorem and necessary and sufficient conditions for the design of nonblocking protocols
- Termination protocol to deal with site failures in nonblocking 3PC protocols