Distributed Transactions

Nicolas Évrard

B_2CK

Outline

Database transactions

Definition

A transaction symbolizes 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*.

Similar systems could be, if managed properly:

- File systems
- git / mercurial
- Sending e-mails

Transaction properties

The *ACID* properties:

Atomicity	Transactions are *all or nothing* ; if one part of the transaction fails, the entire transaction fails.
Consistency	Transactions bring database from one valid state to another
Isolation	determines how transactions see other transactions changes
Durability	ensures that once a transaction has been committed it will remain so.

Transaction properties

The *ACID* properties:

Atomicity	Transactions are *all or nothing* ; if one part of the transaction fails, the entire transaction fails.
Consistency	Transactions bring database from one valid state to another
Isolation	determines how transactions see other transactions changes
Durability	ensures that once a transaction has been committed it will remain so.

Transaction properties

The *ACID* properties:

Atomicity	Transactions are *all or nothing* ; if one part of the transaction fails, the entire transaction fails.
Consistency	Transactions bring database from one valid state to another
Isolation	determines how transactions see other transactions changes
Durability	ensures that once a transaction has been committed it will remain so.

Transaction properties

The *ACID* properties:

Atomicity	Transactions are *all or nothing* ; if one part of the transaction fails, the entire transaction fails.
Consistency	Transactions bring database from one valid state to another
Isolation	determines how transactions see other transactions changes
Durability	ensures that once a transaction has been committed it will remain so.

Transaction properties

The *ACID* properties:

Atomicity	Transactions are *all or nothing* ; if one part of the transaction fails, the entire transaction fails.
Consistency	Transactions bring database from one valid state to another
Isolation	determines how transactions see other transactions changes
Durability	ensures that once a transaction has been committed it will remain so.

Outline

Sending emails

Example

The example does the following:

- Receive an e-mail
- Store in database the e-mail
- Send the email to a list of contact

```
# Into a DB transaction
email = receive_email()
if not email.sent:
    send(email, contacts)
mark_as_sent(email)
store(email)
# Go on within the transaction
```

Sending emails

Example

The example does the following:

- Receive an e-mail
- Store in database the e-mail
- Send the email to a list of contact

```
# Into a DB transaction
email = receive_email()
if not email.sent:
    send(email, contacts)
mark_as_sent(email)
store(email)
# Go on within the transaction
```

What could possibly go wrong?

Outline

The Two-Phase Commit Protocol

Enters the *The Two-Phase commit protocol* or 2PC.

Their are two kind of actors in the protocol.

Coordinator	The coordinator is the node designated as so, he is the one controlling the transaction. It is the node initiating the protocol.
Cohorts	The other nodes are the cohorts, they will respond with an agreement message or an abort message.

The Voting Phase

When the coordinator reach the last step of its transaction, the *Voting Phase* starts.

1. The coordinator sends a query to commit message and waits until it has received a reply
2. The cohorts execute their transaction up to the point where they are asked to commit
3. Each cohort replies with an agreement message or an abort message.

The Voting Phase

When the coordinator reach the last step of its transaction, the *Voting Phase* starts.

1. The coordinator sends a query to commit message and waits until it has received a reply
2. The cohorts execute their transaction up to the point where they are asked to commit
3. Each cohort replies with an agreement message or an abort message.

The Voting Phase

When the coordinator reach the last step of its transaction, the *Voting Phase* starts.

1. The coordinator sends a query to commit message and waits until it has received a reply
2. The cohorts execute their transaction up to the point where they are asked to commit
3. Each cohort replies with an agreement message or an abort message.

The Voting Phase

When the coordinator reach the last step of its transaction, the *Voting Phase* starts.

1. The coordinator sends a query to commit message and waits until it has received a reply
2. The cohorts execute their transaction up to the point where they are asked to commit
3. Each cohort replies with an agreement message or an abort message.

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Completion Phase

If the coordinator received an agreement message from all cohorts

1. The coordinator sends a commit message to all cohorts
2. Each cohorts completes the operation
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator completes the transaction

If the coordinator received an abort message from any cohort

1. The coordinator sends a rollback to all cohorts
2. Each cohort rollbacks its transaction
3. Each cohort sends an acknowledgment to the coordinator
4. The coordinator rollbacks its transaction

The Tryton implementation

The Tryton implemation of *2PC* is heavily inspired by the one used by the Zope framework.

The main ideas:

- The Tryton transaction is the coordinator
- Data managers will join the Tryton transaction
- Upon committing the Tryton transaction the 2PC happens
- A data manager must raise an error to send an abort message

Data manager API

`tpc_begin`	The 2PC is initiated, the data manager should perform any necessary steps for saving the data
`commit`	In this step the data manager must make sure that any conflicts or errors are handled. Changes are not permanent yet!
`tpc_vote`	This is the last chance for the data manager to abort the global transaction. Voting is done by raising (or not) an exception.
`tpc_finish`	This method makes the changes permanent and should never fail.
`tpc_abort`	This method abandons all changes done, just like `tpc_finish` it should never fail.

Data manager API

`tpc_begin`	The 2PC is initiated, the data manager should perform any necessary steps for saving the data
`commit`	In this step the data manager must make sure that any conflicts or errors are handled. Changes are not permanent yet!
`tpc_vote`	This is the last chance for the data manager to abort the global transaction. Voting is done by raising (or not) an exception.
`tpc_finish`	This method makes the changes permanent and should never fail.
`tpc_abort`	This method abandons all changes done, just like `tpc_finish` it should never fail.

Data manager API

tpc_begin	The 2PC is initiated, the data manager should perform any necessary steps for saving the data
commit	In this step the data manager must make sure that any conflicts or errors are handled. Changes are not permanent yet!
tpc_vote	This is the last chance for the data manager to abort the global transaction. Voting is done by raising (or not) an exception.
tpc_finish	This method makes the changes permanent and should never fail.
tpc_abort	This method abandons all changes done, just like tpc_finish it should never fail.

Data manager API

`tpc_begin`	The 2PC is initiated, the data manager should perform any necessary steps for saving the data
`commit`	In this step the data manager must make sure that any conflicts or errors are handled. Changes are not permanent yet!
`tpc_vote`	This is the last chance for the data manager to abort the global transaction. Voting is done by raising (or not) an exception.
`tpc_finish`	This method makes the changes permanent and should never fail.
`tpc_abort`	This method abandons all changes done, just like `tpc_finish` it should never fail.

Data manager API

`tpc_begin`	The 2PC is initiated, the data manager should perform any necessary steps for saving the data
`commit`	In this step the data manager must make sure that any conflicts or errors are handled. Changes are not permanent yet!
`tpc_vote`	This is the last chance for the data manager to abort the global transaction. Voting is done by raising (or not) an exception.
`tpc_finish`	This method makes the changes permanent and should never fail.
`tpc_abort`	This method abandons all changes done, just like `tpc_finish` it should never fail.

The python code

On only one slide!

```python
def commit(self):
    try:
        if self._datamanagers:
            for datamanager in self._datamanagers:
                datamanager.tpc_begin(self)
            for datamanager in self._datamanagers:
                datamanager.commit(self)
            for datamanager in self._datamanagers:
                datamanager.tpc_vote(self)
        self.connection.commit()
    except:
        self.rollback()
        raise
    else:
        try:
            for datamanager in self._datamanagers:
                datamanager.tpc_finish(self)
        except:
            logger.critical('A datamanager raised an exception in'
                ' tpc_finish, the data might be inconsistant',
                exc_info=True)

def rollback(self):
    for datamanager in self._datamanagers:
        datamanager.tpc_abort(self)
    self.connection.rollback()
```

Outline

A data manager for emails

It's a usual requirement for ERP actions that they should send confirmation emails.

Tryton provides an helper function to do it correctly: `sendmail_transactional`.

You just need to specify the following section in your configuration file:

```
[email]
uri = smtp://user:password@host:port
```

sendmail_transactional

```
def sendmail_transactional(
        from_addr, to_addrs, msg, transaction=None,
        datamanager=None):
    if transaction is None:
        transaction = Transaction()
    assert isinstance(transaction, Transaction), transaction
    if datamanager is None:
        datamanager = SMTPDataManager()
    datamanager = transaction.join(datamanager)
    datamanager.put(from_addr, to_addrs, msg)
```

SMTPDataManager

```
class SMTPDataManager(object):

    def __init__(self, uri=None):
        self.uri = uri
        self.queue = []
        self._server = None

    def put(self, from_addr, to_addrs, msg):
        assert isinstance(msg, Message), msg
        self.queue.append((from_addr, to_addrs, msg))

    def __eq__(self, other):
        if not isinstance(other, SMTPDataManager):
            return NotImplemented
        return self.uri == other.uri

    def _finish(self):
        self._server = None
        self.queue = []
```

tpc_begin / commit

Basically there is nothing special to do.

```
def tpc_begin(self, trans):
    pass

def commit(self, trans):
    pass
```

More complicated data manager might want to prepare to save the data.

tpc_begin / commit

Basically there is nothing special to do.

```
def tpc_begin(self, trans):
    pass

def commit(self, trans):
    pass
```

More complicated data manager might want to prepare to save the data.

tcp_vote

```
def tpc_vote(self, trans):
    if self._server is None:
        self._server = get_smtp_server(self.uri)
```

The idea here is that if we successfully connect to the server then everything should be OK.

`get_smtp_server` is the function parsing the `URI` to connect to the server. It uses the `smtplib` module, so you can use plain `SMTP`, `SMTPS` or `SMTP+TLS`.

tcp_finish

```
def tpc_finish(self, trans):
    if self._server is not None:
        for from_addr, to_addrs, msg in self.queue:
            sendmail(from_addr, to_addrs, msg, server=self._server)
        self._server.quit()
        self._finish()
```

This is the function that actually sends the email messages in the queue to the server.
Once the message are sent the connection is closed and the queue emptied.

tcp_abort

```
def tpc_abort(self, trans):
    if self._server:
        self._server.close()
    self._finish()
```

This close the connection to the server (quite abruptly) and then tidy up the queue and server.