Analysis of a Distributed Commit Protocol for Internet Commerce

Tanja Schulz

August 30, 1997

Presentation

Date: 11.09.1997
Time: 15:00
Venue: JCMB 2511

Supervisor: Peter Thanisch
2nd marker: Gordon Brebner
Session Chair: Stuart Anderson

Abstract

In distributed computing environments a process may need to access data on physically distributed computers/sites, each of which is an autonomous database server. When updates have to be done at two or more sites it has to be ensured that either none or all of the required changes will be made permanent at all of the participating sites, even if failures occur.

Transactions are used to aggregate individual modifications to a database into a single atomic modification that appears to occur either entirely in a single moment or not at all and Commit Protocols define the rules that the participants follow to reach consensus about whether to abort or commit a transaction. A transaction may be defined as a 'collection of operations on the physical and abstract application state' ([4]). This means that a transaction is described by all effects that the execution of an operation has on a database.

Two circumstances conspire to make commit protocols complicated. Firstly, database systems are typically multiuser and locking is used to prevent interference. Secondly, failures, such as site crashes and network outages, can occur. The commit protocol for one transaction might be stalled because of a failure and this means that all participating sites must retain locks on its data items. This could cause blocking and even deadlocks.

In traditional distributed environments the participating sites usually belong to one company or companies with a very close relationship. The environment normally uses the same or compatible components (Transaction Manager, Network, Protocols) and they are all adjusted to each other.

Another characteristic of such an environment is that the access pattern to the data on different servers is quite predictable.

Existing standards for distributed commit protocols usually employ a 'one-pipe' model in which messages associated with the transaction protocol flow over the same conduit (pipe) as the application to application communications. The transaction tree is built implicitly as an application component communicates with another application component. To achieve this, the transaction protocol has been embedded within the communications services provided to the application (usually by a Transaction Processing Monitor (TPM)).

The implementation and maintenance of such a TPM is expensive (in terms of money, time and expertise !) as it is a very complex system. Especially small companies can often not afford this and do not even need all of the offered facilities. They need a lightweight system that is easy to understand, implement and maintain.

Another drawback of the 1-pipe model used with traditional protocols is

that the application may only use the communication services supported by the transaction protocol. That implies that for each new communications protocol a new transaction protocol has to be developed.

Furthermore, a product must also implement the communications protocol in order to be able to support a specific transaction protocol. This increases the complexity which in turn inhibits implementation. Instead of a wide implementation of the existing standards, numerous proprietary transaction protocols have been invented. The big disadvantage of these protocols is that they only work between themselves. This is wasteful and restricts inter-operability. The effect is that transactions may only be used where each component of a distributed application is running under the same TP Monitor software. Therefore it must always be known in advance which application components are to be involved in a transaction.

With the increasing importance of Internet and Internet Commerce, the requirements on a distributed commit protocol have changed.

In an internet environment heterogeneous communication protocols may be used, security concerns are much higher and network delays cause longer message transfer times.

One of the big advantages of Internet is its flexibility. As a consequence, the environment is much more unpredictable and heterogeneous than traditional transaction processing environments. The distributed transactions will affect the databases of more than one company and issues like security and blocking become more important. No company wants its system to be blocked because of a failure in the system of another company. The flexibility of Internet also means that the pattern of access to the data is much more unpredictable.

The rigidity of traditional distributed systems does not match the flexibility that people have come to expect from the Internet usage. A protocol that is suitable for Internet commerce should be a simple standard transaction propagation and coordination protocol that works independently of the application to application communication mechanism. This two-pipe model is a model of separation of transaction control and application communications which is used by many existing proprietary transaction protocols.

To meet the new requirements that the changes in distributed environments pose on distributed commit protocols, Microsoft and Tandem have proposed a new protocol. The Transaction Internet Protocol (TIP) has been designed as a simple, easily-implemented protocol for achieving atomic commitments in an Internet environment. It is based on the standard 2 phase commit (2PC) protocol and employs the two-pipe model. It achieves its simplicity by specifying only how different nodes agree on the outcome of a transaction. The applications have to communicate via other protocols. Thereby issues like application communication semantics and data representation can be avoided. An Internet draft, describing the protocol, has been published on the Internet and it is intended that TIP will be adopted by IETF as an Internet standard.

The aim of my project was a critical analysis of the proposed protocol, analysing the protocol itself as well as the available documentation.
TIP is promoted as a simple, easily-implemented protocol, but the documentation available so far does not make it easy to understand at the first instance. The main source for the analysis of the protocol was the Internet draft ([1]) which describes the different commands and states that are possible within the protocol. Lately, this draft has been extended by a second draft that explains some of the requirements that exists for using TIP ([2]). Although this still does not answer all questions, it makes it easier to understand the protocol and how it should be used.
Generally, TIP is understood as a 2PC protocol. Nevertheless it also offers the possibility to use a 1PC instead, a decision which can be made at two points in the protocol.

In order to start a transaction using TIP, the application on a client has to establish a connection to the required server. With the first command (*IDENTIFY*) the two parties agree on the protocol version that will be used and the initiator of the connection (=primary) should supply an endpoint identifier so that in case of a failure the contacted server (=secondary) may try to re-establish the connection (without this identifier no commit is possible in the 2PC).
After the connection has been set up, the primary has to decide for the first time whether a 1PC or 2PC will be used.
If there are no local recoverable resources at the client and only one subordinate is involved in the transaction, then a 1PC may be used (*BEGIN*). If several different servers have to be included in the transaction, the application has to make sure that they are all enlisted, be it at one central site or at a number of different sites. A server may *PUSH* a local transaction to another server or it may *PULL* the transaction from the other server.
Each server is responsible for the correct use of the 2PC with its connections, there is no central coordinator in TIP. Once all connections at a server have agreed on an outcome, it will pass the result on to the server where itself is enlisted.
With TIP, the applications are responsible for the way the communication tree is built. It depends on their decision about which servers they contact

and how they want them to communicate (e.g. it *PUSH*es from one central server to several other servers or it asks one server to *PULL* a transaction from another one). From the TIP-documents that are available so far, it is not obvious in which case a *PUSH* is more appropriate than a *PULL*, but *PUSH* seems to be most suitable for cases in which a client contacted a server and this server has to involve additional other servers in the transaction (and there is no need for the client to be aware of it). If the client actively wants to involve another server in a transaction, it could do so by telling that server to *PULL* the transaction. It is a basic problem with TIP that the documentation does not explain under what circumstances certain commands are issued (e.g. the second point where a 1PC could be employed is after the exchange of *PUSH/PUSHED* or *PULL/PULLED* - but possible reasons are not explained; another question is why a server could reply with a *NOTPUSHED* command on a *PUSH* request).
After the basic setup commands are completed, the protocol proceeds with the 2PC (this is the standard way of using TIP). The primary sends a *PREPARE* command and the secondary replies by either sending *PREPARED* if it is able to commit or *ABORTED* if it cannot commit the transaction. If the secondary replied *PREPARED*, the primary will forward the final decision (*COMMIT/ABORT*).

In case of failures, different actions are possible. If a connection fails before the 2PC has started, the transaction will be aborted. Otherwise the two servers have to re-establish the connection to learn about the final outcome resp. to inform the other site about the final decision. TIP does not offer any special procedures for failures. All communication is strict point-to-point and no information about which servers are included in a transaction is distributed. That means that TIP is very vulnerable to single-point-failures. If one site fails after the subordinates have voted to commit but before sending the final decision, all connections will be blocked. They return to the IDLE-state and could use the connection for other transactions but e.g. locks on data cannot be released. To complete the transaction, the two servers have to get in contact again and find out about the final outcome of the transaction.

This blocking behaviour of TIP is a big disadvantage especially in an internet environment because the reliability of other servers is difficult to assess and no enterprise wants its system to be blocked because of the failure in another enterprises system.
The lack of any intention to overcome this problem is a step backwards compared to the traditional commit protocols.

The performance of TIP also depends on how the communication tree has been built. The protocol allows a very flexible construction of the communication tree. The applications are responsible for contacting the servers and it is therefore possible to have the same sort of transaction executed in completely different communication topologies. As the topology influences e.g. the message delay (messages have to be forwarded down to the last site and answers have to 'travel' all the way back - therefore the longer the chain the slower the response time) this determines the overall performance as well.

To gain further insight into the behaviour of the protocol, a second part of the project was concerned with the implementation of a simulation.

It is important to note that simulating a distributed commit protocol is not a conventional discrete event simulation because we are not interested in measuring time. The simulation gives insight into the behaviour of the protocol by using message passing as the performance metric.

The basis for the simulation was an existing software ([3]) that was written to simulate the behaviour of traditional 2 and 3PC protocols.

It allows to model different scenarios that illustrate the behaviour of the protocol, including abortion of a transaction and failures of specific participants.

The major differences between traditional 2/3PC and TIP (no central coordinator, no static communication pattern) required substantial changes to the existing software.

During the implementation of the protocol it became obvious that a number of issues are not yet covered in the documentation of TIP. This has mainly to do with the internal coordination of different connections that might exist at one server. It is up to the implementor when decisions are passed on (e.g. will a connection already abort although subsequent did not yet complete the transaction or does it wait until the final confirmation (end-of-transaction) arrives ?).

One of TIPs main features is the possibility to generate different communication topologies for a transaction although the same number of sites is involved.
A number of simulations have been performed to show the behaviour of the protocol under different topologies and abort/failure scenarios. By comparing the number of messages that are sent in the different scenarios, the performance of the protocol can be measured (the more messages are sent, the longer the protocol will take to complete). Some basic scenarios have also been compared to the behaviour of a standard 2PC protocol.

References

[1] J. Lyon, K. Evans, J. Klein: Transaction Internet Protocol (ftp://ds.internic.net/internet-drafts/draft-lyon-itp-nodes-01.txt)

[2] J. Lyon, K. Evans, J. Klein: Transaction Internet Protocol - Requirements
(ftp://ds.internic.net/internet-drafts/draft-evans-tip-functions-00.txt)

[3] S. Ramsay: Simulation of Distributed Commit Protocols
University of Edinburgh, May 1997

[4] J. Gray, A. Reuter: Transaction Processing: Concepts and Techniques, Morgan Kaufmann 1993