A Transactional Approach to Configuring Telecommunications Services

Tim Kempster*, Gordon Brebner and Peter Thanisch

Abstract

The trend in the telecommunications industry towards the provision of IP-based services over large-scale networks creates new challenges in network management. In particular, network managers require the ability to flexibly reconfigure the network. This necessitates enhancements to existing network management techniques, and in particular we require a mechanism for enforcing the atomicity of reconfiguration changes that is robust in the face of network failures. Fortunately, atomic commit protocols have already been developed for achieving atomicity in distributed transaction processing. We demonstrate how this technology can be transferred to network configuration management. We also examine the ways in which atomic commit protocols can benefit from new telecommunications services.

1 Introduction

For many years, industry analysts have been announcing the 'imminent' integration of voice and data network services. After several false dawns, it appears that there may be a real trend in this direction, partly driven by the increasing deployment of IP-based services in the telecommunications industry. Telecommunications providers now route voice, data, fax and broadband services such as video over IP networks. As this trend continues telecommunications networks and IP networks will become indistinguishable, however new challenges in network management will emerge. Network management in this new environment will require the configuration of hundreds of complex routers and switches which may be geographically distributed across thousands of miles. Traditional network management techniques, particularly in the field of network configuration, will require revision to cope with both the larger scale and the complexity of networks, as well as the enriched set of configuration parameters within each network component.

The protocols required to achieve this new level of service will be similar to the *atomic commit protocols* found in distributed database technology. Commit protocols are best known in their role of enforcing the *atomicity* of distributed database transactions: they ensure consistent termination of distributed transactions in the face of site and network failures. Although network configuration can benefit from the use of database commit protocols, leveraging ideas founded in database research, these protocol need to be adapted to the different performance and failure regime of network management applications.

*Division of Informatics, University of Edinburgh, Mayfield Road, Edinburgh, EH9 3JZ, Scotland; Tel +44 131 650 5139, Fax +44 131 667 7209. This research was supported by EPSRC grant GR/L74798.

To reinforce the significance of the overlap between these areas, we also review research on new applications for commit protocols in the provision of network services such as atomic broadcast. Finally, we point out ways in which the efficiency of commit protocols can be improved by exploiting new telecommunications services.

2 Setting the Scene

2.1 Network Nodes

Networks are comprised of *nodes* which can be thought of as points of connection together with a *transport medium* connecting these nodes, along which data propagates. Nodes are either the *end points* at which data originates or is consumed, or *redistribution nodes* which store, possibly duplicate or filter, and then forward data to other nodes. Examples of redistribution points are routers, gateways, switches and hubs. These entities can often be configured to change redistribution behaviour and monitored to give a view of data passing through them. The extent of monitoring and configuration that is provided depends on the device. In general however configuration and monitoring capabilities are becoming more comprehensive. One example of this is the advent of the intelligent router[16]. In this paper we will use the generic name *network node* (NN) to describe any redistribution node that can be monitored or configured.

Networks are controlled and monitored from (inter)Network Operation Centers (NOCs)[1]. From a central point an operator gathers information on the state of NNs and the traffic patterns within the network.

Much software has been developed to assist the task of network management. Typically it consists of agents that execute on each NN together with a front end management system which executes at the NOC. HP OpenView's Network Node Manager provides a good example of this. Other vendors have produced interfaces to store network traffic statistics in databases and even analyse this traffic using OLAP style queries. Most software development and research has focused on monitoring rather than control and configuration. In this paper we are interested in the control and reconfiguration of a network service rather than its monitoring.

2.2 Communicating with Network Nodes

The *de facto* standard protocol used for monitoring state and issuing commands to change NN behaviour is the simple network management protocol (SNMP)[4]. This protocol utilises the idea of a Management Information Base (MIB)[11][17]. The MIB describes a set of useful configuration, control and statistical objects. Example objects include routing tables and packet arrival rates. Each object has an access attribute of either `none`, `read-write` or `write-only`. Each NN supports some subset of objects in the MIB called SNMP MIB view. Figure 1 sets the scene. In essence SNMP supports three types of operations.

1. `GetRequest` used to retrieve the value of a MIB object from an NN.
2. `SetRequest` used to set the value of a MIB object in an NN.
3. `traps` enables a NN to generate messages when certain events occur (eg. a restart).

In this paper we are not concerned with traps and will concentrate on the first two methods.

[1]Many NOCs may control a single network or subset of a network. Jurisdiction and arbitration of control is an issue we will not discuss.

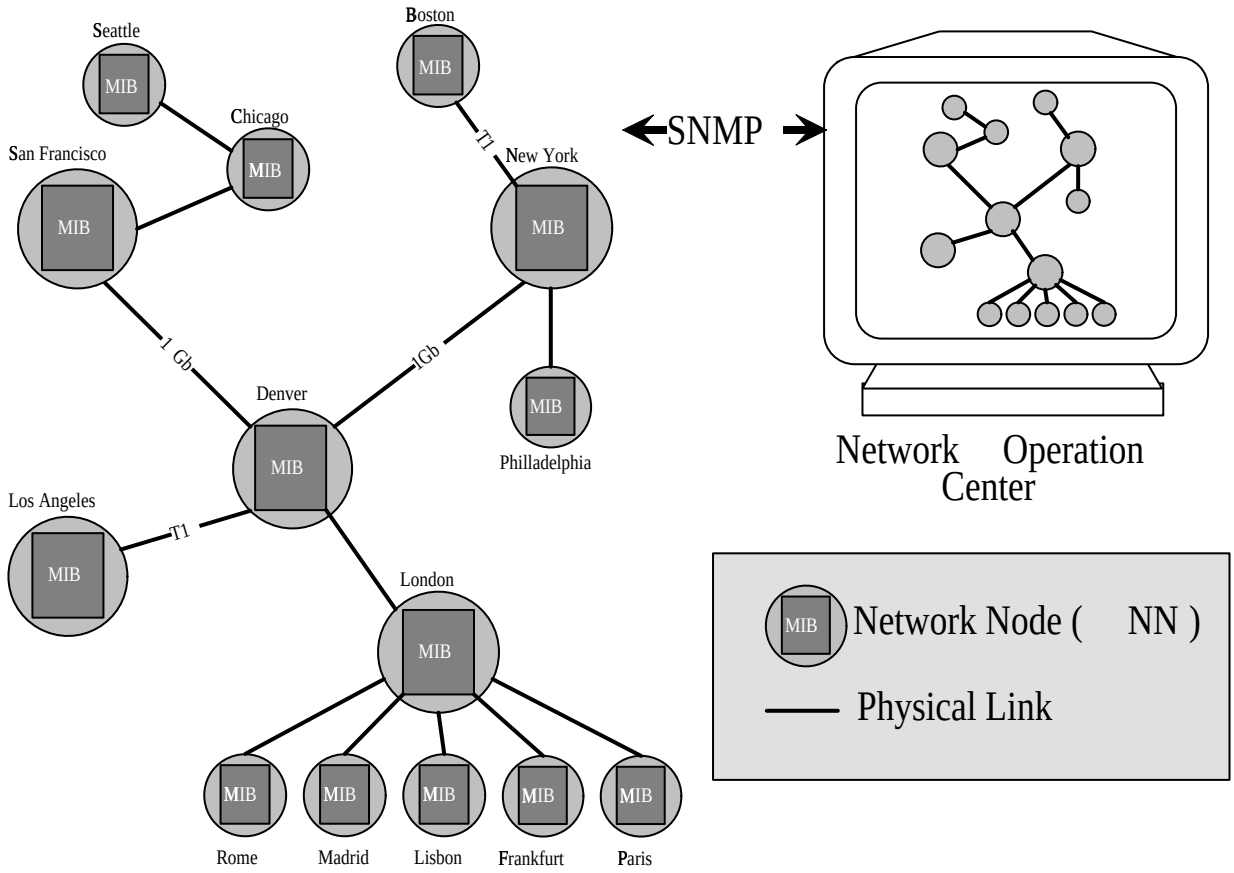


Figure 1: An NOC communicates with the network using SNMP

SNMP need not be implemented on top of a guaranteed message deliver protocol. Monitoring and control software built on SNMP must therefore be resilient to packet loss. SNMP is usually implemented on top of an unreliable, packet-based protocol such as UDP. Each SNMP request or response requires one packet. We will see this raises some interesting issues later.

2.3 Enriching SNMP Services

SNMP allows very simple operations that update and read objects in a NN's MIB. We argue that by slightly extending these operations[2] we acquire much greater power and flexibility. We are motivated by the following two inadequacies of the current implementation.

P1 SNMP does not allow for provisional changes to be requested. Once a request is made it is either granted or denied. If granted by the time the requester is informed of the change it has been applied. In this case it can only be reversed if an inverse operation exists. For the configuration of very simple services this is perhaps sufficient but as NNs become more complex it is unlikely to suffice.

We envisage reconfiguration problems where the proposed change cannot be easily undone but it is important to test the feasibility of this change before applying it. These types

[2]We could build these extensions on top of the existing SNMP methods

updates are very common in other services . In airline reservation systems, seats can be held before they are bought.

P2 The SNMP `GetRequest` method does not enforce persistent views of objects. In other words once an object in a MIB at a NN has been read that object may be immediately updated. It is often very useful to lock out updates once an object has been read for some period of time. We will see later that this allows us to assert a property known as *serializability* in our future network management schemes.

In order to carry out an update to a MIB object the acquisition of a *resource* may be required. For example, suppose router level filtering of certain types of traffic is to be enabled on a particular port. If this is to be enabled, additional CPU may be required to examine packets in more detail for the filtering to take place. If enough CPU resource is present then the change can be made if not the request must be denied.

Many systems provide mechanisms where values can be *locked*[3] The notion of locking can be applied to MIB objects to solve the problems **P1** and **P2** above.

The problem **P1** of provisional updates to configuration can be solved in the following way.

UI A provisional configuration is made by sending a `ProvSetRequest` message to the NN. The NN tries to acquire an exclusive lock on the object to be updated. Once the lock has been obtained a provisional update is made and an acknowledgement is sent to the requester.

UII Step UI is repeated for all the changes that are required on a NN and then a `PrepareRequest` message is sent to the NN. The NN can now decide if it can carry out the cumulative changes requested. If so the required resources for all the changes are reserved[4], and a `Yes` vote is returned to the requester. If the changes cannot be made a `No` vote is returned. Once a `Yes` vote has been made the NN *must* make the changes requested of it, if later asked to commit. At any time up to the time the `Yes` vote is sent the NN may discard all the updates, release locks and respond with a `No` vote when a `PrepareRequest` arrives.

UIII Once the `Yes` vote is received the requester can decide whether or not to go ahead with the request. If so, it sends the NN a `CommitRequest` to commit the changes requested or else an `AbortRequest`. On receiving an `AbortRequest` the NN may release any locks and resources it tentatively acquired. On receiving a `CommitRequest` it makes the changes and then releases any exclusive locks held.

When requesting the values of MIB objects a similar protocol is used.

R I A value is requested by sending a `LockingGetRequest` message to the NN. The NN tries to acquire a shared lock on the object. Once a lock has been acquired the value can be returned to the requester.

R II Step R I is repeated until all the read requests have completed. The requester then sends a `PrepareRequest` message to the the NN. This informs the NN that the value no longer need be protected from updates and the shared lock can be released. The NN acknowledges this release with a `Yes` vote. If the lock could not be held the NN can send a `No` vote to the requester to invalidate the value previously returned.

[3]The simplest implementation of locking provides two types of locks, exclusive and shared. A shared lock can always be granted unless an exclusive lock exists. An exclusive lock can only be granted if no other lock exists.

[4]Locking at the resource level may also be used to implement this reservation but this should not be confused with MIB object locking.

In the example of Figure 2 the NOC first reads a value from an NN in London then provisionally updates a value on a NN in London based on the value read at Frankfurt. The NOC prepares, and then later commits the changes at the London NN. After this a `prepareRequest` is sent to Frankfurt and the read locks are released there.

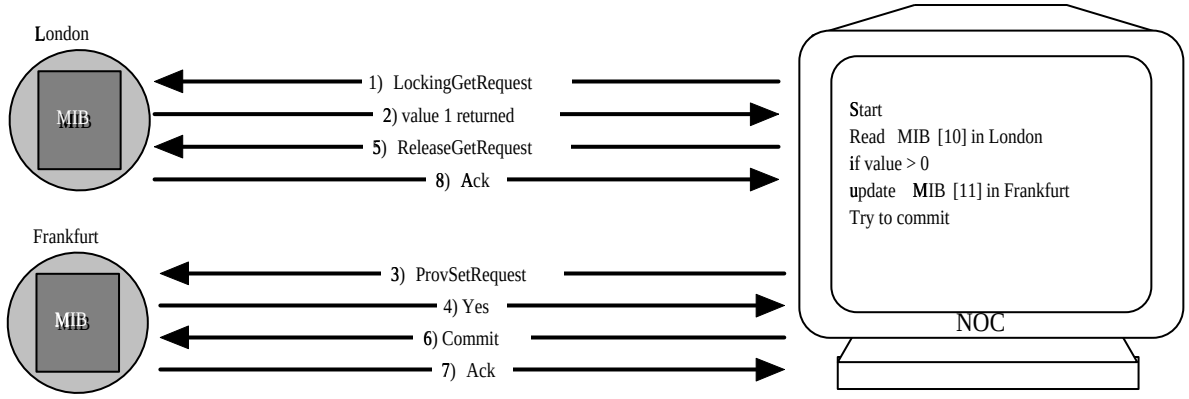


Figure 2: An NOC performs enriched operations.

3 A Centralised Two-Phase Commit Solution

Perhaps the most popular and prevalent commit protocol is the Centralised Two-Phase Commit Protocol[1]. We will describe it in the environment of Network Management. The protocol is comprised of two parts,[5]the operational part and the commit part. During the operational part the operations of the reconfiguration are carried out. This might involve getting object values from MIBs around the network and tentatively updating MIB objects using the methods described in Section 2.3. Once all the operations have been carried out[6] the protocol enters its commit phase.

The centralised commit protocol requires a coordinator for the commit phase. We will assume the same process at the NOC that issued the operations of the transconfiguration also acts as the coordinator. In fact the role of coordinator can be handed over to any process located anywhere on the network. The coordinator solicits votes from each of the NN participating in the transconfiguration using the `PrepareRequest` message. The coordinator collects these votes and then decides on an outcome. If any NN voted `No` or if a timeout expires before all votes are collected the coordinator decides Abort and sends a `AbortRequest` message to any site where a tentative update was performed. If all sites vote `Yes` a `CommitRequest` messages is sent to all non read-only sites. Figure 3 shows the state diagrams of a Coordinator, a NN performing reconfiguration and a NN involved only in read requests.

[5]Strictly speaking, the commit protocol only pertains to the second part.

[6]The feasibility of the transconfiguration may already be established in the operational part if a NN notified the NOC early that the change is impossible. In this case the protocol will terminate here with an aborted outcome and will not enter the next commit part.

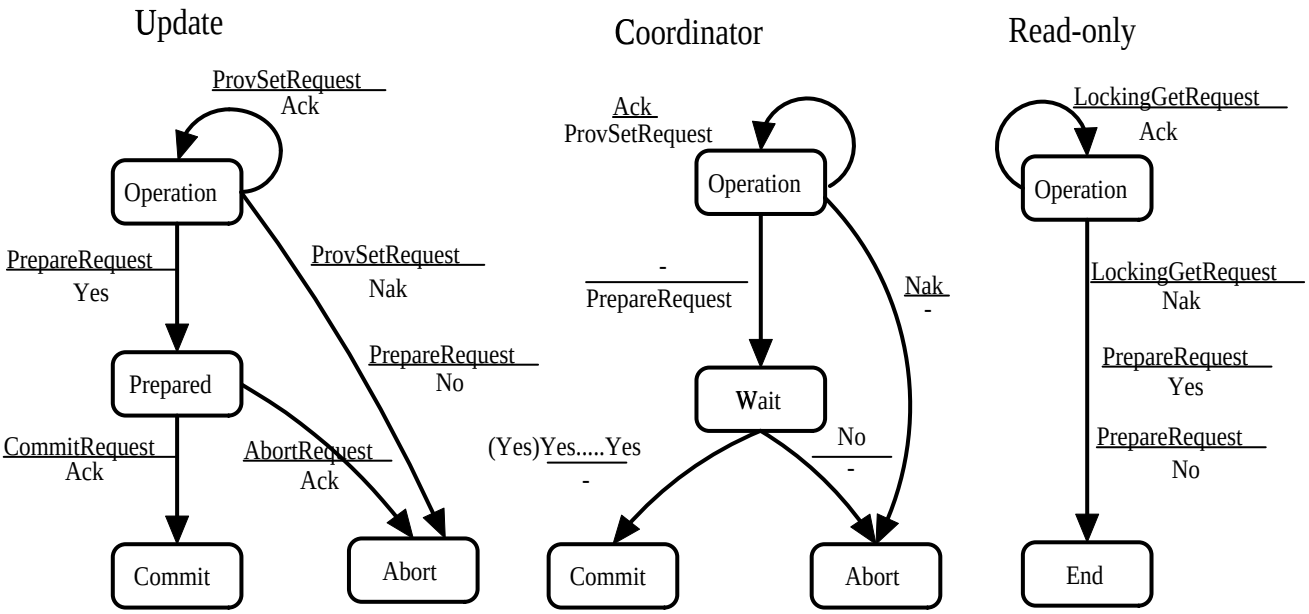


Figure 3: State diagram of the centralised two-phase commit protocol. Transitions are labeled $\frac{\text{rcv msg}}{\text{snd msg}}$ to represent sending 'snd msg' when 'rcv msg' is received.

4 Atomicity and Serializability

The updates and read operations issued at a NOC in order to carry out reconfiguration of the network are analogous to a distributed transaction in the sense that these operations must be grouped together as an atomic unit. The operations of the reconfiguration can be grouped together into a transaction. We call this transactional-style reconfiguration a *transconfiguration*. If implemented carefully we can ensure that the operations within the transconfigurations are atomic.

The atomicity of a transconfiguration is a very powerful tool for the network manager. The network manager need only specify the operations to reconfigure the network service. The atomicity ensures either all or none of the operations are performed. If one or more NN cannot fulfill the network manager's requests, the state of the whole network will remain unchanged. Later, we discuss another powerful property of transconfigurations, namely serializability, but for now we will focus on atomicity.

4.1 Atomicity

Thanisch *et al.*[7] distilled four general properties that are necessary before the use of a commit protocol to enforce transaction atomicity is applicable. These are restated below in Table 1.

Atomic reconfiguration of NNs in a network management environment fulfills all the requirements above. The multiple agents are the NNs. They are semi-autonomous. In general a change may be difficult to undo and finally with the extended operations described in Section 2.3 it is possible to establish the tentative feasibility of their reconfiguration. We can conclude that a commit protocol is appropriate for establishing the atomicity of reconfiguration amongst the NN in a network management environment.

M	Multiple Agents.
T	Tentativeness: Agents can tentatively establish feasibility of the change.
S	Semi-Autonomous Agents: they can always be forced to abort, but they can only be forced to commit if they voted to commit.
P	Permanence: an Agent that is required to make changes cannot easily undo these changes.

Table 1: The MTSP conditions that presage the use of a commit protocol to ensure atomicity.

4.2 Serializability

Serializable level isolation[7] (serializability) arises by combining MIB object locking with a commit protocol. If transconfigurations are serializable a network manager can be sure that if the operations of two transconfigurations execute concurrently on a network, then the effect on the network will be equivalent to running them in some serial order. This is a very useful property especially when multiple network managers have shared control of a network. To illustrate this point further we consider the concurrent executions of two transconfigurations without the property of serializable isolation.

Example 1 *Two routers A and B straddle networks N and M routing two types of traffic, x-traffic and y-traffic, from LAN N to LAN M. The network is initially configured so that A routes x-traffic from N to M while blocking y-traffic. Similarly, B routes y-traffic while blocking x-traffic. Figure 4 describes the situation. It is important that each type of traffic is routed from N to M exactly once. Periodically a transconfiguration T (see Figure 4) is executed at a NOC. The transconfiguration toggles routing of the different traffic types between routers.*

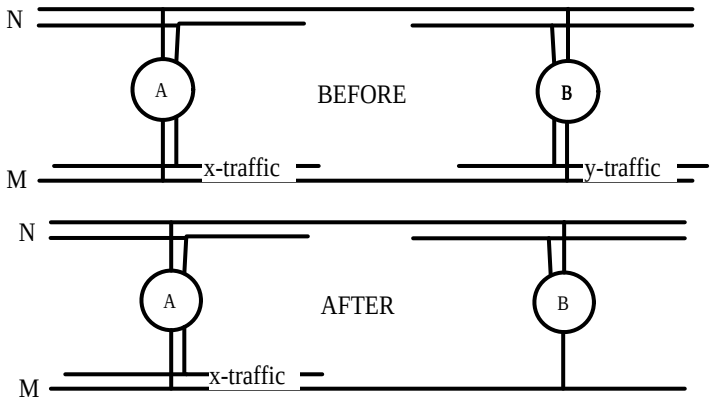
Suppose two transconfigurations of the type T above, denoted T1 and T2, are run concurrently from different NOCs controlling the same network. If isolation is not maintained the operations of the two transconfigurations could be performed as follows. T1 reads A's routing into variable a (x-traffic). T2 reads A's and B's routing configuration into its variables a and b (x-traffic, y-traffic respectively). T2 then updates A and B with the contents of b and a respectively (setting them to y-traffic and x-traffic respectively). Finally T1 reads the value of B's routing (x-traffic) into its variable b and updates routers A and B with the contents of its variables a and b which both contain x-traffic. The result is that the network is configured to route only x-traffic from N to M. Clearly this is incorrect.

□

The fundamental serialization theorem[1] states that if a transaction performs locking in a growing phase (where locks are only acquired) followed by a shrinking phase (where locks are only released), then serializable isolation is achieved[8]. In distributed transaction such as our transconfigurations we require barrier synchronization to provide a window between the two phases. Commit protocols provide just that window. In our centralised two-phase commit protocol locks are acquired up to the point the coordinator sends a `PrepareRequest` message. After that point locks are never acquired only released. We can conclude that our transconfigurations will maintain serializable isolation. If serializability is not important then our transconfigurations need never hold read locks, atomicity will still hold.

[7]Isolation level is a measure of how much interference is allowed between concurrent transactions[3].

[8]This style of lock acquisition and release in a transaction is often referred to as *well formed two-phase locking.*



Transconfiguration T:

BEGIN
a = traffic that A routes
b = traffic that B routes
Set A to route b-traffic only
Set B to route a-traffic only
COMMIT

Figure 4: Transconfigurations interfere producing unexpected results.

5 Blocking

The two-phase commit protocol suffers from blocking. Blocking occurs when a NN is prevented from deciding commit or abort due to inopportune failures in other parts of the system[1]. In the centralised two-phase commit protocol if a NN has voted `Yes` and then loses contact with the coordinator it cannot release the locks and resources it has reserved until it establishes contact with the coordinator once more[9]. Since our NOC might rely on the network it is controlling to pass the messages of the commit protocol, failure of a NN might cause the network to partition leaving the coordinator in one partition and the NN in another. If this was to happen after a NN had voted `Yes` but before it was informed of the outcome the NN would block. Protocols have been designed that are more resilient to failure and we will discuss the appropriateness of these in next.

The three-phase commit protocol[13] is often proposal to solve the problem of blocking. However this proposal needs qualification. In environments where networks can partition and messages can be lost three-phase commit also suffers from blocking. In fact, there exist no protocol resilient to a network partitioning when messages are lost[15]. Although no non-blocking protocols exist the three-phase commit protocol has undergone extensions which guarantee that after a network partition, if any subset of NNs form a connected quorum[10], this subset will not block. The first such protocol was Skeen's quorum based three-phase commit[14]. This was further extend so that nodes could survive repeated partitioning[5] and then again extended to produce commit outcomes more of the time[6].

The choice on whether or not to adopt of a quorum based three-phase commit protocol depends on two factors. The first is connectivity. Often an NOC will have the ability to establish a dialup connection with a NN should it become isolated from the NN by network failure in the network it is configuring. If this is the case then a centralised approach with the NOC acting as coordinator will be appropriate. The second factor is the ability of the NN to elect a coordinator in a partitioned network. For a quorum based protocol to be useful the partitioned NNs must be capable of electing a coordinator and carrying out a termination protocol to decide an outcome. Currently NNs cannot be configured to perform this but in the future this might be possible.

[9]Strictly speaking, if it were able to contact another NN involved in the transconfiguration that had either been informed of the outcome or had not yet voted it could resolve the transconfiguration.

[10]The simplest example of a quorum is a majority.

6 Special Requirements for Network Management

In this section we discuss some special requirements that a network managers might have and how current database research can address these requirements. We also look at a common two-phase commit optimization *early prepare* which is particularly appropriate for network management style commit processing.

6.1 Real-time Commit Protocols

Although the commit protocols we have discussed so far guarantee atomicity they do not guarantee that the changes will take place simultaneously. If this were possible in an the well known impossibility of clock synchronization[10] in a distributed environment with unpredictable message delay would be contradicted. The best we can hope for therefore is that the changes take place within some time window.

Once again the database community has addressed this problem when considering commit processing in Real-Time Distributed Database Systems (RTDDBS). These systems handle workloads where transactions have completion deadlines. The paper[2] describes a centralised two-phase commit protocol where the fate of a transaction is guaranteed to be known to all the participants of the transaction by a deadline, when there are no NN, communication or clock faults. In case of faults, however, it is not possible to provide such guarantees and an exception state is allowed which indicates the violation of a deadline.

Using a real-time commit protocol a network manager can specify a deadline at the point he initiates the commit protocol. In this way he can be sure that the changes will take effect within a specific time window, minimising disruption to the network.

6.2 Order of Commitment

It might be important for a network manager to specify an order[11] in which changes made by a transconfiguration should be applied. This might be important to stop cycles in traffic flow in the network which could result in network storms. We discuss two techniques from the world of transaction processing which allow us to impose such an order.

If a centralised two-phase commit protocol is being used where changes are not applied until commit time the order of commitment can be controlled by the coordinator. The coordinator can dispatch each `CommitRequest` in order waiting for acknowledgements before proceeding to the next request.

Nested transactions[3] provide a mechanism where a transaction can be represented as a tree structure. For an parent transaction to commit each child must be willing to commit. In these schemes a strict the order of commitment is often imposed based on the structure of the nested transaction.

6.3 Early Prepare and Logging

If commit protocols are to be resilient to failure certain state changes must be recorded on stable storage by the participants of a protocol. If a participant crashes, upon recovery a *recovery protocol* executes and utilises this stored information to complete the protocol.

Much work has been carried out to reduce both the number of log writes and the number of messages which must be sent during a commit protocol[12]. These optimizations (particularly

[11] In general a partial order may be more appropriate but for simplicity we assume a total order.

those which reduce log writes) are perhaps not so important in a Network Management scenario but we will briefly discuss the Early Prepare optimization[12] which reduces the message overhead of the centralised two-phase commit protocol.

In this optimization once a NN has tentatively performed its changes it enters its prepared state and votes `Yes`. This saves an extra message round as the coordinator need not solicit the NNs vote and wait for a reply. This optimization is particularly appropriate where all the update requests can be made to a NN in one batch. If however the NOC requires a dialogue with a NN the method can be expensive because forced write to stable storage is required when (re)entering the prepared state.

7 Related Work

Several other proposals have been made for deploying commit protocols as part of a telecommunications service and we give three examples here..

7.1 Atomic Broadcast

Luan and Gligor's atomic broadcast algorithm [9] has each site (end point nodes) accumulating received broadcast messages in a queue. When an application at one of the sites wants to consume a prefix of these messages, it initiates a three-phase commit protocol in which the other sites vote according to whether they have received all messages up to and including that prefix. Messages sequences are only consumed by the application if a quorum of sites confirms that messages up to and including that sequence have been received.

The advantage of this algorithm is that it controls the flow of messages that request progress in the consumption of broadcast message streams. The protocol is designed to detect the situation where two or more sites have concurrently initiated the protocol. It can then combine the voting of the hitherto separate protocol executions.

7.2 Network Control for Mixed Traffic

Li *et al.* have proposed the deployment of a commit style protocol in network management in order to facilitate connection setup in switching systems supporting mixed circuit and packet connections for broadband service [8]. This is part of a control architecture that provides efficient connection setup in a switching system that supports mixed circuit and packet connections for broadband services. Their control architecture is intended for use in systems which allow only one connection to an output port at any given time.

A variety of connection styles, e.g. broadcast, multicast and conferencing, might be demanded by various users. Li *et al.* note that for configuring some of these services, a centralised system is preferable, whereas for other services, a distributed arrangement may be more efficient. Their proposed scheme, using a commit like protocol, is an attempt to get the best of both worlds.

The role of the protocol is to keep the port allocation status information consistent in the various controllers.

Li *et al.* propose a multilevel hierarchical architecture in which Virtual-Centralized Controllers (VCCs) are attached to each level of the control network. The current status of each port can be obtained from each VCC.

[12] Early prepare is sometimes referred to as Unsolicited Vote

Phase 1: (Connection Request Processing). Each VCC, VCC_i, makes a tentative decision on whether the connection requests can be granted, based on the port status at the end of the previous cycle. A tentative allocation, T_i, is made by VCC_i.
Phase 2: (Tentative Allocation Broadcast Phase). Tentative decisions are sent up and down the hierarchy. Thus each VCC collects the tentative decisions made by its parents and children.
Phase 3: (Conflict resolution Phase). Conflicts that occurred in the tentative allocations are resolved and a final allocation is generated. Various conflict-resolution rules are possible, typically assigning different priorities to different levels.

None of the ports on a destination list of a multicast or a broadcast connection will be allocated unless *all* of them are available.

Li *et al*'s protocol works because they are able to assume that the protocol is completely synchronised: the second phase ensures global delivery.

7.3 Active Networks

Zhang *et al.* [18] have investigated the possibility of exploiting an active network in commit protocol execution. They propose to have the active network nodes collecting and examining protocol votes. One active node is designated as the coordinator. Other active nodes can suppress duplicated votes sent towards the coordinator node in a hierarchical network.

We note that such network facilities could be exploited by many of the commit protocols we have discussed in this paper.

8 Conclusions

We have identified a potentially useful area of overlap between research in telecommunications services and research in distributed transaction processing. Telecommunications service providers can adapt transaction processing protocols to the new application of network configuraion management and those developing more efficient commit protocols can exploit new telecommunications services. However, the performance and robustness properties of these technologies could potentially limit the extent of this symbiosis.

References

[1] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[2] S. Davidson, I. Lee, and V. Wolfe. A protocol for timed atomic commitment. In *Proceedings of the Ninth International Conference on Distributed Computing Systems*, pages 199–206, Newport Beach, CA, June 1989. IEEE Computer Society Press.

[3] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, San Mateo, CA, 1993.

[4] M. Schoffstall J. Case, M. Fedor and J. Davin. A simple network management protocol (snmp). Technical Report Internet Draft IETF 1157, IETF, May 1990. On-line, http://www.ietf.cnri.reston.va.us/rfc/rfc1157.txt.

[5] I. Keidar and D. Dolev. Increasing the resilience of atomic commit, at no additional cost. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 245–254, 1995.

[6] T. Kempster, C. Stirling, and P. Thanisch. More committed quorum-based three phase commit protocol. In *Lecture Notes in Computer Science: The Twelth International Symposium on Distributed Computing*, page 246, 1998. On-line, http://www.dcs.ed.ac.uk/home/tdk/.

[7] T. Kempster and P. Thanisch. Atomic commit in concurrent computing. Technical Report ICSA-XX-99, Division of Informatics, University of Edinburgh, June 1999. On-line, http://www.dcs.ed.ac.uk/home/tdk/.

[8] C.-S. Li, C.J. Georgiou, and K.W. Lee. A hybrid multilevel control scheme for supporting mixed traffic in broadband networks. *IEEE Journal on Selected Areas in Communications*, 14(2):306–316, February 1996.

[9] S. Luan and V.D. Gligor. A fault-tolerant protocol for atomic broadcast. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):271–285, July 1990.

[10] J. Lundelius. Synchronizing clocks in a distributed system. Technical Report MIT-LCS//MIT/LCS/TR-335, MIT, Massachusetts Institute of Technology, Laboratory for Computer Science, August 1994.

[11] K. McCloghrie and K. McCloghrie. Management information base for network management of tcp/ip-based internets. Technical Report Internet Draft IETF 2246, IETF, May 1990. On-line, http://www.ietf.cnri.reston.va.us/rfc/rfc2246.txt.

[12] G. Samaras, K. Britton, A. Citron, and C. Mohan. Two-phase commit optimisations in a commercial distributed environment. *Distributed and Parallel Databases*, 3(4):325–360, October 1995.

[13] D. Skeen. Nonblocking commit protocols. In *Proceedings of the ACM SIGMOD Conference on the Management of Data (SIGMOD'81)*, pages 133–142, 1981.

[14] D. Skeen. A quorum-based commit protocol. In *Berkeley Workshop on Distributed Data Management and Computer Networks*, pages 69–80, February 1982.

[15] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, SE-9(3):220–228, May 1983.

[16] David L. Tannenhouse and David J. Weatherall. Towards an active network architecture. *Computer Communications Review*, 26(2), April 1996.

[17] S. Waldbusser. Remote network monitoring management information base. Technical Report Internet Draft IETF 1757, IETF, Febuary 1995. On-line, http://www.ietf.cnri.reston.va.us/rfc/rfc1757.txt.

[18] Z. Zhang, W. Perrizo, and V. Shi. Atomic commitment in database systems over active networks. In *ICDE'99*, pages –. IEEE Comput. Soc. Press, 1999.