Tandem TR 88.6

A COMPARISON OF THE BYZANTINE AGREEMENT PROBLEM AND THE TRANSACTION COMMIT PROBLEM

Jim Gray

May 1988

ABSTRACT

Transaction commit and Byzantine agreement solve the problem of multiple processes reaching agreement in the presence of process and message failures. This paper summarizes the computation and fault models of the two kinds of agreement and shows the difference between them. In particular, it explains that Byzantine agreement is rarely used in practice because it involves significantly more hardware and messages, yet does not give predictable behavior if there are more than a few errors.

TABLE OF CONTENTS

INTRODUCTION

The Workshop on Fault Tolerant Distributed Computing met at Asilomar, California on March 16-19, 1986. It brought together practitioners and theorists. The theorists seemed primarily concerned with variations of the Byzantine Generals problem. The practitioners seemed primarily concerned with techniques for building fault tolerant and distributed systems.

Prior to the conference, it was widely believed that the transaction commit problem faced by distributed systems is a degenerate form of the Byzantine Generals Problem studied by academe. Perhaps the most useful consequence of the conference was to show that these two problems have little in common.

THE TRANSACTION COMMIT PROBLEM

The transaction commit problem was first solved in 1971. Niko Garzado first noticed and solved it while working for IBM on a distributed system for the Italian Social Security Department. The problem was folklore for several years. Five years later, descriptions and solutions began to appear in the open literature [Gray1], [Lampson], [Rosenkrantz].

Solutions to the commit problem make a collection of actions atomic -- either all happen or none happens. Atomicity is easy if all goes well, but the commit problem requires atomicity even if there are failures.

Today, commit algorithms are a key element of most transaction processing systems. Maintenance of replicated objects (all copies must be the same) and maintenance of consistency within a system (a mail message must arrive at one node if it leaves a second one) require atomic agreement among computers.

To state the commit problem more precisely, one needs a model of computation and of failures. Lampson and Sturgis formulated a simple and elegant model of computation and failures [Lampson]. This model is now widely embraced by practitioners. It is impossible to do the model justice here; their paper is well worth reading. In outline:

The Lampson-Sturgis computation model consists of:

- Storage which may be read and written.
- Processes which execute three kinds of actions:
    - change state
    - send or receive a message
    - read or write storage

  Processes run at arbitrary speed, but eventually make progress.

The fault model postulates that:

- Storage writes may fail or corrupt another piece of storage. Such faults are rare, but when they happen, a subsequent read can detect the fault.
- Storage may spontaneously decay, but such faults are rare. In particular, a pair of storage units will not both decay within the repair time for one storage unit. When decayed storage is read, the reader can detect the corruption. These are the fault assumptions of duplexed discs.
- Processes may lose state and be reset to a null state, but such faults are rare and detectable.
- Messages may be delayed, corrupted or lost, but such faults are rare. Corrupted messages are detectably corrupted.

Based on these computation and fault models, Lampson and Sturgis showed how to build single-fault tolerant stable storage which does not decay (duplexed discs), and stable processes which do not reset (process pairs). This single-fault tolerance is based on the assumptions of rare errors and eventual progress (Mean Time To Repair is orders of magnitude less than Mean Time To Failure).

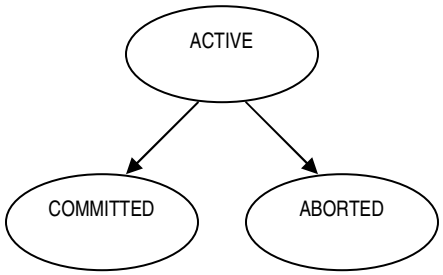
The cost model implicit in this computation model is:

- The computation cost is storage accesses plus messages.
- The delay is serialized storage accesses plus serialized messages.

Good algorithms have low cost and low delay in the average case. In addition they tolerate arbitrarily many lost messages, process resets, and storage decays.

Given this computation and fault model the commit problem can be stated as:

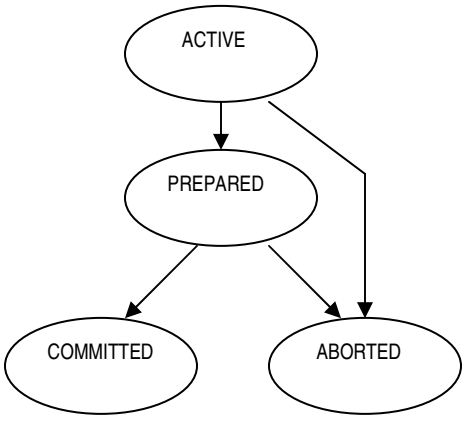
THE COMMIT PROBLEM: Given N stable processes each with the state diagram:



then the commit problem is to find an algorithm which forces ALL processes to the COMMITTED state or ALL to the ABORTED state depending on the input to the algorithm.

The commit problem is easy to solve; one just sends the decision to each process, and keeps resending it until the process changes state and acknowledges. Because messages may be lost and processes may run slowly, there is no limit on how long the algorithm will take. But, eventually all processes will agree and the algorithm will terminate. The expected cost is 2N messages plus N stable (i.e. single-fault tolerant) writes. The expected delay is two message delays plus one stable write delay.

There is a more general version of the commit problem called the two-phase commit problem. It allows an ACTIVE process to unilaterally abort its part of the transaction and consequently abort the whole transaction. After entering the PREPARED state an active process abdicates its right to unilateral abort.



By contrast, the one-phase problem does not allow any of the processes to unilaterally abort. This unilateral abort is very convenient; the CANCEL keys of most user interfaces are examples of unilateral abort.

TWO-PHASE COMMIT PROBLEM: Given N stable processes each with the state diagram above, find an algorithm which forces ALL the processes to the COMMITTED state or all to the ABORTED state depending on the input to the algorithm and whether any process has already spontaneously made the ACTIVE to ABORTED transition. Note that once the process is PREPARED, it cannot make a *unilateral* decision.

Algorithms for two-phase commit are fancier and costlier than the one-phase algorithms described first. Needless to say, three-phase algorithms have been invented and implemented, but it has stopped there.

All of the commit problems have the following properties:

- All processes are assumed to correctly follow the state machine.
- There may be arbitrarily many storage failures, process failures, and message failures.
- Eventually, all processes agree.

THE BYZANTINE GENERALS PROBLEM

The Byzantine Generals Problem grew out of attempts to build a fault-tolerant computer to control an unstable aircraft. The goal was to find a design that could tolerate arbitrary faults in the computer. Since the designers assumed that they could verify the computer software, the only real problem was faulty hardware. Faulty computer hardware can execute even correct programs in crazy ways. So, the designers postulated that most computers functioned correctly, but some functioned in the most malicious way conceivable. Thus, the Byzantine Generals Problem was formulated [Pease], [Lamport].

THE BYZANTINE GENERALS PROBLEM: Assume that there are N generals. Some of them are good and some are faulty. They can communicate only via messages. The bad ones can forge messages, delay messages sent via them, send conflicting or contradictory messages, and masquerade as other generals. If a message from a "good" general is lost or damaged, then the good general is treated as a bad one. An algorithm solves the Byzantine Generals Problem if it gets all the good generals to agree within a bounded time.

The computation model of the Byzantine Generals problem is similar to the Lampson-Sturgis model for processes and messages.

The Byzantine fault model is quite different from the Lampson-Sturgis fault model. The Byzantine Generals Problem assumes that processors and messages may fail undetectably, even maliciously. The Lampson-Sturgis model assumes processes execute correctly or detectably fail and that messages are delivered, detectably corrupted, or lost. Forgeries or undetected corruption are defined as "impossible" (i.e. very rare squared) by Lampson and Sturgis; actually, they just define such an event as a catastrophe.

The gist of the theory on solutions to the Byzantine Generals Problem is:

- If at least 1/3 of the generals are bad, then the good generals cannot reliably agree [Pease].
- If fewer than 1/3 of the generals are bad, then there are many algorithms.

COMPARING THE PROBLEMS

The Commit and the Byzantine Generals problems are identical in the fault-free case. In that case all the participants must agree.

In the typical case (no faults) Commit algorithms send many fewer messages than the Byzantine Generals algorithms.

Fundamental differences between the problems and their solutions emerge when there are faults.

The basic differences are:

- Commit protocols tolerate many faults.

  Byzantine protocols tolerate at most N/3 faults.

- ALL processors agree in the commit case.

  SOME processors agree in the Byzantine case.

- Commit algorithms are fail-fast. They give either a common answer or no answer.

  Byzantine algorithms give random answers without warning if the fault threshold is exceeded.

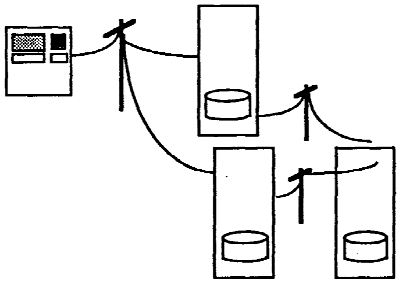
- Commit agreement may require unbounded time.

  Byzantine agreement terminates in bounded time.

- Commit algorithms require no extra processors and few extra messages. Byzantine algorithms require many messages and processors.

The following examples show that neither of these problems is especially realistic

Byzantine ATMs: Consider an Automated Teller Machine (ATM) doing a debit of a bank account. Three computers storing the account plus the ATM give the requisite four processors needed for Byzantine agreement. If the phone line to the ATM fails, then the three computers quickly agree to debit the account but the ATM refuses to dispense the money. This is Byzantine agreement.



Commit ATMs: If the same ATM is controlled by a single computer and they obey a commit protocol, then they will eventually agree (debit plus dispense or no debit and no dispense). But the customer may have to wait at the ATM for many days before the faulty phone line is fixed and the money is dispensed.

Most "real" systems focus on single-fault tolerance because single faults are rare, double faults are very rare (rare squared). Commit algorithms are geared to this single-fault view. They give single-fault tolerance by duplexing fail-fast modules [Gray2].

Byzantine Algorithms require at least four-plexed modules to tolerate a single fault. I believe that this high processor cost and a related high message cost is the reason that no

practitioner has yet implemented Byzantine algorithms. Look at the two pictures above and judge for yourself.

The Lampson-Sturgis model distinguishes between message failure and process failure because long-haul messages typically are corrupted once an hour while processors typically fail once a month. This is an important practical distinction -- especially since Byzantine messages outnumber processors by a polynomial. Amazingly, the Byzantine fault model typically equates message failures with process failures. As the number of nodes grows, the number of message failures grows polynomially and produces a system much less reliable than a single processor. Indeed all Byzantine agreement systems have mean time to failure much below a single processor system [Tay], [Babaoglu].

Commit algorithms are fail-fast while Byzantine algorithms give an answer in any case. Each non-faulty processor executing a Byzantine Generals algorithm gives an answer within a fixed time -- ~N^3 message delays for a typical algorithm. If there are few faults, then all the non-faulty processors will give the same answer. If at least N/3 processors are faulty or if at least N/3 messages are damaged, then two "correct" processors may give different answers.

By contrast, commit algorithms eventually get all the processes to give the same answer. This may take a very long time and many messages. No one wants to wait forever for the right answer. Unfortunately, there is no solution to this dilemma. If all processors must agree, they must be prepared to wait an unbounded time.

The salient properties of the two problems are listed in the chart below. It shows that there is little overlap between the two problems.

	DEGREE OF AGREEMENT	
FAULT TOLERANCE & SPEED	SOME AGREE	ALL AGREE
LIMITED TIME/ERRORS	BYZANTINE	?
UNLIMITED TIME/ERRORS	?	COMMIT

Based on this comparison between the two problems, practitioners embraced the Commit problem over the Byzantine Generals problem because it has an efficient solution to the single-fault case, gives correct answers in the multi-fault case, and has good no-fault performance.

ACKNOWLEDGMENTS

Andrea Borr, Phil Garrett, Fritz Graf, Pat Helland, Pete Homan, Fritz Joern and Barbara Simons made valuable comments on this paper.

REFERENCES

[Babaoglu] Babaoglu, O., "On the Reliability of Consensus Based Fault Tolerant Distributed Computer Systems", ACM TOCS, V. 5.4, 1987.

[Gray1] Gray, J., "Notes on Database Operating Systems", *Operating Systems, An Advanced Course, Lecture Notes in Computer Science*, V. 60, Springer Verlag, 1978.

[Gray2] Gray, J., "Why Do Computers Stop, and What Can We Do About It?", Tandem TR 85.7, Tandem Computers, 1985.

[Lamport] Lamport, L., Shostak, R., Pease, M., "The Byzantine Generals Problem", ACM TPLS, V. 4.3, 1982.

[Lampson] Lampson, B.W., Sturgis, H., "Atomic Transactions", *Distributed Systems Architecture and Implementation; An Advanced Course*, *Lecture Notes in Computer Science*, V. 105, Springer Verlag, 1981.

[Pease] Pease, M., Shostak, R., Lamport, L., "Reaching Agreement in the Presence of Faults", ACM Journal, V. 27.2, 1980.

[Rosenkrantz] Rosenkrantz, D.J., Stearns, R.D., Lewis, P.M. "System Level Concurrency Control for Database Systems", ACM TODS, V. 3.2, 1977.

[Tay] Tay, Y.C., "The Reliability of (k,n)- Resilient Distributed Systems", Proc. 4th Symposium in Distributed Software and Database Systems, IEEE, 1984.