Byzantine Mirroring System

Piotr Zieliński

A fault-tolerant system that guarantees the consistency of mirrored data even if some servers cheat

UNIVERSITY OF CAMBRIDGE
Computer Laboratory
Security Group

We aim to design a new reliable mirroring system. Unlike many currently available techniques, which use a single primary server to handle all incoming updates, our system will consist of many such servers (called "writers"). Besides these writers, the system will contain many additional servers (readers) from which data can only be downloaded.

In contrast to the Usenet news protocol, our system will ensure full consistency of mirrored data. For example, the order of applying updates will be the same on all servers, even if updates were originally sent to different writers. Moreover, the system will be fault-tolerant and be able to operate in malicious environments, even where many primary servers (writers) collaborate in cheating.

High Availability

The system is composed of many servers. Both readers and writers can be contacted to download data, but only writers can accept updates.

"Gossip" Communication

Each update is broadcast epidemically; once every while each host connects to a randomly chosen neighbour and exchanges updates.

Modular Construction

The broadcast mechanism, network protocol, and message acceptance algorithm are all independent and might be developed by different people.

Maintaining Consistency

The system not only detects failures. Its *message acceptance algorithm* maintains the consistency of data among the correct replicas when those failures (even malicious ones) occur.

Protocol Design

The protocol is very uniform and simple. It uses the hosts' computational power rather than network bandwidth. This makes cheating much harder and writing compatible clients easier.

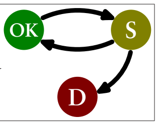
Decision Autonomy

Servers may have different failure-detection mechanisms. In the case of strong disagreement the group may split into two disjoint sets of mutually compatible hosts. Each host is free to choose which fraction to follow.

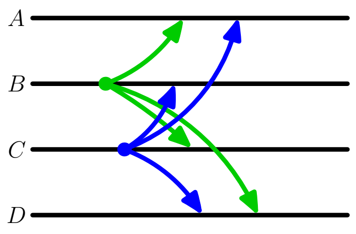
What is the challenge?

Each server can either be online, offline or corrupt. Servers have opinions about each other. Possible opinions and transitions are listed in the box. The figures below show examples of situations that need to be resolved. Irrelevant messages or opinions are not shown.

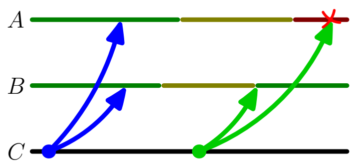
┈┈	Offline	━━	Does *not* suspect	= OK
━━	Online	━━	Suspects	= Probably failed
━━	Corrupt	━━	Considers dead	= Failed



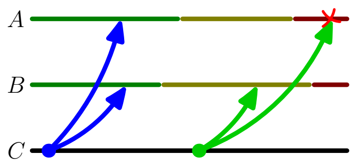
1. Hosts A and D received two messages: green from B and blue from C. However, A considers the green message as the first one, whereas for D the blue one is first. The system must ensure that all processes agree on a total order of all messages.



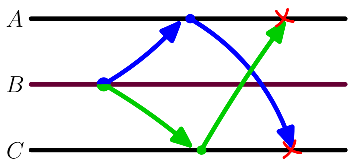
2. After some time has passed since A and B received the latest message from C, they both begin to suspect that C failed. The next message from C made B consider C being available again. However, A decided already that C is dead before it received the message and can not change its mind any more at that point.



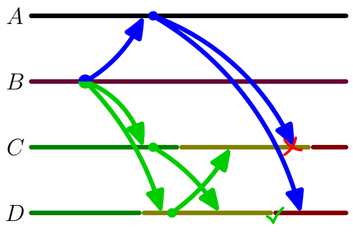
3. Almost the same situation as above, but A and B somehow managed to coordinate their opinions about C. Nevertheless, A accepted both messages from C, but B accepted only the blue one, because the green message arrived when B had already considered C dead.



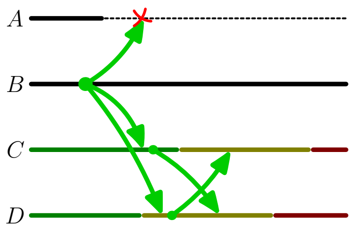
4. Here cheating comes into play. Process B sent two different messages with the same ID. Since each message is broadcast, both A and C echoed received messages and finally found out that B was cheating. However, this was too late because they had already accepted two different messages with the same ID.



5. Waiting for confirmation from all alive hosts does not help either. In this case C and D suspects A of having failed, and B is cheating. Host C learnt that B was corrupt and did not accept its message, but D had received all necessary confirmations and accepted the message. It had not waited for a message from A because D considered A dead.



6. It seems that in the previous example D should have waited longer. But how much longer? For host D, this case (where A really failed) is indistinguishable from the previous one. Yet if D had decided to wait for a confirmation from A even after starting considering it to be dead, it would have waited forever.



Contact details: **Email:** Piotr.Zielinski@cl.cam.ac.uk **WWW:** `http://www.cl.cam.ac.uk/~pz215/`

July 2002