Hybrids on Steroids

SGX-based high-performance BFT

Johannes Behl, Tobias Distler, Rudiger Kapitza

Presented by Federico Mengozzi

12-04-18

The idea

- The protocol is built around Intel SGX (Software Guard Extension), a trusted execution environment that protects software components against malicious behavior of an untrusted operating system

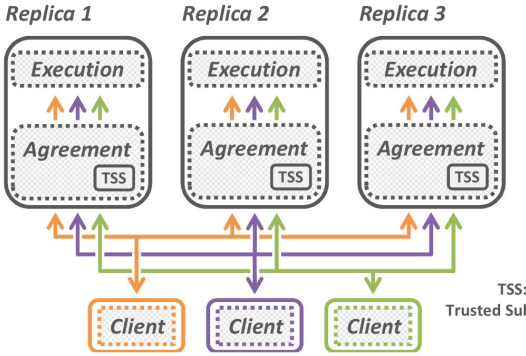

Figure 1. Hybrid BFT state-machine replication.

- All known hybrid protocols rely on a kind of sequential processing that doesn't take advantage of modern multi-core systems

Trusted subsystems

Hybrid protocols require trusted subsystems that are as small as possible and that can provide message certificates

Logs vs Counters

Logs uses an append-only data structure, in general they require more memory (it's the case for A2M-PBFT) while counters expose a simpler interface (for instance USIG) and require less memory (TrInc).
The authors implement the TSS (trusted subsystem) on top of TrInc. The TSS, called TrIncX, generates certificates from a counter value and message digest

```
certificate = cryptography_function(counter, message)
```

Cope with equivocation

Prevention vs Detection

Prevention: Each message is stored in a predetermined place in the virtual timeline of the replica (A2M PBFT) and it's impossible to have conflicting messages

Detection: Each message is stored in position determined at runtime (MiniBFT) and conflicting messages are detected by processing message in the order of their counter values

Preventing equivocation place a further restriction on the protocol and simplifies its design

Sequential processing seems inherent to the hybrid fault tolerance model itself: all hybrid systems **prevent** equivocation by cryptographically binding (by means of the TSS) outgoing messages to an unique monotonically increasing timestamp. In this way if a faulty replica produces conflicting statements by sending different messages it can only do so with different timestamps. With the trusted execution environment provided by the CPU instead of by hardware devices it's easy to have as many TSS as the degree of parallelism

Subprotocols in Hybster

- **Ordering protocol**: Establish a global order in which requests should be executed by assigning a unique order number to each request and by ensuring that a sufficient number of replicas reaches consensus
- **Checkpointing protocol**: It creates consistent snapshot of replica states
- **View-change protocol**: It allows the replica group to safely switch between configurations

Ordering

It’s the main bottleneck in current protocols since it usually requires sequential processing of consensus instances or of all incoming messages

Two vs Three phases

Three phase: Similar to the PBFT model (Pre-Prepare, Prepare and Commit)

Two phase: Leader sends its proposal in a Prepare message and replicas acknowledge it with a Commit message. Requests are executed when a sufficient number of Commit messages are received

Two phase ordering is chosen because it provides better performances even though it increase the complexity of the implementation

Ordering

In Hybster, the leader assign an order number to requests and proposes the assignment to the replicas. If a sufficient number of replicas follow the proposal, consensus is reached and the request is executed

Leader $l = v \bmod n$
Order number **o**

(1) To make impossible for the leader to send different assignment for **o**, a certificate is generated by the TSS that guarantees that only one valid certificate (hence only one valid Prepare) is generated for order number **o**

It might happen that the leader is suspected to be faulty and after some view changes the same leader l might needs to generate a Prepare message for the same order number **o** but with new view v'. To allow this, (1) is extended to generate only one valid certificate for the value $v \mid o$ where **v** is stored in a fixed number of most significant bits and **o** in the remaining bit

```
certificate = create(l, v | o)
```

Checkpointing

To resend messages when required, the replication protocol maintains a log of all ordering messages for a limited number of consecutive consensus instances, between ***low*** and ***high*** water marks

Older messages are discarded once a replica has proof that enough replicas were able to execute the commands up to a certain water mark. The high water mark make sure that a malicious node can’t compromise the network by sending ordering message with order numbers that are too high

View-Changes

To resend message when required the replication protocol maintains a log of all ordering messages for a limited number of consecutive consensus instances, between low and high water marks.

Parallelized consensus

Replicas are composed of equal processing units classed **pillars** that operate mostly independent from each other

Each pillar is responsible for executing the consensus instances from a predefined share of order numbers. Each pillar is equipped with its own instance of TrInx

With a consensus-oriented parallelization, consensus instances are distributed over the pillars in a predefined manner. Thus an order message of a replica ***r*** is only accepted if the certificate is issued by the TrInX instance of the correct pillar

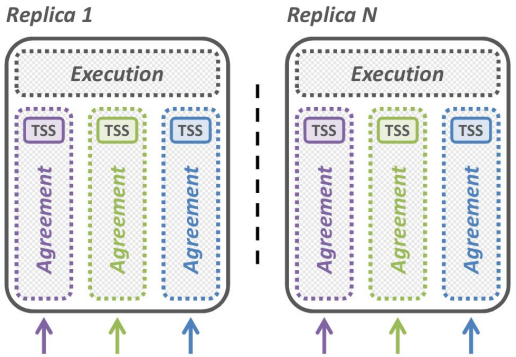

Figure 2. A parallelizable hybrid - the idea.

Multiple timelines

Although in other hybrid protocols a replica can only actively participate in a consensus instance for an ordered number ***o*** if it has already received processed all instances with ***o*** **<** ***o'*** Hybrid is able to circumvent the problem by using different instances of trusted counters and by having multiple independent timelines