

# **DISPOSITION 11: STATE MACHINE REPLICATION**

---

Mathias Ravn Tversted

January 14, 2020

# TABLE OF CONTENTS

State machine

Replicated state machine

Consistency: TOB

- Synchronous implementation

- Async implementation

  - Enten den her

  - Eller den her

Client-centric consistency

# STATE MACHINE

---

# DEFINITION OF A STATE MACHINE

A state machine consists of

- A set of states
- Initial state  $State_0 \in States$
- Set of *Inputs*
- Set of *Outputs*
- Transition function

$$T : States \times Inputs \rightarrow States \times Outputs$$

# REPLICATED STATE MACHINE

---

# IF AND DEFINITION OF REPLICATED STATE MACHINE

We have I/O ports  $RSM_i$  and the following ports

- $\forall i$  *Received<sub>i</sub>* outputs what IF has Received
- *Process* says what should be processed next
- *Deliver<sub>i</sub>* instructs IF to deliver the next message to  $S_i$

It has the following safety requirements:

- **Validity:** If honest server outputs  $(y_1, \dots, y_n)$  Then  $\exists (x_1, \dots, x_n) : (y_1, \dots, y_n) = M(x_1, \dots, x_n)$
- **Agreement:** If honest server outputs  $(y_1, \dots, y_n)$  then all other servers output at least some prefix of that, or vice versa

## IDEAL FUNCTIONALITY OF AN RSM

- Let  $State = State_o$ . Foreach  $RSM_i$ ,  $Q_i = \emptyset$ .
- $Q_i$  is the outputs for  $S_i$ , which has not yet been delivered. Let  $UnProcessed = \emptyset$
- On input  $x$  to  $RSM_i$ , output  $x$  on  $Received_i$ , add  $x$  to  $UnProcessed$
- On input  $x$  on  $Process$ . If  $x \in Unprocessed$ , compute  $(State', y) = T(state, x)$ . Add  $y$  to  $Q_i$ . Pop  $x$  from  $UnProcessed$
- On input  $Deliver_i$ , where  $Q_i \neq \emptyset$ , remove first  $y_i \in Q_i$  and output  $y$  on  $RSM_i$

## **CONSISTENCY: TOB**

---



# CONSISTENCY

In order to keep everyone consistent, we need to build it on *Total-Ordered Broadcast*. Because we have state machines, the order of processing matters, therefore we need to be able to have all the machines process things in the same order. These can be

- Synchronous TOB
- Async TOB
- Eventually synchronous broadcast

## SYNCHRONOUS IMPLEMENTATION OF TOB

- On  $x$  at  $S_i$ , flood it
- All servers  $S_i$  keep *UnQueued<sub>i</sub>* of received messages. These could be received in arbitrary order
- All servers  $S_i$  keep a set *Queued<sub>i</sub>*. When they move messages from *UnQueued* to *Queued*, they do so in the same order
- There is a leader  $L$  (sequencer), who makes the order. Corrupted leader may break liveness but not safety
- If a message is put into *Queued*, they do it in the same order. Leadership goes round-robin. This guarantees liveness
- This happens with blocks

## ADAPTING IF TO ASYNC BROADCAST

Now we have a notion of an epoch, *Unqueued* and we transmit multiple inputs in combined blocks.

1. On input  $x$  to TOB, add it to *Unqueued*
2. Leader of epoch is  $P_i \bmod n$
3. Leader adds *Unqueued* to block and broadcasts it
4. When receiving block, add it to  $Q$

## CORE-SET SELECTION: EKSTRA STUFF DER KAN SPRINGES OVER?

We cannot reliably wait for the leader in each epoch. So we have everyone propose a block, and when a block is *seen by many honest*, we can trust it. We simply have everyone collect these blocks and take the union of them. We have Byzantine Agreement available to us in the async model.

## TIDSMÆSSIGT PROBLEMATISK: CORE-SET SELECTION

Per fig. 11.4

# **CLIENT-CENTRIC CONSISTENCY**

---

# CLIENT VS. SERVER SIDE CONSISTENCY

**Server-side**, which is all honest servers executing commands in the same order

**Client-side**, which is servers being behind other servers. This is why agree requires that for honest servers, that their output is a prefix of another honest parties output