Distributed Systems: Team 8

Presentation and Lecture Scribe

Yashas Samaga
yashas.samaga@research

Shivam Nayak
shivam.nayak@students

Bharathi Ramana Joshi
bharathi.joshi@research

Contents

1 Presentation **1**
- 1.1 Fault, Error and Failure . . . 1
- 1.2 Example . . . 1
- 1.3 Types of Failures . . . 2
- 1.4 Recovering from Failures . . . 2
  - 1.4.1 Forward Recovery . . . 2
  - 1.4.2 Backward Recovery (Rollback Recovery) . . . 2

2 Lecture (19 January 2021) **2**
- 2.1 Recap . . . 2
  - 2.1.1 Message Ordering . . . 2
  - 2.1.2 Global State and Consistency . . . 3
  - 2.1.3 Issues for an snapshot saving algorithm to take care of . . . 3
- 2.2 Chandy-Lamport Algorithm . . . 3
  - 2.2.1 The Algorithm . . . 3
  - 2.2.2 Analysis . . . 4
  - 2.2.3 Correctness . . . 5
  - 2.2.4 Complexity . . . 5
  - 2.2.5 Questions and Answers . . . 5

1 Presentation

1.1 Fault, Error and Failure

Definition 1 (Fault)**.** Anything that causes (or can cause) deviation from the expected behaviour is termed a fault.

Definition 2 (Latent Fault)**.** Faults that are dormant and are not affecting the system yet.

Definition 3 (Active Fault)**.** Faults that have occurred and caused unexpected behavior.

Definition 4 (Error)**.** Active faults lead the system into an error state. Errors are manifestations of faults.

Definition 5 (Failure)**.** A failure is an observed unexpected behavior that was caused by unhandled error(s).

1.2 Example

```
int arr[10], sum = 0;
for (int i = 0; i <= 10; i++)
    sum += arr[i]; // fault: OOB access arr[10]
```

In the above example, there is an out-of-bounds access. It is a latent fault and becomes an active fault upon execution. The memory access could result in a zero by chance and the may system proceed unaffected. It could also result in a segmentation fault which would put the system in an error state. If the application does not trap the segmentation fault and recover, the node crashes.

1.3 Types of Failures

Definition 6 (Timing Failure)**.** The node fails to deliver a response in expected time. The response may be correct but it was delivered earlier or later than expected.

Definition 7 (Omission Failure)**.** The node fails to send or receive response to/from other nodes.

Definition 8 (Crash Failure)**.** The node becomes completely unresponsive.

Definition 9 (Response Failure)**.** The node returns an incorrect response (with respect to expected behavior or specifications). The failure can be a value failure if the response returned incorrect value or a state transition failure if the response is in an incorrect state.

Definition 10 (Byzantine Failure)**.** The node sends different responses throughout the system and/or produces arbitrary messages at arbitrary times. The node can also forge messages/responses from other nodes.

1.4 Recovering from Failures

Recovering from failures is a process of moving from an erroneous state to an error free state.

A distributed system may interact with the outside world. The outside world cannot be expected to recover when the system fails. This is called the output commit problem. The outside world can be modelled as an "outside world process" that communicates through message passing with the system for theoretical analysis.

1.4.1 Forward Recovery

The nature of errors caused by the fault can be completely and accurately accessed and it's possible to remove these errors from the system's state and enable the system to move forward.

1.4.2 Backward Recovery (Rollback Recovery)

The system is restored to the last saved error free state. The rollback of one node can lead to abnormal states in other nodes if nodes are allowed to checkpoint independently. For example, a message that was earlier sent by the failing node might be undone in the recovery. This could force rollbacks in the non-failing nodes. This is called domino effect or rollback propagation.

Methods for domino-free recovery:

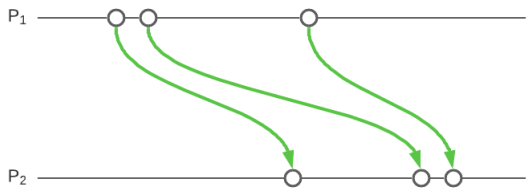
1. **Coordinated checkpointing:** processes coordinate checkpointing to create a global consistent state
2. **Communication induced checkpointing:** processes piggyback recovery protocol information with messages based on which processes decide whether to checkpoint to ensure a possibility of an easy global recovery

2 Lecture (19 January 2021)

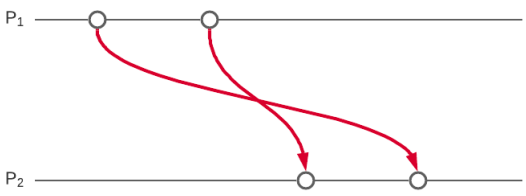
2.1 Recap

2.1.1 Message Ordering

Definition 11 (First-In First-Out)**.** All the messages that are sent from process i to process j are received in the order in which they were sent.



(a) valid example of FIFO communication
All the messages are received at P_2 in the order they were sent by P_1.



(b) example of non-FIFO communication
The message that was sent first by P_1 arrives at P_2 after the second message sent by P_1 arrives at P_2. Therefore, the messages are not received in the order they were sent.

2.1.2 Global State and Consistency

Definition 12 (Global State)**.** Global State of a distributed system is a collection of the local states of its components: constituent processes and communication channels. The local state of a process encompasses the local memory and the other state elements such as registers (the exact constitutions of a local state depend on the component in consideration). The state of a communication consists of messages that are *in transit*.

$$GS = \left\{ \bigcup_{i} LS_i, \bigcup_{i,j} SC_{ij} \right\}$$

where GS is a global state, LS_i is the local state of process i, SC_{ij} is the channel state of a unidirectional channel C_{ij} from process i to process j.

Remark. A global state is also known as global snapshot.

Definition 13 (Consistent Global State)**.** A global state is a consistent global state if and only if it satisfies the following conditions:

1. $send(m_{ij}) \in LS_i \implies m_{ij} \in SC_{ij} \oplus rec(m_{ij}) \in LS_j$
2. $send(m_{ij}) \notin LS_i \implies m_{ij} \notin SC_{ij} \wedge rec(m_{ij}) \notin LS_j$

where m_{ij} is a message sent from process i to process j, LS_i is the local state of process i, SC_{ij} is the communication channel between process i and process j, *send* denotes a send event and *rec* denotes a receive event.

The first condition states that a message $_{ij}$ that was sent by a process i must either be in transit in a communication channel SC_{ij} or must have been received by the process j. The second condition states that a message that every message in a communication channel that was received must have been sent by some process.

Remark. An inconsistent state is not meaningful; a distributed system can never be in an inconsistent global state. However, processes taking saving their states at different physical times can lead to inconsistent global states.

2.1.3 Issues for an snapshot saving algorithm to take care of

1. **Which messages should be recorded?** All messages that were sent before a process took its snapshot must be recorded.
2. **Should a recorded message be stored in the local state or in the channel state?** A message that needs to be saved must be recorded as in transit if it arrived after the destination process took its snapshot; otherwise, it's part of the sender process' and receiving process' local state.
3. **How to determine when a process must take its snapshot?** A process must save its snapshot before processing a message that was sent by another process after it saved its snapshot.

2.2 Chandy-Lamport Algorithm

Chandy-Lamport algorithm records a consistent global snapshot of the distributed system if all communication channels follow FIFO message ordering. The algorithm sends special messages in a specific manner to coordinate the snapshot saving process and ensure consistency. These special messages are called **marker messages**.

2.2.1 The Algorithm

A process initiates — also known as the *initiator process* — the snapshot collection by executing the marker sending rule. The *marker sending rule* dictates the behavior of a process upon receiving a marker message.

Rule 1 (Marker Sending Rule)**.** rule for process P

1. P saves its local state
2. P sends a marker message on each outgoing channel if it hasn't already been sent before sending further messages

Rule 2 (Marker Receiving Rule)**.** rule when process P_i receives a marker message from P_j

When process P_i receives a marker message from process P_j:

if the process has already saved its local state **then**

record the set of messages received through C_{ji} after P_i saved its local state and before this marker message in SC_{ji}

else

execute the Marker Sending Rule

The algorithm terminates when every process has received a marker message on all its incoming channels and all processes have received the local state from every other process (which is required for a process to know the global state). Multiple processes can initiate the snapshot collection concurrently but each initiation must use a unique set of marker messages (identified by a sequence number for example).

Main Idea

We distinguish between three kinds of messages from the algorithm's perspective:

1. **past messages:** messages recorded in both sender's and receiver's local state
2. **in-transit messages:** messages that are recorded in sender's local state but not in receiver's local state
3. **future messages:** messages that are not recorded

Of all the messages that the sender sent before saving their local state, some of them have already been received and some of them are in-transit. All the messages that have been received are recorded in the local state of the receiver. Among the messages that are yet to arrive, we must identify the messages that were sent before and after the sender saved its local state. The former set of messages need to be saved. The marker message is immediately sent after the sender sends after saving the local state. Therefore, all the messages that were received before the marker message were sent before the sender saved their local state. The marker essentially "marks" the position in the FIFO channel that separates the messages that were sent before and after the local state of the sender was saved.

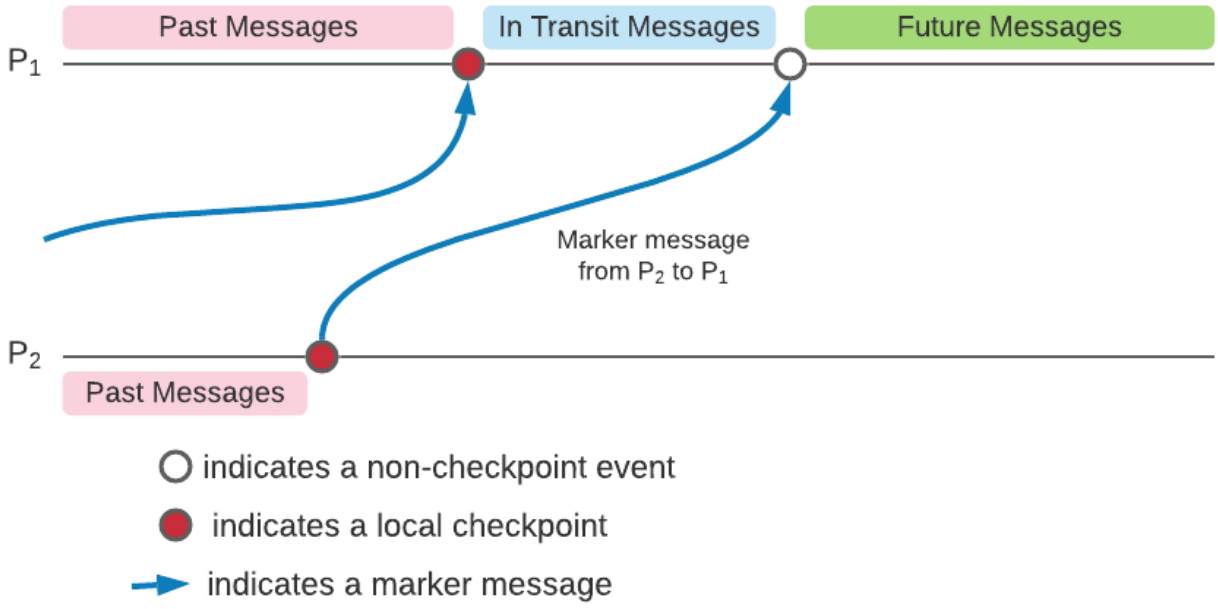


Figure 2

2.2.2 Analysis

The final consistent global state need not correspond to a real physical state as shown in the figure 3. The local states and channel states are saved at different points in physical time.

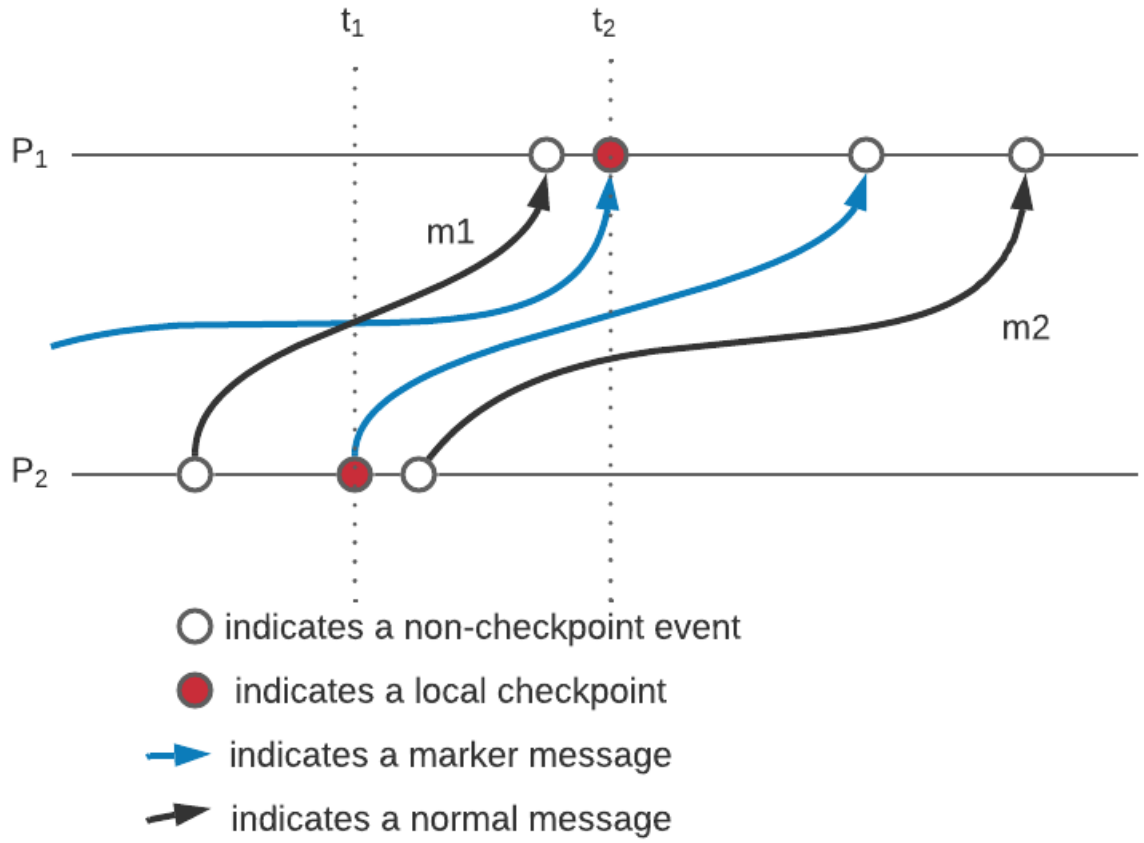


Figure 3: Recorded global state is not a physical state of the system
The final snapshot has $m1$ as received but no mention of $m2$. There is no real physical state where $m1$ has been received and $m2$ hasn't been sent yet.

2.2.3 Correctness

The correctness of the algorithm by showing that the conditions required by the definition of consistent global state are satisfied. Any sent message that is not in the local state of the sender was sent after the local state was saved and hence after the marker message. Only the messages that arrive before the marker message are saved (by FIFO property of the channels). Therefore, second condition of the definition is satisfied. Every message that is in the local state of the sending process either arrived before the receiver saved their local state or arrives after but before the marker message. The receiving process saves the former set of messages in its local state and the latter set of messages in the channel state. Therefore, every message that is registered as sent in the sender's local state is accounted for. This ensures that the first condition is satisfied.

2.2.4 Complexity

> "The recording part of a single instance of the algorithm requires $O(e)$ messages and $O(d)$ time, where e is the number of edges in the network and d is the diameter of the network."
>
> - Page 95, "Distributed Computing Principles, Algorithms, and Systems"

2.2.5 Questions and Answers

When a process Q records its state and further sends the marker to other processes, does that include the initiator also?
Yes, the marker message is sent through to outgoing channel and one of them could be to the initiator process. At the end, every process would have sent a marker message in all its outgoing channels.

What is the purpose of the initiator?

The initiator process triggers the global snapshot collection by broadcasting the marker message to everyone without nudge. Any process can be the initiator. Multiple processes can also initiate the snapshot collection concurrently and save multiple global snapshots.

Does the algorithm always require FIFO message ordering?

Yes. Otherwise, a message sent before the marker message could arrive after the marker message and the receiving will not consider this out-of-order message as in-transit (thereby, violating the conditions in the definition of consistent global state).

What is the relationship between receiving process P_j's first marker, marker from P_i and messages in C_{ij}?

All messages between the two markers are considered in-transit messages in C_{ij} and are stored in SC_{ij}.

Where will messages from $P_j \rightarrow P_i$ be recorded in global state?

They will be recorded at P_i's end. The messages that arrive before P_i saved its local state will be stored in the local state and the messages that arrive between the time P_i saved its local state and the marker message from P_j will be saved in SC_{ji}.

Why concerned about putting marker instead of considering them as transit messages?

We only need to save messages that were sent before the sending process saved its local state. The marker helps separate the messages that were sent before and after the sender saved its local state.

Will Chandy-Lamport algorithm work for non-FIFO channels?

No. The FIFO message ordering is required to distinguish between messages sent before and after the sender saved their state using a marker message.