Distributed Systems Scribe: Lecture 4

Virtual Time

Team no. 7

Siddharth Jain - 2018101038

Aditya Gupta - 2019201067

Shreyas Shankar - 20171128

Presentation Scribes

Introduction

- Virtual Time System is a paradigm for synchronization of distributed systems.
- Virtual time is a global , one-dimensional coordinate system, which executes in coordination with a “virtual clock”.
- Virtual time is implemented as a collection of several loosely synchronized local virtual clocks. As a rule, these local virtual clocks move forward to higher virtual times; however, occasionally they move backwards.
- Virtual time may be related to real time or may be independent of it.

Message Passing

- Processes run concurrently and communicate by exchanging messages, characterized by four values:
    - Name of Sender
    - Virtual Send time
    - Name of Receiver

    - Virtual Receive Time
- Here, conflicts arise when the virtual receive time of the message is less than the local virtual time at the receiver process when the message arrives.

Rules of Virtual Time Systems

- **Rule 1 :** Virtual send time of each message < virtual receive time of that message.
- **Rule 2** : Virtual time of each event in a process < virtual time of next event in that process.

Time Warp Mechanism

The time warp mechanism consists of two major parts: local control mechanism and global control mechanism.

The local control mechanism ensures that events are executed and messages are processed in the correct order.

The global control mechanism takes care of global issues such as global progress, termination detection, I/O error handling, flow control, etc

Local Control Mechanism

- Each Process has a local virtual clock variable. The local virtual clock of a process doesn't change during an event at that process but it changes only between events.
- On the processing of the next message from the input queue, the process increases its local clock to the timestamp of the message.
- At any instant, the value of virtual time may differ for each process but the value is transparent to other processes in the system.

- All arriving messages at a process are stored in an input queue in increasing order of timestamp (receive times).
- Ideally, no messages from the past (called late messages) should arrive at a process. However, processes will receive late messages due to factors such as different computation rates of processes and network delays.
- On the reception of a late message, the receiver rolls back to an earlier virtual time, cancelling all intermediate side effects and then executes forward again by executing the late message in the proper sequence.
- This is described by saying that each process is doing a constant “lookahead,” processing future messages from its input queue.

Antimessages and the rollback mechanism

- Rollback in a distributed system is complicated by the fact that the process that wants to rollback might have sent many messages to other processes, which in turn might have sent many messages to other processes, and so on, leading to deep side effects. For rollback, messages must be effectively “unsent” and their side effects should be undone. This is achieved efficiently by using antimessages.
- For every message, there exists an anti message that is the same in content but opposite in sign. Whenever a process sends a message, a copy of the message is transmitted to the receiver’s input queue and a negative copy (antimessage) is retained in the sender’s output queue for use in sender rollback.
- Whenever a message and its anti message appear in the same queue, regardless of the order in which they arrived, they immediately annihilate each other resulting in shortening of the queue by one message.

Global Control Mechanism

- Handles issues like errors, I/O handling, running out of memory.
- Uses a Global Virtual Time, which is a property of an instantaneous global snapshot of the system at real time “r”.
- Global virtual time (GVT) at real time r is the minimum of:
    - All virtual times in all virtual clocks at time r; and
    - The virtual send times of all messages that have been sent but have not yet been processed at time “r”.
- If every event completes normally, and if messages are delivered reliably, and if there is sufficient memory, then GVT must eventually increase.
- GVT serves as a floor for the virtual times to which any process can ever again roll back.

Use of GVT

- Memory Management: The time warp mechanism uses the concept of fossil detection where information older than GVT is destroyed to avoid memory overheads.
    - Old states in the state queues.
    - Messages stored in output queues.
    - Past messages (in input queues) that have already been processed.
    - Future messages (in input queues) that have not yet been received.
- Snapshots and crash recovery : An entire snapshot of the system at virtual time “t” can be constructed by a procedure in which each process “snapshots” itself as it passes virtual time t in the forward direction and “snapshots” itself whenever it rolls back over virtual time “t”. Whenever GVT exceeds “t,” the snapshot is complete and valid.

References

1. http://cobweb.cs.uga.edu/~maria/pads/papers/p404-jefferson.pdf
2. https://www.youtube.com/watch?v=Gr1EF_CUUQA&ab_channel=DistributedSystems
3. Distributed Computing Textbook (Kshemkalyani and Singhal)

Question:

1. What is the difference between lamport's scheme and virtual time?

   In lamport's scheme all clocks are conservatively maintained so that they never violate causality a process advances its clock as soon as it sees new causal dependency while in virtual time it advances only when there is conflict or violation.

Lecture Scribes

Chandy–Lamport's global state recording algorithm

Each distributed system has a number of processes running on a number of different physical servers. These processes communicate with each other via communication channels using text messaging. These processes neither have a shared memory nor a common physical clock, this makes the process of determining the instantaneous global state difficult.

A process could record it own local state at a given time but the messages that are in transit (on its way to be delivered) would not be included in the recorded state and hence the actual state of the system would be incorrect after the time in transit message is delivered.

Assumptions of the algorithm:

- There are a finite number of processes in the distributed system and they do not share memory and clocks.
- There are a finite number of communication channels and they are unidirectional and FIFO ordered.
- There exists a communication path between any two processes in the system
- On a channel, messages are received in the same order as they are sent.

Algorithm:

Marker sending rule for a process P:

- Process **p** records its own local state
- For each outgoing channel **C** from process **P**,**P** sends marker along **C** before sending any other messages along **C**.

  Note: Process Q will receive this marker on his incoming channel C1.

Marker receiving rule for a process Q:

- If process **Q** has not yet recorded its own local state then
- Record the state of incoming channel C1 as an empty sequence or null.
- After recording the state of incoming channel C1, process **Q** Follows the marker sending rule
- If process **Q** has already recorded its state
- Record the state of incoming channel **C1** as the sequence of messages received along channel C1 after the state of **Q** was recorded and before **Q** received the marker along C1 from process P.

Need of taking snapshot or recording global state of the system:

- **Checkpointing:** It helps in creating a checkpoint. If somehow application fails, this checkpoint can be used re
- **Garbage collection:** It can be used to remove objects that do not have any references.
- It can be used in deadlock and termination detection.
- It is also helpful in other debugging.

Interesting Discussions:

Q.) Will the algorithm work for NON FIFO channels?

Ans.- Consider a process P who is just about to receive his first marker from X. A message sent by X after the marker msg arrives at P first due to non FIFO property. Thus, condition C2 is satisfied where msg not sent is received before snapshot is taken.

Consider a process P which has already marked its state. Now say it is about to receive a marker from process Q. If the channel was not FIFO then the msgs sent from Q aftermarket would reach before the marker and get put into the transit messages list for channel C_{QP}. This defies condition C2 where a message not sent lies in transit.

Q.) When should the algorithm terminate ?

Ans.- The algorithm terminates after each process has received a marker on all of its incoming channels.All the local snapshots get disseminated to all other processes and all the processes can determine the global state.

Homework Questions:

Multiple processes can initiate the algorithm concurrently- Read page 94 of the Distributed Computing Textbook (Kshemkalyani and Singhal).