By Nathan Harvey (997261) and Ziyu Wang (1560831)

Snapshots are used to record a consistent global state of an asynchronous distributed system. This is essential for debugging distributed software, failure recovery, protocol verification, and ensuring correctness of concurrent executions. However, achieving this consistency is non-trivial due to the absence of shared memory and a global clock. Each process operates independently and messages may be delayed. As a result, snapshot implementation must address two main issues:

(I1) Message Inclusion: deciding which messages belong to the snapshot.
(I2) When to Snapshot: determining when each process should record its state.

We selected Venkatesan's Incremental Snapshot method, a variation of Chandy-Lamport algorithm designed for the system with FIFO channels. This algorithm structures a spanning tree from the initiator and recursively aggregates local and channel state information from all its descendants. Unlike Chandy-Lamport, which sends marker to all outgoing channels, Venkatesan's algorithm optimizes communications by sending control messages only to the channels that have been used since the snapshot began. In addition, it piggybacks snapshot id to record in-transit messages, further minimizing the extra messages.

We chose Venkatesan's algorithm for several reasons. Firstly, the FIFO assumption aligns with widely used transport protocols like TCP. Furthermore, the method is memory and communication efficient since it only sends markers to active channels and piggybacks messages in transit. Last but not least, this algorithm supports non-intrusive repeated snapshots, making it highly suitable for systems that require frequent recovery points.

The original paper on snapshot algorithms (Chandy & Lamport, 1985) presents the snapshot algorithm introduced in our lectures, the Chandy-Lamport algorithm. They state its purpose is to detect stable properties in distributed systems, which are properties of a system that once true in a state imply the property will be true in all future states. Venkatesan (1989) introduces an extension to the Chandy-Lamport algorithm that focuses on using incremental snapshots to make assumptions about messages sent before the snapshot, and to send the optimal number of control messages. It also introduces using snapshots for the purpose of crash recovery. Helary (1989) relaxes the FIFO restriction, but requires freezing execution. Mattern (1993) and Lai and Yang (1987) both also relax the FIFO restriction, but does so without needing to freeze execution by piggybacking existing messages. Mattern (1993) references Taylor (1989) as stating that to relax the FIFO restriction you need to use one of these methods.

For our distributed application we are modelling a financial system, such as a stock brokering system. The system is started with a global config that defines the starting amount of money for each process, the total amount of money in the system, and the connections between processes. Processes join independently, and try to contact connected processes. The second process in each connection will open the bidirectional socket. Processes can send money to directly connected processes.

Using a global snapshot algorithm has the benefit of keeping the total amount of money in the system constant. If processes were to take local snapshots, when the system restarted after a crash the total amount of money in the system might change. This is because local snapshots can't accurately capture the state of channels between processes, with messages able to be either recorded twice by both processes involved, or neither processes.

References

Chandy, K. M., & Lamport, L.. (1985). Distributed snapshots. *ACM Transactions on Computer Systems*, *3*(1), 63–75. https://doi.org/10.1145/214451.214456

Venkatesan, S.. (1989). *Message-optimal incremental snapshots*. 53–60. https://doi.org/10.1109/icdcs.1989.37930

Helary, J.-M.. (1989). Observing global states of asynchronous distributed applications. In *Lecture Notes in Computer Science* (pp. 124–135). Lecture Notes in Computer Science. https://doi.org/10.1007/3-540-51687-5_37

Lai, T. H., & Yang, T. H. (1987). On distributed snapshots. *Information Processing Letters*, *25*(3), 153-158. https://doi.org/10.1016/0020-0190(87)90125-6

Mattern, F.. (1993). Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation. *Journal of Parallel and Distributed Computing*, *18*(4), 423–434. https://doi.org/10.1006/jpdc.1993.1075

Taylor, K.. (1989). The role of inhibition in asynchronous consistent-cut protocols. In *Lecture Notes in Computer Science* (pp. 280–291). Lecture Notes in Computer Science. https://doi.org/10.1007/3-540-51687-5_50

Nathan

- Summary of background survey done so far.
    - Chandy-Lamport Snapshot Algorithm
    - Venkatesan, S algorithm
    - Lai, T. H., & Yang, T. H
    - Helary, J.-M
    - Mattern, F.
    - Maybe Acharya, A., & Badrinath, B.R Alagar, S., & Venkatesan, S, I haven't looked into them yet
- Description of the distributed application.
    - Distributed bank with multiple branches
    - Primary process starts the system, other processes join by providing an ip of 1 or more existing processes in the system
        - Primary process defines how much money the system starts with, other processes start with 0
        - Could also have all processes start with 0 and go with a "borrowing" model (processes can send any amount of money, and can have a negative amount of money)
    - Processes (bank branches / entities) can send money to any connected process (that they initially connected to, or that connected to them after startup)
    - Need to think about the order of processes coming back online
        - Maybe processes poll a process if it isn't online yet
    - System always has the same amount of total money
- Explanation on how the algorithm benefits or supports the distributed application.
    - Allows to take globally consistent snapshots of the system, so that if the system crashes you can recover the system
    - If algorithm isn't implemented, an inconsistent global state might be recovered which has a different amount of total money in the system
    -

Chandy, K. M., & Lamport, L.. (1985). Distributed snapshots. *ACM Transactions on Computer Systems*, *3*(1), 63–75. https://doi.org/10.1145/214451.214456

Venkatesan, S.. (1989). *Message-optimal incremental snapshots*. 53–60. https://doi.org/10.1109/icdcs.1989.37930

Helary, J.-M.. (1989). Observing global states of asynchronous distributed applications. In *Lecture Notes in Computer Science* (pp. 124–135). Lecture Notes in Computer Science. https://doi.org/10.1007/3-540-51687-5_37

Lai, T. H., & Yang, T. H. (1987). On distributed snapshots. *Information Processing Letters*, *25*(3), 153-158. https://doi.org/10.1016/0020-0190(87)90125-6

Mattern, F.. (1993). Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation. *Journal of Parallel and Distributed Computing*, *18*(4), 423–434. https://doi.org/10.1006/jpdc.1993.1075

Taylor, K.. (1989). The role of inhibition in asynchronous consistent-cut protocols. In *Lecture Notes in Computer Science* (pp. 280–291). Lecture Notes in Computer Science. https://doi.org/10.1007/3-540-51687-5_50

Snapshot

1. Purpose: consistent global status of a distri. sys.
2. Difficulties: no shared memory and no global clock (asynchronized)
3. Issues: message inclusion *(I1)* + when to snapshot *(I2)*

Algorithms

1. **FIFO** model – bank transaction
    a. Chandy-Lamport – Marker Receiving Rule
    b. Variations
        i. Spezialetti–Kearns method
           – concurrent initiation + improve record efficiency
        ii. Venkatesan's <u>incremental</u> snapshot method
           – repeated snapshots (recovery algo with synchronous checkpointing)
        iii. Helary's wave synchronization method
           – global synchronization (suitable for replay)
2. Non-FIFO model – udp / IoT
    a. Lai-Yang
    b. Mattern's (+Vector Clock)
3. Causal Ordering – message or google doc.
    a. Acharya-Badrinath
    b. Alagar-Venkatesan

Applications

1. Bank with multiple branches or other transactions
   online banking sys; processor state: fund; message: electronic funds in transit
2. Data Failure Recovery

Venkatesan Incremental Snapshot

1. spanning tree [*initialize*]
2. send `init_snap` to children and `marker` to outgoing message destinations
3. wait `snap_completed` and `ack` message
4. send the current snapshot to its parent until to initiator

Advantages

1. Lower communication overhead: only send `marker` to the message in transit node to reduce message amount
2. Suitable for repeated snapshots
3. Support global consistency without sys pause

Chandy, K. M., & Lamport, L.. (1985). Distributed snapshots. *ACM Transactions on Computer Systems*, *3*(1), 63–75. https://doi.org/10.1145/214451.214456

- No shared clock or memory
- Really good analogy
    - “The state-detection algorithm plays the role of a group of photographers observing a panoramic, dynamic scene, such as a sky filled with migrating birds- a scene so vast that it cannot be captured by a single photograph. The photographers must take several snapshots and piece the snapshots together to form a picture of the overall scene. The snapshots cannot all be taken at precisely the same instant because of synchronization problems. Furthermore, the photographers should not disturb the process that is being photographed; for instance, they cannot get all the birds in the heavens to remain motionless while the photographs are taken. Yet, the composite picture should be meaningful. The problem before us is to define “meaningful” and then to determine how the photographs should be taken.”
- Used to solve if stable properties hold
    - Database deadlocks, computation has terminated
    - Any property where it being true in the current state implies it is true in every subsequent state
- The purpose of the algorithm seems to be to detect whether a stable property is true

Spezialetti, M. & Kearns, P. (1986). Efficient Distributed Snapshots.. *ICDCS* (p./pp. 382-388), : IEEE Computer Society. ISBN: 0-8186-0697-5

- I can’t find a copy of this online, closest I got is https://dblp.uni-trier.de/db/conf/icdcs/icdcs86.html#SpezialettiK86

Venkatesan, S.. (n.d.). *Message-optimal incremental snapshots*. 53–60. https://doi.org/10.1109/icdcs.1989.37930

- Purpose is debugging distributed systems, discarding obsolete information??, monitor distributed events, set distributed breakpoints and halt, protocol verification, detect stable properties, record checkpoint, take a log of distributed system, crash recovery
- Algorithm provides incremental snapshots
- Is asymptotically message optimal
- Channels considered bidirectional
- Assumes a primary node initiates all snapshots, but can be relaxed
    - Deeply annoyingly the relaxation cited is Spezialetti, M. & Kearns, which I have already failed to find online. I will have to have another look, because I think that is quite important if we do an impl.
- Sends fewer messages by knowing about which channels have had messages sent on them since the last snapshot
- This one seems like a good candidate for implementation, fairly simple in concept and has a strong use case (snapshot backups for recovery of a banking system)

Helary, J.-M.. (1989). Observing global states of asynchronous distributed applications. In *Lecture Notes in Computer Science* (pp. 124–135). Lecture Notes in Computer Science. https://doi.org/10.1007/3-540-51687-5_37

- Includes freezing computation
- References waves: Helary, J.-M., Jard, C., Plouzeau, N., & Raynal, M.. (1987). *Detection of stable properties in distributed applications*. 125–136. https://doi.org/10.1145/41840.41851
- Which references waves: Chandy, M., & Misra, J.. (1986). An example of stepwise refinement of distributed programs: quiescence detection. *ACM Transactions on Programming Languages and Systems*, *8*(3), 326–343. https://doi.org/10.1145/5956.5958
    - Doesn't mention waves, but describes a marker algorithm for distributed systems

Lai, T. H., & Yang, T. H. (1987). On distributed snapshots. *Information Processing Letters*, *25*(3), 153-158. https://doi.org/10.1016/0020-0190(87)90125-6

- Proposes a method using no control messages
- Only concerned with taking local snapshots, and using them to prove stable properties.
- Not useful for our application as there are no global snapshots
- Markers piggyback on other outgoing messages, and if no outgoing messages are going to be sent we can easily assume no outgoing messages are going to be received
- Makes messages colour coded, alternating white and red
- All processes start white
- When a process receives a red message (or earlier) take a local snapshot
- Messages need to contain the complete message history inside them (or since the last snapshot), which is limiting
    - For some applications (deadlock or termination) you only need the number of messages sent, not the messages themselves
-

Hon Fung Li, Thiruvengadam Radhakrishnan, K. Venkatesh: Global State Detection in Non-FIFO Networks. ICDCS 1987: 364-370

- Can't find this one either https://dblp.org/db/conf/icdcs/icdcs87.html#LiRV87

Mattern, F.. (1993). Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation. *Journal of Parallel and Distributed Computing*, *18*(4), 423–434. https://doi.org/10.1006/jpdc.1993.1075

- States that non-fifo channels require either inhibition (delaying actions, Helary 1989) or piggybacking off of other messages for control messages (Lai and Yang 1987)
    - Taylor, K.. (1989). The role of inhibition in asynchronous consistent-cut protocols. In *Lecture Notes in Computer Science* (pp. 280–291). Lecture Notes in Computer Science. https://doi.org/10.1007/3-540-51687-5_50
- Also uses a red/white system, and introduces concept of a third colour
    - Limit which states a process can receive messages from, if it receives a message from the next state it increments its colour and takes a snapshot

- Algorithm for approximating global virtual time of distributed system
- Make 2 cuts, making sure no message crosses both cuts, to allow ignoring messages that cross a cut the wrong way
    - Minimum timestamp of messages that cross second cut is the first cut

Acharya, A., & Badrinath, B.R. (1992). Recording Distributed Snapshots Based on Causal Order of Message Delivery. *Inf. Process. Lett., 44*, 317-321. https://doi.org/10.1016/0020-0190(92)90107-7

Alagar, S., & Venkatesan, S. (1994). An Optimal Algorithm for Distributed Snapshots with Causal Message Ordering. *Inf. Process. Lett., 50*, 311-316. https://doi.org/10.1016/0020-0190(94)00055-7

Kshemkalyani, A. D., Raynal, M., & Singhal, M. (1995). An introduction to snapshot algorithms in distributed computing. *Distributed Systems Engineering, 2*(4), 224–233. https://doi.org/10.1088/0967-1846/2/4/005