Report on Analysis of Singhal - Kshemkalyani's Vector Clock

Shanmuga Ganesh
04/24/2024

1. Objective

Examining the Singhal-KShemKalyani vector clock algorithm involves altering the number of processes to gauge its efficacy. Evaluation criteria encompass the maximum and average messages generated during send events, alongside the modifications occurring in receive events. Such analysis offers insights into the algorithm's performance across different scalability scenarios, elucidating its behavior under varying process counts.

2. Algorithmic Background

2.1. Random Flooding

The Random Flooding algorithm, a decentralized communication method in distributed systems, facilitates broadcasting messages efficiently while ensuring coverage and scalability. This algorithm operates by randomly selecting neighboring nodes for message propagation, thereby disseminating information throughout the network. It balances the trade-off between efficiency and overhead by optimizing message dissemination while minimizing redundant transmissions. This approach enables widespread coverage of messages across the network, promoting scalability as the system grows in size. However, the randomness inherent in the algorithm introduces a degree of overhead, which must be managed to maintain efficiency in message delivery.

2.2. Singhal-KShemKalyani's Vector Clock

Singhal-KShemKalyani's Vector Clock algorithm plays a pivotal role in distributed systems by providing a mechanism for reasoning about causality and ensuring the consistency of events. At its core, it facilitates event ordering through a partial ordering of events, enabling the system to discern causal relationships among them. Moreover, the algorithm adeptly detects concurrent events, identifying instances where two events neither precede nor succeed each other in the sequence. By leveraging these capabilities, Singhal-KShemKalyani's algorithm establishes a robust framework for maintaining coherence in distributed environments, thus bolstering the reliability and integrity of system operations.

3. Metrics for Performance Evaluation

3.1. Message Construction

Selecting metrics for performance evaluation involves measuring the message construction length in terms of both maximum and average values. This evaluation is conducted while varying the number of processes. Specifically, when a process sends a message to another process, metrics focus on the message content, ensuring that it adheres to the criteria such as (pid_i, ts_i) for all processes i and satisfies the condition $LS_3 < LU_1$ for instance, ensuring correct event ordering.

3.2. Message Updation

Selecting metrics for performance evaluation is crucial. One such metric pertains to message update length, both in terms of maximum and average values. This entails assessing the length of updates made when a process receives a message from another process. For instance, when process P_1 receives a message from P_3 containing timestamp pairs { (pid_i, ts_i) }, it updates its own timestamp (TS_i) if and only if $ts_i > TS_i$ for all elements pidi in the message. By varying the number of processes, analyzing maximum and average message update lengths offers insights into algorithm efficiency and scalability.

4. Understanding the Result

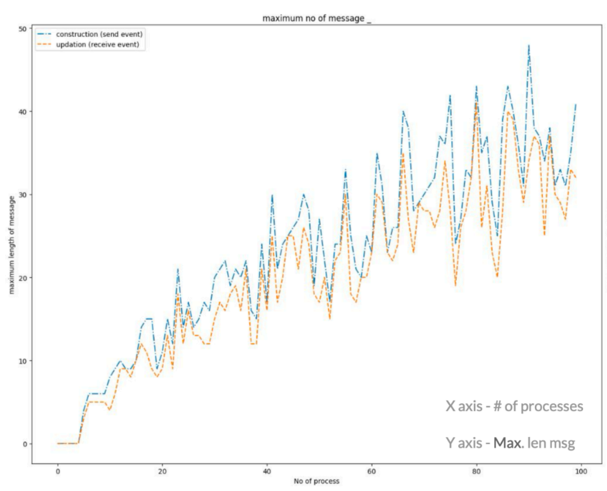
The result indicates that, on average, the length of messages constructed during message construction is approximately 1.27 times greater than the length of messages updated during message updation. This discrepancy stems from the nature of message construction, where each message includes the sender's vector clock encompassing timestamps for all processes, possibly updated previously. During updation, only the greater value of each entry received is considered. Consequently, due to higher timestamp values, only some messages received are updated. This disparity highlights the impact of message construction and updation processes on the overall message lengths in the system.

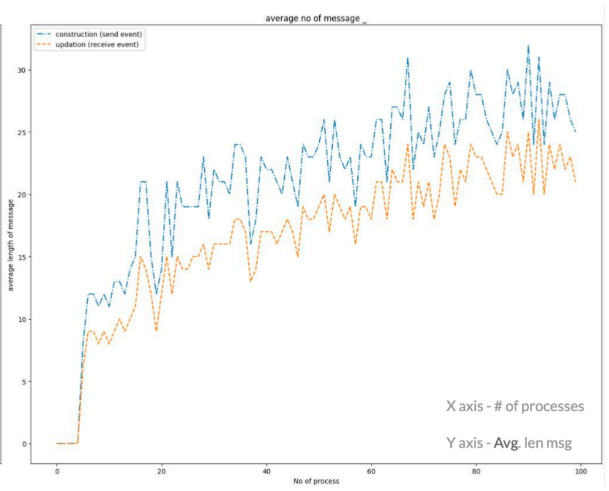
5. Graphical Representation

The visualization illustrates the behavior of message construction and updation lengths in a distributed system across varying numbers of processes. Two plots are presented side by side: the first depicts the maximum lengths of messages for construction (send event) and updation (receive event), while the second illustrates the average lengths of messages for both events. The x-axis represents the number of processes, while the y-axis denotes the maximum or average length of messages. The comparison enables an understanding of how these metrics evolve with increasing process counts, offering insights into the algorithm's performance and scalability.

6. Future Work

Future work entails exploring non-clique topologies such as DFS spanning trees to enhance the realism of distributed algorithms. Additionally, leveraging multi-threading will simulate concurrent conditions more accurately, enabling a deeper understanding of algorithm behavior in complex distributed systems.


maximum no of message _
construction (send event)
updation (receive event)
maximum length of message
No of process
X axis - # of processes
Y axis - Max. len msg


average no of message _
construction (send event)
updation (receive event)
average length of message
No of process
X axis - # of processes
Y axis - Avg. len msg