Efficient Implementations of Vector Clocks

- If the number of processes in a distributed computation is large, then vector clocks will require piggybacking of huge amount of information in messages.
- The message overhead grows linearly with the number of processors in the system and when there are thousands of processors in the system, the message size becomes huge even if there are only a few events occurring in few processors.
- We discuss an efficient way to maintain vector clocks.
- Charron-Bost showed that if vector clocks have to satisfy the strong consistency property, then in general vector timestamps must be at least of size *n*, the total number of processes.
- However, optimizations are possible and next, and we discuss a technique to implement vector clocks efficiently.

Singhal-Kshemkalyani's Differential Technique

- *Singhal-Kshemkalyani's differential technique* is based on the observation that between successive message sends to the same process, only a few entries of the vector clock at the sender process are likely to change.
- When a process p_i sends a message to a process p_j, it piggybacks only those entries of its vector clock that differ since the last message sent to p_j.
- If entries $i_1, i_2, \ldots, i_{n_1}$ of the vector clock at p_i have changed to $v_1, v_2, \ldots, v_{n_1}$, respectively, since the last message sent to p_j, then process p_i piggybacks a compressed timestamp of the form:

$$\{(i_1, v_1), (i_2, v_2), \ldots, (i_{n_1}, v_{n_1})\}$$

to the next message to p_j.

Singhal-Kshemkalyani's Differential Technique

When p_j receives this message, it updates its vector clock as follows:

$$vt_i[i_k] = max(vt_i[i_k], v_k) \text{ for } k = 1, 2, \ldots, n_1.$$

- Thus this technique cuts down the message size, communication bandwidth and buffer (to store messages) requirements.
- In the worst of case, every element of the vector clock has been updated at p_i since the last message to process p_j, and the next message from p_i to p_j will need to carry the entire vector timestamp of size n.
- However, on the average the size of the timestamp on a message will be less than n.

Singhal-Kshemkalyani's Differential Technique

- Implementation of this technique requires each process to remember the vector timestamp in the message last sent to every other process.
- Direct implementation of this will result in $O(n^2)$ storage overhead at each process.
- Singhal and Kshemkalyani developed a clever technique that cuts down this storage overhead at each process to $O(n)$. The technique works in the following manner:
- Process p_i maintains the following two additional vectors:
  - $LS_i[1..n]$ ('Last Sent'):
    $LS_i[j]$ indicates the value of $vt_i[i]$ when process p_i last sent a message to process p_j.
  - $LU_i[1..n]$ ('Last Update'):
    $LU_i[j]$ indicates the value of $vt_i[i]$ when process p_i last updated the entry $vt_i[j]$.
- Clearly, $LU_i[i] = vt_i[i]$ at all times and $LU_i[j]$ needs to be updated only when the receipt of a message causes p_i to update entry $vt_i[j]$. Also, $LS_i[j]$ needs to be updated only when p_i sends a message to p_j.

Singhal-Kshemkalyani's Differential Technique

- Since the last communication from p_i to p_j, only those elements of vector clock $vt_i[k]$ have changed for which $LS_i[j] < LU_i[k]$ holds.
- Hence, only these elements need to be sent in a message from p_i to p_j. When p_i sends a message to p_j, it sends only a set of tuples
$$\{(x, vt_i[x]) | LS_i[j] < LU_i[x]\}$$
as the vector timestamp to p_j, instead of sending a vector of n entries in a message.
- Thus the entire vector of size n is not sent along with a message. Instead, only the elements in the vector clock that have changed since the last message send to that process are sent in the format $\{(p_1, latest\_value), (p_2, latest\_value), \ldots\}$, where p_i indicates that the p_ith component of the vector clock has changed.
- This technique requires that the communication channels follow FIFO discipline for message delivery.

Singhal-Kshemkalyani's Differential Technique

- This method is illustrated in Figure 3.3. For instance, the second message from p_3 to p_2 (which contains a timestamp $\{(3, 2)\}$) informs p_2 that the third component of the vector clock has been modified and the new value is 2.
- This is because the process p_3 (indicated by the third component of the vector) has advanced its clock value from 1 to 2 since the last message sent to p_2.
- This technique substantially reduces the cost of maintaining vector clocks in large systems, especially if the process interactions exhibit temporal or spatial localities.

Singhal-Kshemkalyani's Differential Technique

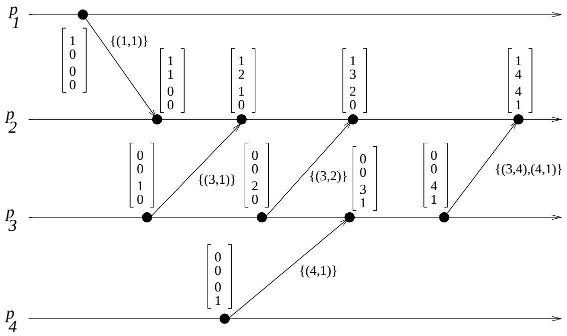

Figure 3.3: Vector clocks progress in Singhal-Kshemkalyani technique.