Distributed Systems Scribe: Lecture 3

Arpan Dasgupta, Shobhit Gautam, Veeravalli Saisooryarao

January 13, 2021

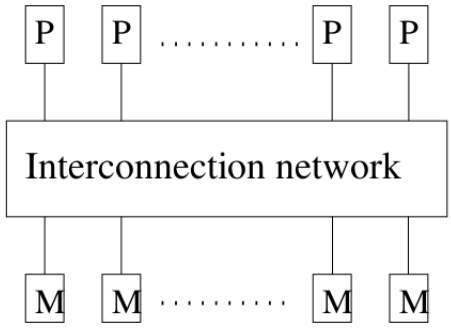
Contents

1 Presentation

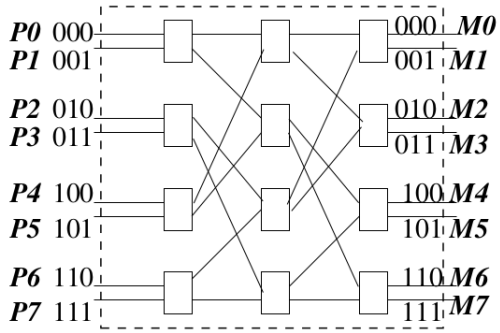
A presentation about **Omega and Butterfly Networks** was given. A summary of the presentation follows with the questions asked after it. The complete presentation can be found separately.

1.1 Multistage Interconnection Networks

- In multiprocessor systems with UMA (Uniform Memory Access), the set of processors are connected to a set of memory addresses via an interconnection network.
- The connection of the processors to the memory blocks can be done in several ways. They are generally done through interconnection patterns using switches.
- There are several interconnection patterns and two of them are omega and butterfly networks.

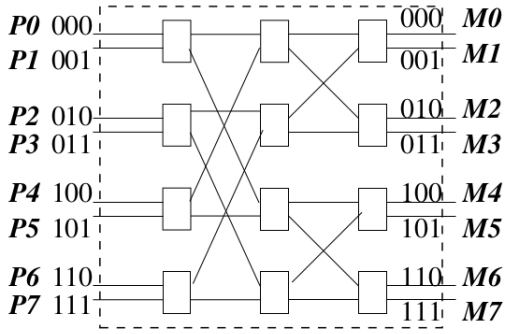


1.2 Omega Network



Omega Routing function - To go from processor i to memory node j, follow the binary representation of j; at stage k, check kth bit of j. Go up if current bit = 0 and go down if bit = 1.

1.3 Butterfly Network



Butterfly Routing function - In a stage s switch, if the $(s+1)$th MSB of j is 0, the data is routed to the upper output wire, otherwise it is routed to the lower output wire.

1.4 Comparison and Facts

1. The butterfly and omega networks can be proven to be isomorphic.
2. These interconnection networks are blocking.

3. These networks connect every processor to memory location and vice versa.

4. Routing in the 2x2 switch at stage k uses only the kth bit, and hence can be done at clock speed in hardware.

1.5 Questions

There were three questions asked verbally and one in chat.

- **Why is there a need for the routing function, is the network itself not sufficient?**
  Ans - If I want to get from processor i to processor j, I need to know the path to take (up or down) at each node to get to my destination. Why we need a routing function to define it for us and not leave it to some search algorithm is to allow hardware to do it at clock speed maintaining the lowest possible latency. Additionally, only 1 path from P_i to M_j is possible so trying to discover it at runtime does not make sense.

- **Why is using an interconnection network like this necessary, why not connect each pair directly?**
  Ans- The number of connections for a point-to-point network is n^2, but for butterfly and omega network it is $O(n \log n)$ $[n(\log 2n + 1)$ precisely]. This is a large difference for large values of n.

- **Is there any advantage of point-to-point vs these?**
  Ans - Yes, lower latency (1 edge vs $\log n$ edges between source and destination)

- **How is the destination path decided at each box in omega network?**
  Ans - The destination path at each switch is decided by the previously discussed routing function.

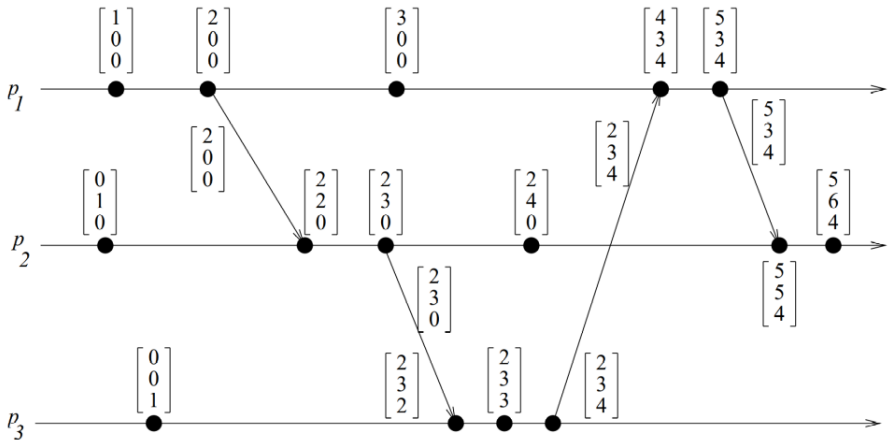
2 Lecture

The lecture was based on the concept of vector time.

2.1 Vector Time

Vector time is represented by n-dimensional non-negative integer vectors. Each process p_i maintains a vector vt_i $i \in [1..n]$, where $vt_i[i]$ is the local logical clock of p_i and describes the logical time progress at process p_i. $vt_i[j]$ represents process p_i 's latest knowledge of process p_j local time. The entire vector shows the global view of time for the process.

The time is updated by taking a max of the time received from the message and that from the event before it in the process and then adding one to the ith element for process i.



For process i, for d=1, are the events internal to the process ordered as a sequence 1,2,3,4,.... Or could there be jumps ?
Ans - No, there can not be jumps as a process cannot receive an updated version of its own local time from a message from another process.

Using vector clocks, can we determine whether the two events are concurrent or whether they are causal ordered?
Ans - Yes, if an event's vector time *strictly dominates* the vector time of another event, then we can say that the first event is causally dependent on the second. Otherwise, we can say that they are concurrent.
Here, *strictly dominates* is defined by - vector a strictly dominates vector b if all elements of a are greater than equal to corresponding elements of b and at least one element of a is greater than the corresponding element of b.

Event Counting - In vector time, how can we find out how many events this event is causally dependent on?
Ans - We can find the number of events one event is causally dependent on by summing up all the elements in the vector time and subtracting 1. (Assuming increment d is 1)

Is it possible for two events to have the same vector clock?
Ans - No, it is not possible for any two events on the same or different processes to have the same vector clock. For two events on the same process, due to addition of 1 to the current element in rule 1, we can never have the same time. For events in different processes (say p_i and p_j, one of the two elements among i or j has to differ.

Vector time, however, needs a lot of memory to be passed around with every message and we need n dimensional vector even if only some processes are active.

Possible optimization - Of the n components of the vector time, only few would have changed since the last time P_i sent a message to P_j. Need to include only those tuples. $(i_1, t_1), (i_2, t_2), \ldots$ However, for this to work, each process has to keep track of what last messages were sent to each of the other processes. Thus memory requirement is $O(n^3)$.

2.1.1 Singhal-Kshemkanlyani Differential Technique

A clever optimization to the previously suggested technique can reduce the required memory greatly.
Process pi maintains the following two additional vectors:
1. Last Sent : $Ls_i[j]$ indicates the value of $vt_i[i]$ when process p_i last sent a message to process p_j.
2. Last Update : $LU_i[j]$ indicates the value of $vt_i[i]$ when process p_i last updated the entry $vt_i[j]$.
$Lu_i[j]$ needs to be updated only when the receipt of a message causes p_i to update entry $vt_i[j]$. $LS_i[j]$ needs to be updated only when p_i sends a message 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.

2.2 Matrix Time

The time is represented by a set of n x n matrices of non-negative integers. A process pi maintains a matrix $mt_i[1..n, 1..n]$ where, $mt_i[i, i]$ denotes the local logical clock of p_i and tracks the progress of the computation at process p_i. $mt_i[i, j]$ denotes the latest knowledge that process p_i has about the local logical clock, $mt_j[j, j]$, of process p_j.
$mt_i[j, k]$ represents the knowledge that process p_i has about the latest knowledge that p_j has about the local logical clock, $mt_k[k, k]$, of p_k.

Process p_i uses below rules to update its clock:
Rule 1 : Before executing an event, process p_i updates its local logical time as follows: $mt_i[i, i] := mt_i[i, i] + d(d > 0)$
Rule 2: Each message m is piggybacked with matrix time mt. When p_i receives such a message (m,mt) from a process p_j, then p_i executes the below sequence: Update its global logical time as follows: (a) $1 \leq k \leq n : mt_i[i, k] := max(mt_i[i, k], mt[j, k])$ (b) $1 \leq k, l \leq n : mt_i[k, l] := max(mt_i[k, l], mt[k, l])$
Execute Rule 1. Deliver message m

2.3 Interesting Discussions

- **How does vector time manage inclusion and failing of nodes?**
  A question by a student 'does vector time limits the system from adding the node dynamically?' led to the discussion about what happens to vector time when a process stops or a new process is to be added.
  This kind of situation of adding nodes happens rarely, but sometimes vector time accommodates for it by starting with extra elements. Further information about this was requested to be researched by the instructor.

- **Can we have a Singhal-Khemkalyani type approach for Matrix time?**
  Since the matrix clock takes even more time, we need optimizations like this to make it viable.

- What will be the event counting algorithm for matrix time?

3 Questions

These questions were given as HW at several points in the class.

- Is vector time strongly consistent? ie. $e_i \rightarrow e_j \iff C(e_i) < C(e_j)$. If so prove, else show a counterexample.
- How do vector clocks work for out of order massages? Do they work if every message is a broadcast message?
- Will Singhal-Khemkalyani technique work if messages are not in FIFO order?
- In the matrix time updation rule (Rule 2) for case $1 \leq k \leq n$, is it feasible to have $mt_i[i,k] := max(mt_i[i,k], max(mt[*,k]))$ instead of $mt_i[i,k] := max(mt_i[i,k], mt[j,k])$