COMPSCI 630 Systems **Spring 2018**

Lecture 20: April 23

Lecturer: Emery Berger *Scribe(s): Abhinav Jangda*

20.1 Clock Synchronization

CPU contains an internal clock, which is powered by a quartz crystal. These clocks have a clock drift of half a second per day. Hence, within 30 days a quartz clock will be 15 seconds ahead or behind the actual clock. These clocks have to be synchronized after some time with a time server. Time servers contains atomic clocks which are several magnitudes accurate than quartz clock. To do synchronization protocols like Network Time Protocol (NTP) is used. However, synchronization by NTP suffers due to problems like network delays. Hence, using absolute time is not meaningful in distributed systems.

20.2 Happens before relation

Happens before relation defines the notion of causality between events. For example, if process P_1 sends a to process P_2 and due to a, P_2 sends b to P_3, then we say that a has happened before b. In other words, a is the cause of b.

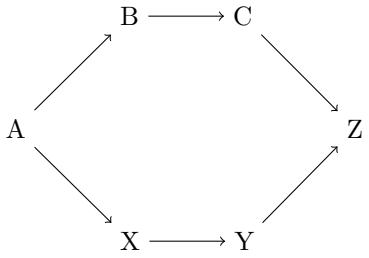

Figure 20.1: Example of Happens before relation

Figure 20.1 shows an example of causality between events `A`, `B`, `C`, `X`, `Y`, `Z`. `A` happens before every other event. `B` happens before `C` and `Z`. However, we cannot say if `X` happens before `B` or not. Hence, happens before relation (also called causal relationship) provides us with partial order but not total order.

20.3 Lamport Clock

Lamport Clock is an implementation of causality. Every machine will have one global integer which is incremented whenever an event has happened on that machine, which is Lamport Clock (`LC`) value for that particular machine. Whenever, machine A sends a message m to machine B, then machine A will also add its `LC` value to the message. Machine B will then update its `LC` value as the $B.LC \leftarrow max(m.LC, B.LC) + 1$. The Lamport Clock value for a machine defines the number of messages it has received. Now, we can see

that for any two messages m and n, if m has happened before n, then $m.LC < n.LC$. However, we cannot say that if $m.LC < n.LC$ then m has happened before n. Hence, we can only achieve partial ordering but not total ordering with Lamport Clock. To achieve Total Ordering we can use Vector Clocks or Totally ordered multicast using Lamport Clock.

20.4 Race Detection

A race condition happens when an application depends on the sequence or timing of processes or threads for it to operate properly. In terms of causality, race condition is an absence of a happen before relation. A benign data-race is an intentional data-race whose existence does not affect the correctness of the program. These data-races may help in getting better performance.

Race detection can be performed statically and dynamically. In order to do static race detection, the tool has to identify reads and writes and establish happens before relationship between them. Static race detection is difficult for compiler to do. In general static analysis is limited by halting problem. In languages using pointers and references, alias analysis of all heap variables (including locks) increases the difficulty of doing static analysis.

Race detection is termed sound, if it says that if there is no race then there is definitely no race in the program. In this case, however, it may be possible that the program analysis gives false positives, i.e., reporting a race condition even if it isn't. A program analysis is complete if it has no false positives, i.e., if it reports a race then the race is real.

Dynamic Race Detection

- Sees the execution trace and code of the program
- Input dependent
- No false positives

Static Race Detection

- Can only see the code of the program
- Input independent
- Can have false positives
- Returns facts regarding program

There are two types of dynamic race detection techniques: (i) lockset analysis, and (ii) happen before analysis. In Lockset Analysis, there is a set of locks for every variable. During the beginning of a program, all the sets are universal set, i.e., lock sets of all variables contains all locks. A set of current locks is maintained for each thread. Whenever lock is obtained, this lock is added to the set of current locks and whenever lock is released, this lock is removed from the set of current locks. Whenever a variable is accessed, it's lockset is updated as the intersection of current lockset and variable's lockset. If the intersection is empty, then this is a potential race condition. Unfortunately, Lockset analysis is expensive and can give false positives.

Happen-Before analysis uses vector clocks. It is based on the idea that certian operations define an ordering between operations of threads and if a variable access can't establish itself as being after the previous one, then we have detected an actual race. Each variable stores the thread clock value for its recent access of each thread. When an access occur, then this access must have a Lamport clock value more than the Lamport clock value of previous thread.