COMP 7970 Storage Systems

Event-Driven Simulation 2

Dr. Xiao Qin

Department of Computer Science and Software Engineering
Auburn University
http://www.eng.auburn.edu/~xqin
xqin@auburn.edu

COMP 7970, Auburn University
Example: A disk system simulator

- A system consists of a disk cache and two disks
- The cache simulator
  - Checks cache contents (takes 0.1 ms)
  - If hit, returns data (takes .2 ms)
  - If miss, forwards the request to a disk
- The disk simulators
  - Analyzes the disk request (takes 1 ms)
  - Seeks (takes $Ts$ ms)
  - Rotational latency (takes $Tr$ ms)
  - Reads/writes data (takes $Ta$ ms)
How to Simulate?

- Cache, Disk 0, and Disk 1 work concurrently

```
Req 1 (R)  Req 2 (W)  Req 3 (R)
miss      miss      Hit
```

Cache
How to Simulate?

- Cache, Disk 0, and Disk 1 work concurrently

```
Req 1 (R)  Req 2 (W)  Req 3 (R)

Cache: miss  miss  Hit

Disk 0: Firmware  Seek  R. Latency  Read

Disk 1: Firmware  Seek  R. Latency  Write
```
Event-Driven Simulation

- We are only interested in the system states at some important moments when some **events** happen
  - E.g.: arrival of a request, cache misses, disk-seek completion, etc.
  - What happens between events? We don’t care
    - Time is not a linear flow. It is quantized.
    - Call an event handler when see/process an event
      - Functions are designed to “wait until be called”
  - An event may generate one or more new events
    - Need reorder these events according to the time stamps

![Diagram](image-url)
Event-Driven Simulation (Cont’d)

- **Event structure**
  - Event type (?)
  - Device ID
  - Time stamp (when will the event happen?)
    - Simulated time, not host machine time

- **Event queue**
  - We always process the next event with the smallest time stamp
  - Need sort the queue every time a new event is added in
    - Performance critical
    - Priority Queue / Heap (or other data structures)
Event-Driven Simulation (Cont’d)

• Current time
  – A global variable
  – Simulated time, not host machine time

• Main event loop

  While event queue is not empty {
    Get next event
    time = event.time
    switch (event.type) ..... 
  }

  

Event-Driven Simulation (Cont’d)

- Trace processor reads a request, inserts an event (req1) to the queue
- Event loop gets the event, updates time, calls cache simulator
- Cache simulator generates a new event
  - event.time = time + 0.1 ms
  - event.type = cache_miss
- Event loop gets the event, updates time, calls disk 0 simulator
  - Disk simulator generates a new event
  - event.time = time + 1 ms
  - event.type = firmware_done
- Trace processor reads a request, inserts an event (req2) to the queue
- Event loop gets the req2 event, updates time, calls cache simulator
- Cache simulator generates a new event
  - event.time = time + 0.1 ms
  - event.type = cache_miss
- ......
Event-Driven Simulation (Cont’d)

Refer to disksim3.0 as an example
http://www.ece.cmu.edu/~ganger/disksim/
void DistrSystem::run( int LS_policy) // run the workload with LS policy  
{
    DOUBLE time_period = 0;
    int node;

    // initialize
    sys_time = first_job_time;

    while (!stop_running()) // stop when all machine's lists empty
    {
        for (node = 0; node < NUM_MACHINES; node++) // each machine in the system
        {
            machine[node]->check_quantum (LS_policy);
        } // for a new quantum
        for (node = 0; node < NUM_MACHINES; node++) // each machine in the system
        {
            machine[node]->run_quantum ();
        }

        sys_time += QUANTUM;
        time_period += QUANTUM;

        process_new_event();

        // change global load information periodically
        if (time_period >= LOAD_UPDATE_PERIOD) {
            update_sys_load();

            time_period = 0;
        }
    }
}
void Machine::check_foreign_list(DOUBLE begin_time, int LS_policy)
{
    // same to future_list
    Process *proc;
    int num_foreign = foreign_list->NumOfItem();

    if (num_foreign > CPU_THRESHOLD)
        printf( "Too many foreign jobs %d, #%d.\n", mid, num_foreign);
    while (num_foreign > 0)
    {
        proc = (Process*) foreign_list->Remove();
        num_foreign--;

        if (proc->arrival_time > begin_time) {
            foreign_list->Append(proc);
            continue;
        }
        // if local execution, allocate memory resources
        CPU_queue_length++;
        memory_usage += proc->memory_size;

        // add to ready_list
        proc->state = READY;
        ready_list->Append(proc);
        reorganize_memory();
    }
}