Real-Time Scheduling in Distributed Environments

Arnab Sarkar
Dept. of Computer Sc. & Engg.
IIT Guwahati
Setting the Stage

- **Real-time Systems**
  - A system whose specification includes both functional as well as temporal notions of correctness.
  
  - Logical Correctness: Produces correct outputs.
  
  - Temporal Correctness: Produces outputs at the right time.
    - *It is not enough to say that “brakes were applied”*
    - *You want to be able to say “brakes were applied at the right time”*
Setting the Stage

Real-time systems enable us to:

- Manage the vast power generation and distribution networks,
- Control industrial processes for chemicals, fuel, medicine, and manufactured products,
- Control automobiles, ships, trains and airplanes,
- Conduct video conferencing over the Internet and interactive electronic commerce, and
- Send vehicles high into space and deep into the sea to explore new frontiers and to seek new knowledge.
Example of a Real-Time System

Many real-time systems are *control systems*.

**Example:** A simple one-sensor, one-actuator control system.

![Control System Diagram]

13-Feb-15

STC Networks & Distrib. Comput., IITG
Example of a Real-Time System

Pseudo-code for this system:

```plaintext
set timer to interrupt periodically with period $T$;
at each timer interrupt do
    do analog-to-digital conversion to get $y$;
    compute control output $u$;
    output $u$ and do digital-to-analog conversion;
end do
```

$T$ is called the **sampling period**. $T$ is a key design choice. Typical range for $T$: seconds to milliseconds.
Another Example

Multimedia

Want to process audio and video frames at steady rates.

- TV video rate is 30 frames/sec. HDTV is 60 frames/sec.
- Telephone audio is 16 Kbits/sec. CD audio is 128 Kbits/sec.

Other requirements: Lip synchronization, low jitter, low end-to-end response times (if interactive).
Characteristics of Real Time Tasks

- **Task**: A sequential piece of code.
- **Job**: Instance of a task.
  - Jobs require **resources** to execute.
  - **Example resources**: CPU, network, disk, critical section.

- **Release time of a job**: The time at which the job becomes ready to execute.

- **Absolute Deadline of a job**: The time instant by which the job must complete execution.
- **Relative deadline of a job**: “Deadline – Release time”.
Example

• Job is released at time 3.
• Its (absolute) deadline is at time 10.
• Its relative deadline is 7.
• Its response time is 6.
Real-Time Periodic Task

- Task: a sequence of similar jobs
  - Periodic task \((p,e)\)
    - Jobs repeat regularly
    - Period \(p\) = inter-release time \((0 < p)\)
    - Execution time \(e\) (maximum execution time; \(0 < e < p\))
    - Utilization \(U = e/p\)
Deadlines: Hard vs. Soft

- **Hard deadline**
  - Disastrous or very serious consequences may occur if the deadline is missed
  - Validation is essential: can all the deadlines be met, even under worst-case scenario?
  - Deterministic guarantees

- **Soft deadline**
  - Ideally, the deadline should be met for maximum performance. The performance degrades in case of deadline misses.
  - Best effort approaches / statistical guarantees
Schedulability

- Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines
What’s Important in Real-Time

- Metrics for real-time systems differ from that for time-sharing systems.

<table>
<thead>
<tr>
<th></th>
<th>Time-Sharing Systems</th>
<th>Real-Time Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>High throughput</td>
<td>Schedulability</td>
</tr>
<tr>
<td>Responsiveness</td>
<td>Fast average response</td>
<td>Ensured worst-case response</td>
</tr>
<tr>
<td>Overload</td>
<td>Fairness</td>
<td>Stability</td>
</tr>
</tbody>
</table>
Real-Time Scheduling

- Determines the order of real-time task executions
- Static-priority scheduling
- Dynamic-priority scheduling
RM (Rate Monotonic)

- Optimal static-priority scheduling
- It assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period

![Diagram showing RM scheduling]

13-Feb-15
RM (Rate Monotonic)

- Executes a job with the shortest period

<table>
<thead>
<tr>
<th>Job</th>
<th>Period</th>
<th>Execution Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1(4,1)$</td>
<td>4</td>
<td>[0, 5]</td>
</tr>
<tr>
<td>$T_2(5,2)$</td>
<td>5</td>
<td>[5, 10]</td>
</tr>
<tr>
<td>$T_3(7,2)$</td>
<td>7</td>
<td>[10, 15]</td>
</tr>
</tbody>
</table>
RM (Rate Monotonic)

- Executes a job with the shortest period

![Diagram showing jobs T1(4,1), T2(5,2), T3(7,2) with a deadline miss marker.]

Deadline Miss!
Response Time

- Response time
  - Duration from released time to finish time

$T_1(4,1)$

$T_2(5,2)$

$T_3(10,2)$
Response Time

- Response time
  - Duration from released time to finish time
Response Time

- Response Time \((r_i)\) [Audsley et al., 1993]

\[
    r_i = e_i + \sum_{T_k \in HP(T_i)} \frac{r_i}{p_k} \cdot e_k
\]

- \(HP(T_i)\): a set of higher-priority tasks than \(T_i\)
RM – Utilization Bound

- Real-time system is schedulable under RM if
  \[ \sum U_i \leq n (2^{1/n} - 1) \]

Liu & Layland,
RM – Utilization Bound

Real-time system is schedulable under RM if

\[ \sum U_i \leq n \left(2^{1/n} - 1\right) \]

Example: \( T_1(4,1), T_2(5,1), T_3(10,1), \)

\[ \sum U_i = \frac{1}{4} + \frac{1}{5} + \frac{1}{10} \]

\[ = 0.55 \]

\[ 3 \left(2^{1/3} - 1\right) \approx 0.78 \]

Thus, \( \{T_1, T_2, T_3\} \) is schedulable under RM.
RM – Utilization Bound

- Real-time system is schedulable under RM if
  - $\sum U_i \leq n \left(2^{1/n} - 1\right)$

RM Utilization Bounds

![Graph showing RM Utilization Bounds](data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAIcAAADpCAYAAAAy2pXAAAACXBIWXMAAAsIAAAAD2AAB1f/1x9AAAABd起身

The Number of Tasks

Utilization

13-Feb-15
EDF (Earliest Deadline First)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline

```
(4,1) (5,2) (7,2)
```

T₁(4,1) T₂(5,2) T₃(7,2)
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline
EDF (Earliest Deadline First)

- Executes a job with the earliest deadline
EDF (Earliest Deadline First)

- Optimal scheduling algorithm
  - if there is a schedule for a set of real-time tasks, EDF can schedule it.

![Diagram showing EDF scheduling of tasks T1(4,1), T2(5,2), and T3(7,2)]
EDF – Utilization Bound

- Real-time system is schedulable under EDF if and only if

\[ \sum U_i \leq 1 \]

Liu & Layland,
Least Laxity First (LLF)

- Dispatch the task with the smallest laxity, which is the largest amount of time that a task can be delayed (some type of procrastination index)
- In a sense, it is similar to EDF, in that it runs the *most urgent* tasks in the set (the metric by which *urgency* is measured, differs though)

<table>
<thead>
<tr>
<th></th>
<th>arrival</th>
<th>duration</th>
<th>deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>T 1</td>
<td>0</td>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>T 2</td>
<td>4</td>
<td>3</td>
<td>28</td>
</tr>
<tr>
<td>T 3</td>
<td>5</td>
<td>10</td>
<td>29</td>
</tr>
</tbody>
</table>

• Requires calling the scheduler periodically, and to re-compute the laxity - many calls of the scheduler
• When tasks have the same laxity - too many preemptions
What is a Distributed System?

- A set of nodes communicating through a network
- Network could be LAN or WAN
- Nodes could be homogeneous or heterogeneous
Why Distributed Systems?

- Applications themselves are distributed
  - E.g., command and control, air traffic control

- High performance
  - Better load balancing

- High availability (fault-tolerance)
  - No single point of failure
Problems with Distributed Systems

- Resource management is difficult
  - No global knowledge on workload
  - No global knowledge on resource allocation

- No synchronized clock (or clocks need to be synchronized)

- Communication related errors
  - Out of order delivery of packets, packet loss, etc.

- Difficult to distinguish network partition from node/link failures
Task Model

- Tasks can be periodic or non-periodic.

- Guaranteed Task – A task which may be assured to meet its deadline under all circumstances.

- A set (possibly null) of guaranteed periodic tasks exists at each node.

- Non-periodic tasks may arrive at any node at any time in the network.

- The scheduling objective is to guarantee all periodic tasks and as many non-periodic tasks as possible, utilizing the resources of the entire network.
Local Scheduler

- Each node in the network contains a local scheduler.
  - When a new task arrives, an attempt is made to schedule the task at that node.
    - Calls Guarantee routine
  - If guaranteed, the Dispatcher is invoked.
    - Dispatcher determines which among the guaranteed periodic and non-periodic tasks is to be executed next.
    - This selection depends on the scheduling policy used.
      - Example: EDF
Local Scheduler

- If the new task cannot be guaranteed, the scheduler tries to execute the task on some other node.

- The local scheduler interacts with other nodes to determine the node to which the task can be sent to be scheduled.
  - Uses techniques such as *bidding*, *focused addressing* etc.

- Upon arrival at the destination node, another attempt is made to schedule the task.

- Eventually, the task is either guaranteed and executed, or discarded.
Surplus

- The amount of *computation time* available on a node between the *arrival time* of a new unguaranteed task and its *deadline*.

- The new task can be guaranteed only if the surplus is greater than the execution time required for the task.

- A local task is guaranteed only if its *latest start time* is greater than its *arrival time*. 
Latest Start Time (S)

- Arrange all guaranteed tasks in order of non-increasing deadlines

- If the 1st task has deadline $D_1$ and execution time $e_1$,
  - $S_1 = D_1 - e_1$

- Let the 2nd task has deadline $D_2$ and execution time $e_2$,
  - If $D_2 > S_1$, $S_2 = S_1 - e_2$
  - Otherwise, $S_2 = D_2 - e_2$

- $S$ values of all other tasks are similarly calculated.
Time Overhead Considerations

- Time spent on scheduling is important in real-time systems.

- Dispatcher's execution time must be included in every task's computation time.
  - Invoked each time any task (including the local scheduler and bidder tasks) completes execution and relinquishes the CPU.

- Newly arriving non-periodic tasks must be examined soon after they arrive.
  - But interrupting a running task to guarantee a newly arriving task could cause the running task to miss its deadline!
Time Overhead Considerations

- **A possible solution:**
  - After dispatcher chooses next task to run, check its *surplus* to verify that running the bidder task / local scheduler in between will not cause deadline misses of guaranteed tasks.

- This solution cannot be used if the running task is non-preemptive. **Solution:**
  - Run the bidder and the local scheduler tasks as periodic tasks on a processor separate from the CPU on which tasks are scheduled.

- **A logical extension:** Allocate a separate processor with limited processing capabilities for scheduling.
Communication Related Overheads

- If a new task cannot be guaranteed locally, it becomes a candidate for remote execution.

- Communication delays depend on the pairs of processes involved:
  - The distance separating them
  - The communication protocol used
  - The communication from other nodes in the system to the two nodes.
Communication Delay Estimation

- Every communication is time-stamped by the sending node.

- The receiving node computes delay by subtracting the time-stamp from the time of receipt.

- Subsequent delays may be estimated based on a linear relationship between message length and communication delay.

- In this talk, we assume that the clocks on different nodes are synchronized.
Focused Addressing

- The scheduler with a task which cannot be guaranteed locally, first attempts to execute it on another node through focused addressing.

Focused addressing works as follows:

- Estimate the arrival time $AT$ of the task (say $T$) at a node $N$.
- If estimated surplus at $N$, between $AT$ and deadline $D$ of $T$ is greater than its execution time $e$ by $FP\%$, the task is sent to the node.

- The receiving node uses the guarantee routine to check whether it can guarantee the arriving task.
- $FP$ is an adaptive parameter.
Bidding

- If there is no node with a significant surplus, a more expensive bidding procedure is invoked.

- The main functions of the bidder are:
  - For a task that cannot be guaranteed locally, the bidder sends out a request for bids (RFBs) to nodes with surplus processing power.
  - Evaluating bids.
  - Responding to the request for bids from other nodes.
Request for Bids (RFB)

- A request for bid (RFB) message is broadcast to all nodes.

- A RFB message contains the following information:
  - Execution time of the task: $e$, Deadline: $D$, Size of the task: $S$, the time $t$ at which the message is being sent and a deadline for responses: $R$.

- $R$ is the time after which the requesting process will examine the bids to choose the best bidder.
Request for Bids (RFB)

- The deadline for responses $R$ should be such that after $R$ there is sufficient time:
  - for the requesting process to evaluate the bids,
  - for the task to reach the best bidder node,
  - for the best bidder to guarantee and schedule the task,
  - once scheduled, for the task to complete computations and meet its deadline.
Request for Bids (RFB)

- $R = D - (P + E + W + e)$
  - $P$: period of the task that evaluates bids (the maximum waiting time before bids are recognized)
  - $E$: expected time taken for the task to reach the best bidder
  - $W$: estimated time after arrival that the task may begin computation.

- If $R$ is insufficient, the bidder may again resort to focused addressing.
  - $FP$ is adjusted to augment chances of finding a node with surplus.

- If a node with surplus still cannot be found, the task cannot be guaranteed.
Request for Bids (RFB)

- A possible improvement:
  - *Do not broadcast RFBs*
  - Send RFBs only to nodes whose estimated surplus matches the requirements of the task to be guaranteed - *Buddies*
    - avoids potentially unnecessary communication

- Drawbacks of the approach:
  - Requires time to check the node-surplus information to determine potential bidders
  - Can prevent bidding by nodes with surplus, if the available information was inaccurate.
Making Bids

- This is carried out in response to an RFB

- The bidder first checks and proceeds only if its response will reach the requestor before the response deadline $R$.
  - This time includes: time of response + transit time of response

- Once a node decides to respond, it first computes $AT$, the estimated arrival time of the task if, indeed, it is awarded the task.
Making Bids

- $AT$ takes into account the following:
  - The fact that bids at the requesting node are evaluated after the response deadline $R$
  - The average delay in evaluating bids (estimated to be one half the bidding period)
  - The estimated time for the task to arrive at the bidder's node.

- Whether the bidder can execute the new task is determined by the surplus $SATD$ at the bidder's node between $AT$ and deadline $D$. 
Making Bids

- **SATD** takes into account:
  - Future instances of periodic tasks
  - Processing time for tasks that may arrive as a result of previous bids
    - **PNB**: % of CPU time used by non-periodic tasks arriving as a result of bidding
  - Processing time needed for non-periodic tasks that may arrive locally in the future
    - **PNL**: % of CPU time used by non-periodic tasks arriving locally

- **SATD = S − (PNB + PNL) × (D - AT)**
  - **S** = Surplus between AT and D
Making Bids

Possible Improvements:

- A node receiving an RFB, determines that another node has a higher probability of being awarded the task.
  - *Do not respond to the RFB*

- Saves communication and computation costs incurred in bidding

- The accuracy of this decision would depend on the accuracy about other nodes’ surpluses
Evaluating Bids

- Bids are processed by the node that originally sent RFBs.
- Queues all bids until the response deadline $R$.
- Calculates Estimated Arrival Time ($EAT$) at each bidder's node.
- For each bidder it estimates $SEATD$, the surplus between ETA and D.
  - $SEATD = SATD \times \left(\frac{D - EAT}{D - AT}\right)$

- The node with the greatest $SEATD$ becomes the best bidder.
  - Identity of the 2nd best bidder may also be sent to the best bidder.
Evaluating Bids

- Intimates to all *but* the best bidder that their bids were not accepted.
  - An alternative: *Bidders may time-out after a predetermined time.*
    - (RFBs may also contain similar time bounds within which bids must be received)

- Surplus information sent on bids may be used for *focused addressing* and selection of *buddies* while sending RFBs
Response to Task Award

- Awardee node treats it as a task that has arrived locally
  - Takes action to guarantee it.

- If the task cannot be guaranteed,
  - Determine if some other node has the surplus to guarantee it.
  - Instead of a broadcast, send the task to the second-best bidder.

- Otherwise, the task is rejected.

- The environment that submitted the task will be responsible for appropriate action if a task is not guaranteed.
  - Resubmit task with a later deadline.
Handling Precedence Constraints

- The bidding algorithm can be extended to handle tasks with precedence constraints.

- Consider the following scenario:
  - Task A has been guaranteed on node 1, B on node 2, C on node 3
  - A and B should precede C
  - Assume $D_A < D_B$

- Try to guarantee C at node 3, with start time of $D_B + T$
  - $T$: Max time required for A and B’s outputs to reach C
Handling Precedence Constraints

- If C cannot be guaranteed at node 3,
  - Send C to node 2 (containing preceding task with latest deadline)
  - Broadcast an RFB to be returned to node 2
- Attempt to guarantee task at node 2.
- If this is not successful, try to modify $D_B$ to $D'_B$ such that,
  - $D'_B < D_B$ and $D_A < D'_B$
  - $B$ remains guaranteed
  - $C$ can be guaranteed
  - A possible extension: Recursively apply the above method.

- If $C$ is still not guaranteed,
  - Send $C$ to the best bidder in a normal bidding process
Time triggered Bus: Flexray

Slot 1 of static segment: Assigned to ECU1
Slot 2 of static segment: Assigned to ECU2
Slot 3 of static segment: Assigned to ECU3
Slot 1 of dynamic segment: Assigned to ECU1
Slot 2 of dynamic segment: Assigned to ECU2
Slot 3 of dynamic segment: Assigned to ECU3
Slot 4 of dynamic segment: Assigned to ECU2
Multiprocessor Scheduling - Partitioning

- Partition tasks so that each task always runs on the same processor

- Steps:
  - Assign tasks to processors (bin packing)
  - Schedule tasks on each processor using uniprocessor algorithms like EDF or LLF.
Partitioning

Assignments of tasks to processors
  - Bin-packing problem (NP-hard problem)
  - Typically done using heuristics

Proposed heuristics
  - First Fit (FF)
  - Best Fit (BF)
  - Worst Fit (WF)
  - First Fit Decreasing (FFD), etc.
Global Scheduling

- A single scheduling algorithm is used that schedules all tasks

- Important difference:
  - Task may migrate among the processors
Partitioned Schedulers ≠ Optimal

- Example: 2 processors; 3 tasks, each with 2 units of work required every 3 time units.
Global Schedulers Succeed

Example: 2 processors; 3 tasks, each with 2 units of work required every 3 time units

Task 3 *migrates* between processors
Problem Classification Methods

- Migration-based Classification
  - No migration
  - Restricted migration
  - Full migration

- Priority-based Classification
  - Static priorities
  - Job-level dynamic priorities
  - Unrestricted dynamic priorities
Global Scheduling Vs. Partitioning

- Trade-off between the two approaches
  - Global scheduling (= no restriction on migration)
    - Good: high utilization
    - Bad: high migration cost (also cache misses)
  - Partitioned scheduling (= strict restriction on migration)
    - Good: no migration cost
    - Bad: low utilization

- Generally, if we restrict more,
  - the run-time overhead is reduced but
  - the schedulability (e.g., utilization bound) is also reduced.
Migration-based Classification

- **No Migration (Partitioned)**
  - Task can not migrate
  - Job can not migrate

- **Restricted Migration**
  - Task can migrate
  - Job can not migrate

- **Full Migration**
  - Task can migrate
  - Job can migrate
Migration-based Classification

- Task migration and/or cache misses become very harmful when
  - CPUs are connected via bus or network
  - Each CPU has its own memory
  - Shared global memory not enough to hold states of all tasks

- For CPU cores on a single chip
  - CPUs are connected via a high-speed on-chip network
  - CPUs share large global memories and caches.
  - Lower migration costs
The Big Goal #1

- Design of optimal scheduling algorithms
  - Intuitively speaking, any task set, whose utilization is less than or equal to the number of processors, is schedulable by some \((3,3)\)-restricted algorithm.

  - Optimal real-time scheduling methodologies on multiprocessors.
Greedy Algorithms Fail on Multiprocessors

At each scheduling point, a greedy algorithm will regularly select the $m$ “best” jobs and run those

- Earliest Deadline First (EDF)
- Least Laxity First (LLF)

EDF and LLF are optimal on a single processor; neither is optimal on a multiprocessor

- Such greedy approaches generally fail on multiprocessors
Greedy Algorithms Fail on Multiprocessors

Example \((n = 3 \text{ tasks}, \ m = 2 \text{ processors})\) :

- **Task 1**: Work = 9, Period = 10
- **Task 2**: Work = 9, Period = 10
- **Task 3**: Work = 8, Period = 40

**Utilization**: \(\frac{9}{10} + \frac{9}{10} + \frac{8}{40} = 2\)
Event-Driven Algorithms Fail on Multiprocessors

At time $t = 0$, Tasks 1 and 2 are the obvious greedy choices.

- Task 1: Work = 9, Period = 10
- Task 2: Work = 9, Period = 10
- Task 3: Work = 8, Period = 40
Even at $t = 8$, Tasks 1 and 2 are the only "reasonable" greedy choices.
Greedy Algorithms Fail on Multiprocessors

Yet if Task 3 isn’t started by $t = 8$, the resultant idle time eventually causes a deadline miss.
How can we “see” this critical event at $t = 8$?
Proportioned Algorithms Succeed

Subdivide Task 3 with a period of 10

Task 1: Work = 9, Period = 10
Task 2: Work = 9, Period = 10
Task 3: Work = 2, Period = 10

CPU 1

CPU 2
Proportioned Algorithms Succeed

Now Task 3 has a zero laxity event at time $t = 8$
The Other Big Goals

- **Big Goal #2: Proportional Fairness**
  - Jobs having equal priority (same utilization) are said to be scheduled with equal fairness if their rates of execution progress are same.

- **Big Goal #3: Low Overheads**
  - Task migration and Context Switches
  - Scheduling Complexity
Current State of the Art

- Pfair, ERfair, PD² satisfy goals #1 and #2

- Bfair, EKG, LLREF, DP-Fair satisfy goals #1 and #3

- POFBFS*, POES*, SERF*, ESSM* satisfy goals #1, #2 and #3

- There are other algorithms like EDF-fm (2005) which trades-off goal #1 to achieve goal #3.
ERFair Scheduling

- A work-conserving global multi-processor scheduling methodology for hard real-time repetitive tasks sets with fully dynamic priorities.

- Divides tasks into unit length sub-tasks; schedules the most urgent sub-tasks at each time-slot to ensure fairness.

- *Early Release fair (ERfair) Scheduling*: At the end of any time-slot $t$, at least $(wt_i * t)$ time-slots of execution of each task $T_i$ must complete.
ERfair Scheduling - Idea

- Early Release fair (ERfair):
  - Given the task weights, finds pseudo-deadline \( d_i^j \) of the \( j^{th} \) sub-task of task \( I \) as:
    \[
    d_i^j = \left\lfloor \frac{j \times p_i}{e_i} \right\rfloor - 1
    \]

- Algorithm:
  - Schedule task with earliest pseudo-deadline first.
    - Arrange tasks in a min heap.
    - Extract the task at the root and execute.
    - Calculate pseudo-deadline of next sub-task.
    - Insert the task into the heap and re-heapify.
  - Ties between multiple tasks having same pseudo-deadline is broken using tie-breaking rules.
  - Complexity: \( O(\log n) \) per time-slot per processor.
Strengths

- **Schedulability**: Optimal
- **Quality of Service (QoS)**: Guarantees QoS: reserve $X$ time units for task $A$ out of every $Y$ time units.

- **Temporal Isolation**: Provides temporal isolation to each client task from the ill-effects of other "misbehaving" tasks attempting to execute for more than their prescribed processor shares.
  - Makes it applicable in a wide range of domains – CPU, networks, embedded systems

- Graceful degradation for all tasks in times of overload.
- Efficient handling of dynamic task arrivals and departure
Weaknesses

- **Scheduling Overheads**
  - **High Scheduling Complexity:** Uses a min-heap to determine the most urgent operation deadlines of sub-tasks at each time-slot. Hence, for $n$ given tasks, they suffer a high scheduling complexity of $O(lg n)$ per time-slot per task.
  - **Unrestricted Migrations and Preemptions:** A direct consequence of global scheduling and ignorance of affinities:
    - of tasks towards the processor where it executed last
    - of processor caches towards tasks it executed recently.

- Dearth of techniques to incorporate practical and emerging design metrics like power, overload management, fault tolerance, etc.
Thank You