Scheduling on Large-Scale Platforms

Henri Casanova
(casanova@cs.ucsd.edu)

Lecture Outline

- What is Scheduling?
- Application Scheduling
  - State-of-the-art
  - Case-study: divisible loads
- Modeling and Evaluation
- Conclusion
What is Scheduling?

- Context: parallel/distributed (scientific) applications that run on distributed computing platforms
  - a matrix-multiplication parallel algorithm on a cluster
  - a complex, multi-component, climate modeling application on a grid that consists of 6 clusters and a data warehouse
- **Scheduling:** the decision process by which application components (computation, data) are assigned to resources (compute node, disk, network link)

What is Scheduling?

- Scheduling is a well-known hard problem in computer science
  - NP-hard in most instances (with a few exceptions)
- Many scheduling heuristics have been proposed
  - various application models
  - various platform models
- Most general application model: DAG
- Most general platform model: Graph
  - Not to say it's a good model, and there are many subtleties there
Where do DAGs come from?

- **Example**: Solving a triangular systems of linear equations
  - $A \cdot x = b$, where $A$ is lower triangular

  ```
  for i = 1 to n
  x(i) = b(i) / a(i,i)
  for j = i+1 to n
  b(j) = b(j) - a(j,i) * x(i)
  ```

Where do DAGs come from?

- **Example**: Solving a triangular systems of linear equations
  - $A \cdot x = b$, where $A$ is lower triangular

  ```
  for i = 1 to n
  x(i) = b(i) / a(i,i)
  for j = i+1 to n
  b(j) = b(j) - a(j,i) * x(i)
  ```

  - **i=1**: $x(1) = \frac{b(1)}{a(1,1)}$ // easily compute $x(1)$
    $b(2) = b(2) - a(2,1) \cdot x(1)$ // remove it from
    $b(3) = b(3) - a(3,1) \cdot x(1)$ // the system
    ...
  
  - **i=2**: $x(2) = \frac{b(2)}{a(2,2)}$ // easily compute $x(2)$
    $b(3) = b(3) - a(3,2) \cdot x(2)$ // remove it from
    $b(4) = b(4) - a(4,2) \cdot x(2)$ // the system
    ...
DAG and Linear System Solve

**Bernstein Condition**

- Let \( \text{In}(T) \) be the input variables of task \( T \)
- Let \( \text{Out}(T) \) be the output variables of task \( T \)
- Two tasks \( T \) and \( T' \) are not independent when
  - \( \text{In}(T) \cap \text{Out}(T') \neq \emptyset \)
  - or, \( \text{Out}(T) \cap \text{In}(T') \neq \emptyset \)
  - or, \( \text{Out}(T) \cap \text{Out}(T') \neq \emptyset \)

**DAG and Linear System Solve**

- Let ‘<‘ denote the sequential order
  - \( T_{1,1} < T_{1,2} < T_{1,n} < T_{2,2} < T_{2,3} < \ldots T_{n,n} \)
- *Independent* tasks can be done in //
  - How do we determine that tasks are independent?

\[
\text{for } i = 1 \text{ to } n \\
\text{Task } T_{i,i}: x(i) = b(i) / a(i,i) \\
\text{for } j = i+1 \text{ to } n \\
\text{Task } T_{i,j}: b(j) = b(j) - a(j,i) * x(i)
\]
Back to the Example

\[ \text{for } i = 1 \text{ to } n \]
\[ \text{Task } T_{i,i}: \ x(i) = \frac{b(i)}{a(i,i)} \]
\[ \text{for } j = i+1 \text{ to } n \]
\[ \text{Task } T_{i,j}: \ b(j) = b(j) - a(j,i) \times x(i) \]

- Task \( T_{i,i} \):
  - \( \text{In}(T_{i,i}) = \{b(i), a(i,i)\} \)
  - \( \text{Out}(T_{i,i}) = x(i) \)
- Task \( T_{i,j}, \text{ for } j>i \)
  - \( \text{In}(T_{i,j}) = \{b(j), a(j,i), x(j)\} \)
  - \( \text{Out}(T_{i,j}) = \{b(j)\} \)

- Dependences?
  - \( \text{Out}(T_{i,i}) \cap \text{In}(T_{i,j}) = \{x(i)\} \text{ for all } j>i \)
  - \( \text{Out}(T_{i,j}) \cap \text{Out}(T_{i+1,j}) = \{b(j)\}, \text{ for all } j > i+1 \)

DAG obtained from instruction-level analysis

Back to the Example

\[ \text{for } i = 1 \text{ to } 5 \]
\[ \text{Task } T_{i,i}: \ x(i) = \frac{b(i)}{a(i,i)} \]
\[ \text{for } j = i+1 \text{ to } 5 \]
\[ \text{Task } T_{i,j}: \ b(j) = b(j) - a(j,i) \times x(i) \]

DAG obtained from instruction-level analysis

(root)

(end node)
More DAGs

- A DAG (or Task Graph) may just be inherent to an application at a higher level
- Workflow applications explicitly couple components together into a “coarse-grain” DAG
  - image processing
  - climate modeling
  - database operations

Scheduling Problem

- Although there is a lot of formalism, it comes down to putting slots into Gantt charts:
Scheduling Problem

Although there is a lot of formalism, it comes down to putting slots into Gantt charts:

- Although there is a lot of formalism, it comes down to putting slots into Gantt charts:
DAG Scheduling Problem

- Theorem: there exists a valid schedule iff the DAG has no cycles
- Critical Path:
  - Let $X$ be a path in the DAG from the root to the end-node.
  - Let $F(X)$ be the sum of all node weights along the path.
  - The makespan is larger than $F(X)$ for all paths.
  - The path that maximizes $F(X)$ is called the *critical path*.

DAG Scheduling problem

- If we have an infinite number of processors, then “schedule as early as possible” is optimal.
- Otherwise, it’s NP-hard
  - NP-hard on 2 identical processors
- Things can get more complicated
  - cost of communication over some network
  - heterogeneous processors
- People have proposed many heuristics
- One common approach: list scheduling
  - compute task “priorities” and schedule tasks in order of these priorities.
  - priorities may, for instance, be higher for tasks on the critical path.
Job Scheduling

- So far we’ve talked about a single application
- But in many cases, resources are shared by multiple users
  - “jobs” arrive at different times
- Performance metric should be some type of aggregate, rather than the makespan
  - Minimize average “turn-around time”
  - Minimize average “slow-down”
    - Slow-down is: “time in the system / time in the system if alone in the system”
  - Minimize maximum “slow-down”
    - Try to enforce some notion of fairness
- There is a wealth of complexity results, as well as open problems

Scheduling in Practice

- Very large literature
- Very large gap between theory and practice
- Application Scheduling:
  - Most published algorithms make unrealistic assumptions
    - Especially when looking at large-scale Grid platforms
  - Therefore, they are almost never used
  - Question: Should one just give up and use simple (e.g., greedy) approaches, because there is no hope of doing anything clever in practice?
- Job Scheduling:
  - State-of-the-art: Batch-schedulers (PBS, LQS, LoadLeveler, SGE)
  - NOT designed with user-centric metrics in mind
    - e.g., resource utilization vs. average slowdown
  - Very rigid request model: “I want N processors for H hours”
  - Question: should we have more flexible interfaces to job schedulers, and should metrics be more user-centric?
Lecture Outline

- What is Scheduling?
- Application Scheduling
  - State-of-the-art
  - Case-study: divisible loads
- Modeling and Evaluation
- Conclusion

Application Scheduling

- Context: “Grid” platforms
  - Application should run on multiple resources over the wide-area
  - The “scheduler” component is present in many systems
    - But: “We’ll put something clever in there at some point”
- Why is practical application scheduling difficult?
  - “Algorithm in paper A assumes homogeneous resources”
  - “Algorithm in paper B assumes a fully-connect network with no bandwidth sharing”
  - “Algorithm in paper C assumes that all resources are stable throughout application execution”
  - “Algorithm in paper D assumes that we know everything about the application”
  - “Algorithm in paper E assumes that there are no latencies”
- Assumptions in the literature do not match the real world
Lecture Outline

- What is Scheduling?
- Application Scheduling
  - State-of-the-art
  - Case-study: divisible loads
- Modeling and Evaluation
- Conclusion

Divisible Load Applications

- Definition
  - Application consists of “many” independent elemental tasks
  - Each task has a data and computation load
  - Application approximated as a continuous load W, which is measured in “load units”
  - Amount of data to transfer is significant (i.e., not purely compute intensive)
- Application examples
  - MPEG encoding: 1 load unit = 1 frame
  - Bioinformatics: 1 load unit = 1 sequence
  - Volume rendering: 1 load unit = 1 voxel
  - Datamining, image processing, file compression, database joins, genetic algorithms, etc.
- Great candidates for Grid deployment
  - No dependencies (MUCH more simple than DAGs)
  - Should be deployable on a large scale
  - Like a data-intensive SETI@home
Divisible Load Scheduling

- **DLS problem**
  - pick a number of chunks
  - pick a size for each chunk
  - which chunk on which compute resource and when?
- **Variations**
  - **Objective**
    - minimize “makespan”
    - maximize steady-state throughput
  - **Compute and Network Resources**
    - homogeneous
    - heterogeneous
  - **Topology**
    - linear array, star, tree, hypercube, mesh, ...

Single-Round DLS

- **N workers, N “chunks”**

![Diagram showing single-round DLS with three workers and their tasks](image)

- **Some assumptions**
  - Communication from master is serialized
  - Communication and Computation “costs” are commensurate to chunk size (in load units)
Single-Round DLS

Optimal Schedule

worker #1
worker #2
worker #3

Drawback: idle time
- most common bane of parallel application performance
**Multi-Round DLS**

- N workers, MxN chunks (M: # of rounds)
- Increasing and then decreasing chunk sizes
- Well-known “pipelining” technique
- Additional assumption
  - Overlap of communication and computation at each worker

**Multi-Installment (MI) Algorithm**

- Multi-installment algorithm [Veeravalli’95]
- Worker finishes receiving data for a chunk exactly when computation for that chunk can start
- leads to a “tight” schedule
  - no gaps in network and worker utilization
  - Induction on chunk sizes to compute the schedule
MI for Large-Scale Platforms?

- Assumes linear costs
  - $T_{\text{comm}} = \frac{\text{chunk\_size}}{\text{transfer\_rate}}$ (in load units)
  - $T_{\text{comp}} = \frac{\text{chunk\_size}}{\text{compute\_rate}}$ (in load units)
- Assumes a homogeneous platform
  - homogeneous network and workers
- Assumes 100% accurate performance prediction for both network and workers

→ MI is not applicable to large-scale platforms
Affine Costs

- Known to be more realistic than linear costs
- \( T_{\text{comm}} = \beta + g / B \)
  - \( \beta \): overhead of establishing connection, network latency, authentication/authorization, etc.
  - Can be significant in wide-area platforms (e.g., TeraGrid) using Grid middleware (e.g., GridFTP) (up to several seconds)
- \( T_{\text{comp}} = \alpha + g / S \)
  - \( \alpha \): overhead of establishing connection, network latency, authentication/authorization, process initiation, etc.
  - In practice can be several seconds
  - Grid Benchmarking projects report high values (up to 45 seconds on some production Grid platforms)

XML: eXtended Multi-Installment
XMI: eXtended Multi-Installment

∀ i ≥ N, \( \alpha + g_i = \frac{(g_{i-1} + g_{i-2} + \ldots + g_{i-N})}{R} + N\beta \)
**XMI: eXtended Multi-Installment**

∀ \( i \geq N \), \( \alpha + g_i = \frac{(g_{i-1} + g_{i-2} + \ldots + g_{i-N})}{R} + N \beta \)

∀ \( 0 \leq i < N \), \( \alpha + g_i = \frac{(g_{i-1} + \ldots + g_0)}{R} + i \beta + g_0 + \alpha \)

∀ \( i \leq 0 \), \( g_i = 0 \)

---

**Recursion** (\( R = B / S \))

∀ \( i \geq N \), \( \alpha + g_i = (g_{i-1} + g_{i-2} + \ldots + g_{i-N}) / R + N \beta \)

∀ \( 0 \leq i < N \), \( \alpha + g_i = (g_{i-1} + \ldots + g_0)/R + i \beta + g_0 + \alpha \)

∀ \( i \leq 0 \) \( g_i = 0 \)
Solving the XMI recursion

- How to solve?
  \[ \forall i \geq N, \quad \alpha + g_i = (g_{i-1} + g_{i-2} + \ldots + g_{i-N}) / R + N \beta \]
  \[ \forall 0 \leq i < N, \quad \alpha + g_i = (g_{i-1} + \ldots + g_0) / R + i \beta + g_0 + \alpha \]
  \[ \forall i \leq 0 \quad g_i = 0 \]

- Generating function: \( G(x) = \sum_{i=0}^{\infty} g_i x^i \)

- Compute \( G(x) \)?
  - sum over the equations above
  - property: \( \sum_{i=0}^{\infty} g_i x^i = n^{th} \text{ coeff. of } G(x)/(1-x) \)

\[
G(x) = \frac{g_0 (1-x^N) + \alpha x^N + \beta (x + x^2 + \ldots + x^N)}{(1-x) - x(1-x^N) / R}
\]

Solving the XMI recursion

- The problem is reduced to finding the coefficients of \( G(x) \)
- Simple rational expansion theorem \([GKP'94]\)
  - roots of the denominator polynomial
  - can only work if all roots of degree 1
    - true is \( R \neq N \)
    - if \( R = N \), turns out one can use the General theorem
- Roots can be computed numerically
- We obtain chunk sizes as linear combinations of geometric series

\[
g_i = g_0 \sum_{j=0}^{N} \eta_j \theta_j^i + \sum_{j=0}^{N} \zeta_j \theta_j^i \quad (\theta_j: \text{inverse of } j^{th} \text{ root})
\]
XMI: Still not there

Remaining limitations
- only applicable to homogeneous platforms
- difficult to develop the recursion in the heterogeneous case
- assumes 100% accurate performance prediction for both network and workers
UMR: Uniform Multi-Round

- Key issue: make the chunk size recursion simpler to solve
- Key idea: use “uniform” chunks:
  - chunks are the same size within a round
- Homogeneous case:

\[
\forall 0 \leq j < M-1 \quad \alpha + \frac{\text{chunk}_j}{S} = N \left(\frac{\text{chunk}_{j+1}}{B}\right) + N \beta
\]

(last round: same as for XMI)
- if \( NS = B \) we obtain an arithmetic series
- if \( NS \neq B \) we obtain a geometric series
\[ \forall 0 \leq j < M-1 \quad \alpha + \frac{\text{chunk}_j}{S} = \frac{1}{N} \left( \frac{\text{chunk}_{j+1}}{B} \right) + N \beta \]

(last round: same as for XMI)

- if \( NS = B \) we obtain an arithmetic series
- if \( NS \neq B \) we obtain a geometric series

**Recursion:**

\[ \text{chunk}_j = \left( \frac{B}{NS} \right)^j (\text{chunk}_0 - \Delta) \]

\[ \Delta = \frac{BS}{B - NS} (N\beta - \alpha) \]

We have

\[ N \sum_{j=0}^{M-1} \text{chunk}_j = W \]

Therefore we can compute all chunk sizes
One can compute the makespan

\[ T(M, \text{chunk}_0) = \frac{W}{NS} + M\alpha + \frac{1}{2} N(\beta + \frac{\text{chunk}_0}{B}) \]

- 1/2 factor to account for the last round

Constrained minimization problem

\[ \text{MINIMIZE} \quad T(M, \text{chunk}_0) = \frac{W}{NS} + M\alpha + \frac{1}{2} N(\beta + \frac{\text{chunk}_0}{B}) \]

\[ \text{SUBJECT TO} \quad G(M, \text{chunk}_0) = \sum_{j=0}^{M-1} N\text{chunk}_j - W = 0 \]

Lagrange Multiplier method

\[ L(M, \text{chunk}_0, \lambda) = T(M, \text{chunk}_0) + \lambda G(M, \text{chunk}_0) \]

\[ \begin{aligned} 
\frac{\partial L}{\partial \lambda} &= G = 0 \\
\frac{\partial L}{\partial M} &= \frac{\partial E_x}{\partial M} + \lambda \times \frac{\partial G}{\partial M} = 0 \\
\frac{\partial L}{\partial \text{chunk}_0} &= \frac{\partial E_x}{\partial \text{chunk}_0} + \lambda \times \frac{\partial G}{\partial \text{chunk}_0} = 0 
\end{aligned} \]
UMR: Computing M

- Solving the Lagrange system gives

\[ N\Delta = \frac{W_{total} - NM\Delta}{1 - \left(\frac{B}{NS}\right)^M} \ln\left(\frac{B}{NS}\right) - 2\alpha \times B \frac{1 - \left(\frac{B}{NS}\right)^M}{1 - \frac{B}{NS}} = 0 \]

\[ \text{chunk}_0 = \frac{(1 - \frac{B}{NS})(W_{total} - NM^*\Delta)}{N \times (1 - (\frac{B}{NS})^{M^*})} + \Delta \]

- The first equation can be solved numerically
- We then obtain M and chunk\(_0\)
- With the recursion we have all the chunk sizes

UMR vs. XMI

![Graph comparing UMR and XMI performance](image-url)
Heterogeneous UMR

- Chunks within a round are “uniform”
  - different chunk sizes
  - same processing times on all workers
- Main issue: selection and ordering of workers
- We know that selection is NP-hard
- We reuse a heuristic from [Beaumont’02]
  - Sort workers by decreasing bandwidth
- Effective:
  - within 20% of the “ideal” makespan to 1000-fold heterogeneity
  - as compared to 80+% with selection

Robust UMR (RUMR)

- All this is great but...
- We still assumes 100% accurate performance prediction for both network and workers

⇒ RUMR: Robust UMR
Performance Prediction Errors

- Uncertainty on chunk transfer and computation time
  - They are random variables
- Several Possible causes
  - undedicated platform
    - common for networks
    - CPUs on network of workstations
  - data-dependent chunk computation
    - e.g., MPEG encoding: 10% uncertainty
    - e.g., HMMER: 9% uncertainty
    - e.g., VFleet volume rendering: 1% uncertainty

Decreasing chunk sizes

- Question: How can we account for uncertainties that disrupt the schedule?
- Factoring Idea [Hummel’92]
  - Start with large chunks and decrease chunk sizes
  - N chunks for 1/2 load, N chunks for 1/3 load, etc.
  - Greedy allocation of chunks
- Example with 3 rounds
Decreasing chunk sizes

- Question: How can we account for uncertainties that disrupt the schedule?
- Factoring Idea [Hummel'92]
  - Start with large chunks and decrease chunk sizes
  - N chunks for 1/2 load, N chunks for 1/3 load, etc.
  - Greedy allocation of chunks
- Example with 3 rounds

<table>
<thead>
<tr>
<th>Worker</th>
<th>Round 1</th>
<th>Round 2</th>
<th>Round 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>worker #1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>worker #2</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>worker #3</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Question: How can we account for uncertainties that disrupt the schedule?

Factoring Idea [Hummel'92]
- Start with large chunks and decrease chunk sizes
- N chunks for 1/2 load, N chunks for 1/3 load, etc.
- Greedy allocation of chunks

Example with 3 rounds
Decreasing chunk sizes

- Question: How can we account for uncertainties that disrupt the schedule?
- Factoring Idea [Hummel'92]
  - Start with large chunks and decrease chunk sizes
  - N chunks for 1/2 load, N chunks for 1/4 load, etc.
  - Greedy allocation of chunks
- Example with 3 rounds

```
worker #1
   |   |   |   |
   3 5 3

worker #2
   |   |   |   |
   3

worker #3
   |   |   |   |
   3
```

RUMR: UMR + Factoring

- Limitation of Factoring
  - Does not account for network transfers
  - Starting with large chunks hinders pipelining

UMR

- Good overlap of communication and computation
- Good robustness to performance prediction errors

Factoring

- Good overlap of communication and computation
- Good robustness to performance prediction errors
RUMR: 2 phases

- Execution is split into two phases
  - UMR phase
  - Factoring phase
- When should the UMR phase end?
  - large uncertainty: short UMR phase
  - small uncertainty: long UMR phase
- Heuristic:
  - Give an amount of load to each phase “proportionally” to the error
    - defined as the stdev of actual/predicted

RUMR: Evaluation [HPDC’03]
What now?

- We have addressed the four limitations of previous work on multi-round divisible load scheduling
  - affine costs
  - number of rounds
  - heterogeneous platform
  - robustness to uncertainty
- Implemented as part of real software for real applications on real platforms

Lesson

- Going from theoretical scheduling to practical scheduling is ... A LOT of work
- In some cases it is worth it
  - Divisible Loads
- In some cases, it is unclear
  - Workflow applications?
  - Should one try to reuse the DAG scheduling literature?
Lecture Outline

- What is Scheduling?
- Application Scheduling
  - State-of-the-art
  - Case-study: divisible loads
- Modeling and Evaluation
- Conclusion

Platform Modeling

- Platform modeling is key to the relevance of a scheduling algorithm
- A reason why published scheduling algorithms go unused is because they assume unrealistic platform models
  - simplistic models make it easier to find scheduling “results”
  - but the sad truth is that the real-world is complex
- Particularly true for large-scale networks
  - We’ve seen latencies
  - What about bandwidth sharing using something like TCP on wide-area networks?
Banwidth Sharing

- Traditional Assumption: Fair Sharing
  - Good approximation for LANs
  - Some people look at network paths [F. Vivien’s talk this morning]
  - But what about WANs?
    - That’s what Grids are made of, so we should probably pay attention

A Simple Experiment

- Sent out files of 500MB up to 1.5GB with TTCP
- Using from 1 up to 16 simultaneous connections
- Recorded the data transfer rate per connection
### Experimental Results

#### Number of concurrent TCP connections

#### Normalized data rate per connection

#### Bandwidth Sharing

- **Explanations:**
  - TCP congestion windows
  - Backbone links have so many connections that interference among a few selected connections is negligible
  - None of this is surprising to networking people...
  - This empirical data makes it possible to develop a simple model of bandwidth sharing over single network paths, usable at the application level:
    - $bw(i) = \frac{bw(1)}{1 + (i - 1) \cdot \gamma}$
    - $\gamma = 0$: ideal WAN
    - $\gamma = 1$: ideal LAN
    - in between: fits experimental data within 10%
Why do we care?

- Has an impact of even simple scheduling problems:
  - 20 tasks to do
  - Can do them locally in 5 seconds
  - Or do them remotely in 5 seconds, but pay the communication time (1000Kb and 100kb/sec), with possible simultaneous connections

But it’s more complex..

- WANs are what large-scale systems (Grids, P2P) are made of
- Therefore, we have to account for TCP WAN properties to conduct relevant scheduling research, until other protocols arise
- But we only looked at two end-points!
  ➡️ We need a model of bandwidth sharing that will work over arbitrary topologies
Bandwidth Sharing

- What about more complex topologies?
  - Packet-level simulation
    - Too low-level to think about when designing scheduling algorithm
    - Slow simulations (e.g., NS)
  - Macroscopic TCP models (it’s a field)
    - “Fluid in Pipe” analogy
    - Turns out TCP implements some type of Proportional Fairness

```
[ naïve example]
```

- Many details necessary for dealing with a complex topology
  - Round-trip times
  - Bottleneck links
  - Etc..

- One can developed a fast algorithm that computes bandwidth shares according to accurate TCP macroscopic models [Kelly,Massoulie,Roberts]

- **Question:** What do we do with such a model?
Modeling for Scheduling

- Physical topology is complex
  - complex bandwidth sharing
- It is unlikely that one can design a scheduling algorithm that can account for all details effectively
- It is unlikely that one can obtain all the necessary information anyway!
- Therefore, the scheduling algorithm uses an abstract model
Modeling for Scheduling

Abstract

Physical

Modeling for Scheduling

Abstract

Physical
Modeling for Scheduling

**Abstract**

- Fully connected
- Heterogeneous
- No latencies

**Physical**

- Router

---

39
Major (Open) Questions

- How Abstract can we get?
  - When does the abstraction lead to unreasonable schedules?
  - Different authors take different approaches
    - Depends on the scheduling problem
    - Trend: going less abstract

- How Real can we get?
  - How much information are we able to provide the scheduling algorithm with?
  - There are tools for network measurements, topology discoveries, etc.

- How Real do we want to get?
  - Not clear one can design a good scheduling algorithm that account for all routers
  - Even though, information is never perfect/static and trying to account for all details may in fact be damageable

- What is the right trade-off?

Evaluation

- How does one evaluate/compare scheduling algorithms on large-scale platforms?

- Option #1: real-world experiments
  - labor-intensive at best
    - "give me your platform for 10 days so that I run experiments"
  - Often unrepeatable
    - prevents back-to-back runs
  - Limited to physical infrastructure
    - limits statistical significance of results
  - Limited by time (needs thousands of runs)
    - limits statistical significance of results

→ One has to resort to **Simulation**
Need for a Simulation model!

- What is the right model?
  - nobody agrees
  - validation is VERY difficult

- Close to physical:
  - Network
    - Packet-level simulation (DaSSF, MicroGrid, ModelNet)
    - Abstract model (SimGrid)
  - Computation
    - Emulation (MicroGrid)
    - Abstract model (SimGrid)

- Trade-off: Accuracy vs. Speed
  - Not such an obvious trade-off

Two approaches

- Scenario #1
  - Fully-connected, heterogeneous, no latencies
  - Macroscopic bandwidth-sharing model, latencies, abstract model of computation
  - Physical

- Scenario #2
  - Macroscopic bandwidth-sharing model, latencies
  - Packet-level simulation, emulation of computation
  - Physical
Two Fundamental Questions

- Simulation speed
- Simulation accuracy
- Abstraction of platform model used by the scheduling algorithm

- A few data points
- Still far from known
- Gigantic amount of work
  - Validation in the real world?

- Application performance
- Abstraction of platform model used by the scheduling algorithm

- A few data points
- Still far from known
- Application dependent

Many other questions

- Simulated network topologies?
  - Discovered from real networks?
    - A lot of difficulty there
  - Topology generators (BRITE, Tiers, etc.)
    - Incomplete vision
  - Several research efforts

- Fundamentally, simulation models must come from real-world datasets
  - Measurements, benchmarking
  - We’re doing this for desktop availability for instance
Lecture Outline

- What is Scheduling?
- Application Scheduling
  - State-of-the-art
  - Case-study: divisible loads
- Modeling and Evaluation
- Conclusion