Roadmap

- **Which theory models are more closer to data-center settings.**

  Srikanth Kandula  
  Ratul Mahajan  
  Amar Phanishayee  
  Monia Ghobadi

- **Focus on algorithms**
  Even complex algorithms can have algorithmic intuitions which are useful in practice.

- **One example**
  One system heuristic and one complex provable algorithm (Using LP Hierarchies) that has good heuristic value.
Luleå FB Data Center, **South of Arctic Circle**
Luleå FB Data Center, South of Artic Circle

It is beautiful like this for 3 days....
Luleå FB Data Center, South of Artic Circle, a cold, cold place...
Efficiency Matters a Lot

“as large as cities”

5%
Efficiency Matters a Lot: *Simplicity is not everything!*

“as large as cities”

**Emphasis on Principled Algorithms**

- Simple heuristics
- Theoretically Sound Algorithms

<table>
<thead>
<tr>
<th>Cost</th>
<th>Time</th>
</tr>
</thead>
</table>
How we Measure Efficiency

- **Makespan**

  Minimizing the maximum completion time among a set of jobs.

  *Length of the schedule.*

- **Average (or total) Flow-time (aka, Job Completion-time)**

  \[ F_j = C_j - r_j \]

  - same as *response time*
  - measures the time a job spends in a system
How we Measure Efficiency

- **Makespan**
  - Minimizing the maximum completion time among a set of jobs.
  - Length of the schedule.

- **Average (or total) Flow-time (aka, Job Completion-time)**
  - $F_j = C_j - r_j$
  - same as **response time**
  - measures the time a job spends in a system

Throughput, energy, fairness, utilization, etc..
Challenges of Data Center Scheduling

**Resources**
- Heterogeneous (FPGA + CPU, GPU + CPU)
- Multidimensional (CPU, memory, network)

**Algorithms**
- Fast, simple, often online.

**Jobs**
- Complex dependencies: DAGs, Co-flows, etc.
Challenges of Data Center Scheduling

Resources
- Heterogeneous (FPGA + CPU, GPU + CPU)
- Multidimensional (CPU, memory, network)

Jobs
- Complex dependencies: DAGs, Co-flows, etc.

Algorithms
- Fast, simple, often online.

Rich theory with many nice algorithms when jobs have simple structures.
Scheduling on Heterogeneous Clusters
Scheduling on Heterogeneous Clusters

Why are clusters heterogeneous?

- Special purpose hardware
- Data locality
- Geographic location
- Privacy concerns
Modeling Heterogeneity

jobs run faster on some clusters and slower on others
Modeling Heterogeneity

jobs run faster on some clusters and slower on others

Jobs arrive over time

<table>
<thead>
<tr>
<th>machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 15 1000 ..... 10</td>
</tr>
<tr>
<td>66 100 5 ..... 98</td>
</tr>
<tr>
<td>1 15 88 ..... 13</td>
</tr>
<tr>
<td>100 788 9 ..... 13</td>
</tr>
</tbody>
</table>
Heterogeneous == “Unrelated Machines Scheduling”

Assign (match) jobs to clusters + schedule to Minimize QoS.

Jobs arrive over time
**Beautiful Algorithms For Unrelated Machines Scheduling Problems**

**Offline, Online, Multidimensional, Clairvoyant, Non-Clairvoyant, Stochastic, Truthfulness...**

<table>
<thead>
<tr>
<th>Makespan</th>
<th>Flow-time</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>LST’87</td>
<td>CGK’09</td>
<td>AGK’12</td>
</tr>
<tr>
<td>ST’89</td>
<td>AGK’12</td>
<td>KLS’10</td>
</tr>
<tr>
<td>Svensson’12</td>
<td>BK’15</td>
<td>I KMP’14</td>
</tr>
<tr>
<td>AAFPW’97</td>
<td>IKMP’14</td>
<td>P’07</td>
</tr>
<tr>
<td>KD’18</td>
<td></td>
<td>A’06</td>
</tr>
</tbody>
</table>

Has lead to development of very nice ideas: Use of vertex solutions and duality in design of algorithms, configuration LPs, potential functions, connections to game theoretic ideas...
Offline, Online, Multidimensional, Clairvoyant, Non-Clairvoyant, Stochastic, Truthfulness…

**RESEARCH DIRECTION:**

**Few Machine types:** Can we get better algorithms for some classic unrelated machines scheduling?

Has lead to development of very nice ideas: Use of vertex solutions and duality in design of algorithms, configuration LPs, potential functions, connections to game theoretic ideas…
Challenges of Data Center Scheduling

Resources

- Heterogeneous (FPGA + CPU, GPU + CPU)
- Multidimensional (CPU, memory, network)

Algorithms

- Fast, simple, often online.

Jobs

- Complex dependencies: DAGs, Co-flows, etc.
The plan

One Heuristic

GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. OSDI 2016.


Very general, works well in practice, as bad as any other algorithm on paper 😊

One Complex Theoretical Framework

Levey and Rothvoss ’16.

Garg, Kulkarni, Li’18.
Garg, Kukarni, Li’18.

Very specific, provable, and quite complex.
The plan

One Heuristic

GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. OSDI 2016.


Very general, works well in practice, as bad as any other algorithm on paper 😊

One Complex Theoretical Framework

Levey and Rothvoss ’16.

Garg, Kulkarni, Li’18.

Garg, Kukarni, Li’18.

Very specific, provable, and quite complex.
The plan

One Heuristic

GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. OSDI 2016.


Very general, works well in practice, as bad as any other algorithm on paper 😊

One Complex Theoretical Framework

Levey and Rothvoss ’16.

Garg, Kulkarni, Li’18.

Garg, Kukarni, Li’18.

Very specific, provable, and quite complex.
A Directed Acyclic Graph (DAG) Scheduling Problem in Large Clusters

GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. OSDI 2016.

DAG Model Supported in Hadoop

Multidimensionality

Heterogeneity of clusters
Cluster Scheduling

Resources of a cluster

D types of resources

\[(1, 1, \cdots, 1)\]

A single job represented as a DAG
Cluster Scheduling

Resources of a cluster

D types of resources

(1, 1, ⋮, 1)

Demand Vector

(1, 0, ⋮, 1/2)
(1/4, 1, ⋮, 1/10)
(1/2, 1/2, ⋮, 1/2)

A single job represented as a DAG

(task)
Cluster Scheduling

Resources of a cluster

\[ (1, 1, \ldots, 1) \]

**D** types of resources

Demand Vector

A single job represented as a DAG

Processing length (duration)

(task)
Cluster Scheduling: Minimize Makespan

A single job represented as a DAG

Cluster

Cluster (1, 1)
Is There a Good Algorithm?

**Theorem: Bansal and Khot ‘09.**

It is unlikely *(UGC-hard)* that a polynomial time algorithm can achieve better than *D* approximation to the DAG scheduling problem. This holds even if all tasks of the DAG have 1) same length, 2) require exactly one resource.

- Any non-idling algorithm is *equally good or equally bad!*

Not a useful intuition for system designers.
Is There a Good Algorithm?

**Theorem: Bansal and Khot ‘09.**

It is unlikely (UGC-hard) that a polynomial time algorithm can achieve better than $D$ approximation to the DAG scheduling problem. This holds even if all tasks of the DAG have 1) same length, 2) require exactly one resource.

**Optimal Algorithm:**
*Do a greedy schedule respecting precedence constraints*

At least one resource is used. **Congestion** for that resource decreases.
Is There a Good Algorithm?

**Theorem: Bansal and Khot ‘09.**

It is unlikely (UGC-hard) that a polynomial time algorithm can achieve better than $D$ approximation to the DAG scheduling problem. This holds even if all tasks of the DAG have 1) same length, 2) require exactly one resource.

**Optimal Algorithm:**
Do a greedy schedule respecting precedence constraints

At least one resource is used. **Congestion** for that resource decreases.
Is There a Good Algorithm?

**Theorem: Bansal and Khot ‘09.**

It is unlikely (UGC-hard) that a polynomial time algorithm can achieve better than $D$ approximation to the DAG scheduling problem. This holds even if all tasks of the DAG have 1) same length, 2) require exactly one resource.

**Optimal Algorithm:**
Do a greedy schedule respecting precedence constraints

---

At least one resource is used. *Congestion* for that resource decreases.
When did System Designers Care for Lowerbounds?

GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. OSDI 2016.


✧ Could find almost optimal solutions on MS data sets.

✧ Improves makespan by 30% at least compared to simple greedy heuristics.
“pathologically bad schedules in today's approaches mostly arise due to two reasons: (a) long-running tasks have no other work to overlap with them, which reduces parallelism, and (b) the tasks that are runnable do not pack well with each other, which increases resource fragmentation.”
Our approach is to identify the potentially troublesome tasks, such as those that run for a very long time or are hard to pack.
Intuition of Graphene

Main Steps

- Our approach is to identify the potentially troublesome tasks, such as those that run for a very long time or are hard to pack.

- Place the troublesome tasks first onto a virtual resource-time space. This space would have $d + 1$ dimensions when tasks require $d$ resources; the last dimension being time.
Intuition of Graphene

**Main Steps**

- Our approach is to identify the potentially **troublesome tasks**, such as those that run for a very long time or are hard to pack.

- *Place the troublesome tasks first* onto a virtual resource-time space. This space would have \( d + 1 \) dimensions when tasks require \( d \) resources; the last dimension being time.

- Our intuition is that placing the troublesome tasks first leads to a good schedule since the remaining tasks can be placed into resultant holes in this space.
Schedule Construction

- Identify tasks that can lead to a poor schedule (troublesome tasks) - T

Nearly optimal for over three quarters of our analyzed production DAGs

- Place tasks in T on a virtual time space; overlay the others to fill any resultant holes in this space
Can we formalize this intuition?
A \((1 + \varepsilon)\)-Approximation for Makespan Scheduling with Precedence Constraints using LP Hierarchies.

Levey and Rothvoss ‘16
Identical Machines Scheduling

(Special case of DAG scheduling)

A single DAG. Each task needs to be scheduled on exactly one machine. Each task needs 1 unit of CPU.

$m$ identical machines (or CPUs)

Minimize Makespan
Identical Machines Scheduling

A single DAG. Each task needs to be scheduled on exactly one machine. Each task needs 1 unit of CPU.

$m$ identical machines (or CPUs)
Identical Machines Scheduling

A single DAG. Each task needs to be scheduled on exactly one machine. Each task needs 1 unit of CPU.

$m$ identical machines (or CPUs)
Identical Machines Scheduling

A single DAG. Each task needs to be scheduled on exactly one machine. Each task needs 1 unit of CPU.

$m$ identical machines (or CPUs)
Identical Machines Scheduling


Greedy or List-Scheduling is 2 approximation for minimizing makespan.
Identical Machines Scheduling


Greedy or List-Scheduling is 2 approximation for minimizing makespan.
Identical Machines Scheduling


Greedy or List-Scheduling is 2 approximation for minimizing makespan.
Identical Machines Scheduling


Greedy or List-Scheduling is 2 approximation for minimizing makespan.
Identical Machines Scheduling


Greedy or List-Scheduling is 2 approximation for minimizing makespan.

\[ \text{OPT} \geq \text{BAD SLOTS} \]

\[ \text{OPT} \geq \text{GOOD SLOTS} \]

\[ \frac{n}{m} \]

\[ \text{Makespan} \leq \text{Length of Longest chain} \]
Identical Machines Scheduling


Greedy or List Scheduling is a 2-approximation for minimizing makespan.

- Optimal theoretically. But conveys very little information in practice.
- Does not work well in practice when there are more than one resource type.

\[ \text{OPT} \geq \frac{n}{m} \]

Length of Longest chain

OPT \geq \text{BAD SLOTS} + \text{GOOD SLOTS} \quad \frac{n}{m}
Identical Machines Scheduling

Theorem. Levy and Rothvoss’16. There is a quasi-polynomial time \((1 + \epsilon)\) approximation for minimizing makespan when jobs have unit lengths, when number of machines is a constant.

Garg’17 made it strictly quasi-polynomial time.
Identical Machines Scheduling

Theorem. Kulkarni, Li’18.

There is a quasi-polynomial time $(1 + \epsilon)$ approximation for minimizing makespan when jobs have arbitrary lengths, when number of machines is a constant. The algorithm schedules jobs on a single machine and may preempt jobs within a machine.
There is a polynomial time optimal \((2 + \epsilon)\) approximation for minimizing weighted completion time of jobs, when number of machines and job sizes are uniform.
Identical Machines Scheduling


Greedy or List-Scheduling is 2 approximation for minimizing makespan.
Crucial Observation

\[
\text{Makespan} \leq \epsilon \cdot OPT + \frac{OPT}{n/m} \leq (1 + \epsilon) \cdot OPT
\]
Crucial Observation

\[ \leq \epsilon \cdot OPT \]

**BAD SLOTS**
Length of Longest chain

\[ \leq (1 + \epsilon) \cdot OPT \]

How to schedule troublesome tasks?
Framework

Partition the tasks into a set of *bottom tasks* and a single set of *top tasks*. For each set of bottom tasks we find a sub-interval where they should be scheduled.

*Then do a recursive scheduling of bottom tasks.*
Precedence constraints across bottom tasks are automatically satisfied.
Framework

Bottom tasks

Precedence constraints going from bottom to top tasks are loose.
Framework

Bottom tasks

Precedence constraints going from bottom to top tasks are loose.

[\[T_2, T_3\]]
For every task in the set of top tasks we have $[r_j, d_j]$ based on the tentative assignment of bottom jobs.

Precedence constraints going from bottom to top tasks are loose.

There is enough space to schedule top tasks.
Precedence constraints going from bottom to top tasks are loose.

For every task in the set of top tasks we have \([r_j, d_j]\)
based on the tentative assignment of bottom jobs.

There is enough space to schedule top tasks if there are no precedence constraints between top tasks.
**Precedence constraints going from bottom to top tasks are loose.**

For every task in the set of top tasks we have \([r_j, d_j]\)
based on the tentative assignment of bottom jobs.

EDF will schedule all top tasks in the empty space but may violate the precedence constraints between top tasks.

There is enough space to schedule top tasks if **there are no precedence constraints between top tasks.**
Main Steps

- Our approach is to identify the potentially troublesome tasks, such as those that run for a very long time or are hard to pack.

- Place the troublesome tasks first onto a virtual resource-time space. This space would have $d+1$ dimensions when tasks require $d$ resources; the last dimension being time.

- Our intuition is that placing the troublesome tasks first leads to a good schedule since the remaining tasks can be placed into resultant holes in this space.
**Framework**

- **Time Interval**
  - $T_0$
  - $T_1$
  - $T_2$
  - $T_3$
  - $T$

**Bottom tasks**

- Precedence constraints going from bottom to top tasks are loose.

$[T_2, T_3]$
The chain length among top tasks is very small.

Precedence constraints going from bottom to top tasks are loose.
The chain length among top tasks is very small.

Precedence constraints going from bottom to top tasks are loose.
The algorithm has recognized a crude schedule for troublesome tasks. That’s why chain length among top tasks is small.
Intuition of Graphene

Main Steps

- Our approach is to identify the potentially troublesome tasks, such as those that run for a very long time or are hard to pack.

- Place the troublesome tasks first onto a virtual resource-time space. This space would have \( d + 1 \) dimensions when tasks require \( d \) resources; the last dimension being time.

- Our intuition is that placing the troublesome tasks first leads to a good schedule since the remaining tasks can be placed into resultant holes in this space.
The chain length among top tasks is very small.

Precedence constraints going from bottom to top tasks are loose.
Precedence constraints going from bottom to top tasks are loose.

For every task in the set of top tasks we have $[r_j, d_j]$

all except few

EDF will schedule all top tasks in the empty space but may violate the precedence constraints between top tasks.

There is enough space to schedule top tasks if there are no precedence constraints between top tasks.
How to partition the DAG?

1. Precedence constraints between bottom tasks should be implied.
2. The precedence constraints between top and bottom tasks are loose.
3. The chain length among top tasks is small.
Linear Programming Formulation

For every task $j$

$$\sum_{t=1}^{T} x_{jt} = 1$$

is scheduled.

For time slot $t$

$$\sum_{j} x_{jt} \leq m$$

has at most $m$ jobs.

For precedence relation $i \rightarrow j$, $\sum_{t' < t} x_{it'} \geq \sum_{t' \leq t} x_{jt'}$ is satisfied at each time step $t$.

All variables $x_{jt} \geq 0$ are non-negative.

Binary search the optimal makespan as $T$.
Optimal makespan is 4 but LP can complete in 3 time slots.

LP can schedule a job fractionally in a time slot.
Consider the LP solution. Interval of a task is *smallest interval* that contains fractional schedule of the task.
What LP gives?

An interval for each task.
What LP gives?

An interval for each task.

We use these intervals to partition the DAG into top and bottom tasks.
Building Binary Tree

LP Schedules all tasks between $[0, T]$
Building Binary Tree

LP Schedules all tasks in \([0, T]\)

Assign each task to the smallest interval node in the tree that fully contains it.
LP Schedules all tasks in $[0, T]$.

Assign each task to the smallest interval node in the tree that fully contains it.
Building Binary Tree

LP Schedules all tasks in $[0, T]$.

Assign each task to the smallest interval node in the tree that fully contains it.
Building Binary Tree

LP Schedules all tasks in $[0, T]$

Assign each task to the smallest interval node in the tree that fully contains it.
Defining Top and Bottom Tasks

\[ [0, T] \]

\[ [0, T/2] \]

\[ [T/2 + 1, T] \]

\( \left( \log \log T \right)^2 \)
Defining Top and Bottom Tasks

\[ [0, T] \]

\[ [0, T/2] \]

\[ [T/2 + 1, T] \]

\((\log \log T)^2\)

\(\log \log T\)

Throw Them Away!!
Defining Top and Bottom Tasks

Top Tasks

- \([0, T]\)
- \([0, T/2]\)
- \([T/2 + 1, T]\)

Bottom Tasks

- \((\log \log T)^2\)
- \(\log \log T\)

Throw Them Away!!
Defining Top and Bottom Tasks

- Precedence constraints between bottom tasks should be implied.
- The precedence constraints between top and bottom tasks are loose.
- The chain length among top tasks is small.
Defining Top and Bottom Tasks

Top Tasks

Bottom Tasks Sets

Throw Them Away!!

\[ \log \log T \]
Every top task can lose one interval to the left and one interval to the right in terms of space in which it should be scheduled. But, bottom intervals are tiny compared to top, so this is not a big loss.

Precedence constraints going from bottom to top tasks are loose.
Precedence constraints going from bottom to top. Tasks are loose. Every top task can lose one interval to the left and one interval to the right in terms of space in which it should be scheduled. But, bottom intervals are tiny compared to top, so this is not a big loss.

1. Precedence constraints between bottom tasks should be implied.
2. The precedence constraints between top and bottom tasks are loose.
3. The chain length among top tasks is small.
Lift and Project Method (LP Hierarchies)

All the variables are integral.

A systematic way of placing troublesome tasks!
Lift and Project Method (LP Hierarchies)

Dimensions

Number of variables in LP that you want integral

Running time
Increases by a factor of $n$.

$O(n^S)$

All the variables are integral.

A systematic way of placing troublesome tasks!

Original LP
Lift and Project Method (LP Hierarchies)

“Conditioning”

Touch a variable, and it becomes integral!
Lift and Project Method (LP Hierarchies)

“Conditioning”

"Touch a variable, and it becomes integral!"
Lift and Project Method (LP Hierarchies)

“Conditioning”

Touch a variable, and it becomes integral!
Lift and Project Method (LP Hierarchies)

“Conditioning”

Touch a variable, and it becomes integral!
Lift and Project Method (LP Hierarchies)

“Conditioning”

Touch a variable, and it becomes integral!

The LP solution changes in such a way that, for every other task on, the interval in which it is scheduled in the new solution only shrinks.

I have a better understanding of where this task got scheduled.
Reducing Chain Length of Top Tasks

[0, T]

[0, T/2]

[T/2 + 1, T]
Reducing Chain Length of Top Tasks

The interval is of length $T$. We will make sure that there is no chain of length $\epsilon T$ assigned to this interval.
Reducing Chain Length of Top Tasks

The interval is of length $T$. We will make sure that there is no chain of length $\epsilon T$ assigned to this interval.
Reducing Chain Length of Top Tasks

The interval is of length $T$. We will make sure that there is no chain of length $\epsilon T$ assigned to this interval.
Reducing Chain Length of Top Tasks

The interval is of length $T$. We will make sure that there is no chain of length $\epsilon T$ assigned to this interval.
Reducing Chain Length of Top Tasks

\[ [0, T] \]

\[ [0, T/2] \]

\[ [T/2 + 1, T] \]
Reducing Chain Length of Top Tasks

The interval is of length $T$. We will make sure that there is no chain of length $\epsilon T$ assigned to this interval.

How many conditioning are required? $m/\epsilon$

Now recall that number of intervals in top tasks is $2(\log \log T)^2 \approx (\log T)^{\log \log T}$
Reducing Chain Length of Top Tasks

The interval is of length \( T \).
We will make sure that there is no chain of length \( \epsilon T \) assigned to this interval.

How many conditioning are required?

Running time.

Now recall that number of intervals in top tasks is

\[
2 (\log \log T)^2 \sim (\log T)^{\log \log T}
\]
There is a quasi-polynomial time \((1 + \epsilon)\) approximation for minimizing makespan when jobs have arbitrary lengths, when number of machines is a constant. The algorithm schedules jobs on a single machine and may preempt jobs within a machine.

There is a polynomial time \(2 + \epsilon\) approximation for minimizing weighted completion time of jobs, when number of machines and job sizes are uniform.

*More sophisticated use of conditioning and new algorithms for scheduling top tasks.*
Our approach is to identify the potentially troublesome tasks, such as those that run for a very long time or are hard to pack.

*Place the troublesome tasks first* onto a virtual resource-time space. This space would have d +1 dimensions when tasks require d resources; the last dimension being time.

Our intuition is that placing the troublesome tasks first leads to a good schedule since the remaining tasks can be placed into resultant holes in this space.

**Lift and Project Algorithms**

- Using Lift and Project to figure out placing long tasks. Is there a simple, say DP approach to it?
- Can we use LP support for placing tasks?
- Can recursion help in Graphene setting?
Identical Machines Scheduling and Training Neural Networks

PipeDream: Fast and Efficient Pipeline Parallel DNN Training
Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, Phil Gibbons
Training Deep Learning Models

- Large fraction of the data center workloads for many companies.
- Improving training time is considered very important.
- DAGs are good abstractions of DNN training computations.
- Connections to DAG scheduling and communication delay problems.
Two Paradigms

Figure 2: Example data-parallel setup with 4 machines. Timeline at one of the machines shows communication stalls during model parameter exchanges.

Figure 4: An example pipeline-parallel assignment with four machines and an example timeline at one of the machines, highlighting the temporal overlap of computation and activation / gradient communication.
Model Parallelism

- Schedule the layers among a set of machines. Typically Identical.

- Or at most 2 types: CPU + FPGA, CPU + GPUs etc.

Figure 4: An example pipeline-parallel assignment with four machines and an example timeline at one of the machines, highlighting the temporal overlap of computation and activation / gradient communication.
Model Parallelism

- Schedule the layers among a set of machines. Typically Identical.

- Or at most 2 types: CPU + FPGA, CPU + GPUs etc.

- There is communication between layers. Communication cost is crucial.
Model Parallelism

These problems are quite similar to scheduling with communication delays, when there are precedence constraints. (PY’90, VLL’90, MH’95, HLV’94)

Very poorly understood.

Good scheduling has same effect as caching!

PipeDream: Fast and Efficient Pipeline Parallel DNN Training
Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, Phil Gibbons

Zhicheng Yin, Jin Sun, Ming Li, Jaliya Ekanayake, Haibo Lin, Marc Friedman, José A. Blakeley, Clemens A. Szyperski, Nikhil R. Devanur.
Bubble Execution: Resource-aware Reliable Analytics at Cloud Scale. PVLDB 11(7).
Summary: Data Center Scheduling

Resources
- Heterogeneous (FPGA + CPU, GPU + CPU)
- Multidimensional (CPU, memory, network)

Jobs
- Complex dependencies: DAGs, Co-flows, etc.

Algorithms
- Fast, simple, often online.
Summary: Data Center Scheduling

Resources
- Heterogeneous (FPGA + CPU, GPU + CPU)
- Multidimensional (CPU, memory, network)

Algorithms
- Fast, simple, often online.

Jobs
- Often hard in worst case. What’s the right model?
- Understand DAGs that arise in practice. Say DNNs.
- What are the high-level algorithmic intuitions?

Complex dependencies: DAGs, Co-flows, etc.
Summary: Data Center Scheduling

Resources

- Heterogeneous (FPGA + CPU, GPU + CPU)
- Multidimensional (CPU, memory, network)

Jobs

- Complex dependencies: DAGs, Co-flows, etc.
- Often hard in worst case. What’s the right model?
- Understand DAGs that arise in practice. Say DNNs.
- What are the high-level algorithmic intuitions?