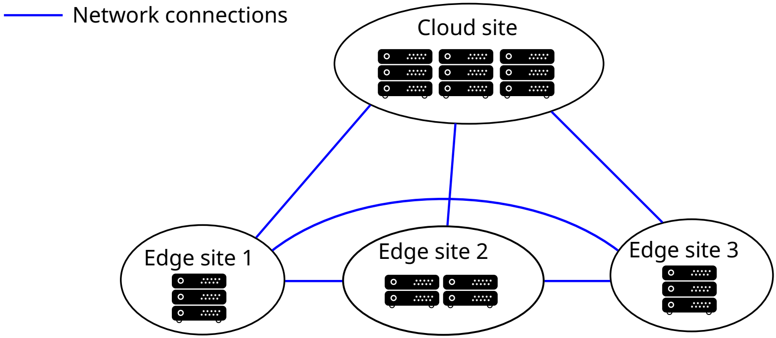
Scheduling Tasks with Precedences on Edge-Cloud Platforms Partially Powered with Renewable Energy

Clément Mommessin
Remous-Aris Koutsiamanis Jean-Marc Menaud

IMT Atlantique, Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, F-44000 Nantes, France
{clement.mommessin, remous-aris.koutsiamanis, jean-marc.menaud}@imt-atlantique.fr

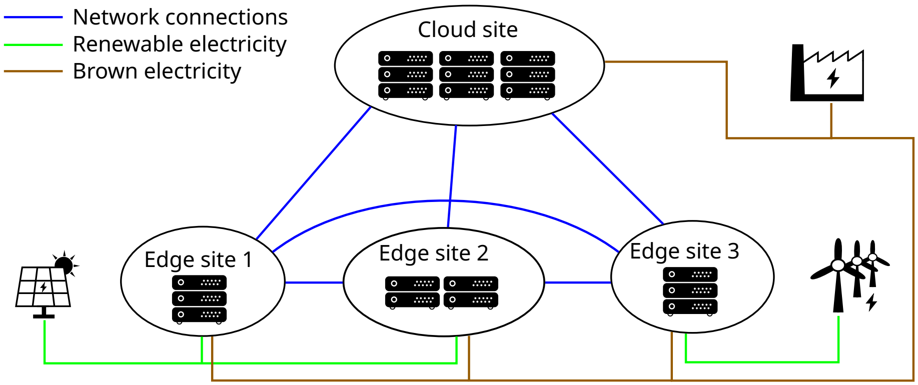
May 16, 2024

General Model: Platform



Network: a complete graph with latency value for each edge

General Model: Platform



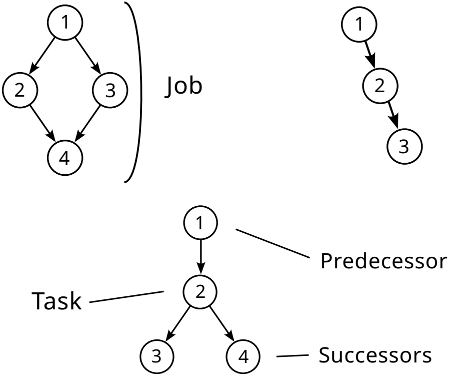
Network: a complete graph with latency value for each edge

For each computing site:
- #CPUs
- #memory units
- speed (work/second)
- Pstatic
- Pdynamic (per CPU)

Renewable sites:
prediction of power production
assumed to be constant

General Model: Jobs and Tasks

Jobs are submitted over time, their execution must start **immediately**.
Tasks are allocated on **reservations** that can be changed until they start.
Executing tasks cannot be preempted/migrated.

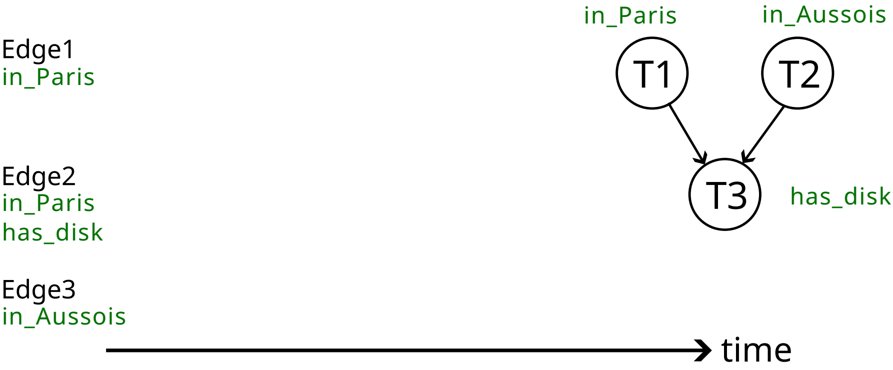


For each job:
- DAG of tasks
- submission time
- deadline

For each task:
- #CPUs
- #memory
- #work to perform

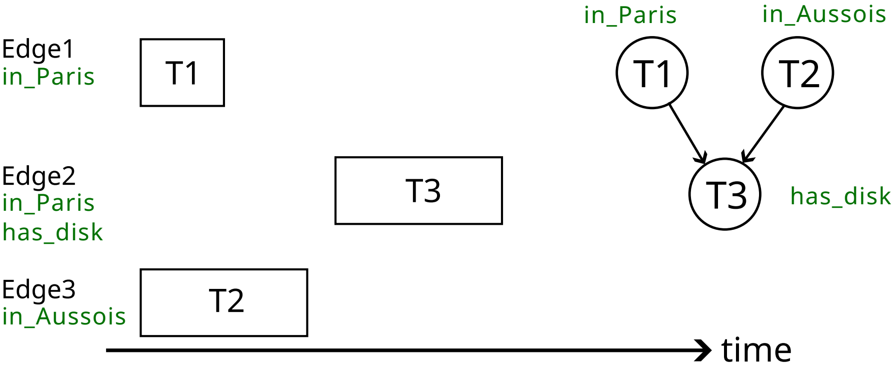
Additional Constraints: Labels

Labels are associated to tasks and computing sites for **affinity** filtering (*à la Kubernetes*).
→ A task can only be placed on a computing site containing its labels.



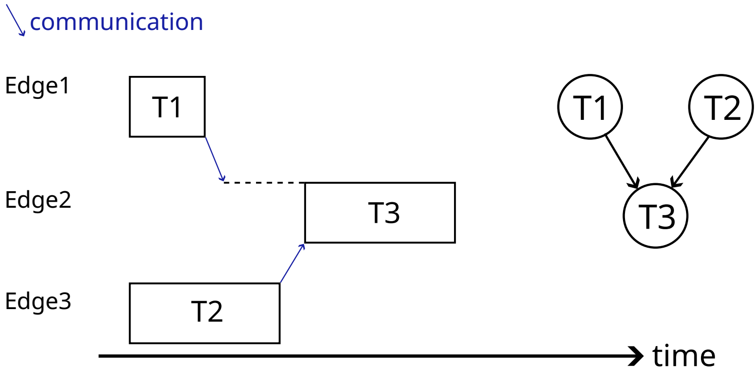
Additional Constraints: Labels

Labels are associated to tasks and computing sites for **affinity** filtering (*à la Kubernetes*).
$\rightarrow$ A task can only be placed on a computing site containing its labels.



Additional Constraints: Immediate Starting Times

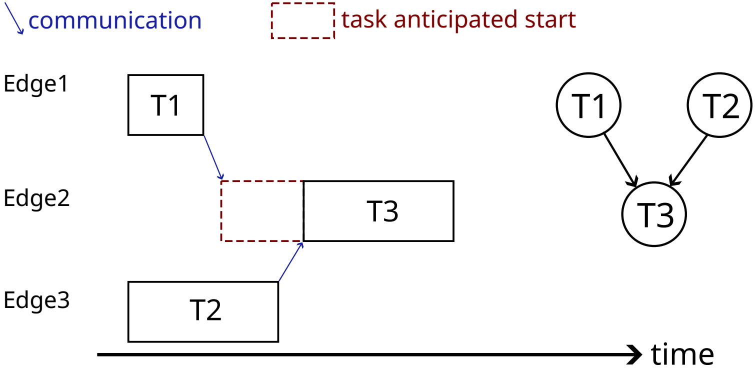
A task with no predecessor **must start immediately** at the job submission. Communications from a task to its succesor(s) **must start immediately** at the completion of the task.
A successor task **must start its execution immediately** after all communications have finished.



Additional Constraints: Immediate Starting Times

A task with no predecessor **must start immediately** at the job submission.
Communications from a task to its succesor(s) **must start immediately** at the completion of the task.
A successor task **must start its execution immediately** after all communications have finished.

$\rightarrow$ A task may start **before** its actual execution (**holding the resources**).



Green Scheduling at the Edge

Problem recap'

- Computing sites composed of one server with limited resources
- Jobs with submission times, deadlines, DAG of tasks
- Renewable energy sources + the regular grid (brown energy)
- Labels on tasks and servers for filtering (*à la* Kubernetes)
- Dynamic setting with a time window of 15 minutes

Decisions to take

- Determine the allocation of tasks for each job (or reject the whole job)
- Determine the repartition of renewable/brown power drawn by each computing site for each time interval

The tasks start/finish times are fixed by the allocation decisions

Objective: Minimize the total brown energy consumption
and the number of jobs rejected

Some Formulae

Processing time of a task:

$$\text{exec} = \frac{\text{task work}}{\text{server speed}}$$

Communication time between two tasks executed on sites i and m:

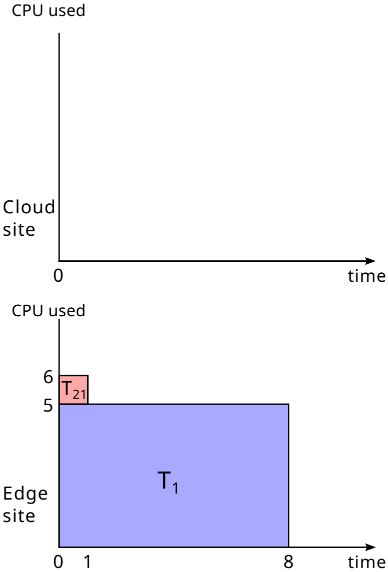
$$\text{comm} = \begin{cases} \text{latency}_{im} & \text{if } i \neq m \\ 0 & \text{if } i = m \end{cases}$$

(We assume infinite bandwidth to avoid network sharing problems)

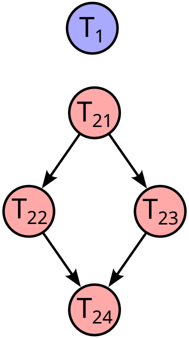
Power consumption of a computing site when u CPUs are used:

$$\text{power} = \begin{cases} P_{\text{static}} + u \times P_{\text{dynamic}} & \text{if } u > 0 \\ 0 & \text{if } u = 0 \end{cases}$$

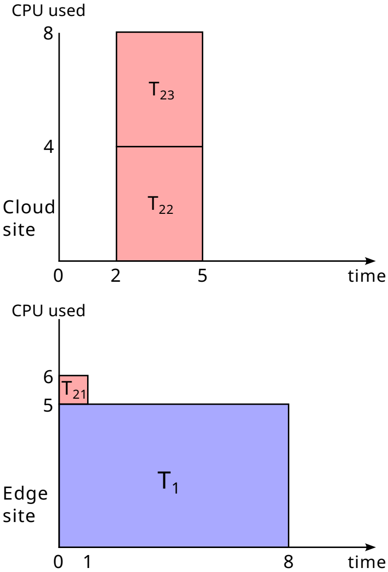
Example: Schedule



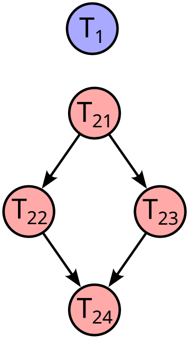
Solution 1



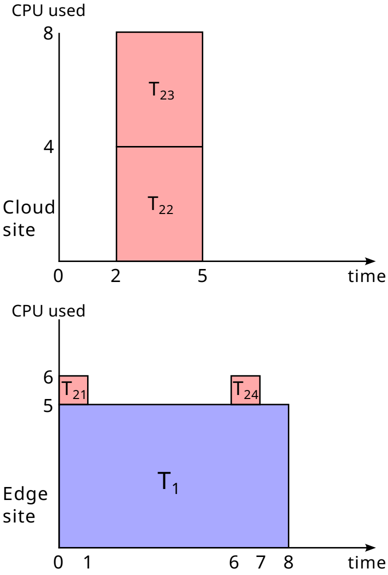
Example: Schedule



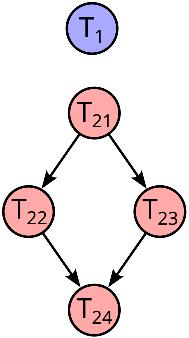
Solution 1



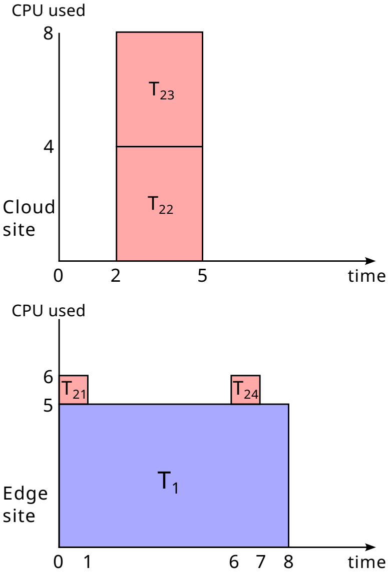
Example: Schedule



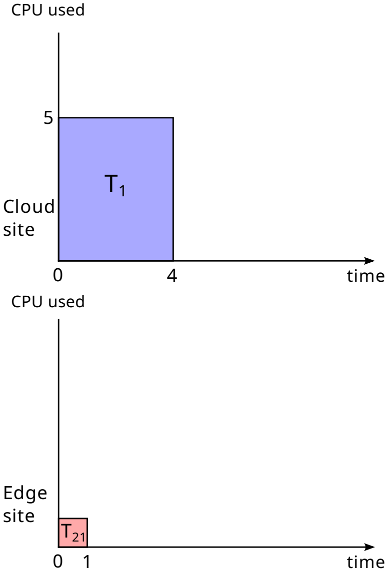
Solution 1



Example: Schedule

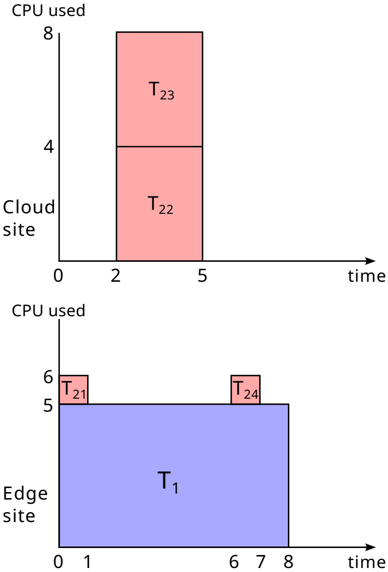


Solution 1

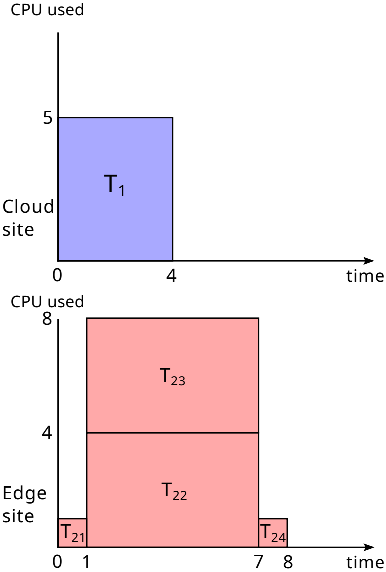


Solution 2

Example: Schedule

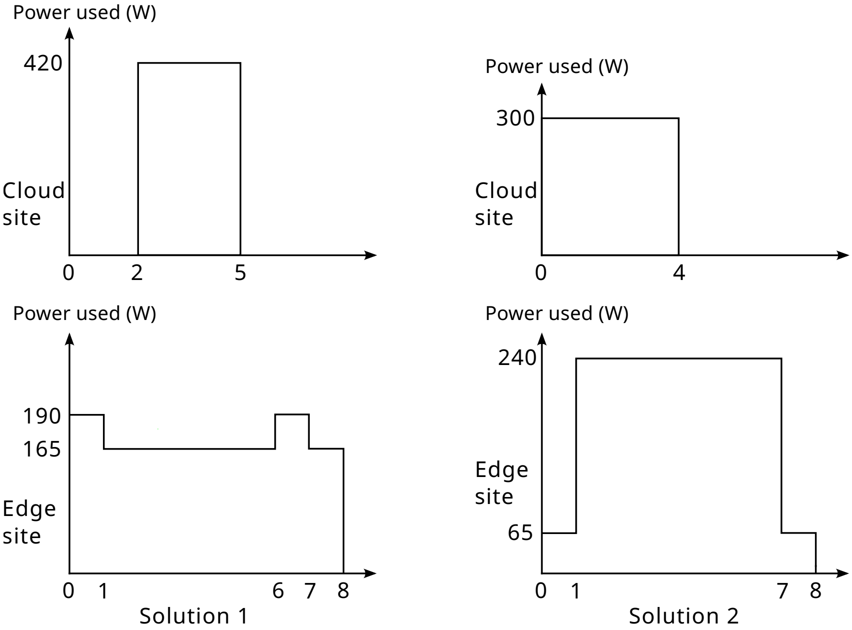


Solution 1

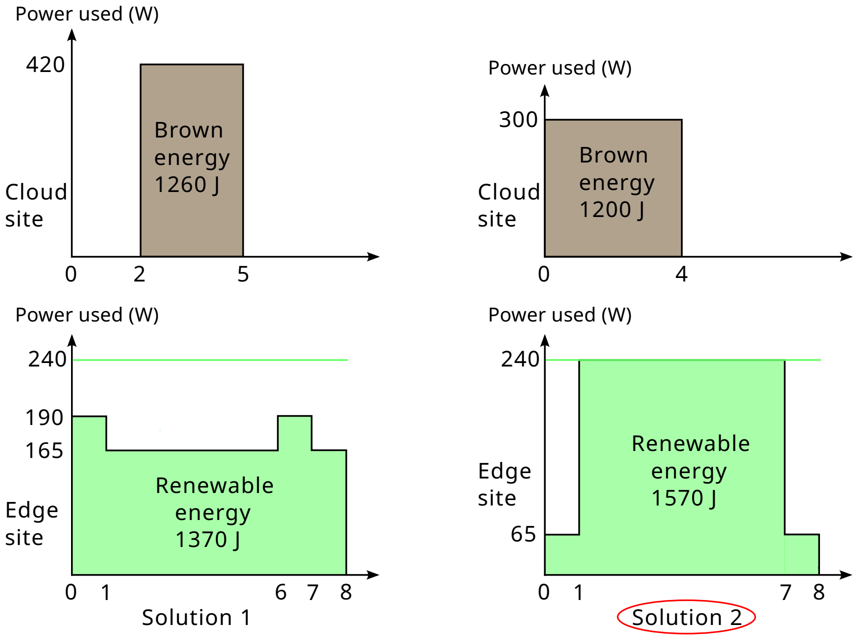


Solution 2

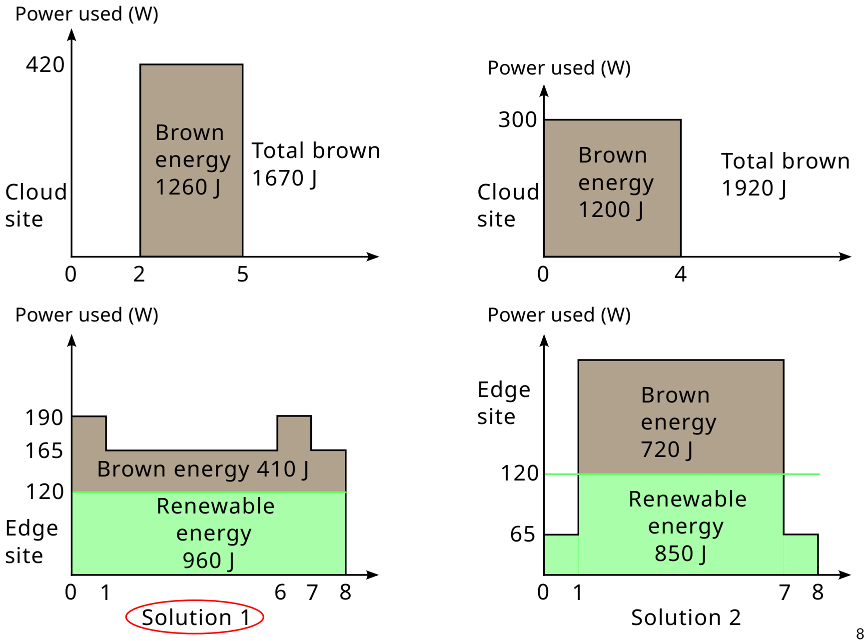
Example: Energy Consumption



Example: Energy Consumption



Example: Energy Consumption



Solving the Problem

So how to actually solve the problem?

- **First try** with Constraint Programming
- **Second try** using 'Classical' Scheduling Algorithms

First Try: with Constraint Programming

MiniZinc

An open-source constraint modelling language which can be used in conjunction with multiple back-end solvers (*Gecode*, *OR-Tools*, etc.).

Algorithm: Upon submission of a job, instanciate a MiniZinc model and solve it to schedule all job tasks, given the current state of the platform.

Current MiniZinc models:

- Exact model: Optimal objective value (additionnal brown energy consumed) but slow to solve (several minutes)
- Approximate model: Each task consumes a fraction of the P_{static}
  $\rightarrow$ Approximate objective value but fast to solve (less than a second)

Second Try: using 'Classical' Scheduling Algorithms

Develop greedy algorithms and their variants:

- Greedy algorithms
- Local search and exhaustive search variants
- (Metaheuristics)

Definitions

Rank (*à la* HEFT)

The rank of a task denotes its average critical path to the last finishing task of the job. For a task k:

$$\text{rank}_k = \overline{\text{exec}_k} + \max_{p \in \text{successors of } k} \{\overline{\text{comm}_{kp}} + rank_p\}$$

→ Used to order tasks

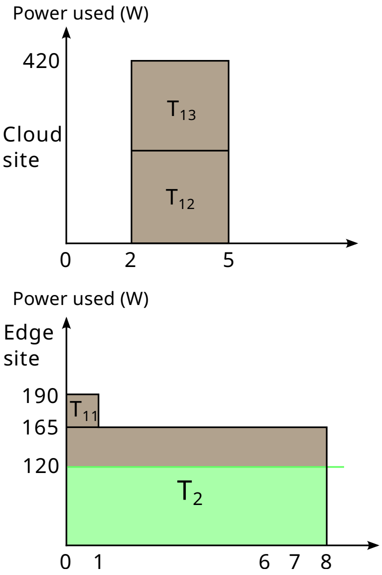
Candidate Locations

Define the possible placements of a task

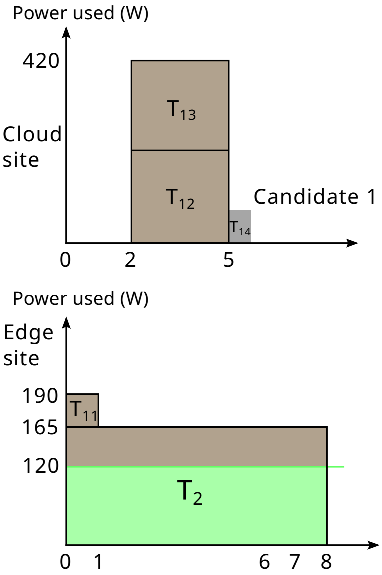
- the allocation (the 'where')
- the *anticipated* start time, *real* start time and finish time (the 'when')
- some metrics (additional total/brown/renewable energy consumption)

The locations are *feasible*: they respect the resource demands and deadline

Example: Candidate Location

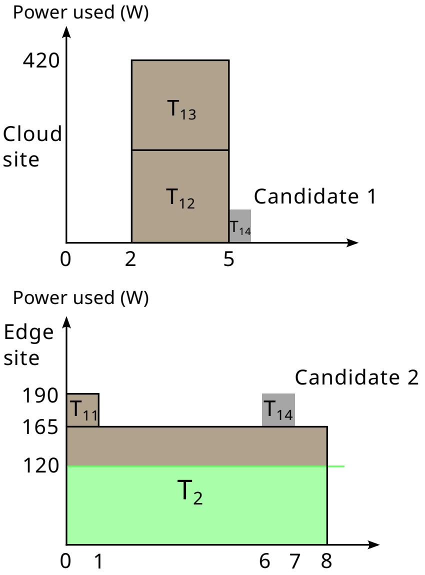


Example: Candidate Location



- Candidate 1: Cloud site
  Start time: 5, finish time: 5.5
  70J additional brown energy
  $(P_{\text{static}} + P_{\text{dynamic}})$

Example: Candidate Location



- Candidate 1: Cloud site
  Start time: 5, finish time: 5.5
  70J additional brown energy
  ($P_{\text{static}} + P_{\text{dynamic}}$)
- Candidate 2: Edge site
  Start time: 6, finish time: 7
  25J additional brown energy
  (P_{dynamic} only)

Algorithm: Greedy

Greedy Algorithm Skeleton

Upon job submission:

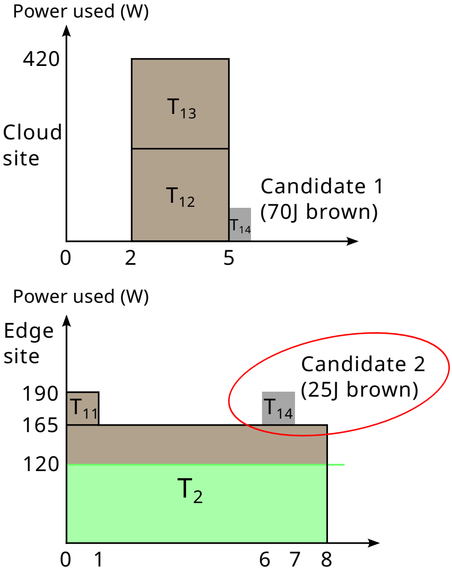
1. Sort its tasks by **decreasing rank** (follows the dependencies)
2. For each task:
3. Compute all feasible candidate locations
4. Place the task on the **best location**

If a task cannot be allocated (lack of free resource or deadline not met)
$\rightarrow$ Reject the whole job

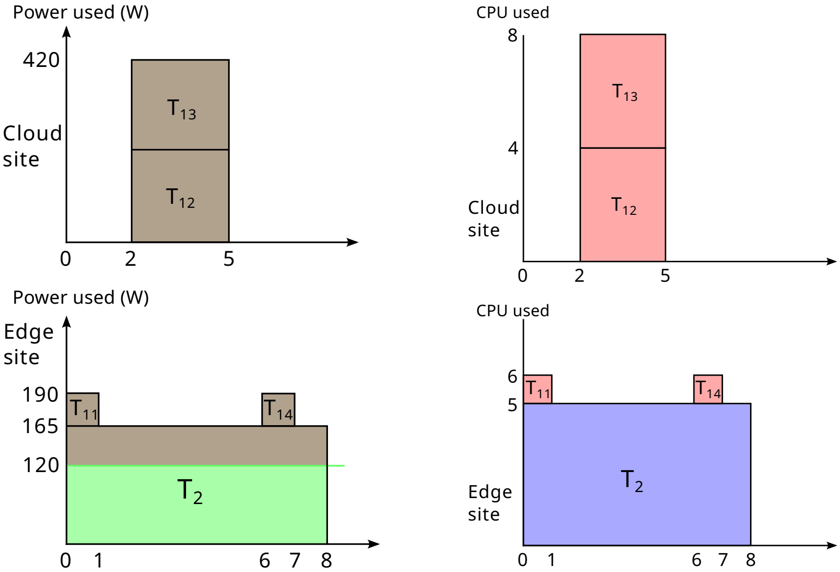
Selection policy for the best location:

- Smallest additionnal energy
- Smallest additionnal brown energy
- Some other combination (resource usage, energy, finish time)

Example: Greedy Algorithm



Example: Greedy Algorithm



Power usage CPU usage

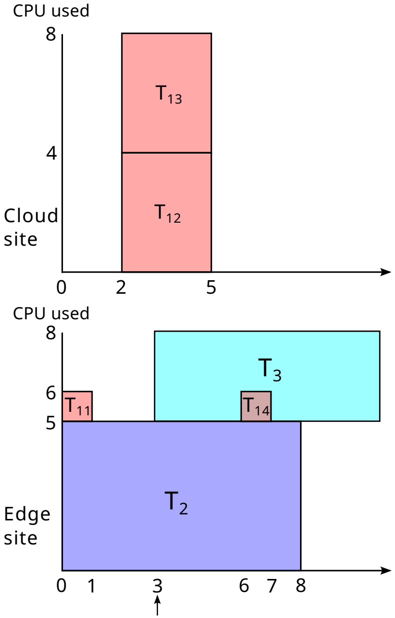
Example: Greedy Algorithm

→ New job submission at time 3

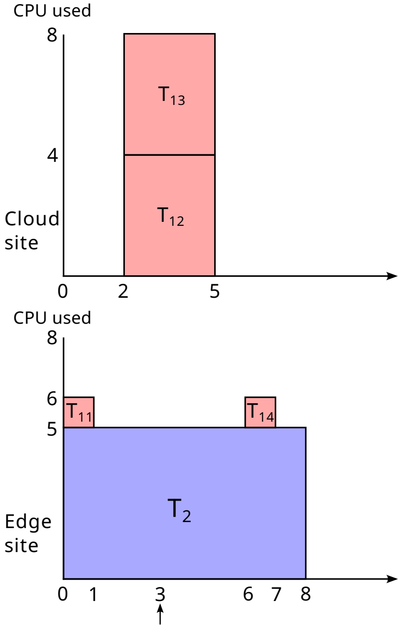
One task, Edge only, 3 CPUs

No space for T_3

- Reject the job (Greedy algorithm)
- Try to rearrange the schedule (Local Search)



Local Search variant



Local Search procedure

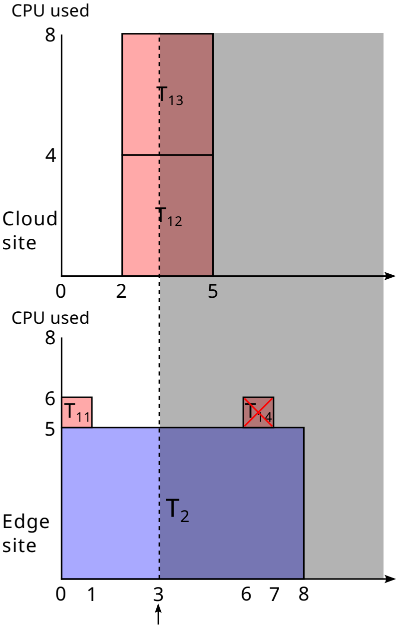
1. Make a backup of the platform state
2.
3.
4.

Local Search variant

Local Search procedure

1. Make a backup of the platform state
2. Remove all non-running tasks in the **"neighborhood"** of the problematic task
3.
4.

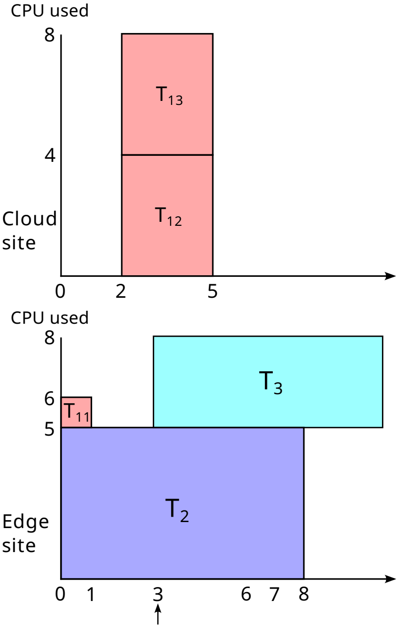
Neighborhood of the task:
From its earliest possible starting time onwards.



Local Search variant

Local Search procedure

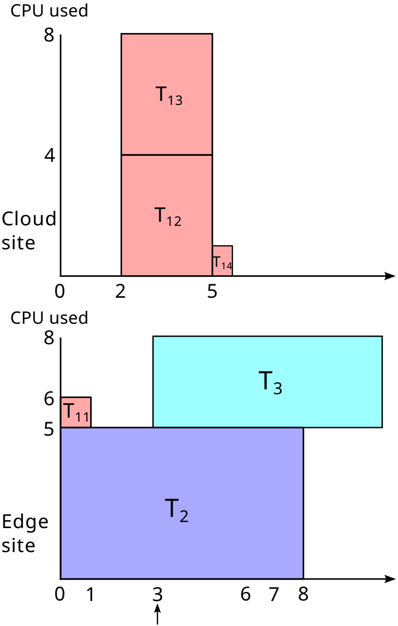
1. Make a backup of the platform state
2. Remove all non-running tasks in the **"neighborhood"** of the problematic task
3. Schedule the problematic task
4.



Local Search variant

Local Search procedure

1. Make a backup of the platform state
2. Remove all non-running tasks in the **“neighborhood”** of the problematic task
3. Schedule the problematic task
4. (Try to) schedule all tasks in the queue using **a scheduling procedure**



Local Search variants

Scheduling procedures

- **Greedy**: Simply apply the greedy algorithm.
- **Greedy with LS**: Perform greedy but apply the Local Search procedure if necessary.
- **Exhaustive search**: Recursively try all candidate locations for all tasks. Apply the best solution found.

$\rightarrow$ If a task cannot be allocated, revert to backed up state and reject the job of first problematic task.

Experimental Campaign

Not really started yet
(but algorithms are implemented)

- Try different types of platforms
- Try different collections of jobs

For that we need datasets
(maybe look at *Azure* ones?)

$\rightarrow$ Do you have ideas of datasets?
Please come and talk to me

Conclusion

A classical scheduling problem with uncommon constraints:

- Renewable power sources
- Allocation filtering with labels
- Immediate starting times (no delay)

On-going and future work:

- Focus on the experimental part
- Continue the algorithm design/implementation with new ideas
- Design good search heuristics for the Constraint Programming models

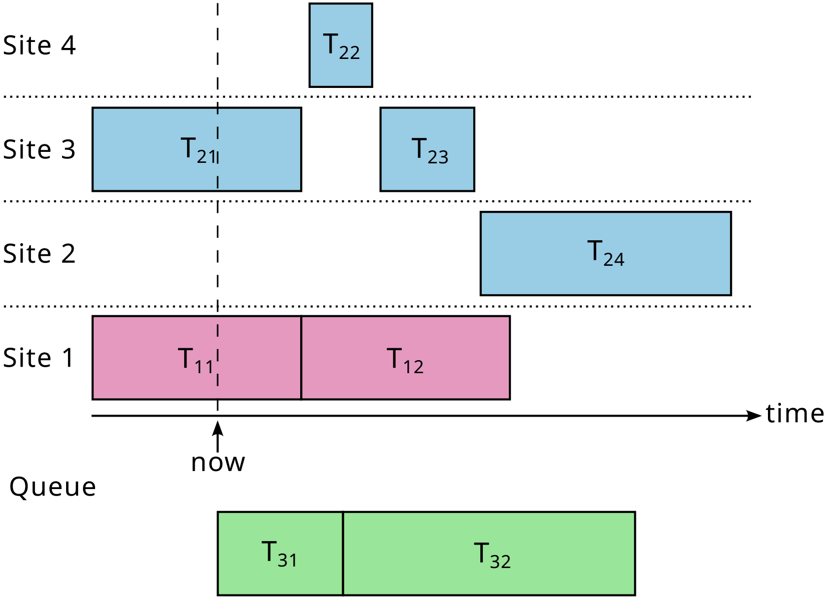
Scheduling Tasks with Precedences on Edge-Cloud Platforms Partially Powered with Renewable Energy

Clément Mommessin
Remous-Aris Koutsiamanis   Jean-Marc Menaud

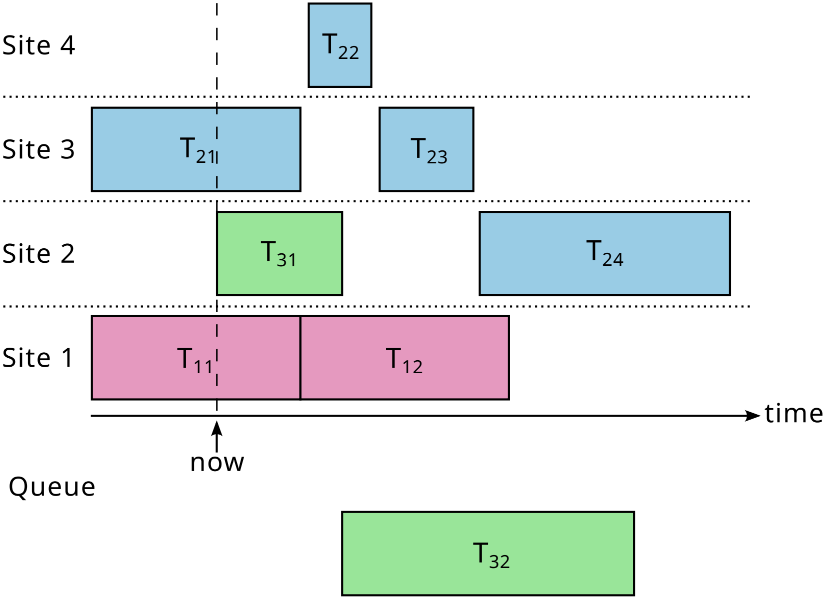
IMT Atlantique, Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, F-44000 Nantes, France
{clement.mommessin, remous-aris.koutsiamanis, jean-marc.menaud}@imt-atlantique.fr

May 16, 2024

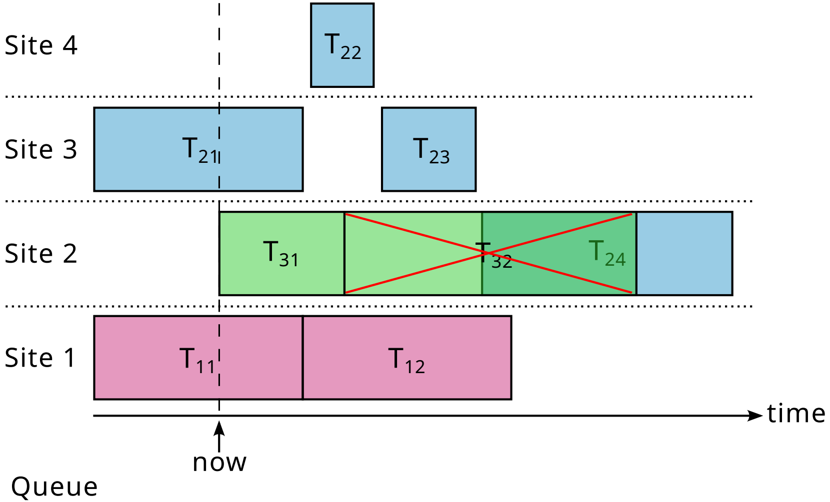
New Local Search Example



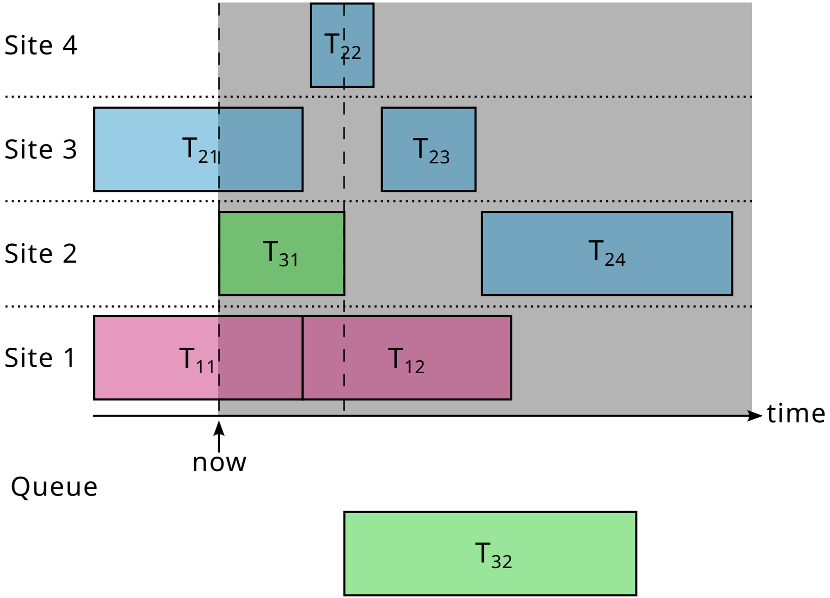
New Local Search Example



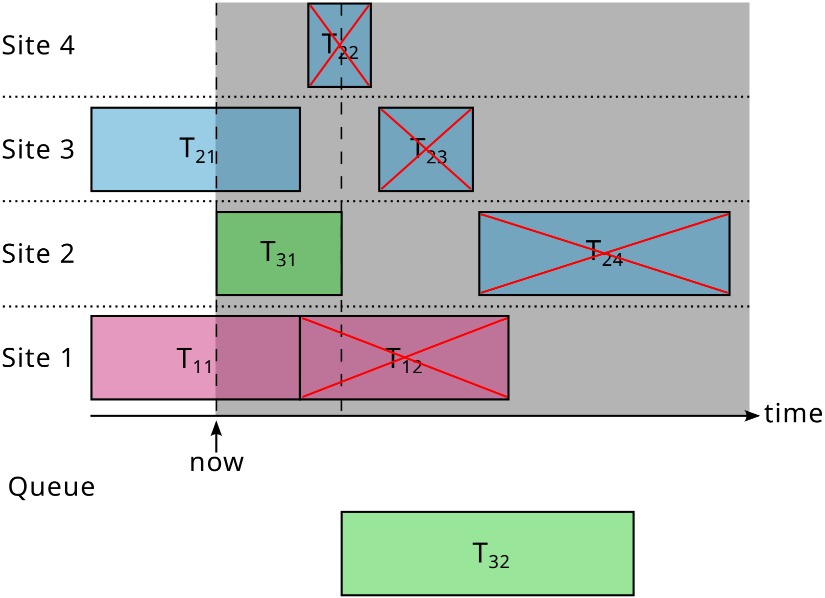
New Local Search Example



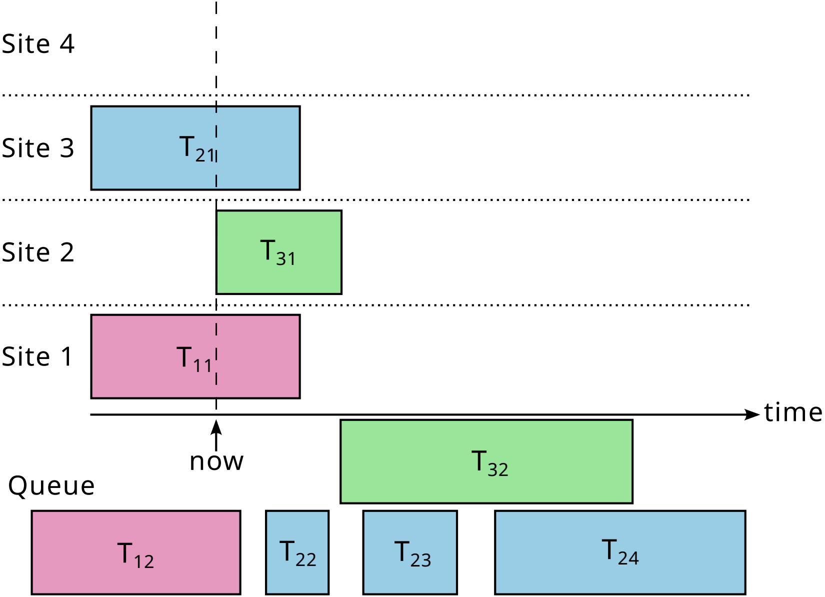
New Local Search Example



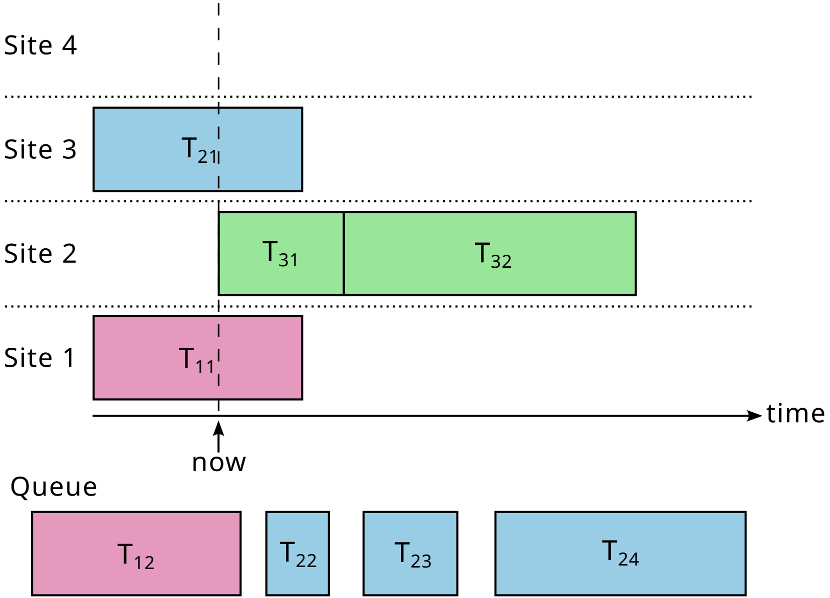
New Local Search Example



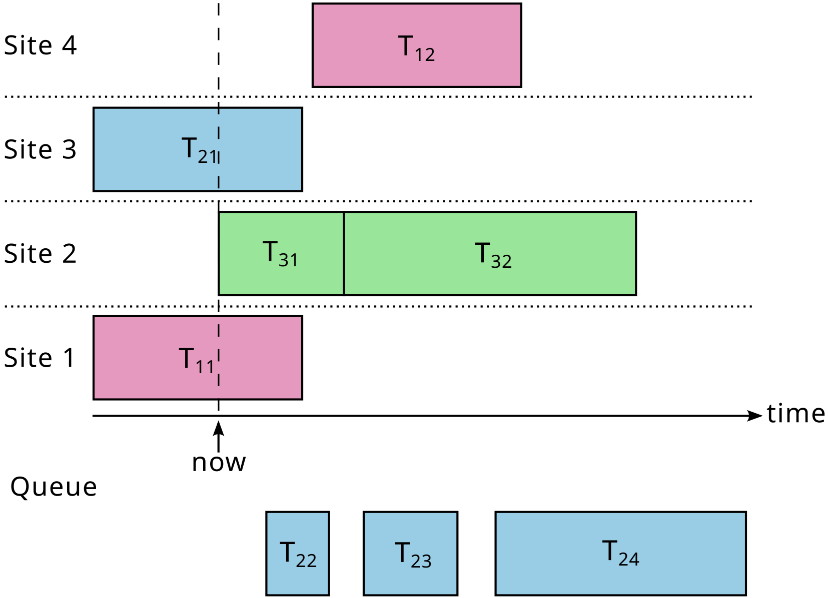
New Local Search Example



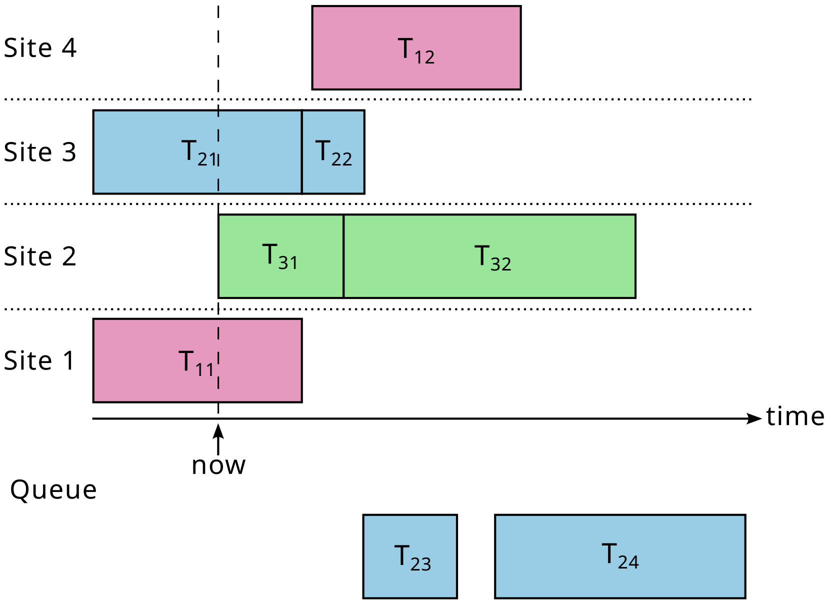
New Local Search Example



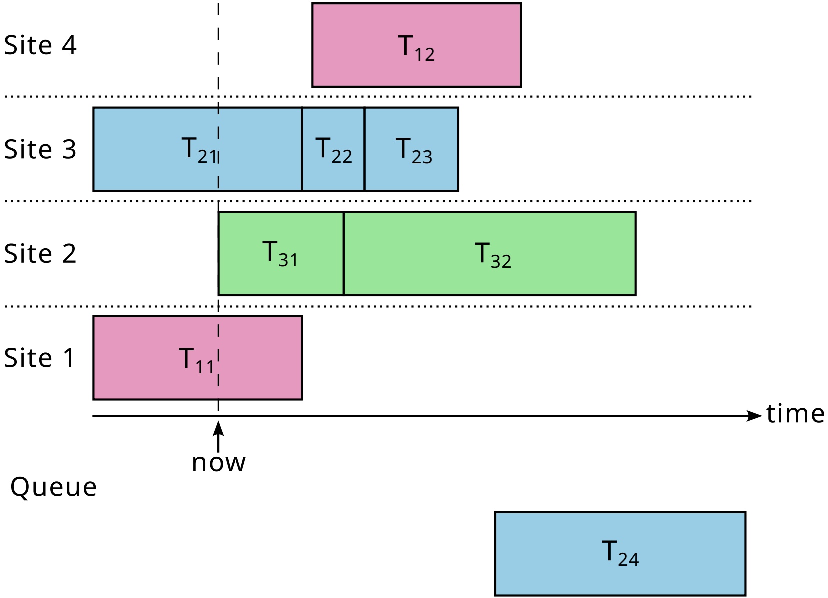
New Local Search Example



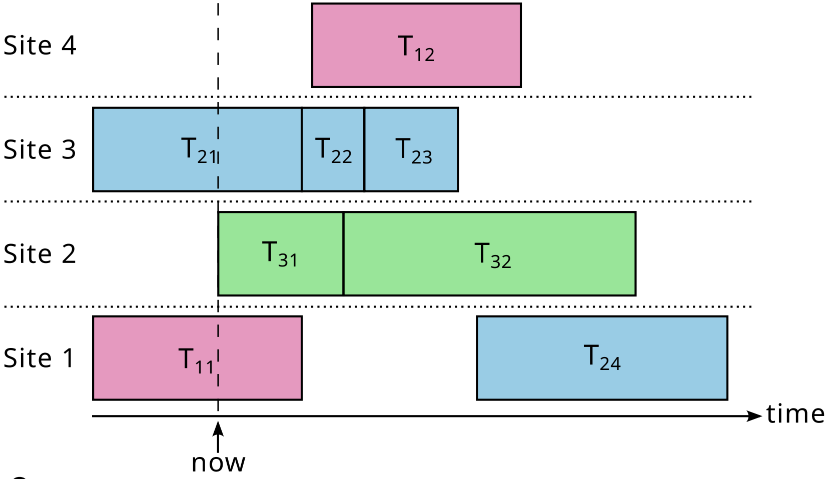
New Local Search Example



New Local Search Example



New Local Search Example



Queue