Max-stretch minimization on an edge-cloud platform

Anne Benoit\textsuperscript{1}, Redouane Elghazi\textsuperscript{1,2}, Yves Robert\textsuperscript{1,3}

1. LIP, ENS Lyon, France
2. FEMTO, Univ. Franche Comté, Besançon, France
3. UT Knoxville, TN, USA

IPDPS, May 2021, Virtual presentation
Introduction and motivation

Edge-Cloud computing:

- Execute some jobs in-situ, directly on edge server where they originate from
- Delegate some jobs to a powerful cloud platform to avoid overloaded edge servers

For instance: smart radiators, mobile gaming, autonomous vehicles, flying drones, ...

Decide which job to communicate to the cloud platform
**Objective function**

- **Response time** (or flow time) for a job: time spent by that job in the system, starting from its release date and up to final completion
  → Classical objective: minimize maximum response time

- **Stretch**: response time normalized by job length
  → ensures fairness among jobs

- **Two jobs released at same time, durations 1min / 10min, with long comm. time:**
  - Long job first → maximum stretch 11
  - Short job first → maximum stretch 1.1
  - Both cases → maximum response time 11
Framework

Two-level platform: $P^c$ homogeneous processors in a cloud (speed 1), and $P^e$ edge computing units (speed $s_j \leq 1$)

Independent jobs $J_1, \ldots, J_n$; For $1 \leq i \leq n$:

- $o_i$: origin processor on the edge ($1 \leq o_i \leq P^e$)
- $w_i$: amount of work required to complete the job
- $r_i$: release date
- $u_p_i$ and $d_n_i$: communication times required to send the job to the cloud and get the result back (uplink/downlink comms)

Processing times:

- $t_i^e = \frac{w_i}{s_{o_i}}$ on the edge
- $t_i^c = u_p_i + w_i + d_n_i$ on the cloud

Preemption is possible, but not migration
A single edge processor, with speed $\frac{1}{3}$, and six jobs:

$J_1$: $r_1 = 0$, $w_1 = 1$, $up_1 = dn_1 = 5$;
$J_2$: $r_2 = 0$, $w_2 = 4$, $up_2 = dn_2 = 2$;
$J_3$: $r_3 = 3$, $w_3 = 2$, $up_3 = 2$, $dn_3 = 1$;
$J_4$: $r_4 = 5$, $w_4 = 4/3$, $up_4 = dn_4 = 5$;
$J_5$: $r_5 = 5$, $w_5 = 2$, $up_5 = 2$, $dn_5 = 1$;
$J_6$: $r_6 = 6$, $w_6 = 1/3$, $up_6 = dn_6 = 5$. 
Optimization problem

**Goal:** Minimize the maximum stretch:
- $C_i$: time at which execution of $J_i$ is completed
- $S_i = \frac{C_i - r_i}{\min(t_i^e, t_i^c)}$
- $S_i = 1$ if job executed with minimum possible time
- Objective: Minimize $\max_{1 \leq i \leq n} S_i$

**Constraints:**
- Overlap computations and comms; full-duplex comm. channels
- Sequentialize comms involving a common processor

**MinMaxStretch-EdgeCloud problem:** Find a schedule that respects all constraints, with the aim of minimizing the max. stretch

![Diagram of the MinMaxStretch-EdgeCloud problem with jobs and scheduling]

- $S_2 = 1$
- $S_3 = S_5 = \frac{6}{5}$
- $S_1 = S_6 = 1$; $S_4 = \frac{5}{4}$
Online vs offline

Example: Decisions more difficult to take when there is no knowledge about jobs that will be released in the future.
One could schedule job $J_3$ either on the edge or on the cloud: would complete at time 9 in both cases!
Depending on the jobs that come next (computation-intensive vs comm-intensive), one decision would be better than the other...

- **Online** case: problem where jobs are not known in advance
- **Offline** case: all job parameters are known in advance

We prove that the problem is NP-complete even in the offline case (see paper or long presentation), and we derive heuristics to address the general online problem.
Heuristics

- **Event-based algos**: Reconsider decisions only when event occurs
- **Polynomial-time algorithms**, inspired from existing algorithms in the homogeneous case, but need to carefully choose proc. for each job

- **Edge-Only**: All jobs are executed **locally on the edge**
- **Greedy**: Schedules first the job that would currently achieve the highest stretch
- **SRPT**: Builds on classical SRPT strategy, i.e., assigns to a processing unit the job that it can finish the earliest
- **SSF-EDF**: Set a target stretch, give priorities to job according to deadlines, execute highest priority job on processor that **minimizes its stretch**
Simulations

- Implementation of simulation tool and heuristics in C++
- Use of parameters from real edge-cloud platforms

**Random instances:** 20 cloud processors, 10 slow edge processors with speed $s = 0.1$, and 10 fast edge processors with speed $s = 0.5$; jobs generated using a uniform distribution for the execution and communication times, as well as the release date and the origin processor; CCRs ranging from 0.1 (compute-intensive scenario) to 10 (communication-intensive scenario)

**Kang instances:** different types of edge processors, depending on whether their computational unit is a GPU or a CPU, and their communication channel is 3G, LTE, or Wi-Fi; jobs created according to these values.

[github.com/Redouane-Elghazi/Max-Stretch-Minimization-on-an-Edge-Cloud-Platform.git](https://github.com/Redouane-Elghazi/Max-Stretch-Minimization-on-an-Edge-Cloud-Platform.git)
Results: Random instances

- **Load**: Average number of jobs originating from edge, simultaneously in the system (default load = 0.05, default CCR=1)

- New heuristics much better than **Edge-Only** for small CCRs

- **SSF-EDF** is the best in all scenarios, **SRPT** very close for small loads

- **Greedy** slightly behind, **SRPT** and **Greedy** do not scale well with load
Results: Kang instances

- **SRPT**, closely followed by **SSF-EDF**, is clearly the best.
- **Edge-Only** cannot keep up when the number of jobs increases.

**Key Points**

- CCR dictated by platform parameters
- **SSF-EDF**, closely followed by **SRPT**, is clearly the best
- **Edge-Only** cannot keep up when the number of jobs increases
The different algorithms may be useful in different situations:

- Note that all heuristics do not exceed a few seconds
- **SSF-EDF** gives the best solutions overall, but is the most costly
- **SRPT** is easier to implement and it is the fastest, very close to **SSF-EDF** with reasonable load
- **Greedy** can be better than **SRPT** with high loads, but more costly
- **Edge-Only**: costly solution that does not exploit the cloud
- **Importance of using cloud resources when available**, in particular when communication costs are not too important
Problem of scheduling independent jobs on an edge-cloud platform:

- Design of general model with realistic communication model
- Minimizing the maximum stretch is NP-complete, even without release dates and on a homogeneous platform
- Design of heuristic algorithms in online setting
- Algorithms delegating jobs to cloud much better than Edge-Only
- SSF-EDF very efficient, SRPT is an interesting (cheaper) alternative

Future work:

- Derive theoretical bounds for online algorithms (competitive results), for instance for some specific job distributions
- Address more complicated framework where cloud processors are not available full-time