Max-stretch minimization on an edge-cloud platform

Anne Benoit[1], Redouane Elghazi[1,2], Yves Robert[1,3]

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

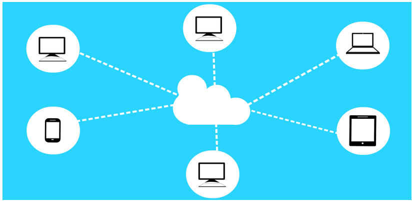
IPDPS, May 2021, Virtual presentation

ENS DE LYON

Introduction and motivation

Edge-Cloud computing:

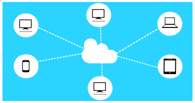
- Execute some jobs in-situ, directly on edge server where they originate from
- Delegate some jobs to a powerful cloud platform to avoid overloaded edge servers



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
  $\rightarrow$ Classical objective: minimize maximum response time
- Stretch: response time normalized by job length
  $\rightarrow$ ensures fairness among jobs
- Two jobs released at same time, durations 1min / 10min, with long comm. time:
  Long job first $\rightarrow$ maximum stretch 11
  Short job first $\rightarrow$ maximum stretch 1.1
  Both cases $\rightarrow$ maximum response time 11

Outline

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
- up_i and dn_i: communication times required to send the job to the cloud and get the result back (uplink/downlink comms)

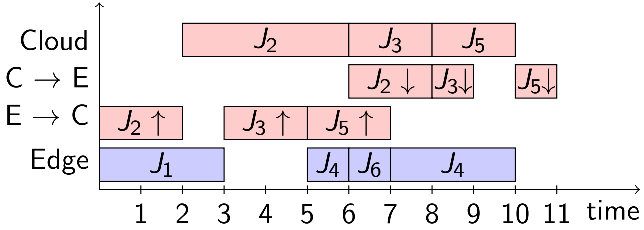
Processing times:

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

Preemption is possible, but not migration

An example

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

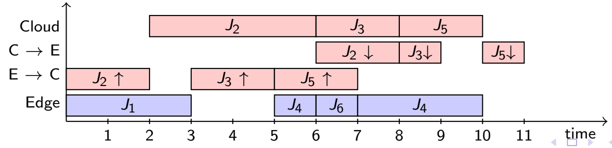
Goal: Minimize the maximum stretch:

- 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

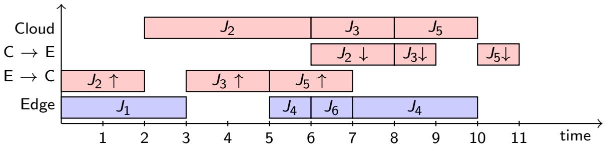


$S_2 = 1$

$S_3 = S_5 = \frac{6}{5}$

$S_1 = S_6 = 1$; $S_4 = \frac{5}{4}$

Online vs offline



$S_2 = 1$

$S_3 = S_5 = \frac{6}{5}$

$S_1 = S_6 = 1$; $S_4 = \frac{5}{4}$

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

In the following, we prove that the problem is difficult even in the offline case, and then we derive heuristics to address the general online problem

Outline

NP-Completeness

Complexity of MinMaxStretch-EdgeCloud in the offline case (MMSECO problem); MMSECO-Dec is the corresponding decision problem (is it possible to achieve a target max. stretch?):

- We prove that MMSECO-Dec is in NP
- For a fixed number of processors, MMSECO-Dec is NP-complete in the weak sense
- For a variable number of processors, MMSECO-Dec is NP-complete in the strong sense

The results remain true even without release dates (consider that all jobs are released at time 0)

NP-Completeness proofs outline

Proofs for a fixed number of homogeneous processors (forgetting about edge-cloud):

- We prove that, on each processor, jobs are ordered from the shortest to the longest, without preemption
- With 2 processors: 2-Partition with two large jobs (one per processor), executed last, each responsible for the maximum stretch of the processor
- With p processors: 3-Partition with p large jobs

MINMAXSTRETCH-EDGECLOUD setting harder than homogeneous processors (take one edge processor, speed 1, and no communication costs)

NP-Completeness proof: Example

Example of the reduction with two processors:

- Instance of 2-Partition: $A = \{1, 3, 5, 6, 10, 11\}$, with $n = 6$ and $S = 18$;
- We propose the instance of scheduling: $W = \{M+1, M+3, M+5, M+6, M+10, M+11, M+S, M+S\}$, where M is a large number that we take equal to 108 ($M = nS$), i.e., $W = \{109, 111, 113, 114, 118, 119, 126, 126\}$;
- We want to find a schedule with maximum stretch at most $\frac{44}{7}$ ($\frac{n^2+n+2}{n+1}$).

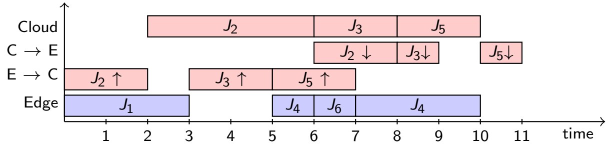
And in this case, there indeed exists a schedule with maximum stretch $\frac{44}{7}$ and a 2-partition of A.

Outline

Heuristics

- MinMaxStretch-EdgeCloud problem: NP-complete even in the offline case
- Design of heuristics for general online setting

- Event-based algos: Reconsider decisions only when event occurs
- At most $4n$ events; for job J_i, $1 \leq i \leq n$: (1) release date; (2) end of execution; (3) end of uplink communication; (4) end of downlink communication



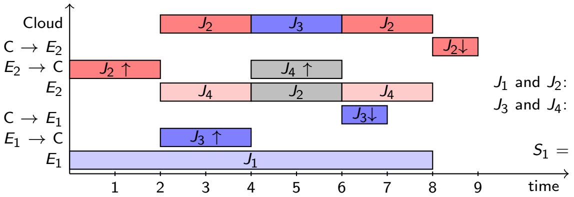
- Polynomial-time algorithms, inspired from existing algorithms in the homogeneous case, but need to carefully choose proc. for each job

Baseline: Edge-Only strategy

- *Edge-Only*: All jobs are executed locally on the edge
- Might be good when edge processing units have good processing speed and/or when communications are costly
- Independent processors: Minimize max-stretch on one processor
- Use Bender's algorithm Stretch-So-Far Earliest-Deadline-First: Δ-competitive with a single processor, where Δ is the ratio between the longest and the shortest job
- Account for edge-cloud framework: consider potential execution time of job on the cloud when computing its stretch

Greedy

- *Greedy* heuristic: Schedules first the job that would currently achieve the highest stretch
- At each event, as long as there are available resources, compute for each job the minimum stretch that can be achieved using an available resource immediately
- Select the job that maximizes this value, and execute the job on the resource (edge or cloud) on which it achieves the minimum stretch
- Worst case complexity: $O(n^2 P^c)$



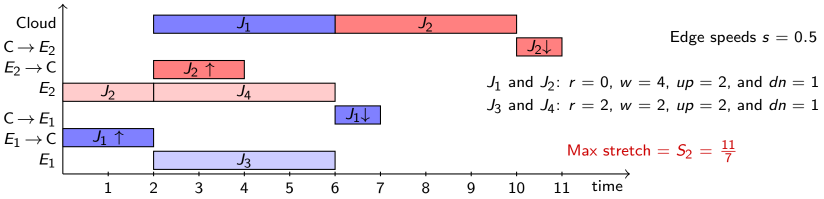
Edge speeds $s = 0.5$

J_1 and J_2: $r = 0$, $w = 4$, $up = 2$, and $dn = 1$
J_3 and J_4: $r = 2$, $w = 2$, $up = 2$, and $dn = 1$

$S_1 = \frac{8}{7}$; $S_2 = \frac{9}{7}$; $S_3 = \frac{5}{4}$; $S_4 = \frac{3}{2}$

Shortest Remaining Processing Time

- *SRPT*: Builds on classical SRPT strategy, i.e., assigns to a processing unit the job that it can finish the earliest
- Approach that has already be proven to be efficient to minimize the average stretch in the homogeneous case: $O(1)$-competitive for average stretch
- Shortest jobs will be executed with little delay
- For maximum stretch, a long job could be blocked by short jobs for an arbitrary long duration



Edge speeds $s = 0.5$

J_1 and J_2: $r = 0$, $w = 4$, $up = 2$, and $dn = 1$

J_3 and J_4: $r = 2$, $w = 2$, $up = 2$, and $dn = 1$

Max stretch $= S_2 = \frac{11}{7}$

Stretch-so-far Earliest-deadline-first

Algorithm of Michael A. Bender in the case of one processor:

- Give deadlines to jobs, based on estimate of maximum stretch
- Binary search on the stretch to find the optimal one
- Given target stretch, use Earliest-deadline-first (*EDF*) algorithm, optimal on a single processor
- Competitive ratio $\Delta = \frac{w_{max}}{w_{min}}$

Communication times: *EDF* is not optimal anymore!
Two jobs and one cloud processor:
J_1 with $up_1 = 3$, $w_1 = 1$, $dn_1 = 0$, and deadline $d_1 = 5$
J_2 with $up_1 = 1$, $w_2 = 3$, $dn_2 = 0$, and deadline $d_2 = 6$

- *EDF*: sends J_1 first; J_2 starts communication at time 3 and completes at time $3 + 1 + 3 = 7 > d_2$
- Executing J_2 first: both jobs meet their deadline

Need to adapt this algo. for MinMaxStretch-EdgeCloud

Stretch-so-far Earliest-deadline-first

SSF-EDF heuristic:

- *EDF* tells us which job to consider (highest priority)
- Need to decide on which processor to execute this job
- $\rightarrow$ Execute the job on processor that minimizes its stretch
- Iterate on other jobs sorted by non-decreasing deadlines
- Binary search on target stretch

Running example: *SSF-EDF* outputs the same schedule as *SRPT*, with the same maximum stretch of $\frac{11}{7}$

Outline

Simulations

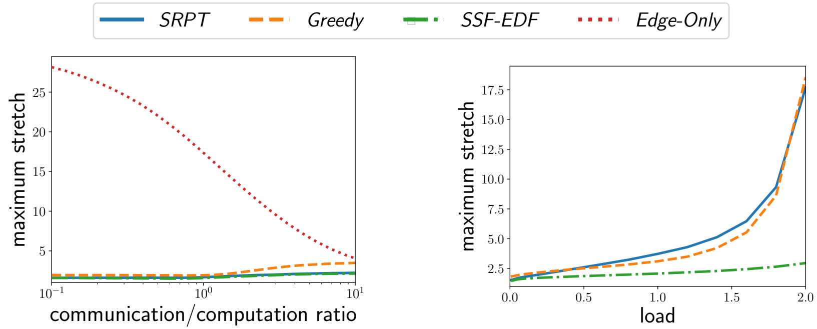
- Implementation of simulation tool and heuristics in C++
- Use of parameters from real edge-cloud platforms

- 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.

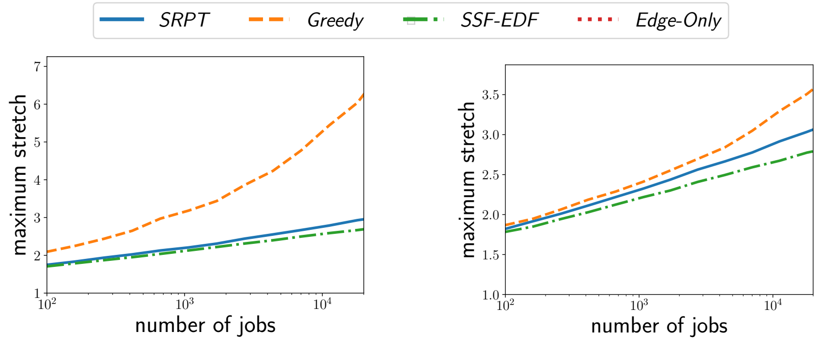
`github.com/Redouane-Elghazi/Max-Stretch-Minimization-on-an-Edge-Cloud-Platform.git`

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



20 edge proc. and 10 cloud proc. 100 edge proc. and 10 cloud proc.

- 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

Trade-off: Solution quality vs execution time

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

Outline

Conclusion

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