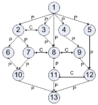
# Minimizing rental cost for multiple recipe applications in the Cloud

F. Hana, L. Marchal, J.-M. Nicod, L. Philippe, V. Rehn-Sonigo and H. Sabbah

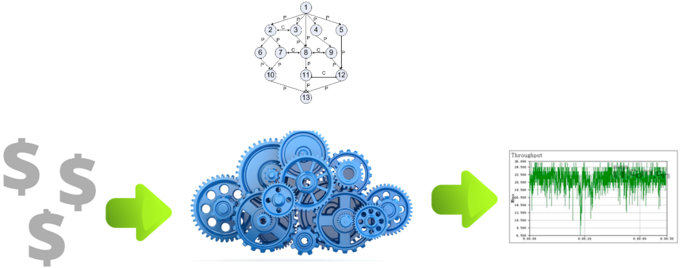
*LIP-ENS Lyon – FEMTO-ST institute - UFC/ENSMM Besançon*

Nashville - May 18th, 2016

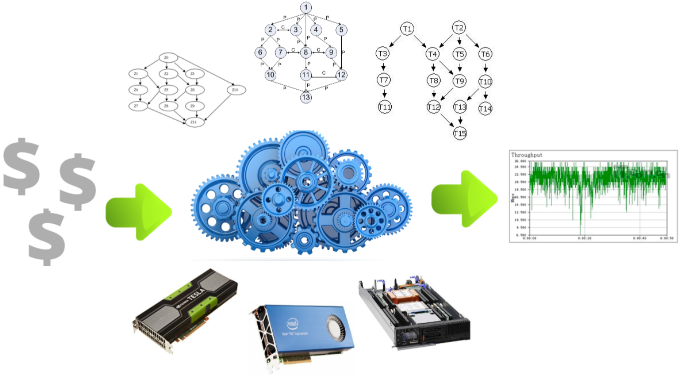
# Introduction and motivation



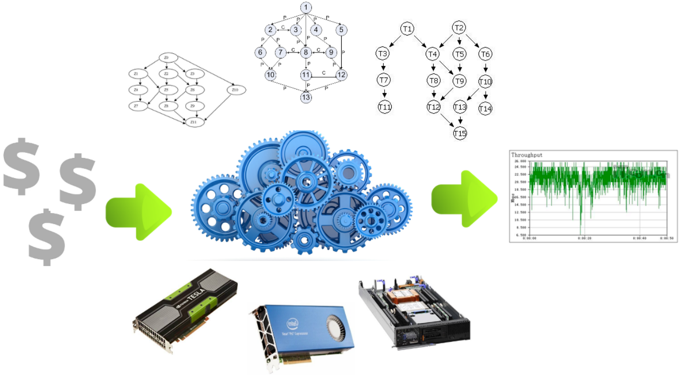

# Introduction and motivation



# Introduction and motivation



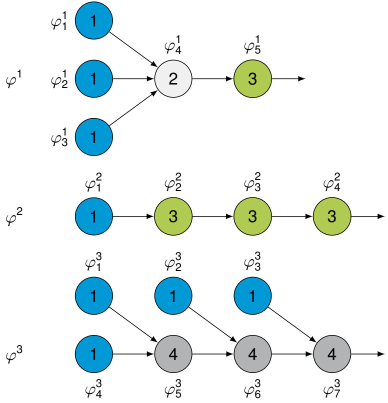
# Introduction and motivation




**Overall objective:** To provision just enough resources to reach the target throughput for a given DAG-based streaming application

# Application framework

Each workflow application $\varphi^j$ produces the same result $\Phi$.




- Each task $\varphi^j_i$ has a task type $q$
- Target throughput $\rho$
- Each application can be run at a different throughput $\rho_j$
- $\rho = \sum_j \rho_j$

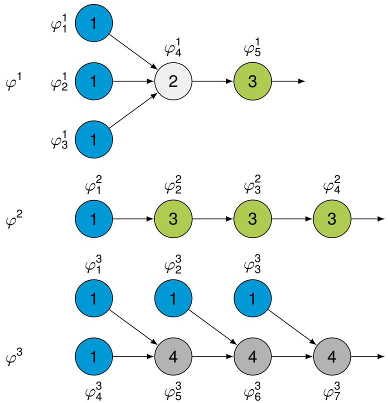
**Target platform**

One processor type per task type

- $c_q$: Rental cost for type $q$
- $r_q$: Throughput of type $q$

# Application framework

Each workflow application $\varphi^j$ produces the same result $\Phi$.




- Each task $\varphi_i^j$ has a task type $q$
- Target throughput $\rho$
- Each application can be run at a different throughput $\rho_j$
- $\rho = \sum_j \rho_j$

**Target platform**

One processor type per task type

- $c_q$: Rental cost for type $q$
- $r_q$: Throughput of type $q$

➡ Find the cheapest configuration to reach the target throughput

# Problem definition

**MinCOST: Minimize the global rental cost $C$**

Given

- an application described by $J$ graphs
- a platform described by processor cost $c_q$ and throughput $r_q$
- a target throughput $\rho$

$\Rightarrow$ select which graphs $\varphi^j$ are used

$\Rightarrow$ choose their output throughput $\rho_j$ ($\rho_j = 0$ if unused)

$\Rightarrow$ deduce the number of processors $x_q$ of each type

to reach the prescribed throughput within minimal cost.

# Problem resolution

## Simple case

- The application is described by a single graph

## General case

$$\rho = \sum_j \rho_j$$

- Black box application
- Application graphs without shared task types
- Application graphs with shared task types

# Simple case: Single application graph

- One application described by one single graph $\varphi^1$
- $\forall q$, the number of machines $x_q$ can be easily computed:

$$x_q = \left\lceil \frac{n_q}{r_q} \cdot \rho \right\rceil$$

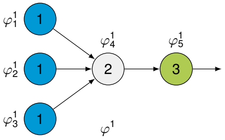
- The associated cost $C_q$

$$C_q(\rho) = \left\lceil \frac{n_q}{r_q} \cdot \rho \right\rceil \times c_q$$

- The final cost $C$:

$$C(\rho) = \sum_{q=1}^{Q} C_q(\rho) = \sum_{q=1}^{Q} \left\lceil \frac{n_q}{r_q} \cdot \rho \right\rceil \times c_q$$

$n_q$: number of tasks, $r_q$: throughput, $c_q$: cost of type q



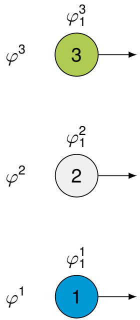
## General case: Black box applications

- Each graph $\varphi^j = \varphi^j_1$ is one complex task

$$\forall j \text{ and } \forall j' (1 \leq j, j' \leq J): \; t(1,j) = t(1,j') \Rightarrow j = j'$$

- Let $\rho_q$ the output of $\varphi^j = \varphi^q$
- $x_q$ can be found by solving the following linear program:

$$\begin{cases} \text{Minimize } C(\rho) = \sum_{q=1}^{Q} x_q c_q \\ \text{Under the constraint } \sum_{q=1}^{Q} x_q \rho_q \geqslant \rho \end{cases}$$




$\Rightarrow$ This resembles a knapsack problem with repetition using negative weights and values

# General case: Black box applications

## Unbounded Knapsack Problem

Given $n$ objects with value $v_i$ and weight $w_i$, and a total capacity of $W$, how many copies of each object should we select to maximize the total value without exceeding weight $W$ ?

ILP formulation with $x_i$ the number of copies of item $i$ included in the solution

$$\begin{cases} \text{Maximize } \sum x_i v_i \\ \text{Under the constraint } \sum x_i w_i \leqslant W \end{cases}$$

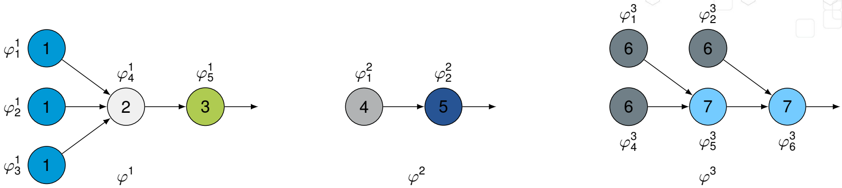
Our problem is thus equivalent to a knapsack problem where:

- Items have value $(-c_q)$ and weight $(-\rho_q)$
- The total capacity is $(-\rho)$.

$\Rightarrow$ The knapsack problem is a (unary) NP-complete problem

$\Rightarrow$ There exists a pseudo-polynomial dynamic program (time complexity $O(J\rho)$)

# Application graphs without shared task types




- Application $\Phi$ can be described by $\varphi^1, \ldots, \varphi^j, \ldots, \varphi^J$ with the same output result
- Each task $\varphi_i^j$ from one graph $\varphi^j$ has a different type from every other task of an other graph $\varphi^{j'}$

$$t(i,j) \neq t(i',j') \text{ with } 1 \leq j, j' \leq J \text{ and } j \neq j' \text{ and } 1 \leq i \leq I_j \text{ and } 1 \leq i' \leq I_{j'}$$

- This problem is unary NP-complete (includes Black Box applications)

$\Rightarrow$ There exists a pseudo-polynomial dynamic program to solve it

# Application graphs without shared task types

## A dynamic program to solve this problem

- Let $C(\rho, j)$ be the optimal platform cost to reach $\rho$ using the first $j$ application graphs

$$C(\rho, j) = \begin{cases} \sum_{i=1}^{I_1} \left\lceil \frac{n^1_{t(i,1)}}{r_{t(i,1)}} \cdot \rho \right\rceil \times c_{t(1,k)} \text{ if } j = 1 \\ \min_{0 \leq \rho_j \leq \rho} \Bigg( C(\rho - \rho_j, j-1) + \\ \qquad \sum_{i=1}^{I_j} \left\lceil \frac{n^j_{t(i,j)}}{r_{(t(i,j))}} \cdot \rho_j \right\rceil \times c_{t(i,j)} \Bigg) \\ \text{otherwise} \end{cases}$$

$\Rightarrow$ The solution is given by $C(\rho, J)$

# Application graphs without shared task types

## Complexity analysis

- As $\forall q$, $r_q \in \mathbb{N}$, $\forall j$ $\rho_j \in \mathbb{N}$

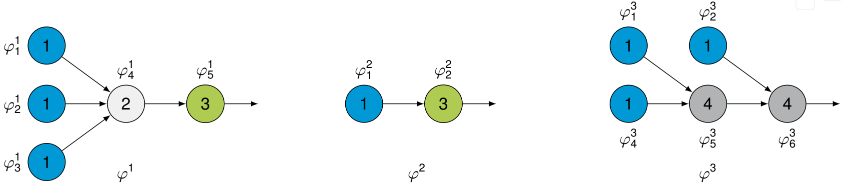
$\rightarrow$ There exists a finite number of $\rho_j$ to test in the previous formulation

- To compute $C(\rho, j)$, all $C(\rho', j')$ with $\rho' \leq \rho$ and $j' \leq j$ has to be computed

$\Rightarrow$ The complexity of the elementary computation is $O(\rho I)$

$\Rightarrow$ The complexity of computing $C(\rho)$ is $O(\rho^2 IJ)$

# Application graphs with shared task types




- One application is described by several graphs which share task types

$$\exists\, \varphi^j, \varphi^{j'} (1 \leqslant j, j' \leqslant J, j \neq j'), \exists\, i\ (1 \leqslant i \leqslant I_j),$$
$$\exists\, i' (1 \leqslant i' \leqslant I_{j'})\ \text{ s.t. }\ t(i,j) = t(i',j')$$

$\Rightarrow$ A processor may be shared between several application graphs

# Application graphs with shared task types

## ILP formulation

Minimizing $C(\rho) = \sum_{q=1}^{Q} x_q \cdot c_q$
under the constraints

- $\rho$ has to be at least the sum of $\rho_j$

$$\sum_{j=1}^{J} \rho_j \geqslant \rho$$

- For each type $q$ we have to provision enough resources ($x_q$)

$$\forall q \; x_q \cdot r_q \geqslant \sum_{j=1}^{J} \left( \sum_{i=1 \,|\, t(i,j)=q}^{I_j} \rho_j \right),$$

with $q = t(i,j)$ and $x_q \in \mathbb{N}$

# Application graphs with shared task types

## ILP formulation

Minimizing $C(\rho) = \sum_{q=1}^{Q} x_q \cdot c_q$
under the constraints

- $\rho$ has to be at least the sum of $\rho_j$

$$\sum_{j=1}^{J} \rho_j \geqslant \rho$$

- For each type $q$ we have to provision enough resources ($x_q$)

$$\forall q \; x_q \cdot r_q \geqslant \sum_{j=1}^{J} \left( \sum_{i=1 \mid t(i,j)=q}^{I_j} \rho_j \right),$$

with $q = t(i,j)$ and $x_q \in \mathbb{N}$

The complexity of this case is still open
unary or binary NP-complete

# Application graphs with shared task types

## Heuristic approaches

$$\rho = (\rho_1, \rho_2, \ldots, \rho_J)$$

1. H0: random
2. H1: best graph
3. H2: random walk
4. H31: stochastic descent
5. H32/H32Jump: steepest gradient

# Application graphs with shared task types

## Heuristic approaches

$$\rho = (\rho_1, \rho_2, \ldots, \rho_J)$$

1. H0: random $\rho = (\rho_1, \rho_2, \ldots, \rho_J)$
2. H1: best graph
3. H2: random walk
4. H31: stochastic descent
5. H32/H32Jump: steepest gradient

# Application graphs with shared task types

## Heuristic approaches

$$\rho = (\rho_1, \rho_2, \ldots, \rho_J)$$

1. H0: random
2. H1: best graph — $\rho = (0, \ldots, \rho, \ldots, 0)$
3. H2: random walk
4. H31: stochastic descent
5. H32/H32Jump: steepest gradient

# Application graphs with shared task types

## Heuristic approaches

$$\rho = (\rho_1, \rho_2, \dots, \rho_J)$$

1. H0: random
2. H1: best graph
3. H2: random walk
   - $\varphi_{j1}$ and $\varphi_{j2}$ are randomly chosen

$$(\dots, \rho_{j1}, \dots, \rho_{j2}, \dots) \to (\dots, \rho_{j1} - \delta, \dots, \rho_{j2} + \delta, \dots)$$

$$(\dots, \rho_{j1}, \dots, \rho_{j2}, \dots) \to (\dots, 0, \dots, \rho_{j2} + \rho_{j1} \dots) \quad \text{if } \rho_{J1} < \delta$$

4. H31: stochastic descent
5. H32/H32Jump: steepest gradient

# Application graphs with shared task types

## Heuristic approaches

$$\rho = (\rho_1, \rho_2, \ldots, \rho_J)$$

1. H0: random
2. H1: best graph
3. H2: random walk
4. H31: stochastic descent
   - Same as H2 except that we keep the same solution as long as we do not obtain any improvement
5. H32/H32Jump: steepest gradient

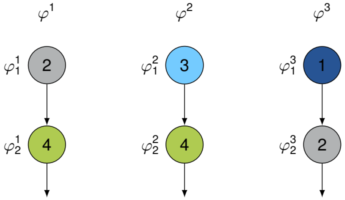
# Application graphs with shared task types

## Heuristic approaches

$$\rho = (\rho_1, \rho_2, \ldots, \rho_J)$$

1. H0: random
2. H1: best graph
3. H2: random walk
4. H31: stochastic descent
5. H32/H32Jump: steepest gradient
   - H32/H32Jump same as H2 except we test all possible throughput fraction exchanges and keep the best until no more improvement is possible
   - H32Jump allows to explore solution that increases $C(\rho)$

# Illustrating example

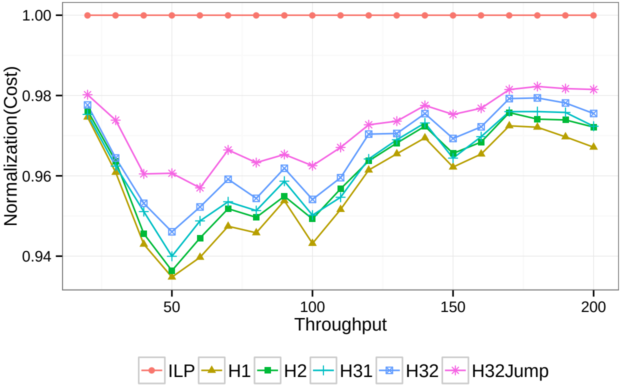



| Processor | type | $\rho$ | cost |
|---|---|---|---|
| $P_1$ | $t_1$ | 10 | 10 |
| $P_2$ | $t_2$ | 20 | 18 |
| $P_3$ | $t_3$ | 30 | 25 |
| $P_4$ | $t_4$ | 40 | 33 |

## Results

| | I L P | | | | H1 | | | | H 2 | | | | H 32 JUMP | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\rho$ | $\rho_1$ | $\rho_2$ | $\rho_3$ | cost | $\rho_1$ | $\rho_2$ | $\rho_3$ | cost | $\rho_1$ | $\rho_2$ | $\rho_3$ | cost | $\rho_1$ | $\rho_2$ | $\rho_3$ | cost |
| 10 | 0 | 0 | 10 | **28** | 0 | 0 | 10 | **28** | 0 | 0 | 10 | **28** | 0 | 0 | 10 | **28** |
| 40 | 40 | 0 | 0 | **69** | 40 | 0 | 0 | **69** | 40 | 0 | 0 | **69** | 40 | 0 | 0 | **69** |
| 50 | 10 | 30 | 10 | **86** | 0 | 0 | 50 | 104 | 10 | 30 | 10 | **86** | 10 | 30 | 10 | **86** |
| 130 | 30 | 90 | 10 | **220** | 0 | 0 | 130 | 256 | 30 | 90 | 10 | **220** | 90 | 30 | 10 | 224 |
| 140 | 0 | 120 | 20 | **237** | 0 | 140 | 0 | 257 | 0 | 120 | 20 | **237** | 0 | 120 | 20 | **237** |
| 150 | 0 | 150 | 0 | **257** | 0 | 150 | 0 | **257** | 0 | 150 | 0 | **257** | 0 | 150 | 0 | **257** |
| 200 | 20 | 180 | 0 | **333** | 0 | 200 | 0 | 340 | 20 | 180 | 0 | **333** | 20 | 180 | 0 | **333** |

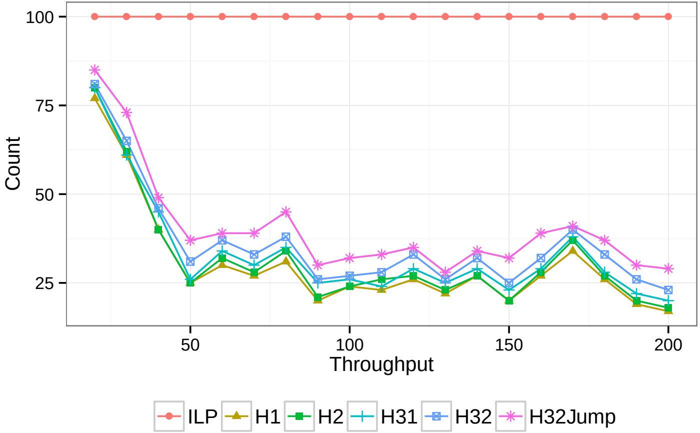
# Experiments: small application graphs




ILP: Gurobi
Simulator: Python

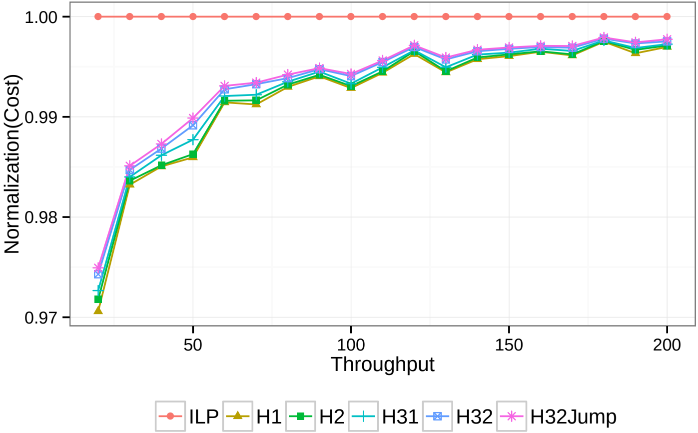
Normalization of cost with the optimal solution
20 alternative graphs, between 5 and 8 tasks for each graph

# Experiments: small application graphs



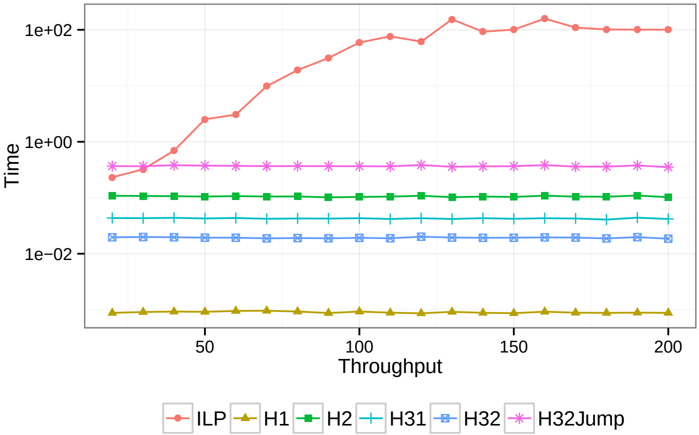

Number of times where each algorithm finds the best
20 alternative graphs, between 5 and 8 tasks for each graph

# Experiments: large application graphs




Normalization of cost with the optimal solution
20 alternative graphs, between 50 and 100 tasks for each graph

# Experiments: large application graphs




Computation time for the heuristics
20 alternative graphs, between 100 and 200 tasks for each graph

# Summary

- Efficient ILP solver:
  - Optimal solutions for small and medium sized problems
  - Fails for applications with more than 100 tasks

- The naive heuristic H1 gives a good solution with minimal overhead
- More sophisticated heuristics only improve H1 up to 5%
- H1 approach gives solutions whose costs are asymptotically close to the optimal (ILP if possible)

# Conclusion

Find the cheapest configuration to reach the target throughput for a given DAG based streaming application

- The issue was to find a suitable distribution between DAGs

⇒ We deduce the platform to rent on the Cloud (minimize the rental cost)

# Conclusion

Find the cheapest configuration to reach the target throughput for a given DAG based streaming application

- The issue was to find a suitable distribution between DAGs

$\Rightarrow$ We deduce the platform to rent on the Cloud (minimize the rental cost)

- In some cases we exhibit algorithms to optimally solve the problem (even if NP-complete in the weak sens)
- The complexity of the most general case remains open

$\Rightarrow$ ILP gives a characterization of an optimal solution

# Conclusion

Find the cheapest configuration to reach the target throughput for a given DAG based streaming application

- The issue was to find a suitable distribution between DAGs

$\Rightarrow$ We deduce the platform to rent on the Cloud (minimize the rental cost)

- In some cases we exhibit algorithms to optimally solve the problem (even if NP-complete in the weak sens)
- The complexity of the most general case remains open

$\Rightarrow$ ILP gives a characterization of an optimal solution

- Heuristics with good performance (6% from the optimal and asymptotically optimal)

# Perspectives

economical cost $\Longleftrightarrow$ energy cost

**Green computing**

- How to take energy into account when we rent resources in the Cloud ?
- How to associate both economical and energetical criteria