Dynamic Partitioned Scheduling of Real-Time **DAG Tasks** on **ARM big.LITTLE** Architectures

A. Mascitti (1), T. Cucinotta

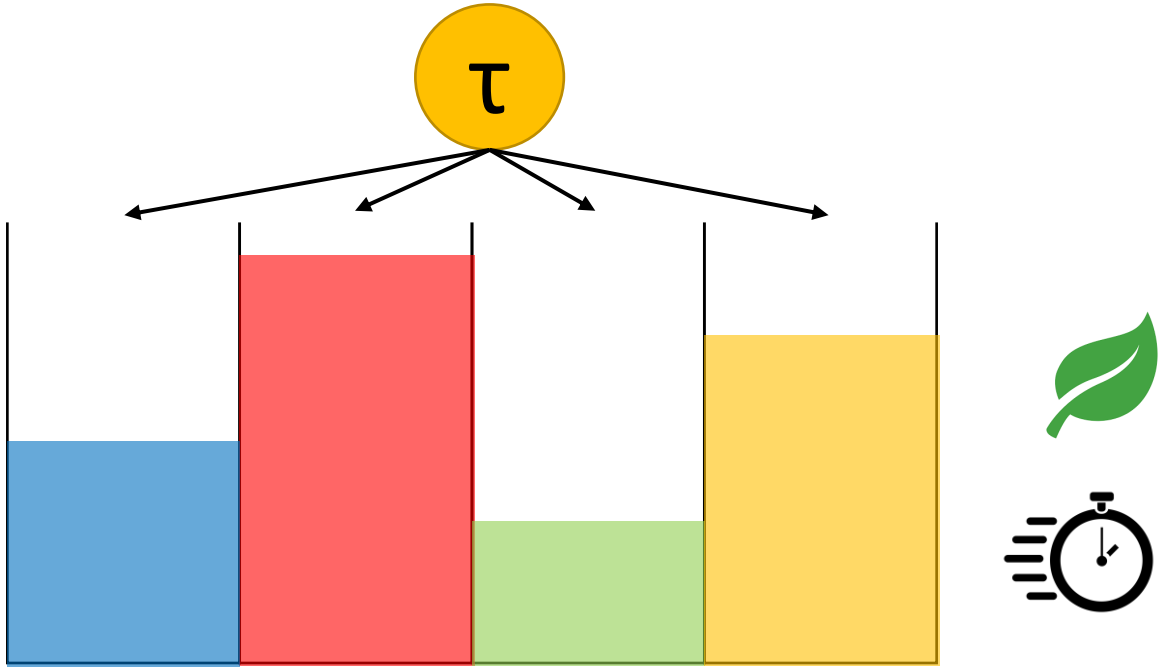
(1) PhD student Scuola Superiore Sant'Anna in Pisa

Sant'Anna

Scuola Universitaria Superiore Pisa

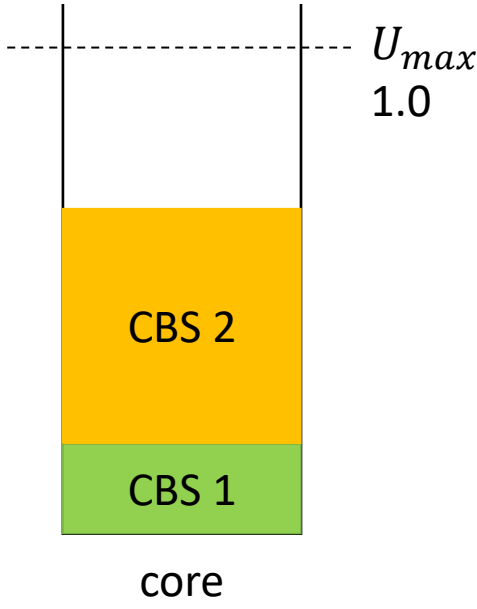
Dynamic Partitioned Scheduling

- More and more **multi-processor** devices
  - **Android** on billion of devices
- More and more **interactive application** → **real-time**



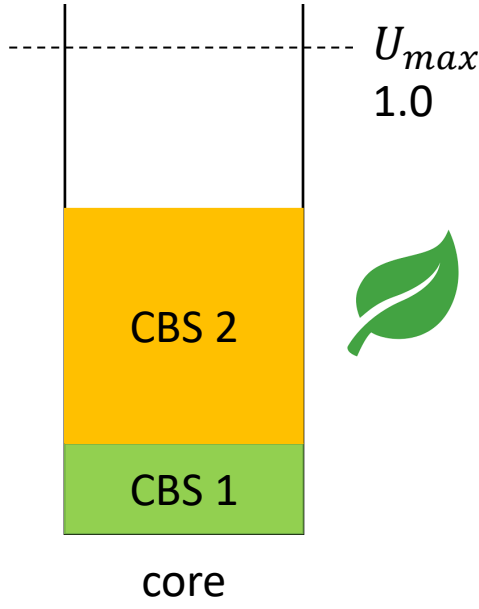
Background

- CBS servers
  - We make use of *resource reservations* to enforce *temporal isolation* among tasks
  - A CBS reservation σ_i is associated parameters (Q, P), where Q is the *budget* and P the *reservation period*
  - While a reservation is scheduled, the budget is decreased accordingly
  - If tasks in the server try to execute for more than Q time units, the server deadline is postponed by P
  - Core schedulability condition: $\sum_i U_i = \sum_i \frac{Q_i}{P_i} \leq 1$



Background

- **GRUB-PA**
  - *Greed Reclamation of Unused Bandwidth – Power Aware*
  - It is an **energy-aware** variant of CBS server
    - Implements unused bandwidth reclamation
    - Exploits DVFS capabilities
  - Implemented in mainline Linux running SCHED_DEADLINE CBS reservations from version 3.14 (Sept 2017)



State of Art

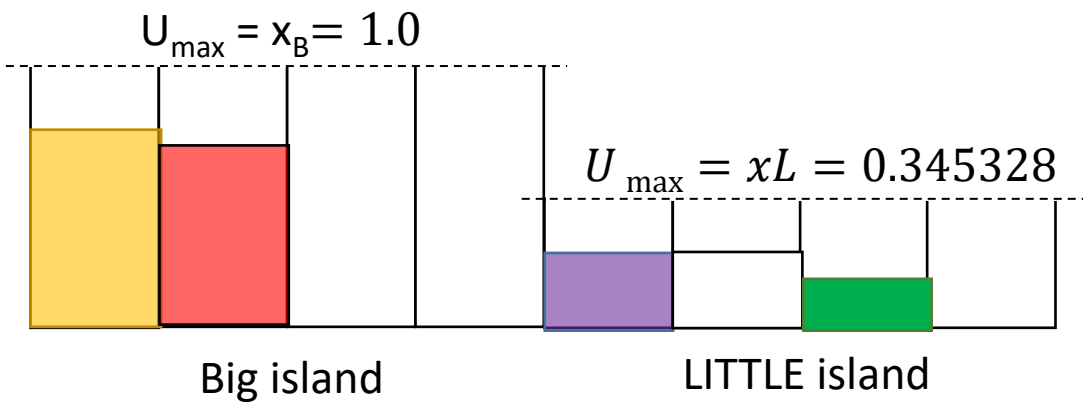
- *Energy-aware* scheduling of sequential tasks
  - Linear Programming-based Methods
  - Others consider heuristics and DVFS capabilities
  - Also explored in a **previous work of ours** (deeply explained later)
- *Thermal-aware* scheduling of sequential tasks
  - ILP methods
  - Minimizing peak temperature
  - Pattern-based approaches (go idle often to reduce temperature)
  - Feedback-loop-based approaches

State of Art

- Non-energy-aware DAG scheduling
  - Some works only consider analysis and schedulability issues
  - Partitioning techniques based on Semi-/Federated scheduling
  - Gang scheduling
- Energy-aware DAG scheduling
  - Not much in the literature
  - We base our solution on Guo et al. (deeply explained later)
  - Based on the concept of Speed-Profiles (we don't use)
  - Introduces the **Task Decomposition technique** that we use

Notation - CPU

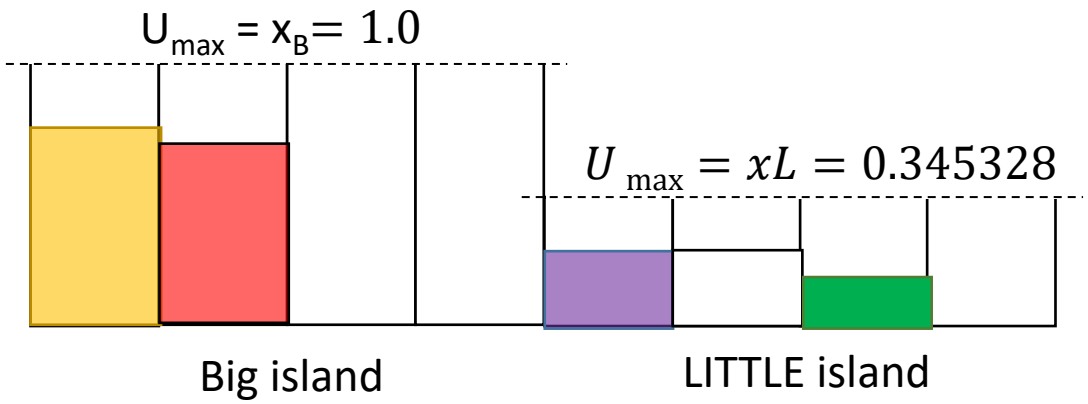
- ARM big.LITTLE platform



- Power model already implemented in RTSim as in [*]
  - Tries to be a good compromise between representativeness and complexity
  - Power consumption depends on voltage (quadratic) and frequency, and task workload type
  - Tuned on ODROID XU3
  - Does not consider interference due to other tasks, cache and memory

Notation - CPU

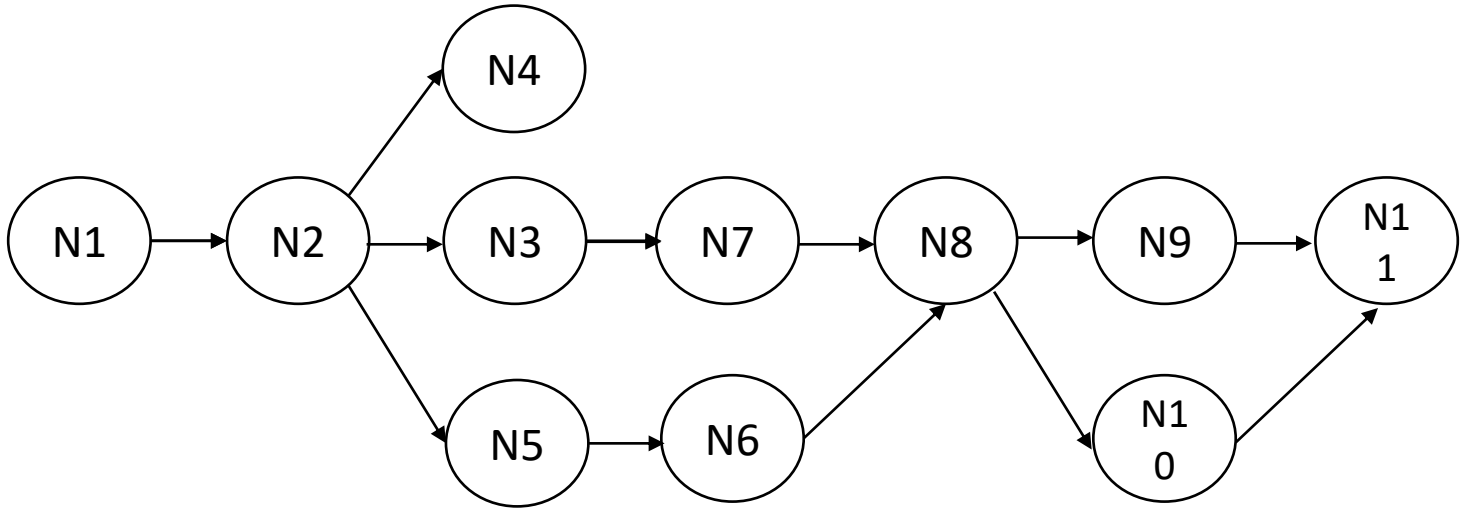
- ARM big.LITTLE platform



- Power model
  - Tries to be a good **compromise** between representativeness and complexity
  - Power consumption depends on voltage (quadratic) and frequency, and task workload type
  - Tuned on **ODROID XU3**
  - Does not consider interference due to other tasks, cache and memory

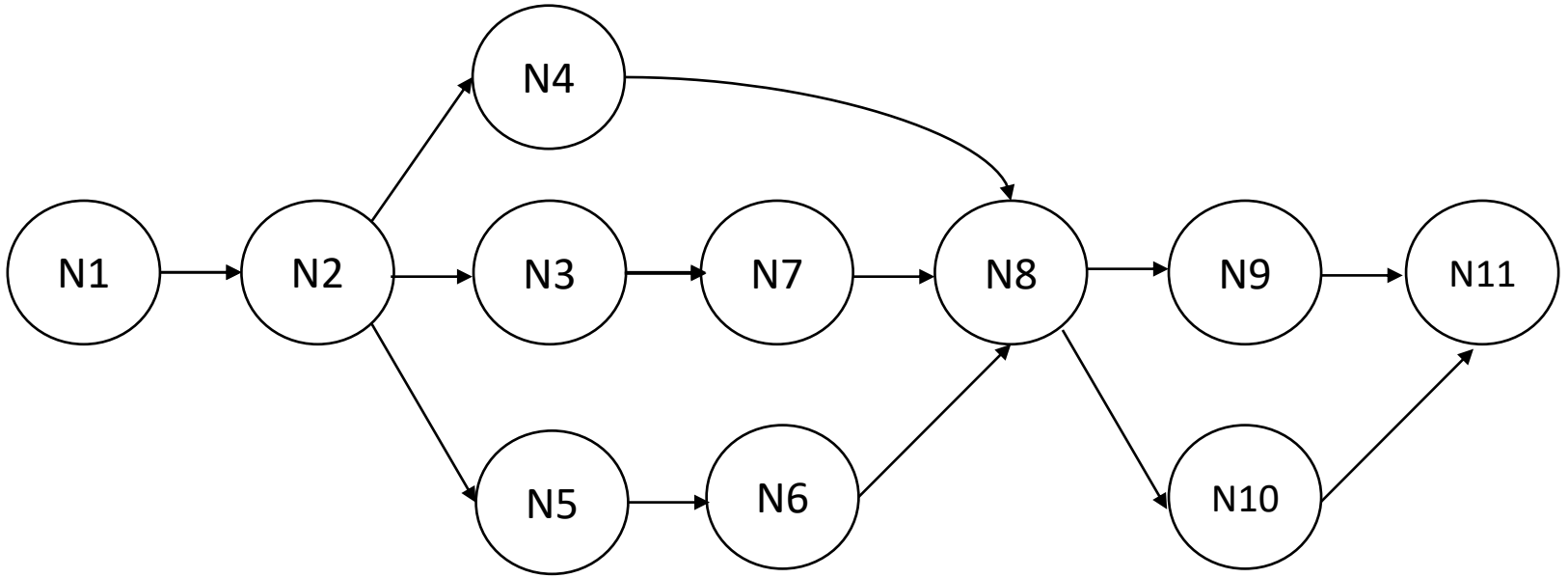
Notation - task

- Set of n **soft** real-time DAG task: $\tau_1 .. \tau_n$
- Implicit deadline (DAG task period = deadline)
- τ_i contains a set of nodes $\tau_i = \left\{ N_i^j \right\}_{j=1}^{n_i}$ with associated nominal WCET C_i^j and period T_i
- **Open and dynamic system** – user opens and closes apps



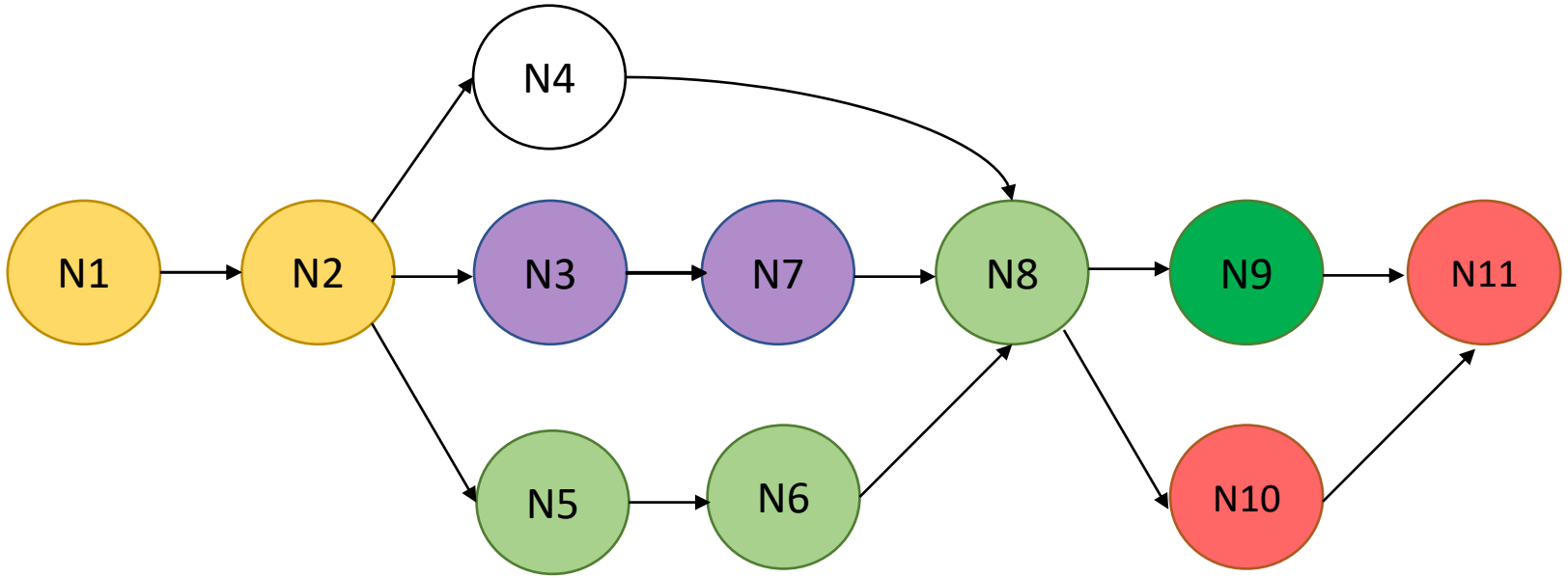
Problem

Given a DAG task τ_i :

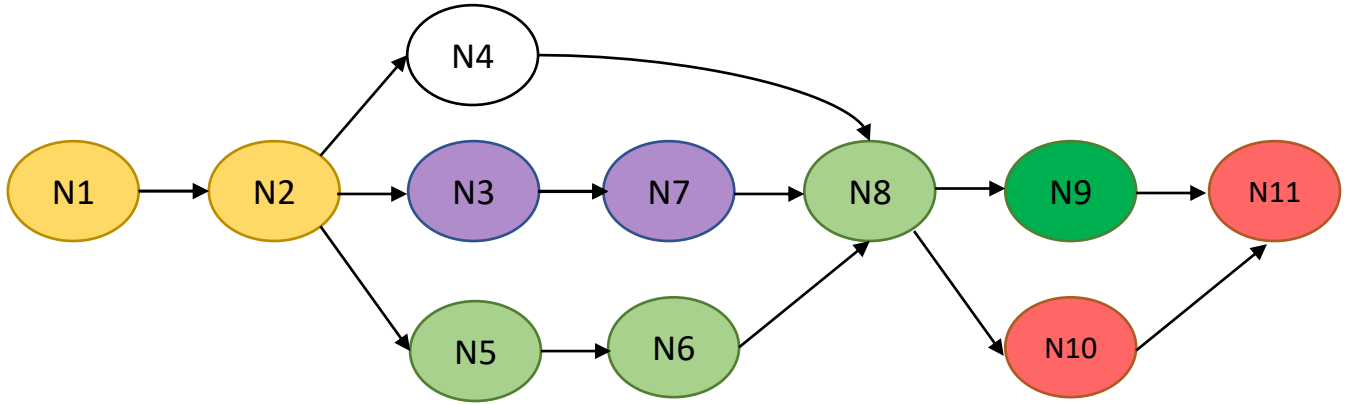


Problem, Novelty

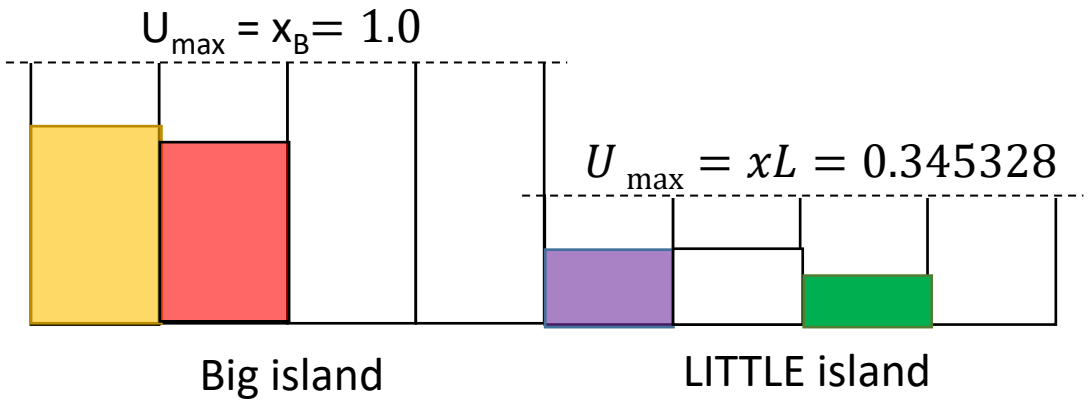
How to split the DAG nodes into groups:



Novelty



and **place** CBS servers
to achieve the **lowest energy consumption**
& guarantee **deadlines**

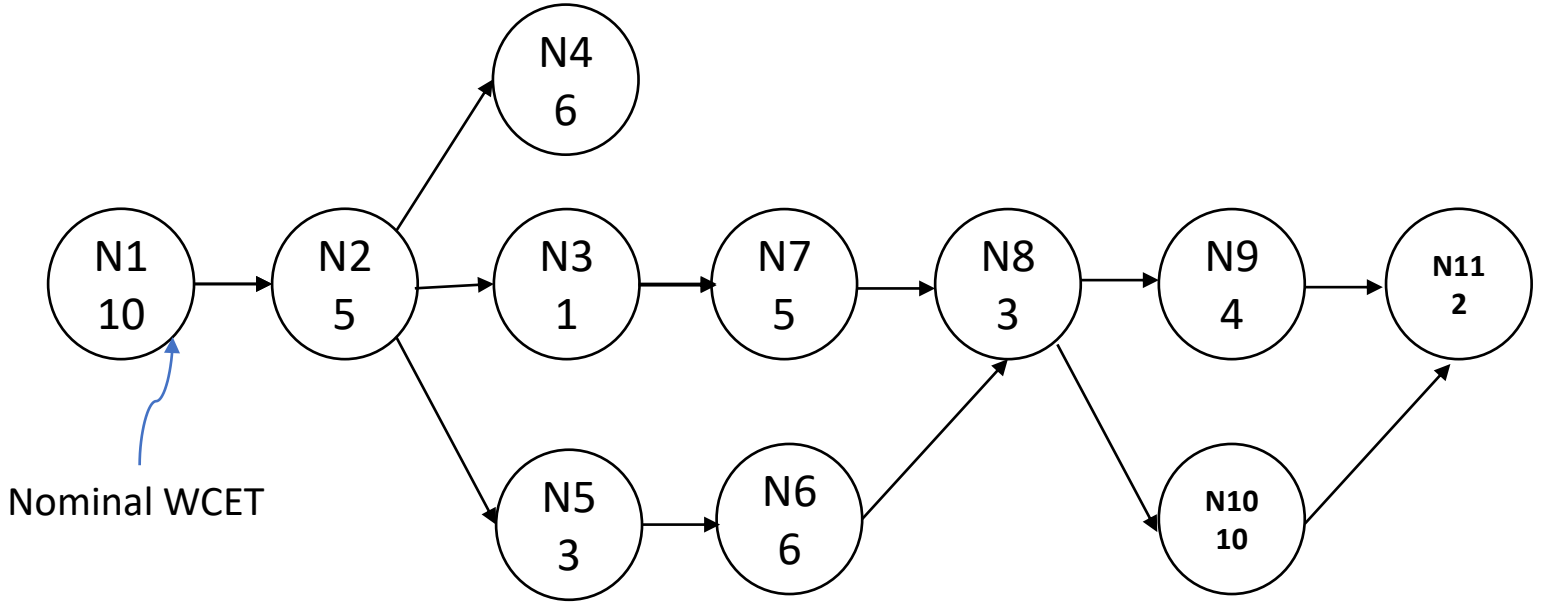


Our solution

Approach

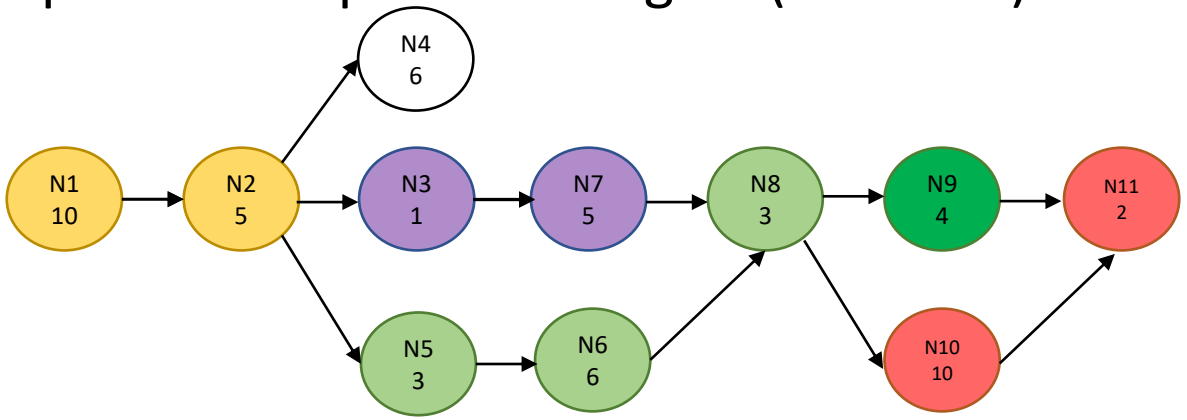
- **Given a DAG task**, we need:
  - A way to split it into **groups of CBS servers**
  - A way to **partition into the CPUs** the CBS servers

  - **Energy**-efficiently
  - Respecting the **soft deadlines**



Approach – DAG to { CBS server }

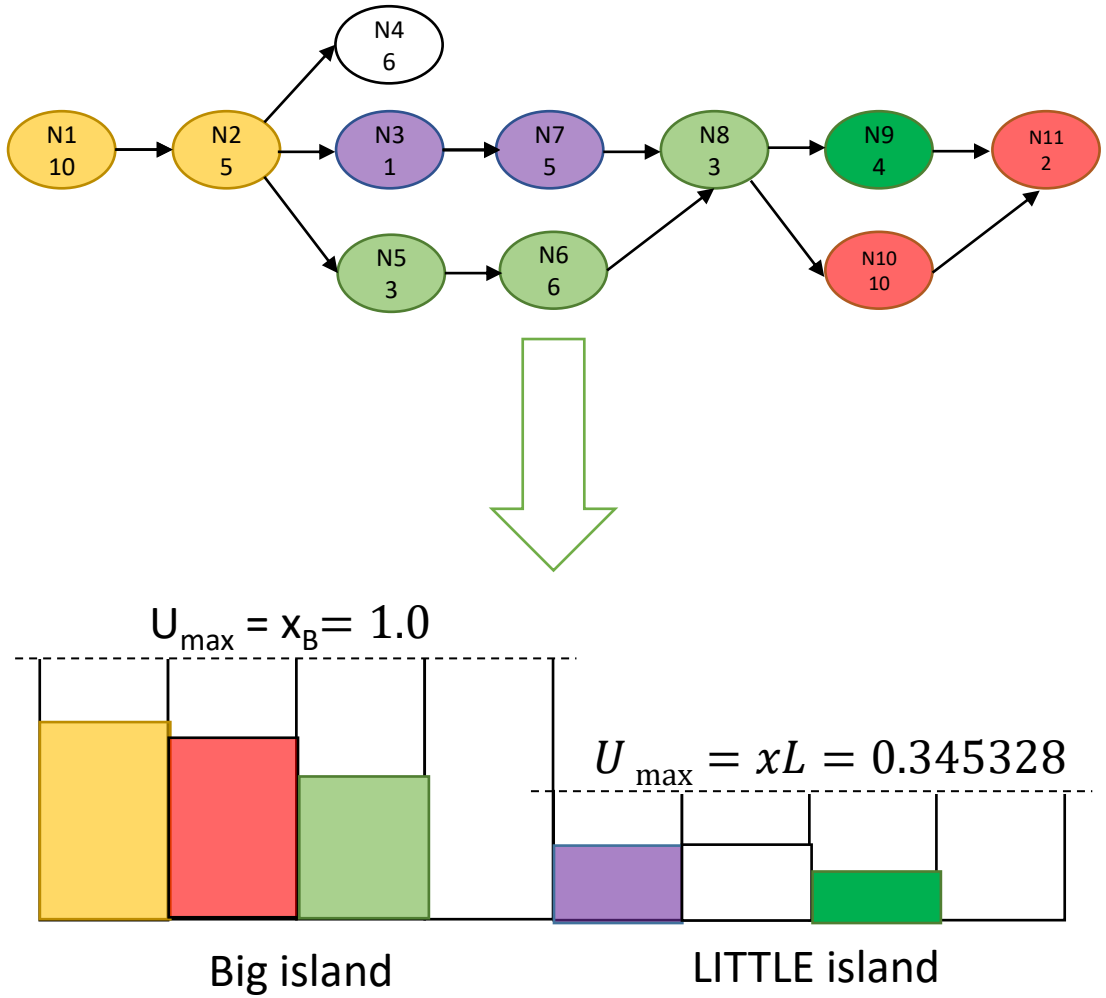
- How to split DAG task τ_1 into groups of CBS servers?
- **DAG Task Decomposition Technique** as in [*]
  - Guo et al. is based on “Speed-Profile”
  - Guo et al. is not about utilizations (we are)
- **Optimized** for our use-case
  - **Decrease utilization of CBS servers**
  - Changes some phases to optimize the goal (see later)



* Guo et al., *Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms*. RTAS 2019

Approach – { CBS server } to CPUs

- How to **place CBS servers onto the CPUs**?
- Dynamic partitioning as in [*]
  - Previous work was about *sequential* tasks
  - Extended to DAG tasks
- **Transparently** assigns CBS parameters
- Decides **core and frequency** for each CBS server
- **Provides the heuristically best core giving min (additional) power consumption**



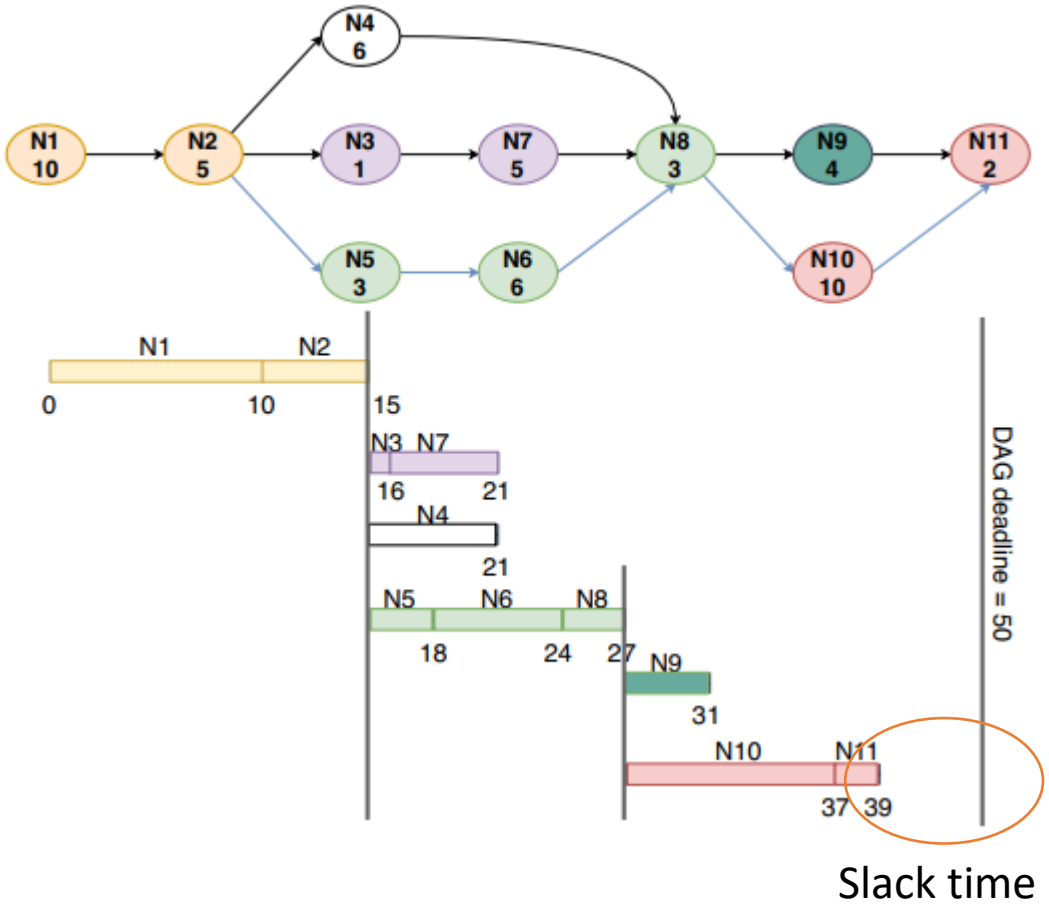
* Mascitti et al, *An adaptive, utilization-based approach to schedule real-time tasks for ARM big. LITTLE architectures*. Ewili 2020

Approach – decrease CBS servers' utilization

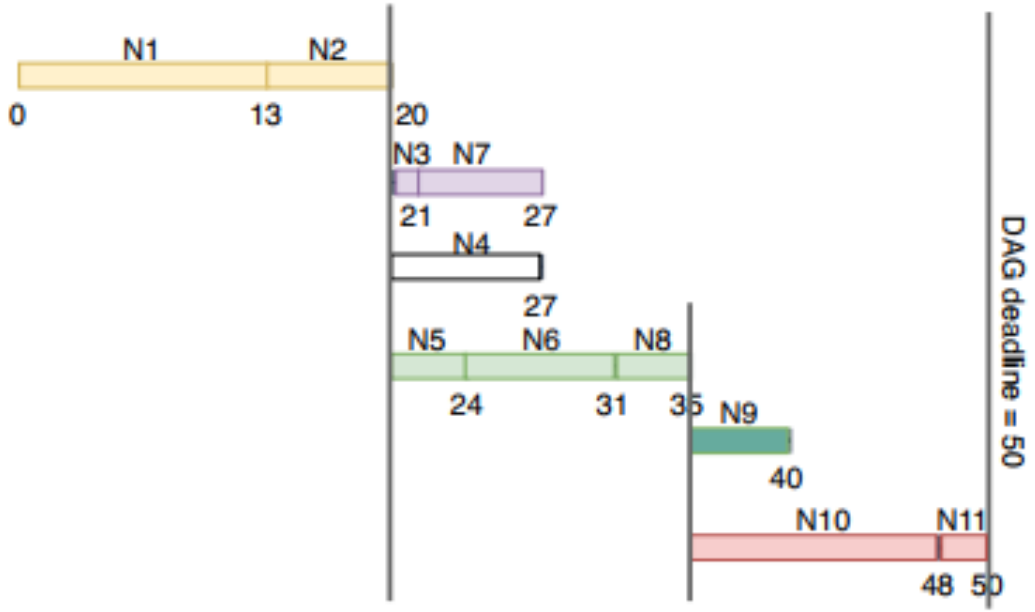
- Smaller CBS utilization -> **smaller core utilization** -> smaller frequency -> **less energy consumption**
- Revisited and optimized version of DAG Decomposition Technique (original by Guo et al.)
- Performed **transparently** at DAG task first arrival
- Many phases:

Phase 1 – DAG Task Decomposition

- In this phase, nodes are divided into groups, which are assigned **initial Arrival and Finishing Time**
  - DAG task has deadline 50
  - E.g., N3 and N7 (violet) belong to one CBS server
  - N3 first job (WCET = 1) begins at t=15 and finishes at t=16
  - N7 first job (WCET = 5) begins at t=16 and finishes at t=21



Phase 2 – Slack allocation



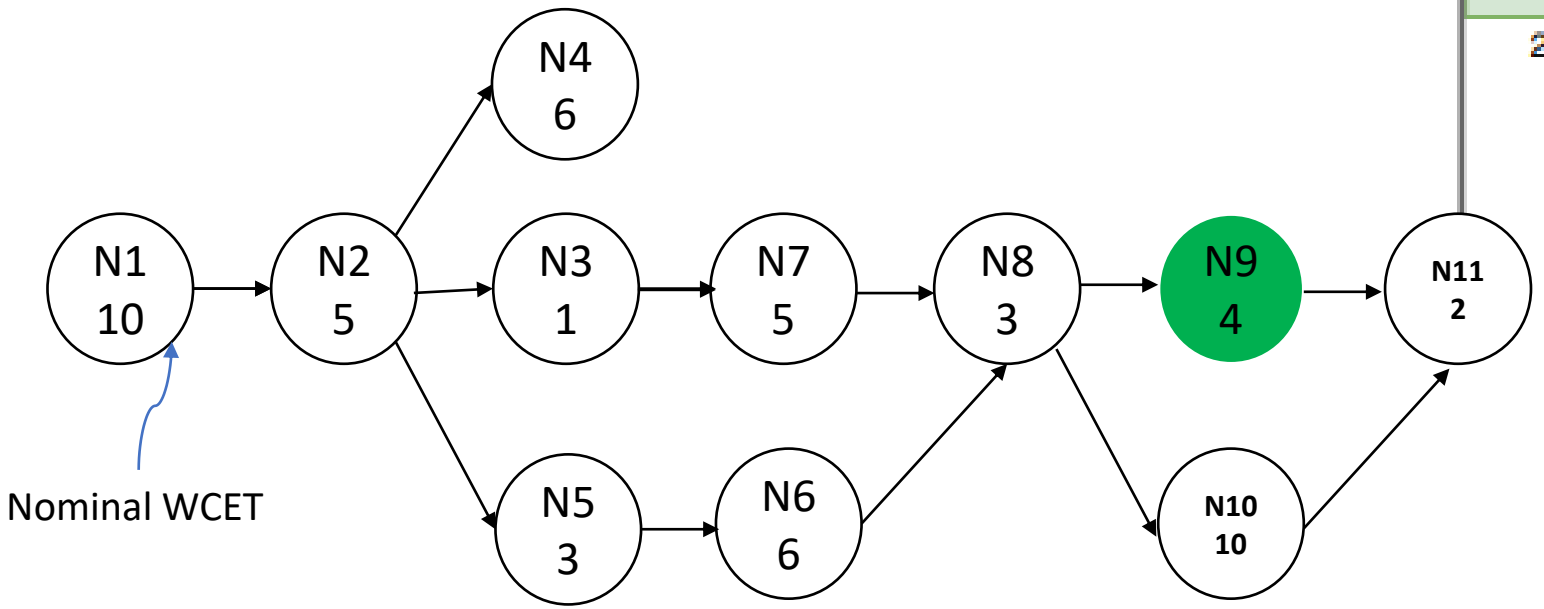
- Slack time is used to relax nodes finishing times and thus **decrease CBS utilizations**
- Allocated uniformly among nodes. E.g. (L_i is DAG critical path length):

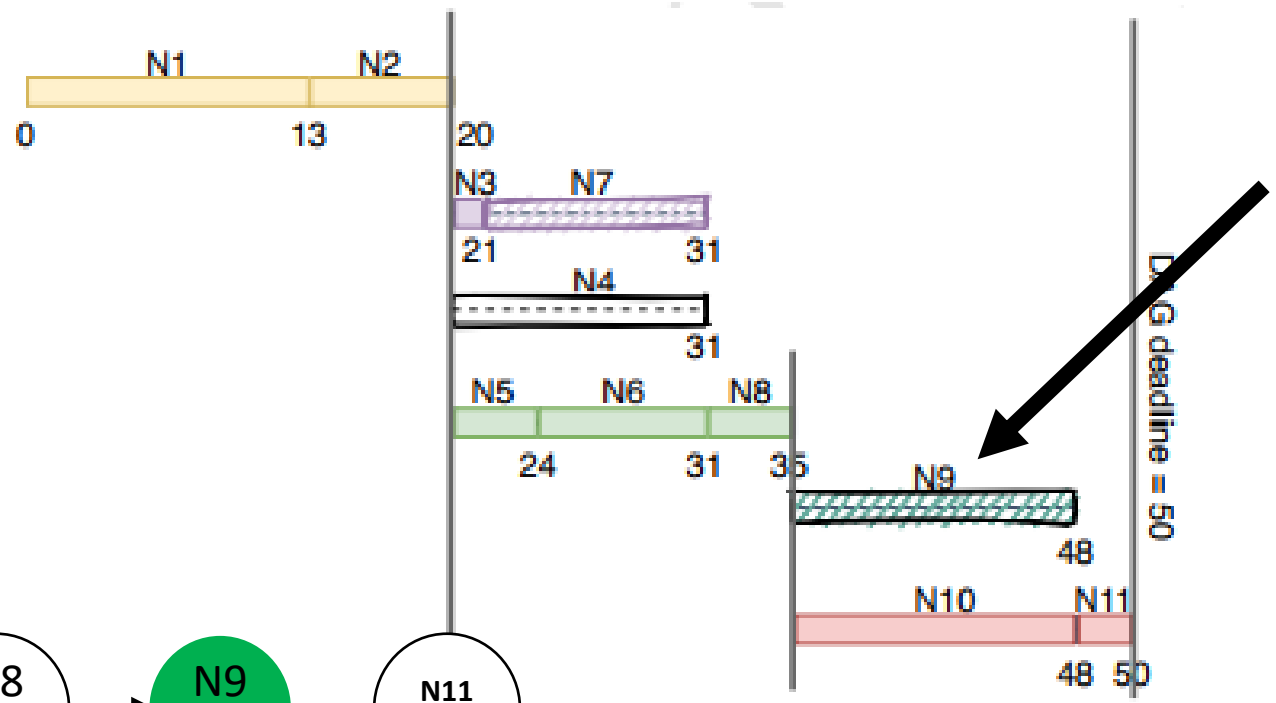
$$\delta \equiv \frac{T_i}{L_i} = \frac{50}{39} = 1.28$$

$$f_i^{N11} = \lceil 39 \times 1.28 \rceil = 50$$
$$f_i^{N10} = \lceil 37 \times 1.28 \rceil = 48$$
$$f_i^{N9} = \lceil 31 \times 1.28 \rceil = 40$$

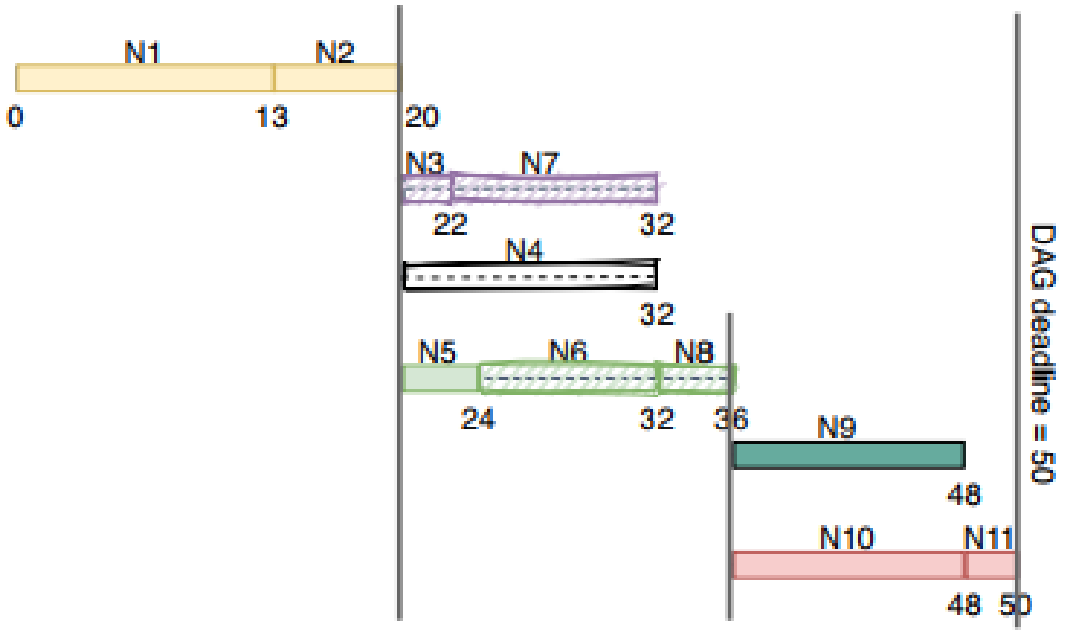
Phase 3 – Segment Extension

- N7 had no reason to finish within the former $f_i^{N9} = 40$
- $\Rightarrow\ f_i^{N9} = a_i^{N11} = 48$
- **CBS utilization is reduced**





Phase 4 – Relaxing finishing times (ours)



- **Further reduces CBS utilization**
- Distribute the time window of each CBS to its nodes, proportionally to their WCET
- E.g., for CBS S = { N3 = 1, N7 =5 } :

$$f_i^{N3} = \lceil a_i^{N3} + C_i^{N3} \times \mu \rceil = \lceil 20 + 1 \times 1.83 \rceil = 22$$

$$f_i^{N7} = \lceil 22 + 5 \times 1.83 \rceil = 32$$

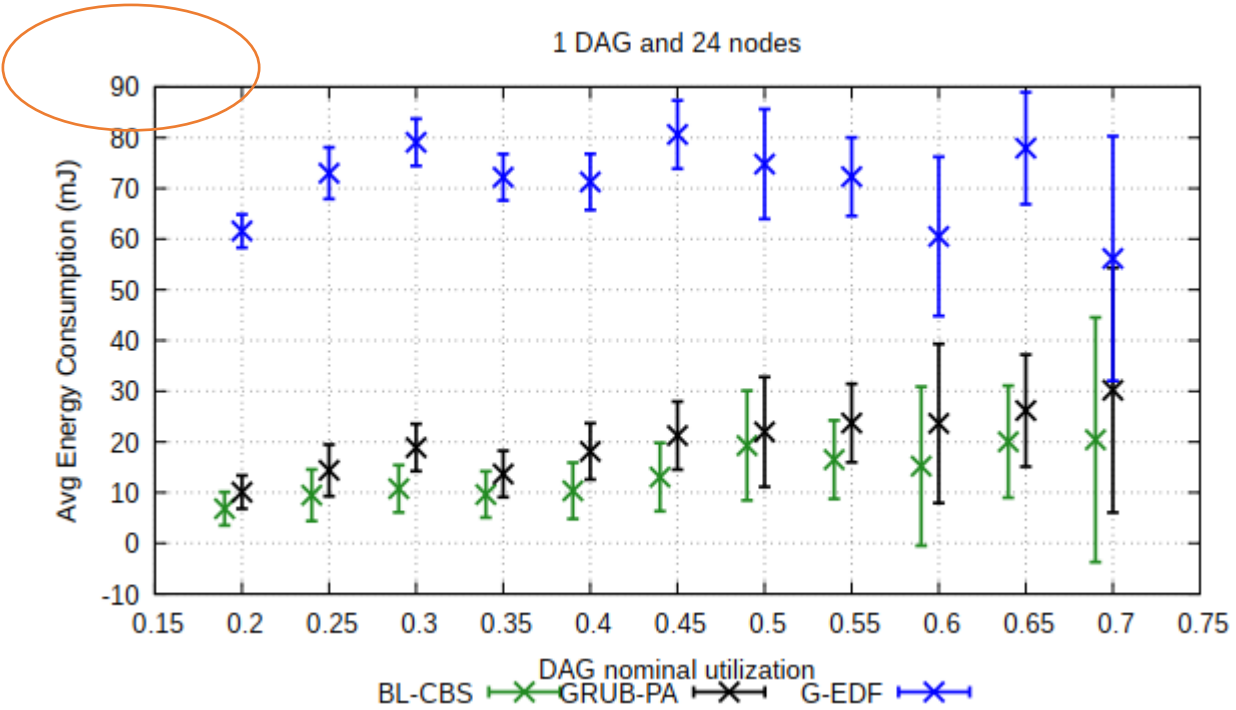
$$\mu \equiv \frac{f_S^{N_S^n} - a_S^{N_S^1}}{\sum_{N_S^j} C_S^j} = \frac{31 - 20}{1 + 5} = 1.83$$

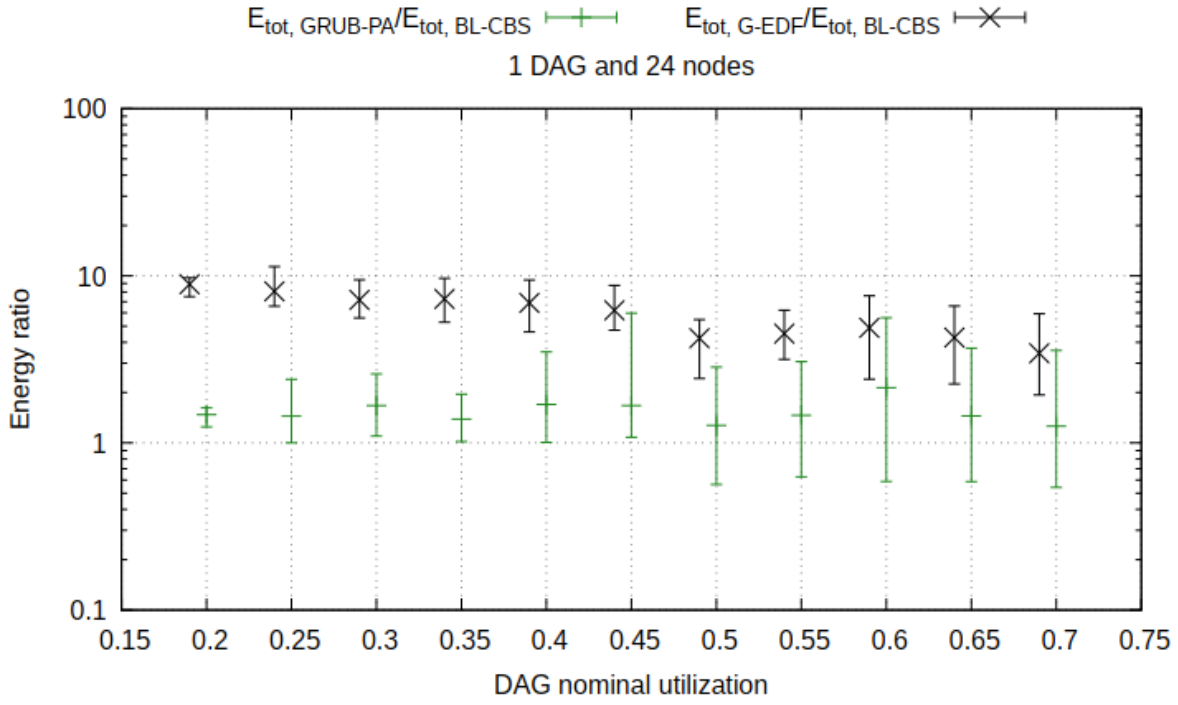
Evaluation

- Compare BL-CBS (i.e., this technique) with **GRUB-PA**
- **Random** tasksets of {1,2,3} DAG tasks and {24,12,8} nodes respectively ; Different DAG utilizations { 0.2, 0.25, …, 0.7 }
- **Execution time** of node instances is set to be uniformly distributed between 0.1 ms and the node nominal WCET => **more dynamic environment**
- For a given CBS server $\sigma_{i,k}$ containing a set of nodes $\{ N_i^j \}$:

$$\begin{cases} Q_{i,k} = \sum_{N_i^j \in \sigma_{i,k}} C_i^j \\ P_{i,k} = \sum_{N_i^j \in \sigma_{i,k}} (f_i^j - a_i^j) \end{cases}$$

Evaluation (Avg energy)

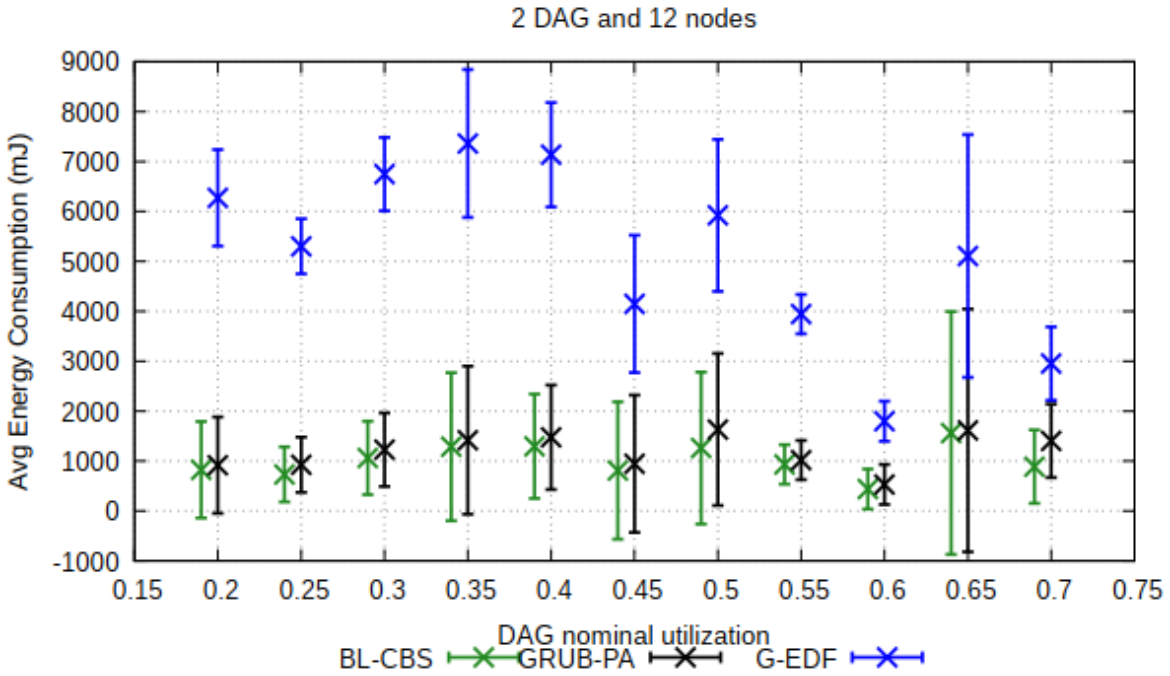


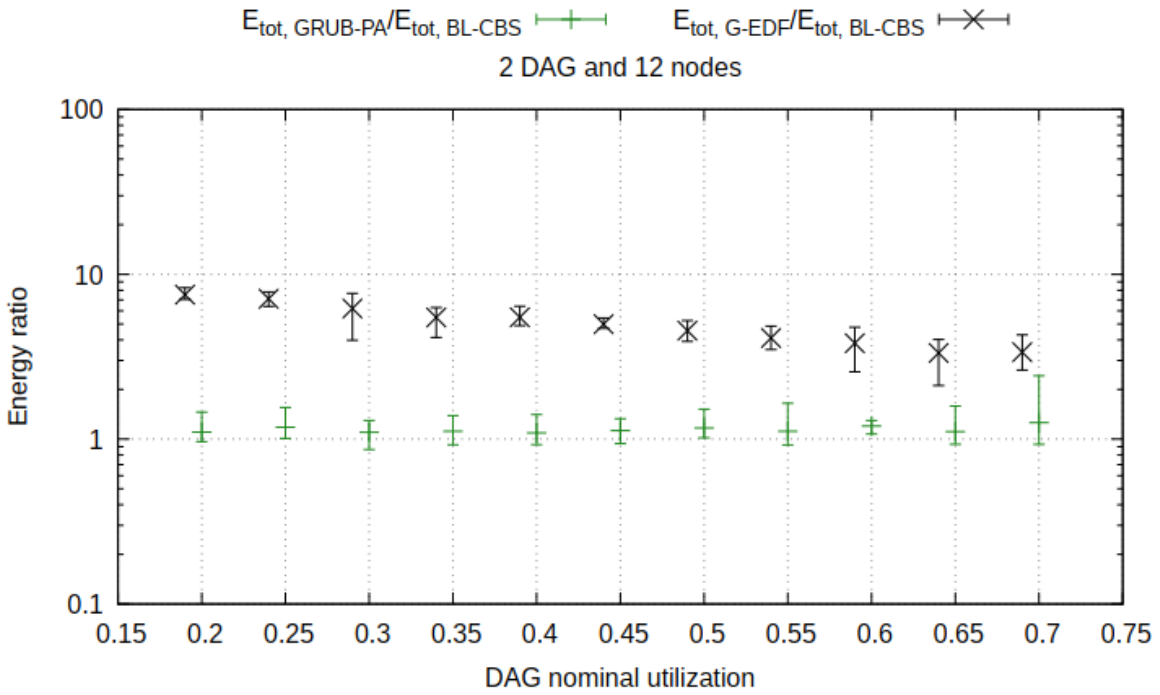


BL-CBS < GRUB-PA

BL-CBS << G-EDF

Evaluation (Avg energy)



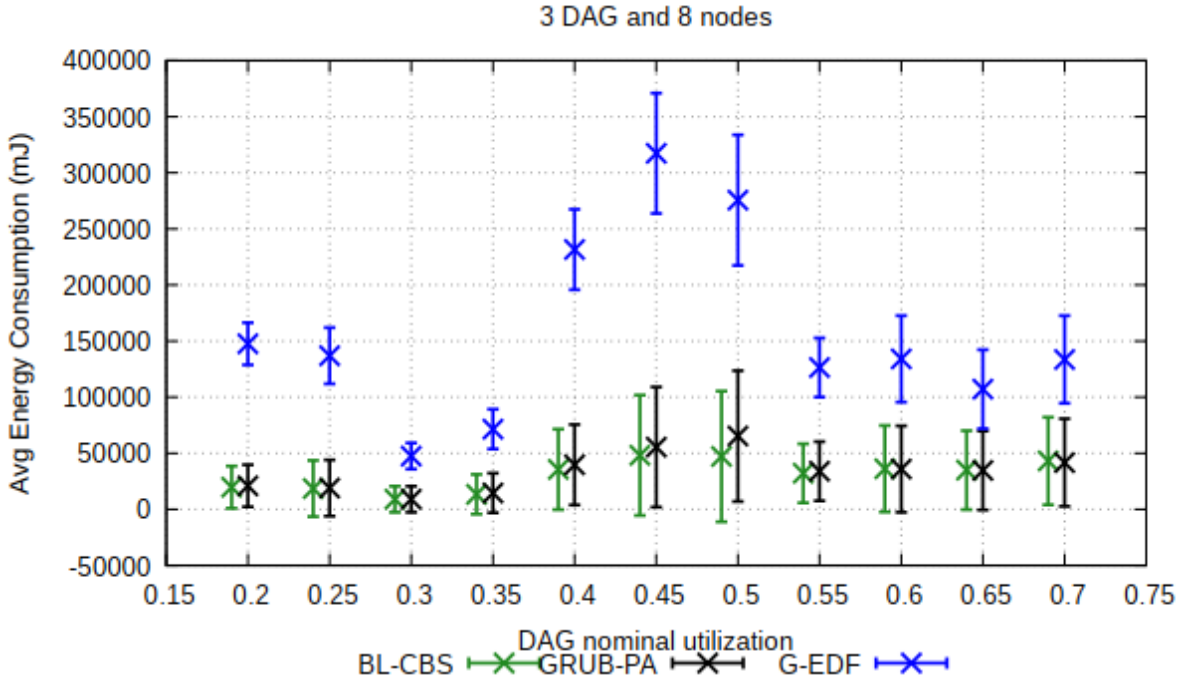


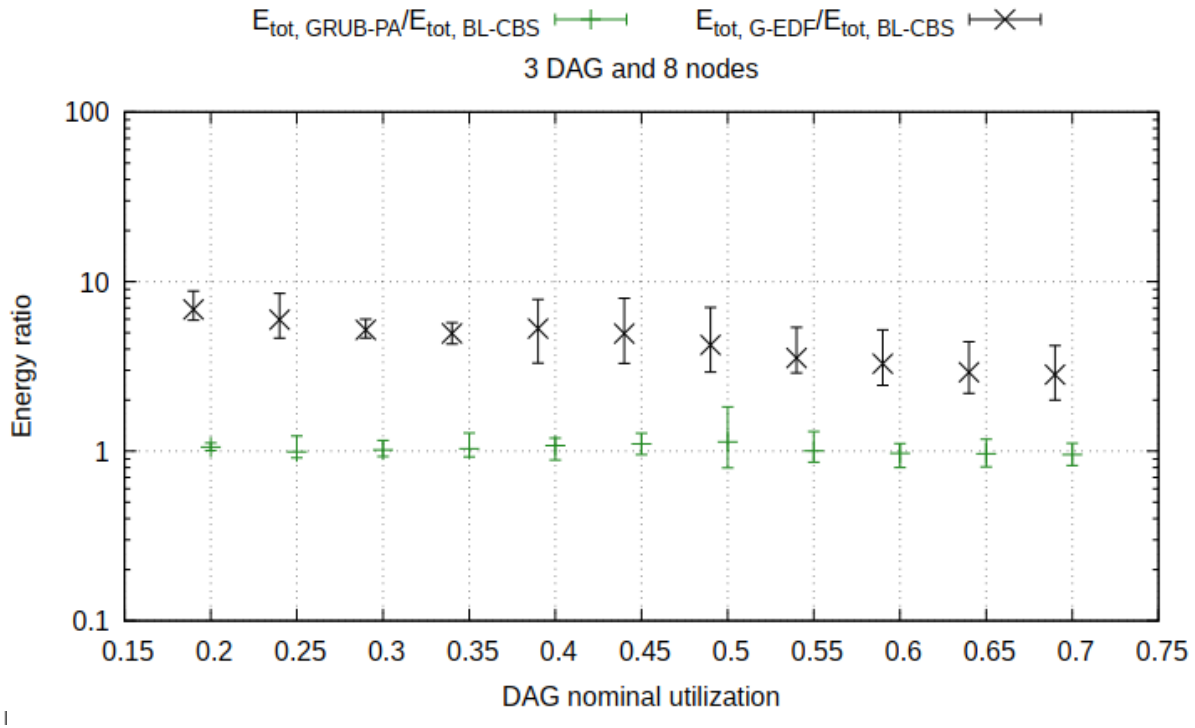
9000 mJ = 90mJ **x 100**

BL-CBS < GRUB-PA

BL-CBS << G-EDF

Evaluation (Avg energy)



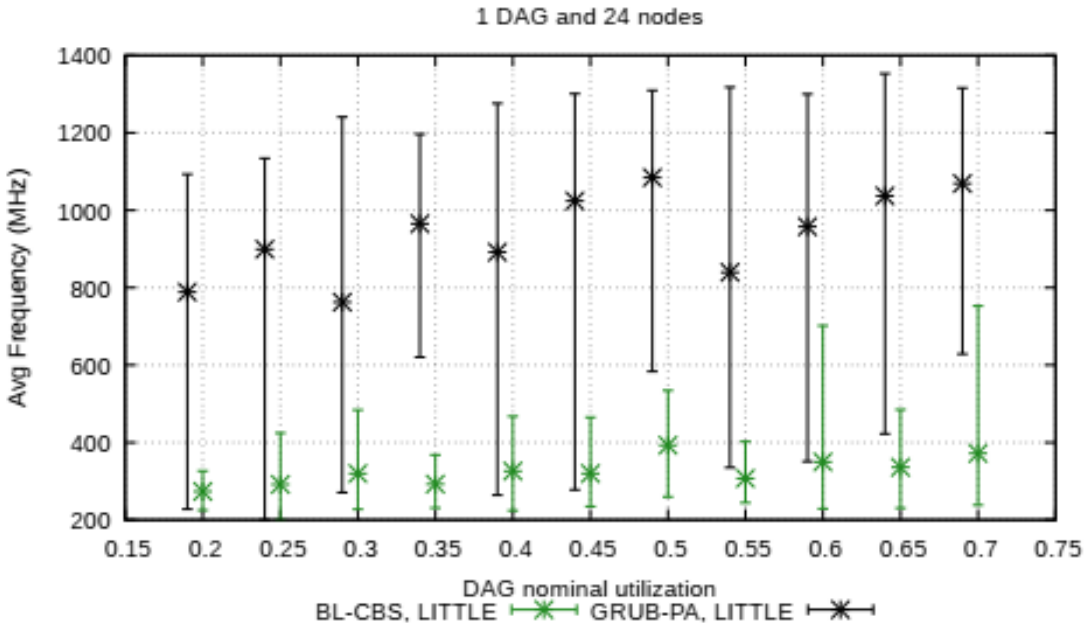


400.000 mJ = 90 mJ **x 4.444**

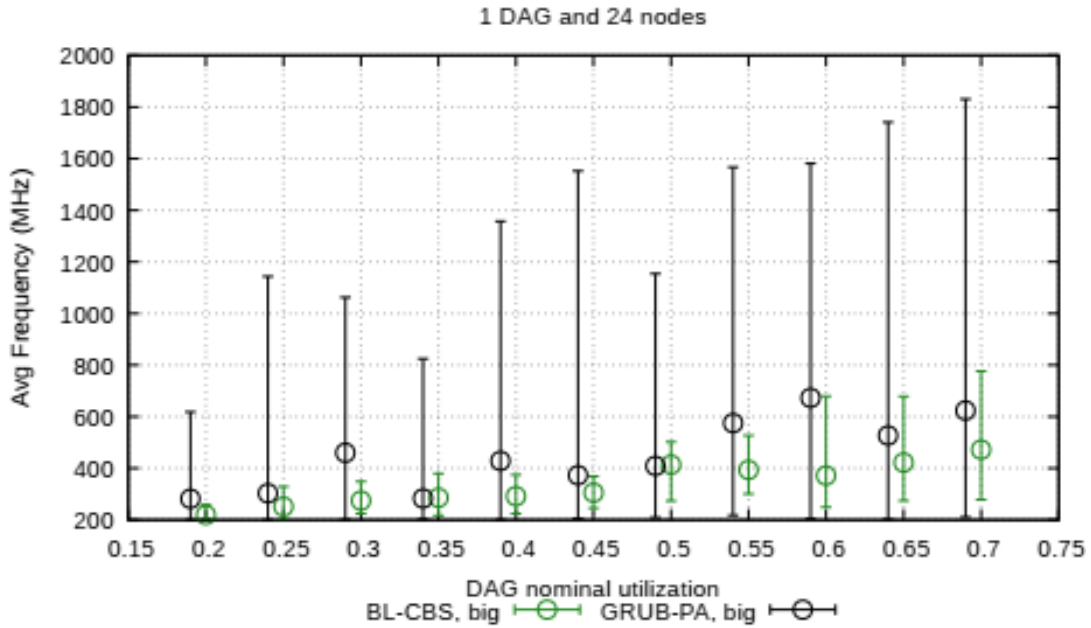
BL-CBS <= GRUB-PA

BL-CBS << G-EDF

Evaluation (avg frequency)



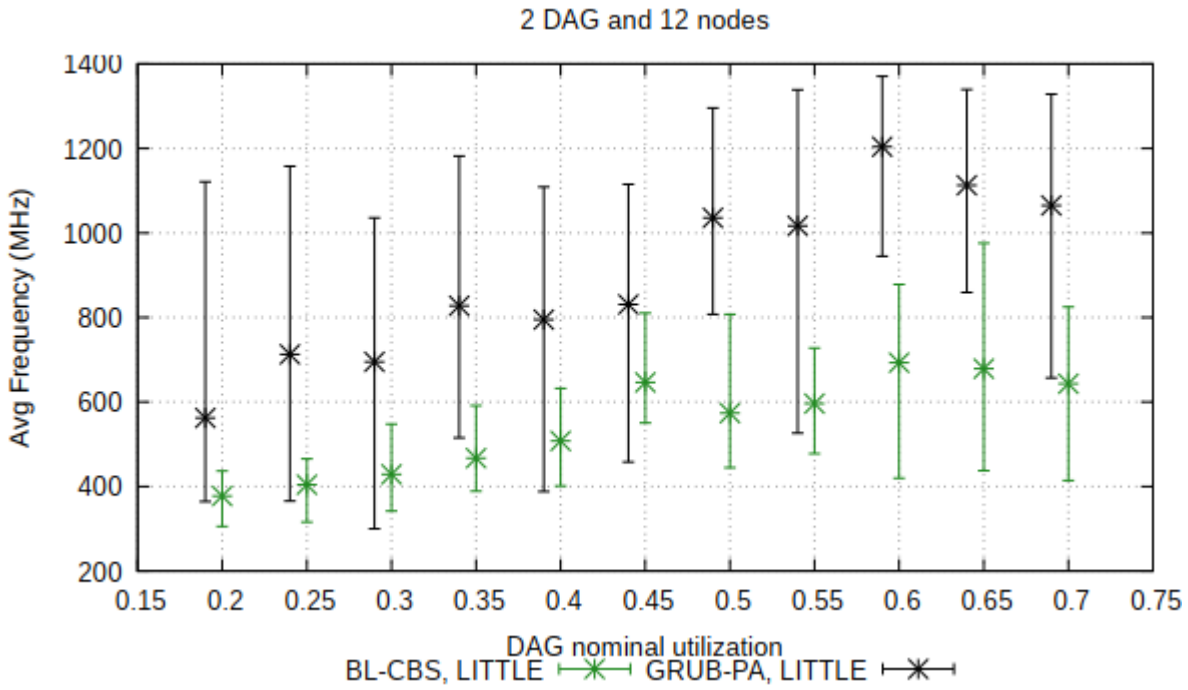
(a) LITTLE island, 1 DAG and 24 nodes

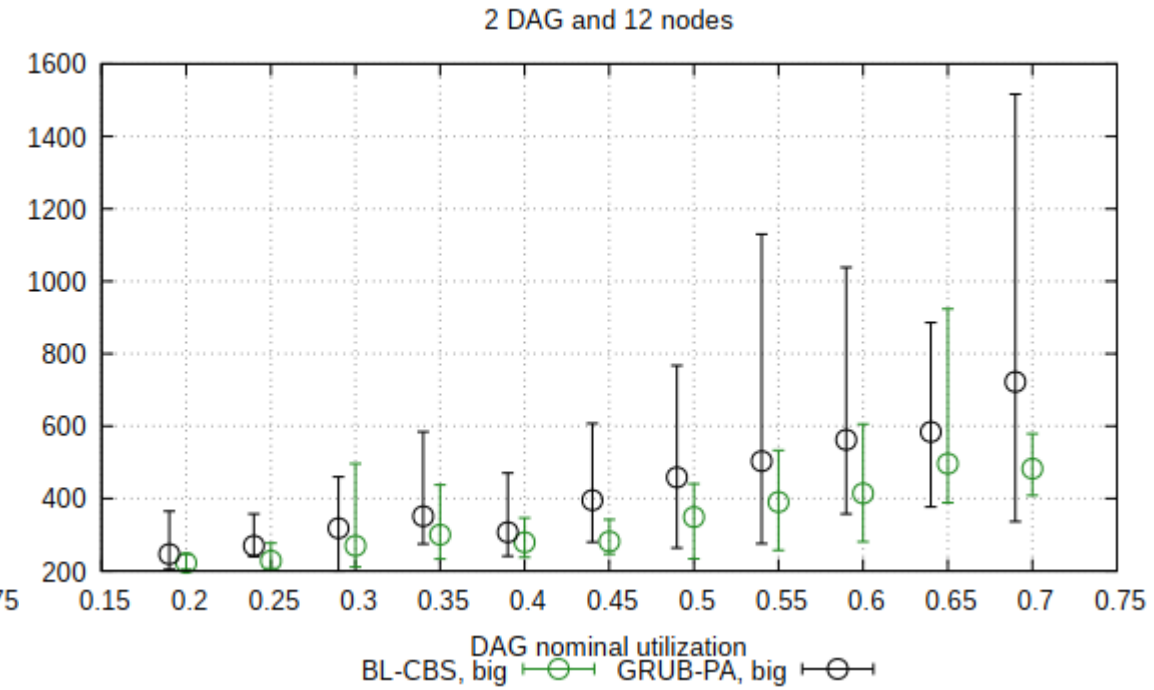


(b) big island, 1 DAG and 24 nodes

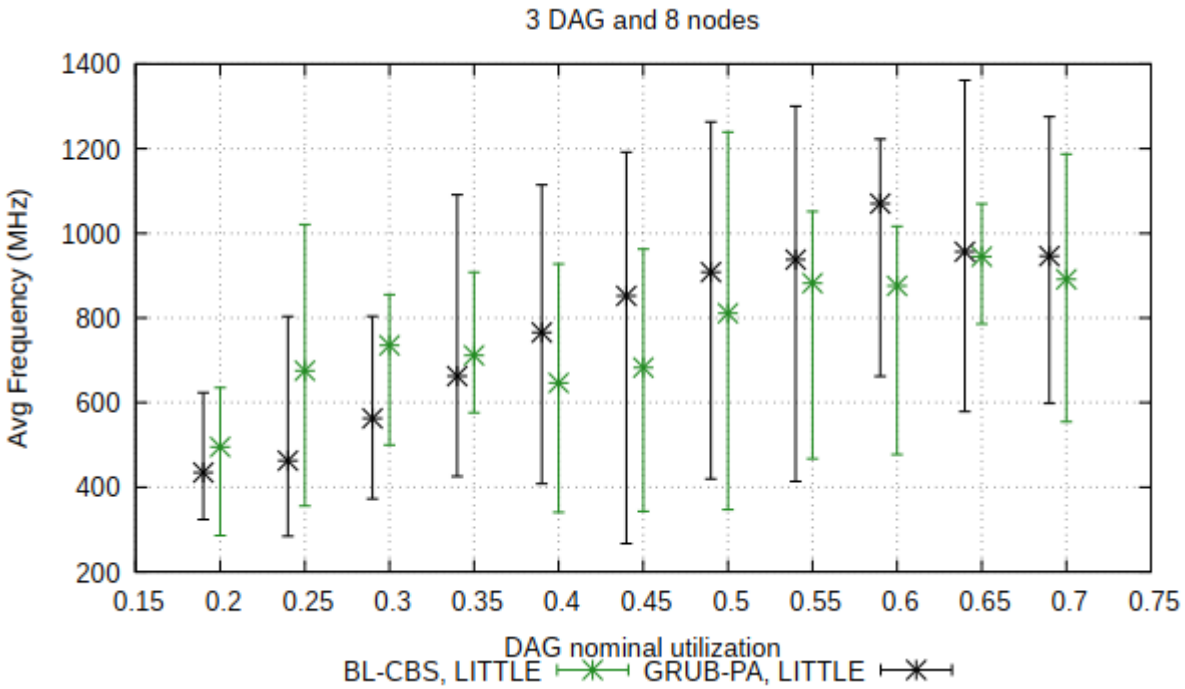
BL-CBS < GRUB-PA

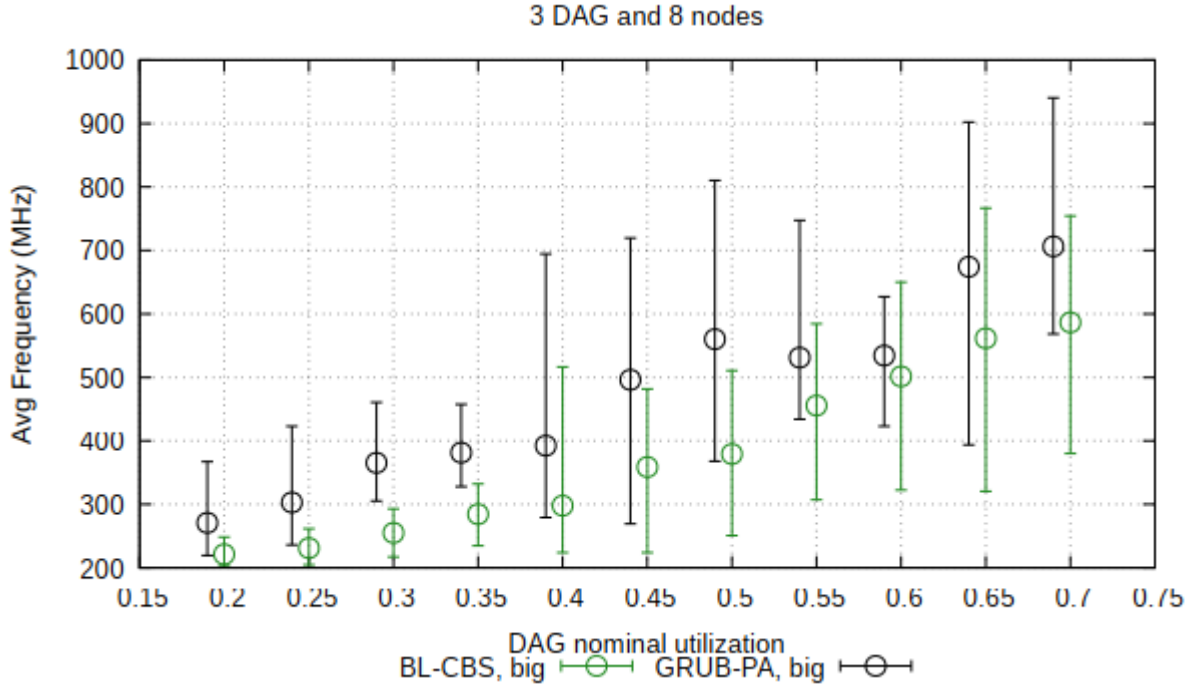
Evaluation (avg frequency)





Evaluation (avg frequency)





LITTLE island has higher frequencies than big island

Conclusions

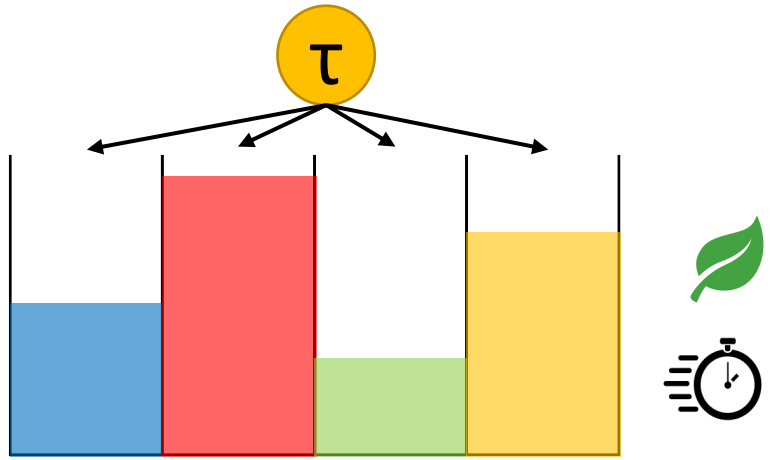
Incorporate **DAG placement and splitting strategies**

Improvement for energy saving

- **BL-CBS for DAG** tasks
- Support to “open” and dynamic system (Android use-cases)
- Made for **ARM big.LITTLE** architecture

Experimental results

- On random tasksets
- **Energy saving in average 10%** over all the experiments with respect to the state-of-the-art GRUB-PA



Future works

- Improve the **performance** of the placement algorithm
- Place the servers on the cores **also considering**
  - The nodes relationships
  - The memory consumption and different workload types
- Consider CPU **deep-idle states**
- Incorporate **Bandwidth reclaiming and feedback** mechanisms

Thank you!

a.mascitti@santannapisa.it

Slide & paper
https://owncloud.retis.santannapisa.it/index.php/s/py1WwfF2aSUjciV