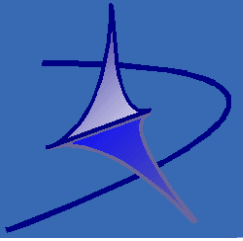


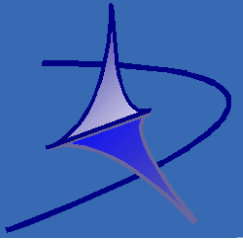
Adaptive Partitioning of Real-Time Tasks on Multiple Processors

Luca Abeni

Tommaso Cucinotta

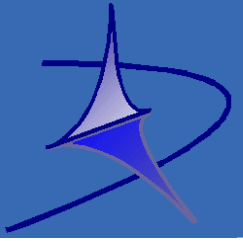
ISTITUTO
DI TECNOLOGIE DELLA
COMUNICAZIONE,
DELL'INFORMAZIONE
E DELLA
PERCEZIONE

Scuola Superiore
Sant'Anna



Real-Time Applications

- Real-Time Applications: characterised by temporal constraints
  - Need for appropriate scheduling algorithms to respect such constraints (deadlines)
- Real-time task: stream of activations (*jobs*) associated to deadlines
  - Here, focus on periodic activations for simplicity
  - The task periodically wakes up (period P), executes for a time (smaller than C) and must finish within a deadline
  - Task utilisation: $U = C/P$

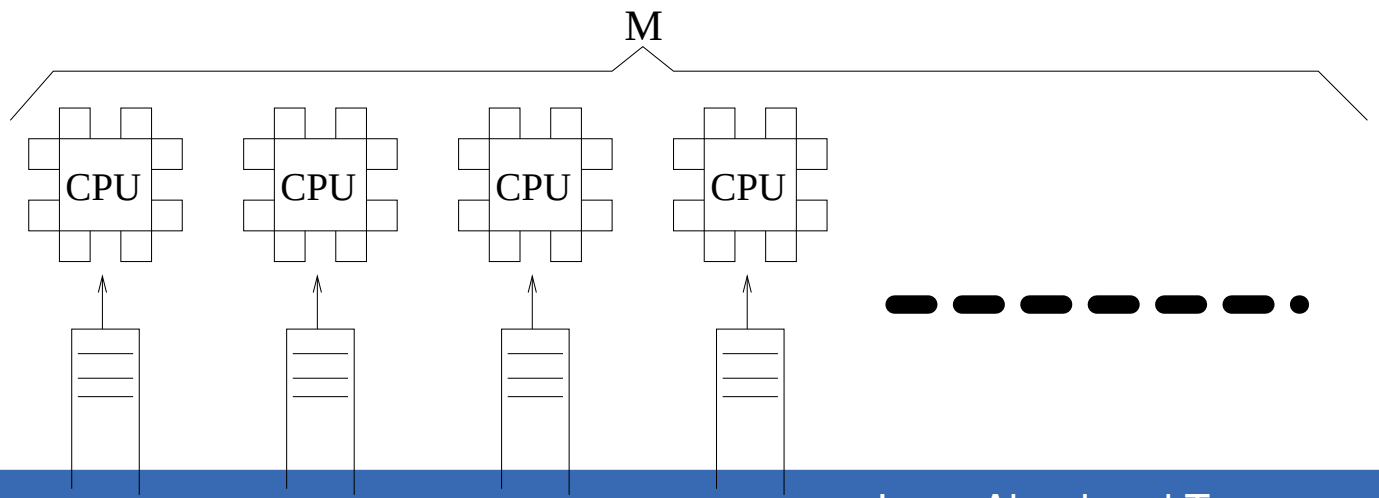


Multiprocessor Real-Time Scheduling

- Uni-processor systems: schedule tasks with fixed priorities or EDF
  - Possible to guarantee that all deadlines are respected, based on the system utilisation
  - System real-time utilisation: sum of the utilisations of all the real-time tasks
- What about multi-cores/multi-processors?
  - Huge amount of literature...
  - ...But still a lot of open issues!
- Generally, 2 different approaches are used:
  - Partitioned scheduling
  - Global scheduling

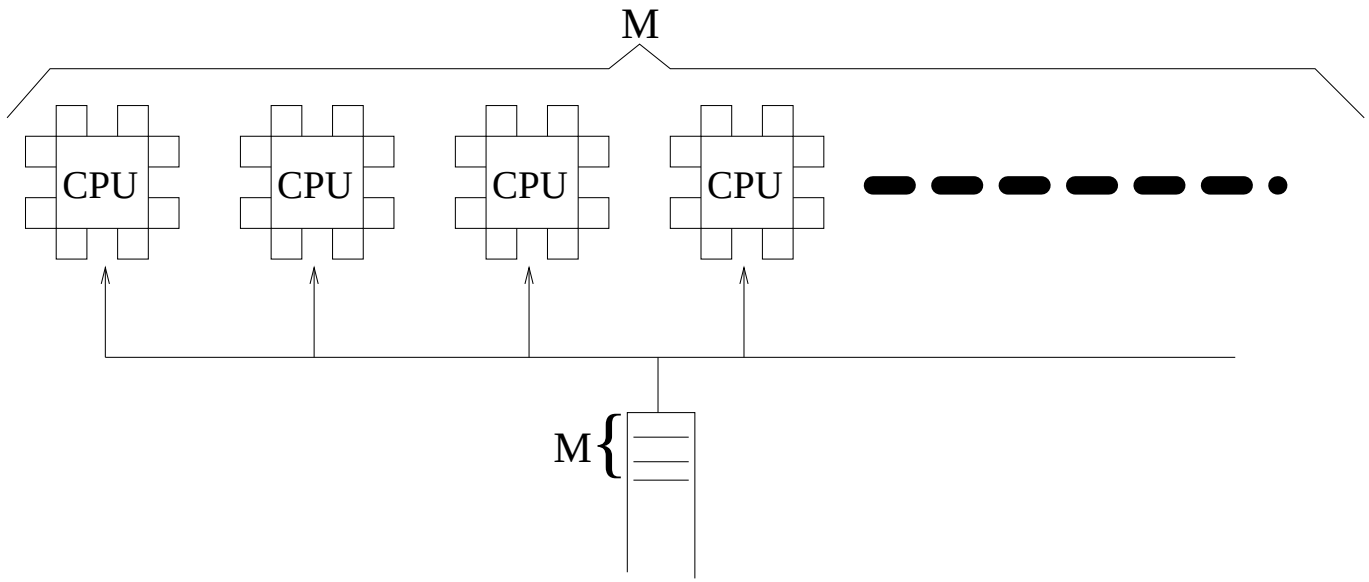
Partitioned Scheduling

- Reduce a multi-core scheduler to multiple single-processor schedulers
  - Statically assign tasks to CPUs
  - Reduce the problem of scheduling on M CPUs to M instances of uniprocessor scheduling
    - Optimal uniprocessor schedulers are easy!!!
  - Problem: system underutilisation



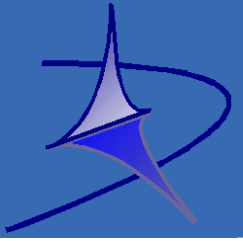
Global Scheduling

- One task queue (conceptually!), shared by M CPUs
  - The first M ready tasks are selected
  - What happens using fixed priorities (or EDF)?
  - Tasks are not bound to specific CPUs
  - Tasks can often migrate between different CPUs
- Problem: low schedulability bound (Dhall's effect)



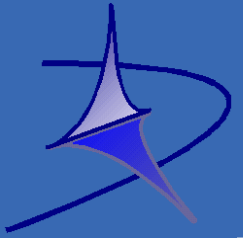
Partitioned vs Global

- Partitioned scheduling: can use EDF (optimal!)
  - Bin-packing to assign tasks to cores so that every core has utilisation ≤ 1
  - Issue: non partitionable tasksets — Think about 2 cores and 3 tasks with execution time 6 and period 10
- Global scheduling: tasksets with utilisation > 1 might be unschedulable!
  - Moreover, migrations are overhead!
- However, global EDF is good for soft tasks...
  - Deadlines can be missed, but by a limited amount of time



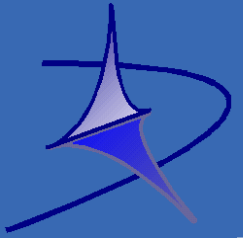
A Possible Trade-Off?

- Is it possible to combine the good properties of partitioned scheduling with the good properties of global scheduling?
  - Obviously, avoiding the drawbacks of the two approaches...
- Goal: if the taskset is partitionable, then we want partitioned scheduling!
  - So that migrations are avoided and deadlines are respected
- Goal: if the taskset is not partitionable, then fallback to global EDF
  - Deadlines missed by a limited amount of time



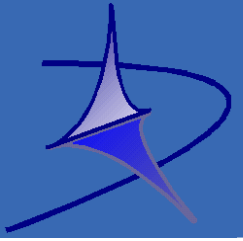
Adaptive Partitioning — Basic Idea

- A task can be migrated only ad the beginning of a new instance $\leftarrow$ restricted migration
- If a task wakes up on a core with $U \leq 1$ (schedulable by EDF), do not migrate it! $\leftarrow$ new idea
- If a task wakes up on an overloaded core ($U > 1$), search for a core where the task is schedulable ($U \leq 1$ after migrating the task) $\leftarrow$ use First Fit
  - If such a core is found, migrate the task
- If the task cannot be migrated to any core without overloading it, select a core based on gEDF
  - Try the core where the task with the largest deadline has been scheduled



The apEDF Algorithm

- Based on the basic adaptive partitioning ideas
  1. Migrations only on wakeup
  2. Migrations only from overloaded cores
  3. When migrating:
     - Use FF to search for a non-overloaded core
     - Fallback to gEDF if all cores are overloaded
- Property based on FF (formal proof exists): if $U < (M+1)/2$, then no deadline is missed!
- Conjectures (only verified through simulations)
  - If schedulable partitioning exists, apEDF converges to it
  - Response times have an upper bound

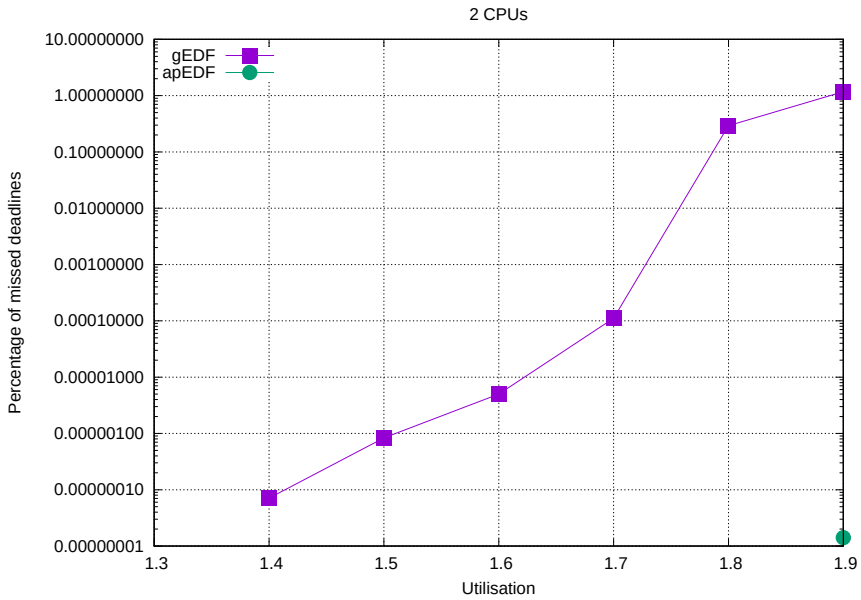


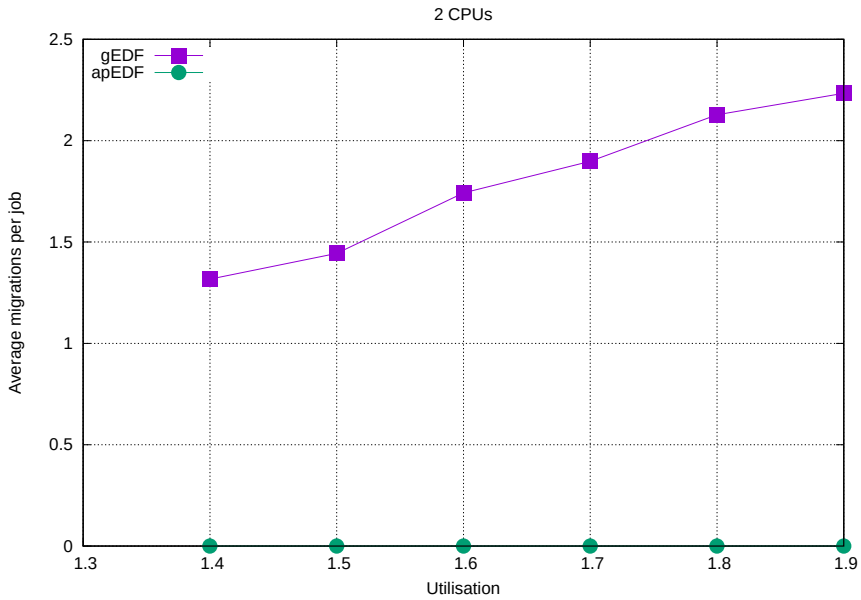
Reducing the Tardiness Bound: a^{2}pEDF

- An upper bound for response times seem to exist...
  - ...But it can be quite large!!!
- Issue: when a schedulable task partitioning cannot be found, migrating tasks only on wakeup can be problematic
  - Non-work-conserving algorithm: a core might be idle when other cores are overloaded!
- Solution: add a “pull” phase to migrate tasks to a core when it becomes idle
  - Migrate tasks to idle cores based on their deadlines (to emulate gEDF)

Soft Real-Time Performance

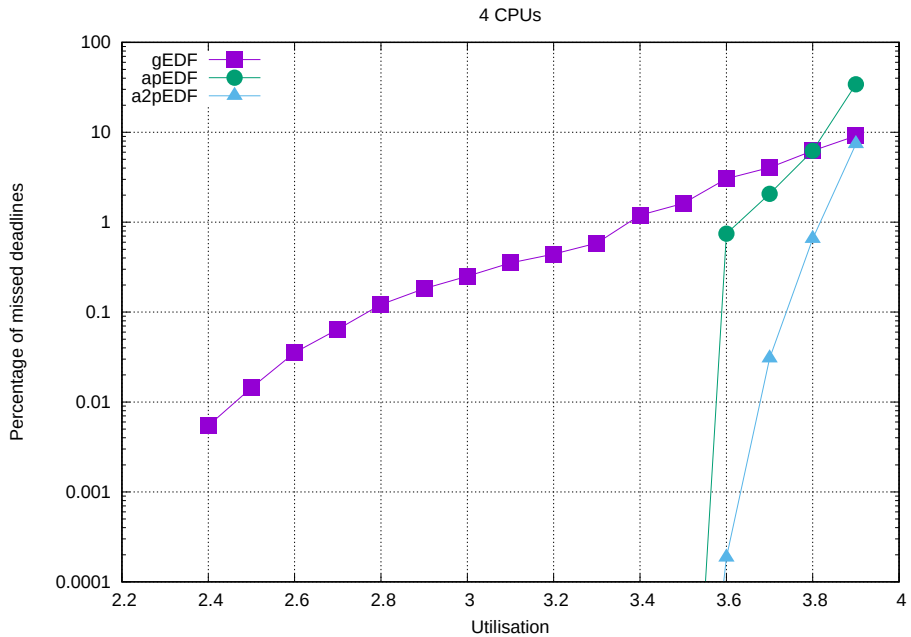
- Soft real-time tasks $\rightarrow$ deadline misses are allowed
- Measure the percentage of missed deadlines

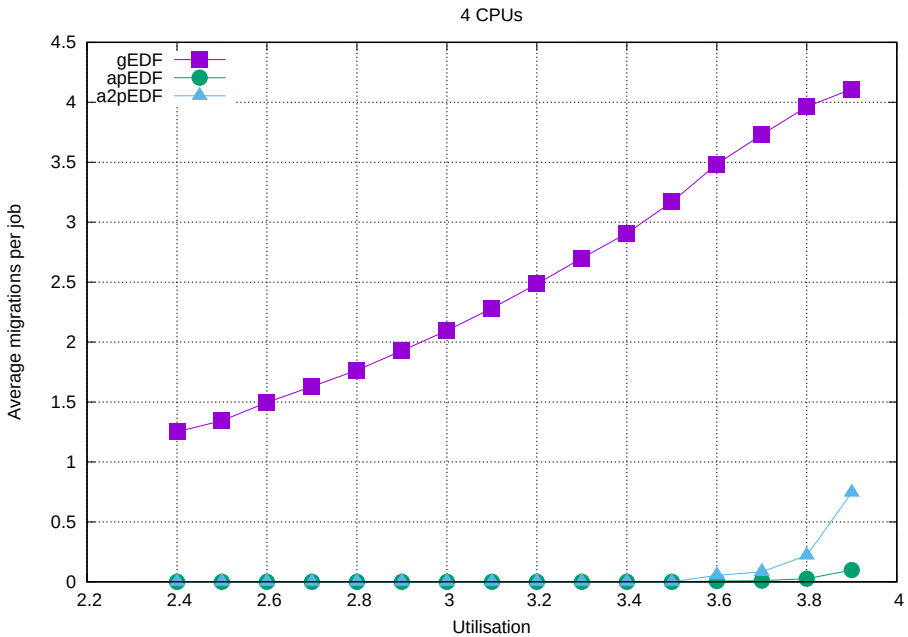




- First test: 2 cores, 16 tasks
- Utilisation ranging from 1.3 to 1.9
- apEDF always much better than gEDF!

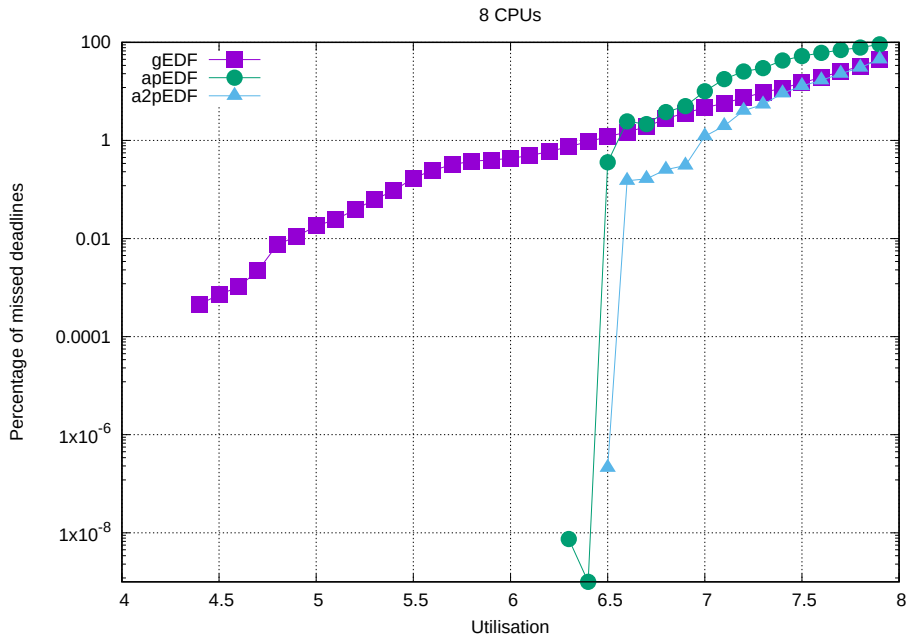
Increasing the Number of Cores

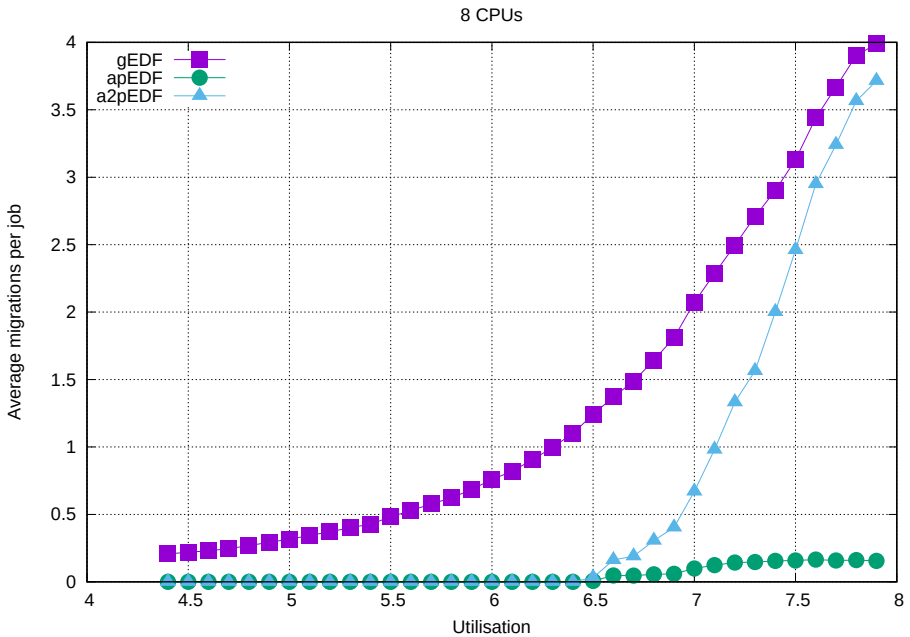




- Second test: 4 cores, 16 tasks
- Utilisation ranging from 2.4 to 3.9
- For very high utilisations (≥ 3.8), apEDF misses more deadlines than gEDF
  - a^{2}pEDF comes to rescue!

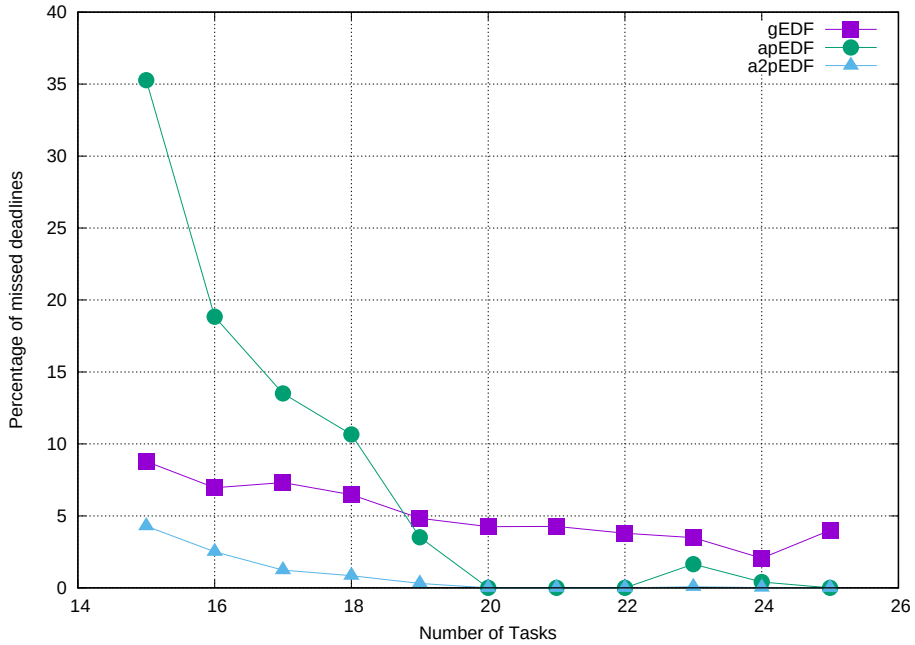
Even More Cores

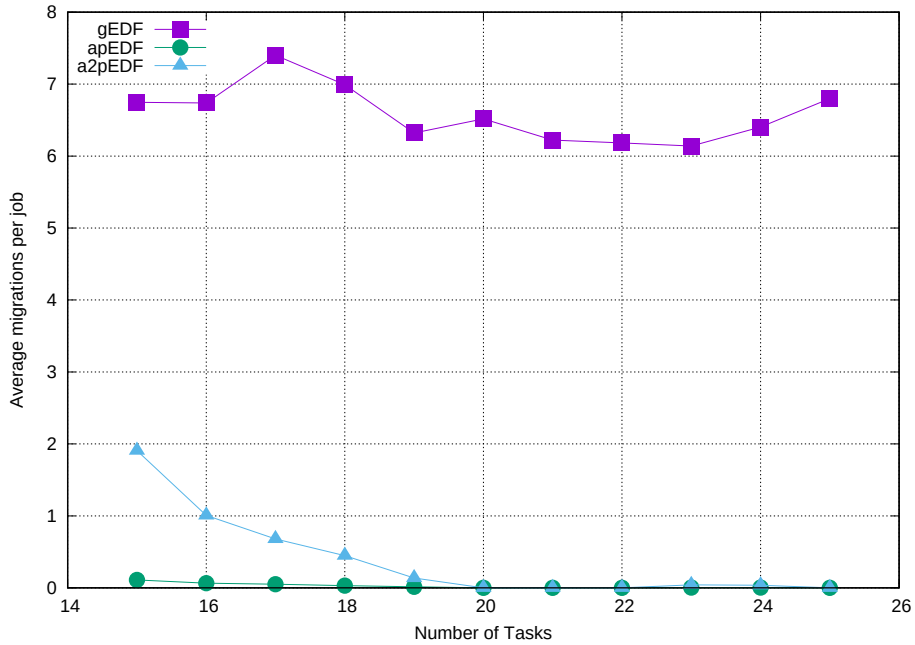




- Third test: 8 cores, 16 tasks
- Utilisation ranging from 4.4 to 7.9
- a^{2}pEDF needed for utilisation > 6.5
- For very high utilisations (> 7), a^{2}pEDF has performance very similar to gEDF

Varying the Number of Tasks

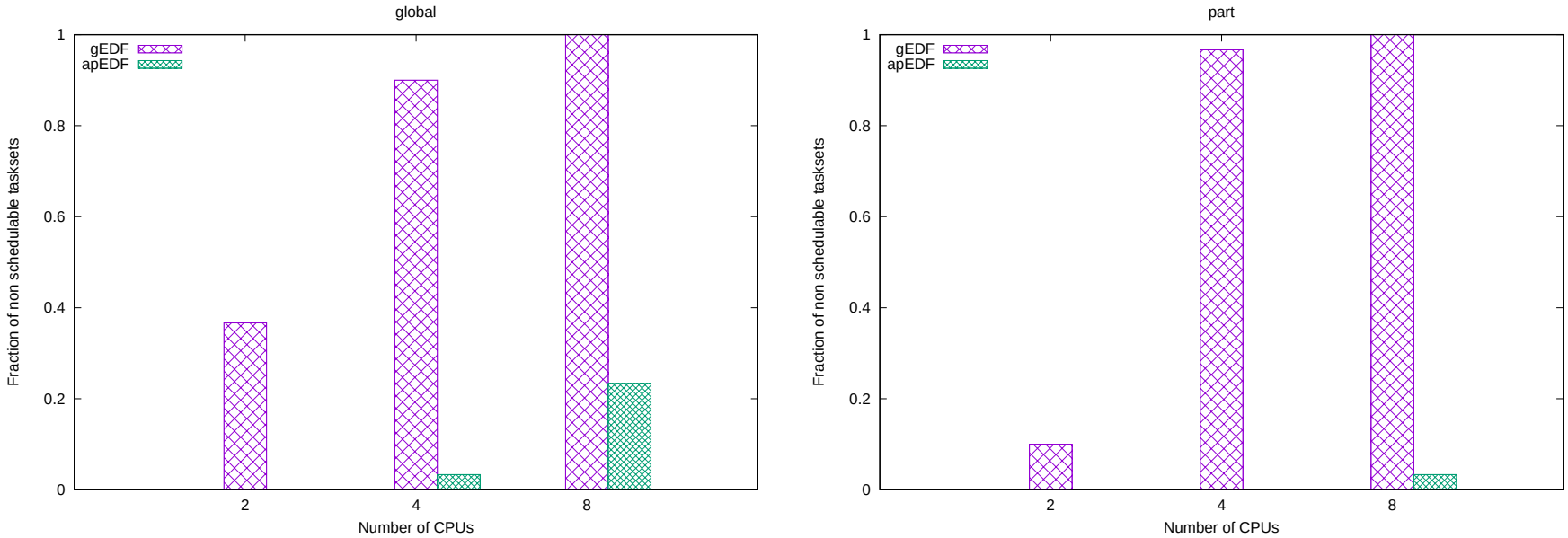




- 8 Cores, Utilisation 7.6
- Number of tasks varying between 15 and 25
- a^{2}pEDF always misses less deadlines than gEDF; apEDF misses less deadlines than gEDF for more than 18 tasks

Hard Real-Time Performance

- Generate 30 tasksets, and measure how many of them miss at least a deadline



- $U = 0.8M$ (U: utilisation; M: number of cores), 16 Tasks

Soft Real-Time Performance, Again

- Some setup as previous slide
- Measure the percentage of missed deadlines

CPUs	gEDF		apEDF	
	part	global	part	global
2	0.000000008	0.0000067	0	0
4	0.000008264	0.8935803	0	0.000000004
8	0.000022046	1.5758920	0.000000004	0.000000209

- And average migrations per job

CPUs	gEDF		apEDF	
	part	global	part	global
2	1.887913	1.965759	0.0000000056	0.0000000048
4	3.157379	3.188243	0.0000000041	0.0000000048
8	3.655992	2.795994	0.0000000060	0.0000000048

Conclusions

- Adaptive partitioning: new migration strategy for multi-core real-time schedulers
- Class of adaptively partitioned EDF algorithms: apEDF and a^{2}pEDF
- Performance evaluated through simulations
  - Better than gEDF for both soft and hard real-time
  - Small number of migrations compared to gEDF
  - Depending on the taskset utilisation, “pull” migrations might be needed
- Future work, both theoretical and practical:
  - Formally prove some conjectured properties
  - Implement in the Linux kernel