Uncertainty-Aware Resource Allocation for Multi-Path Programs with In-Kernel Predictions

Abigail Eisenklam[1*] Carlos A. Montenegro G.[2*] Xian Wang[1] Yifan Cai[1]
Robert Gifford[1] Linh Thi Xuan Phan[1] Ricardo G. Sanfelice[2]

[1]University of Pennsylvania

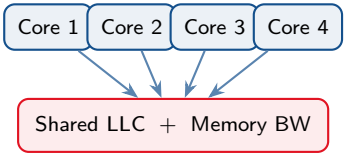
[2]University of California, Santa Cruz

38th European Conference on Real-Time Systems (ECRTS 2026)

*Equal contribution

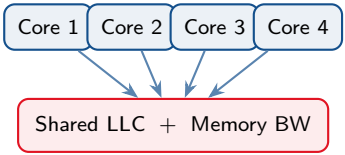
Multicore real-time systems share hardware resources

- Many real-time systems run on multicore platforms
- Cores contend for shared resources such as
  - last-level cache (LLC)
  - memory bandwidth (BW)
- Contention $\Rightarrow$ timing interference $\Rightarrow$ unpredictable execution times



Multicore real-time systems share hardware resources

- Many real-time systems run on multicore platforms
- Cores contend for shared resources such as
  - last-level cache (LLC)
  - memory bandwidth (BW)
- Contention ⇒ timing interference ⇒ unpredictable execution times



One approach to improve predictability

Divide shared resources into partitions and distribute them among tasks based on demand.

Existing allocation strategies make static decisions or program assumptions

Allocation strategy	Efficient utilization?	Unrestricted program behavior?
Static		
Dynamic		

Existing allocation strategies make static decisions or program assumptions

Allocation strategy	Efficient utilization?	Unrestricted program behavior?
Static		✓ allows dynamic behavior as long as WCET holds
Dynamic		

Existing allocation strategies make static decisions or program assumptions

Allocation strategy	Efficient utilization?	Unrestricted program behavior?
Static	✗ no adaptation to actual demand	✓ allows dynamic behavior as long as WCET holds
Dynamic		

Existing allocation strategies make static decisions or program assumptions

Allocation strategy	Efficient utilization?	Unrestricted program behavior?
Static	✗ no adaptation to actual demand	✓ allows dynamic behavior as long as WCET holds
Dynamic	✓ looks up current demand at runtime	

Existing allocation strategies make static decisions or program assumptions

Allocation strategy	Efficient utilization?	Unrestricted program behavior?
Static	✗ no adaptation to actual demand	✓ allows dynamic behavior as long as WCET holds
Dynamic	✓ looks up current demand at runtime	✗ relies on fixed code path or program input

Existing allocation strategies make static decisions or program assumptions

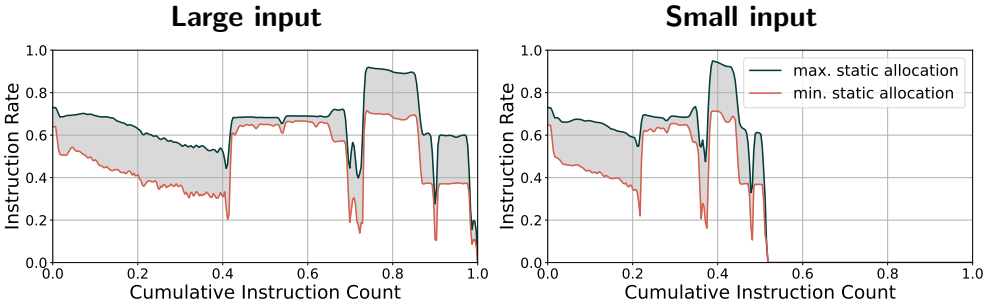
Allocation strategy	Efficient utilization?	Unrestricted program behavior?
Static	✗ no adaptation to actual demand	✓ allows dynamic behavior as long as WCET holds
Dynamic	✓ looks up current demand at runtime	✗ relies on fixed code path or program input

Reality is more complex

Resource demand

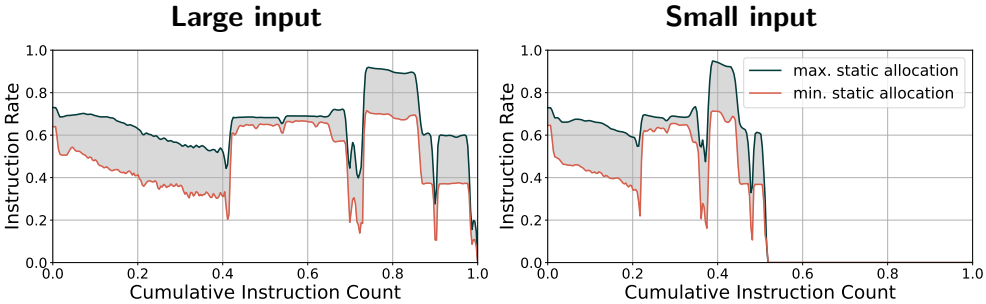
1. depends on program inputs and code paths
2. fluctuates across program phases
3. is affected by scheduling/ deadline urgency

Execution behavior varies within a program *and* across inputs



SPEC `omnetpp`: shaded region represents achievable instruction rates under different resource allocations.

Execution behavior varies within a program *and* across inputs



SPEC `omnetpp`: shaded region represents achievable instruction rates under different resource allocations.

Opportunity

Dynamically allocate shared resources to tasks based on their runtime resource demand under the current input.

Challenges

Challenge	**What we need**
Inputs are not known until the moment they arrive	Extremely efficient estimation of task execution behavior online

Challenges

Challenge	What we need
Inputs are not known until the moment they arrive	Extremely efficient estimation of task execution behavior online
It is often impractical to obtain the set of all valid inputs to a task	A model that generalizes over the input set

Challenges

Challenge	What we need
Inputs are not known until the moment they arrive	Extremely efficient estimation of task execution behavior online
It is often impractical to obtain the set of all valid inputs to a task	A model that generalizes over the input set
Generalization results in imperfect models	Resource allocation that accounts for model uncertainty

Challenges

Challenge	**What we need**
Inputs are not known until the moment they arrive	Extremely efficient estimation of task execution behavior online
It is often impractical to obtain the set of all valid inputs to a task	A model that generalizes over the input set
Generalization results in imperfect models	Resource allocation that accounts for model uncertainty
Input distributions *shift* at runtime	Uncertainty quantification for non-stationary input distributions

Challenges

Challenge	**What we need**
Inputs are not known until the moment they arrive	Extremely efficient estimation of task execution behavior online
It is often impractical to obtain the set of all valid inputs to a task	A model that generalizes over the input set
Generalization results in imperfect models	Resource allocation that accounts for model uncertainty
Input distributions *shift* at runtime	Uncertainty quantification for non-stationary input distributions

The above represents challenges in 1) offline task modeling and 2) online uncertainty-aware decision making.

Outline

System model

- n tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$
  - each releases jobs $\tau_{i,j}$ with arbitrary release $r_{i,j}$ and arbitrary deadline $d_{i,j}$
- Scheduled on M identical cores under some scheduling policy (e.g., global EDF or FP)
- Cores share B resource types
  - each split into equal-sized partitions (e.g., LLC via CAT and bandwidth via MemGuard)
- A resource allocation is a partition of the B resources to cores
- Resource allocations are recomputed every Δ time units

System model

- n tasks $\Gamma = \{\tau_1, \ldots, \tau_n\}$
  - each releases jobs $\tau_{i,j}$ with arbitrary release $r_{i,j}$ and arbitrary deadline $d_{i,j}$
- Scheduled on M identical cores under some scheduling policy (e.g., global EDF or FP)
- Cores share B resource types
  - each split into equal-sized partitions (e.g., LLC via CAT and bandwidth via MemGuard)
- A resource allocation is a partition of the B resources to cores
- Resource allocations are recomputed every Δ time units

Objective

Distribute the B resources across M cores at each timestep $t \in \{k\Delta : k \in \mathbb{N}\}$ to maximize resource utilization, subject to deadline and resource constraints.

Reference point: the clairvoyant optimal solution

If the inputs, release times, and dynamics of a (finite) set of jobs are known ahead of time, the ideal offline allocation under a deterministic scheduling policy is the solution to:

$$\max_{\{\beta_k\}} \sum_k \sum_{\tau_{i,j} \in S_k} \text{instructions retired in } [k, k+1) \text{ by } \tau_{i,j}$$

$$\begin{aligned} \text{subject to} \quad & R_{i,j} \le d_{i,j} && \text{(response time} \le \text{job deadlines)}, \\ & \textstyle\sum_{\tau_{i,j} \in S_k} \beta_{i,j,k} \le \beta^{\max} && \text{(resource usage} \le \text{capacity)}, \\ & x_{i,j,k+1} = G_i(x_{i,j,k},\, \beta_{i,j,k}) && \text{(job dynamics)}. \end{aligned}$$

- $G_i(x_{i,j,k}, \beta_{i,j,k})$ describes how $\tau_{i,j}$ evolves over $[k, k+1)$
  - given current state $x_{i,j,k}$ and allocation $\beta_{i,j,k}$

Reference point: the clairvoyant optimal solution

If the inputs, release times, and dynamics of a (finite) set of jobs are known ahead of time, the ideal offline allocation under a deterministic scheduling policy is the solution to:

$$\max_{\{\beta_k\}} \sum_k \sum_{\tau_{i,j} \in S_k} \text{instructions retired in } [k, k+1) \text{ by } \tau_{i,j}$$

$$\begin{aligned} \text{subject to} \quad & R_{i,j} \le d_{i,j} && \text{(response time} \le \text{job deadlines)}, \\ & \textstyle\sum_{\tau_{i,j} \in S_k} \beta_{i,j,k} \le \beta^{\max} && \text{(resource usage} \le \text{capacity)}, \\ & x_{i,j,k+1} = G_i(x_{i,j,k},\, \beta_{i,j,k}) && \text{(job dynamics)}. \end{aligned}$$

- $G_i(x_{i,j,k}, \beta_{i,j,k})$ describes how $\tau_{i,j}$ evolves over $[k, k+1)$
  - given current state $x_{i,j,k}$ and allocation $\beta_{i,j,k}$
- A solution gives the allocation vector β_k at each step k

Reference point: the clairvoyant optimal solution

If the inputs, release times, and dynamics of a (finite) set of jobs are known ahead of time, the ideal offline allocation under a deterministic scheduling policy is the solution to:

$$\max_{\{\beta_k\}} \sum_k \sum_{\tau_{i,j} \in S_k} \text{instructions retired in } [k, k+1) \text{ by } \tau_{i,j}$$

$$\begin{aligned} \text{subject to} \quad & R_{i,j} \le d_{i,j} && \text{(response time} \le \text{job deadlines)}, \\ & \textstyle\sum_{\tau_{i,j} \in S_k} \beta_{i,j,k} \le \beta^{\max} && \text{(resource usage} \le \text{capacity)}, \\ & x_{i,j,k+1} = G_i(x_{i,j,k},\, \beta_{i,j,k}) && \text{(job dynamics)}. \end{aligned}$$

Reality is more uncertain

Inputs, release times, and dynamics are not known in advance ⇒ our online method approximates this formulation without clairvoyance.

MPORA: Multi-path Online Resource Allocation

MPORA is an uncertainty-aware dynamic resource allocation framework for multi-path, input-dependent real-time tasks.

1. Task modeling	2. Decision making	3. Handling uncertainty
Learn execution behavior given a job's current state, input, and allocation	Receding-horizon optimization inspired by model predictive control (MPC)	Weighted conformal prediction (WCP) to quantify prediction error

Implemented as a Linux kernel module with microsecond-scale inference.

Our contribution: multicore allocation as *uncertainty-aware MPC over learned job dynamics*.

Online allocation: receding-horizon control

At each allocation step:

$$\max_{\{\beta_k\}_{k=0}^{N-1}} \sum_{k=0}^{N-1} \sum_{\tau_{i,j} \in S_k} \text{instructions retired in } [k, k+1) \text{ by } \tau_{i,j}$$

1. Sample state of executing jobs

Online allocation: receding-horizon control

At each allocation step:

$$\max_{\{\beta_k\}_{k=0}^{N-1}} \sum_{k=0}^{N-1} \sum_{\tau_{i,j} \in S_k} \text{instructions retired in } [k, k+1) \text{ by } \tau_{i,j}$$

1. Sample state of executing jobs
2. Roll out predicted state over an N-step horizon

Online allocation: receding-horizon control

At each allocation step:

$$\max_{\{\beta_k\}_{k=0}^{N-1}} \sum_{k=0}^{N-1} \sum_{\tau_{i,j} \in S_k} \text{instructions retired in } [k, k+1) \text{ by } \tau_{i,j}$$

1. Sample state of executing jobs
2. Roll out predicted state over an N-step horizon
3. Solve for allocations maximizing rate s.t. deadline constraints

Online allocation: receding-horizon control

At each allocation step:

$$\max_{\{\beta_k\}_{k=0}^{N-1}} \sum_{k=0}^{N-1} \sum_{\tau_{i,j} \in S_k} \text{instructions retired in } [k, k+1) \text{ by } \tau_{i,j}$$

1. Sample state of executing jobs
2. Roll out predicted state over an N-step horizon
3. Solve for allocations maximizing rate s.t. deadline constraints
4. Apply only the first step, then re-sample and re-solve

Online allocation: receding-horizon control

At each allocation step:

$$\max_{\{\beta_k\}_{k=0}^{N-1}} \sum_{k=0}^{N-1} \sum_{\tau_{i,j} \in S_k} \text{instructions retired in } [k, k+1) \text{ by } \tau_{i,j}$$

1. Sample state of executing jobs
2. Roll out predicted state over an N-step horizon
3. Solve for allocations maximizing rate s.t. deadline constraints
4. Apply only the first step, then re-sample and re-solve

What do we need to predict?

Given a job's state and a candidate future resource allocation, we must predict the

- instructions retired in the next interval of length Δ (for objective function), and
- remaining execution time (for deadline constraint).

Defining job execution state

The execution state must be expressive enough to predict near-term rate and remaining time.

Defining job execution state

The execution state must be expressive enough to predict near-term rate and remaining time.

We define the **execution state** $x_{i,j,k} \in \mathbb{R}^{p_i}$ of a job $\tau_{i,j}$ at step k with three components:

Component	**Purpose**
relevant features of the job's input (e.g., input size or sparsity; must be easily extractable at runtime)	enables generalization over input set

Defining job execution state

The execution state must be expressive enough to predict near-term rate and remaining time.

We define the **execution state** $x_{i,j,k} \in \mathbb{R}^{p_i}$ of a job $\tau_{i,j}$ at step k with three components:

Component	**Purpose**
relevant features of the job's input (e.g., input size or sparsity; must be easily extractable at runtime)	enables generalization over input set
cumulative instructions retired by the job	represents progress along the path

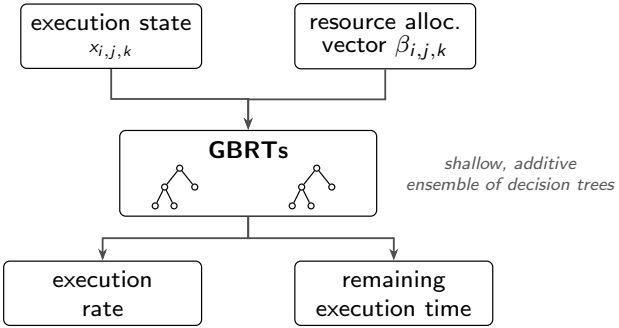
Defining job execution state

The execution state must be expressive enough to predict near-term rate and remaining time.

We define the **execution state** $x_{i,j,k} \in \mathbb{R}^{p_i}$ of a job $\tau_{i,j}$ at step k with three components:

Component	**Purpose**
relevant features of the job's input (e.g., input size or sparsity; must be easily extractable at runtime)	enables generalization over input set
cumulative instructions retired by the job	represents progress along the path
a short lag of the job's recent allocations and execution rates	effective for time-series forecasting

Choosing a model: gradient-boosted regression trees (GBRT)



Why GBRT?

- expressive: can outperform deep learning models at time-series regression tasks
- kernel-friendly: shallow trees, simple comparisons + adds, no dense matrix multiplication
- cheap to train: efficient training on CPU with XGBoost library, even for large datasets

Handling prediction error with conformal prediction

Problem. Learned models carry irreducible error which threatens deadline constraints.

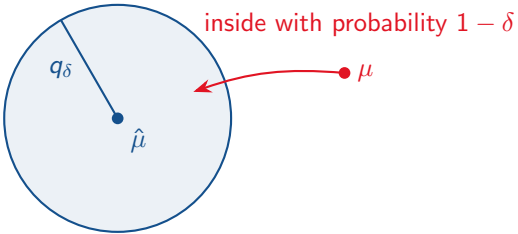
Handling prediction error with conformal prediction

Problem. Learned models carry irreducible error which threatens deadline constraints.

Conformal prediction (CP). Given a calibration dataset D_{cal} and desired coverage δ, CP computes an error bound q_δ such that

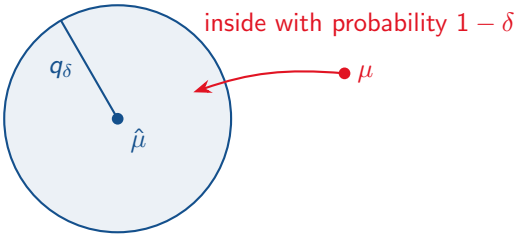
$$\mathrm{P}[\mu \in \hat{\mu} \pm q_\delta] \geq 1 - \delta$$

without any assumption on the distribution of the training data.



Handling prediction error with conformal prediction

Problem. Learned models carry irreducible error which threatens deadline constraints.



Uncertainty-aware deadline constraint

$$\texttt{rem\_time}(x_{i,j,k}, \beta_N) \; + \; \underbrace{q_{\delta_i} + \rho_N}_{\text{safety margin}} \; + t_0 + N\Delta \; \leq \; r_{i,j} + d_{i,j}$$

Tighten the deadline by the CP error bound and propagated roll-out error (details in paper).

Robustness to distribution shift: weighted CP

Problem. The guarantee given by CP is violated when the runtime input distribution differs from the calibration input distribution.

Robustness to distribution shift: weighted CP

Problem. The guarantee given by CP is violated when the runtime input distribution differs from the calibration input distribution.

Compensating for distribution shift. In weighted CP, we can give different weights to samples from the calibration dataset to get

$$\mathrm{P}[\mu \in \hat{\mu} \pm q_\delta] \geq 1 - \delta - \underbrace{\sum_{x_{\mathrm{cal}} \in D_{\mathrm{cal}}} w_{x_{\mathrm{cal}}} \underbrace{\mathrm{d_{TV}}(x_{\mathrm{cal}}, x_{\mathrm{online}})}_{\text{distribution shift}}}_{\text{distribution shift penalty}}$$

where $w_{x_{\mathrm{cal}}}$ is the weight assigned to calibration data point $x_{\mathrm{cal}} \in D_{\mathrm{cal}}$.

Robustness to distribution shift: weighted CP

Problem. The guarantee given by CP is violated when the runtime input distribution differs from the calibration input distribution.

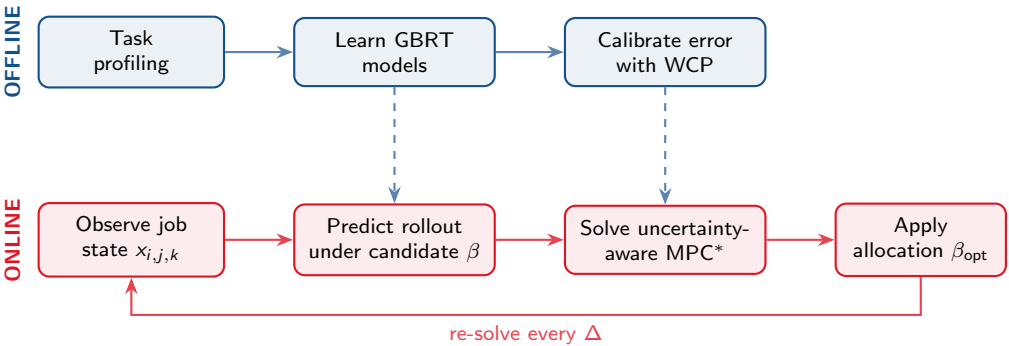
Compensating for distribution shift. In weighted CP, we can give different weights to samples from the calibration dataset to get

$$\mathrm{P}[\mu \in \hat{\mu} \pm q_\delta] \geq 1 - \delta - \underbrace{\sum_{x_{\mathrm{cal}} \in D_{\mathrm{cal}}} w_{x_{\mathrm{cal}}} \underbrace{\mathrm{d_{TV}}(x_{\mathrm{cal}}, x_{\mathrm{online}})}_{\text{distribution shift}}}_{\text{distribution shift penalty}}$$

where $w_{x_{\mathrm{cal}}}$ is the weight assigned to calibration data point $x_{\mathrm{cal}} \in D_{\mathrm{cal}}$.

- Systems often have finite operating scenarios (e.g., modes) with similar input features
- In this case, we can learn per-scenario weights that minimize the distribution shift penalty
- At runtime, store just one bound per scenario
  - negligible memory footprint, no need to retrain or store a separate model per scenario

MPORA: overall framework



*While we calibrate prediction error with WCP to formulate uncertainty-aware deadline constraints, we do not provide hard real-time guarantees. Our focus is on adaptation to input-dependent execution to improve resource utilization and deadline miss ratios.

Experimental setup

Workloads

- SPEC CPU 2017 Intspeed: `mcf`, `omnetpp`, `xalancbmk`, `x264`, `exchange2`, `xz`
- Profiling data collected for 200 diverse inputs $\times$ 20 distinct resource allocations $\times$ 10 runs per task

Experimental setup

Workloads

- SPEC CPU 2017 Intspeed: `mcf`, `omnetpp`, `xalancbmk`, `x264`, `exchange2`, `xz`
- Profiling data collected for 200 diverse inputs $\times$ 20 distinct resource allocations $\times$ 10 runs per task

Platform

- Intel Xeon, 8 cores (4 used for experiments), 20 MB L3
- CAT and MemGuard for resource control
- Linux 6.12 + `PREEMPT_RT`

Experimental setup

Workloads

- SPEC CPU 2017 Intspeed: `mcf`, `omnetpp`, `xalancbmk`, `x264`, `exchange2`, `xz`
- Profiling data collected for 200 diverse inputs $\times$ 20 distinct resource allocations $\times$ 10 runs per task

Platform

- Intel Xeon, 8 cores (4 used for experiments), 20 MB L3
- CAT and MemGuard for resource control
- Linux 6.12 + `PREEMPT_RT`

Protoype implementation

- GBRTs trained on 10% of inputs
- GBRTs converted to fixed-point inference
- Kernel module extracts input features, tracks job state, and runs **MPORA**:
  - on timer fire, make predictions, solve MPC, and apply allocations to cores

Model evaluation: accuracy, size, and speed

Across all benchmarks:

- Between 2.3–7.6 million data points in our test sets
- NMAE between 0.02–0.12 on *unseen* inputs
  - higher error for more complex inputs: simple input features cannot fully describe execution
- Negligible accuracy loss between floating- and fixed-point
- Median prediction time $\leq 1.23\,\mu s$
- All models between 3–8 KB
- GBRTs most effective when programs have distinct phases / high resource sensitivity
- GBRTs overkill when programs are CPU-bound: rate is a "flat line" across allocations

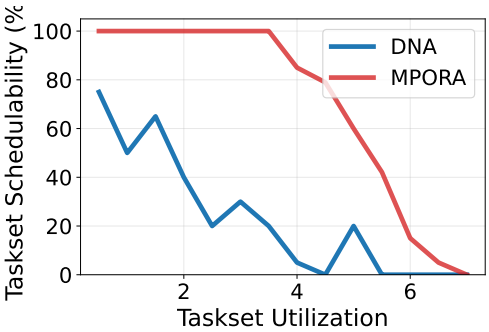
See paper for full results (reported per-benchmark).

Prototype evaluation: schedulability and overhead

Baseline: DNA [RTAS 2021]

Performs dynamic resource allocation using phase tables computed offline (stored for online lookup).

Prototype evaluation: schedulability and overhead



Empirical schedulability vs. reference utilization.

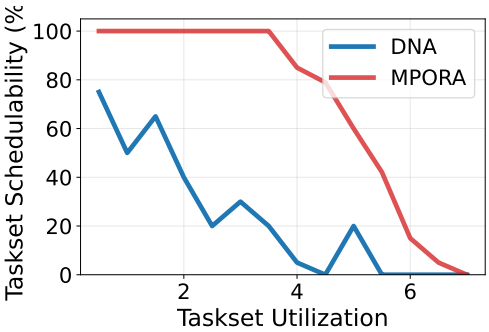
Baseline: DNA [RTAS 2021]

Performs dynamic resource allocation using phase tables computed offline (stored for online lookup).

Schedulability results

MPORA schedules 3× more tasksets than **DNA** across utilizations, since **DNA** is not input-aware.

Prototype evaluation: schedulability and overhead



Empirical schedulability vs. reference utilization.

Baseline: **DNA** [RTAS 2021]

Performs dynamic resource allocation using phase tables computed offline (stored for online lookup).

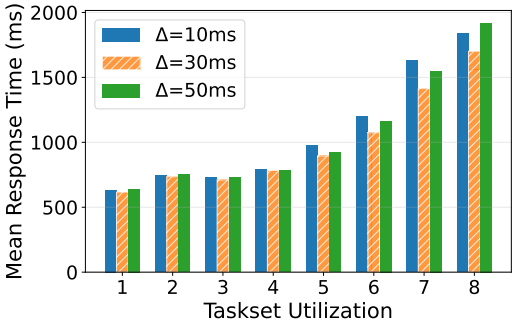
Schedulability results

MPORA schedules 3× more tasksets than **DNA** across utilizations, since **DNA** is not input-aware.

Overhead (window Δ=10 ms, horizon N=1)

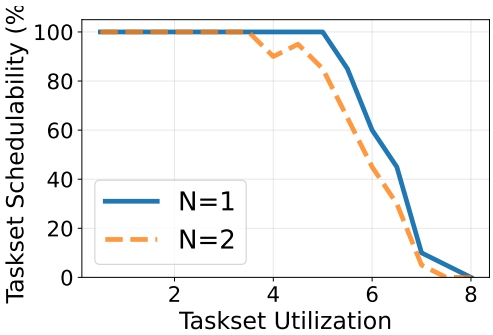
Mean 117 μs per allocation (max 367 μs).
On 1 of 4 cores: avg 0.29%, max 0.92%.

Prototype evaluation: effect of Δ



- As Δ increases, mean response times first decrease, then increase.
- The trade-off: reconfiguration overhead vs. responsiveness.
- The difference between settings grows at higher utilization.

Prototype evaluation: effect of N



- Larger N *hurts* schedulability.
- Predictions grow exponentially in N.
- Need to trim or sample the search space to enable larger N.

Conclusion

MPORA: Multi-path Online Resource Allocation

Learned execution dynamics + receding-horizon model-predictive control + weighted conformal prediction (WCP), implemented as a Linux kernel module.

Results

- Accurate, microsecond-scale, fixed-point predictions with 3–8 KB models
- WCP gives error bounds that are empirically robust to distribution shift (see paper)
- Schedulability improvements with $< 1\%$ overhead

Demonstrates that uncertainty-aware learning and control can operate inside the OS kernel.

Artifact reproduced, available at:
`github.com/phan-lab/MPORA`