Towards Breathable Distributed Systems: An Architecture for Cross-Platform Task Migration

Octavio Delgadillo[1,2] Bernhard Schwarzberg[2] Yinbo Zhou[2] Uwe Baumgarten[2]

[1]fortiss GmbH, Munich [2]Technical University of Munich

RAGE Workshop 2026, 11.05.2026, Saint Malo - France

Outline

Motivation

Automotive transformation / ECU Consolidation:

- 100s of single-purpose ECUs $\rightarrow$ few multi-purpose powerful platforms
- Static allocation cannot handle: overload · hardware failures · varying MEC availability
- *Breathability* would allow systems to dynamically adapt

Contributions

1. **Architectural design principles** for cross-platform task migration
2. **Axcan OS** OS framework (FreeRTOS-based) enabling dynamic deployment & stateful migration

Our Vision

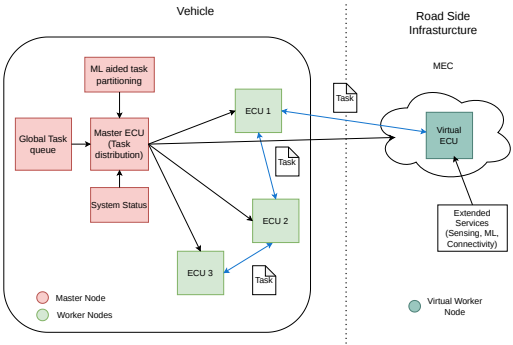
Breathable Distributed Systems

- Adapt to workload changes and dynamic hardware constraints at runtime by **migrating tasks** across heterogeneous nodes — physical ECUs *and* MEC resources.

Challenges

- Task allocation should dynamically adapt to workload and node availability
- OS should have minimal overhead and prioritize real-time capability
- Tasks should be deployed and (re) loaded dynamically
- Real-time deadlines should be met in a distributed system with global clock

Breathable Distributed System Architecture



Master–Worker Model:

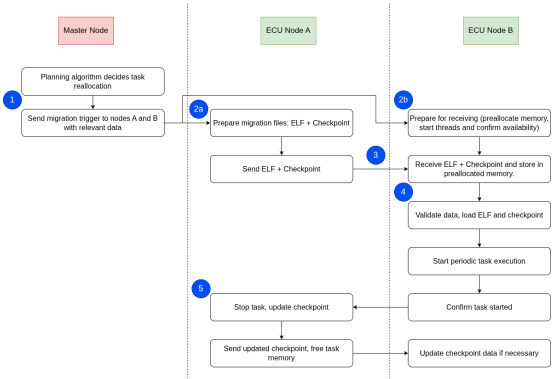
- **Master:** global view, ML-supported task placement & migration decisions
- **Physical ECU workers:** embedded ARM devices on vehicle network
- **Virtual ECU workers:** MEC-hosted containers (roadside units, edge servers)

Core principle

Any task ⇒ any compatible node, migrated at runtime.

Scope: independent tasks, common ISA (ARM aarch64).

Migration Protocol



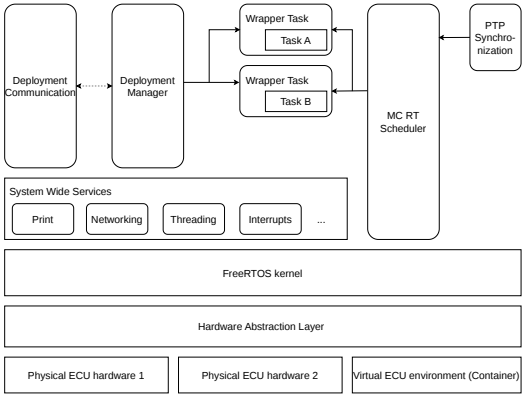
Coordinated protocol:

1. Master signals source & target
2. Source prepares ELF + checkpoint. Target allocates memory, confirms readiness
3. Source transmits executable + state
4. Target loads & validates.
5. Source halts *only after* confirmation from target, then sends updated checkpoint

Failure handling

If target fails: source continues, master retries on alternative node.

Axcan OS: Architecture Overview

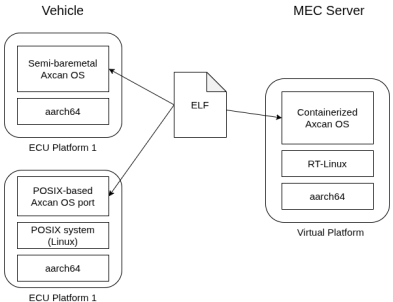


FreeRTOS extended with four main mechanisms:

1. **Dynamic task loading**
   PIE ELF binaries, no recompilation
2. **Cross-platform abstraction**
   Same binary: bare-metal ECU & Linux/POSIX port
3. **Checkpoint-based migration**
   Serialised task state, 1–10 KB typical
4. **Real-time coordination**
   PTP time sync + deadline compensation

Axcan: Nahuatl for "now"/"immediate"

Dynamic Loading & Cross-Platform Compatibility



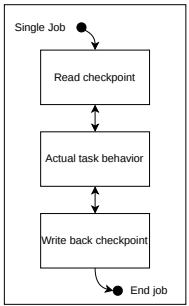
Dynamic Task Loading

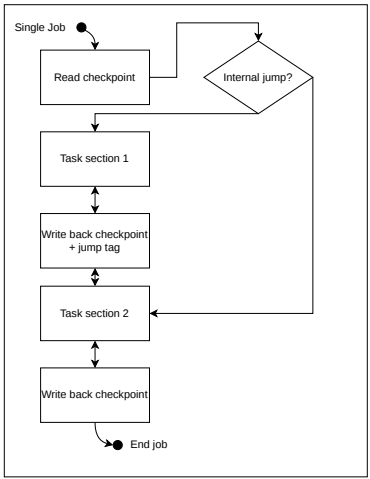
- Compile once as **PIE ELF**
- ELF loader
- System services via **resource table** (function pointers), no platform libs
- Same binary on ECU *and* MEC node

Cross-Platform Support

- **Embedded ECU**
  FreeRTOS HW port (Cortex-A53)
- **MEC / Linux**
  FreeRTOS POSIX port + RT-PREEMPT
- ARM aarch64 covers both tiers

Checkpoint Protocol: Task Architecture

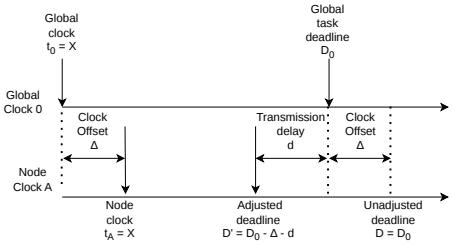




Standard task structure:

- Periodic tasks
- 1 job = 1 full execution of task
- Checkpoint restored at beginning of job
- Checkpoint saved at end of job
- Possible definition of intermediate checkpoints

Real-Time Coordination



Time synchronisation:

- Software PTP: $\Delta_{\max} \leq 10\,\text{ms}$
- Hardware PTP (TSN): planned ($< 1\,\mu\text{s}$)

Deadline compensation:

$$D' = D - \Delta_{\max} - d$$

- d: bounded network latency
- Applied uniformly on each node
  $\rightarrow$ relative EDF order preserved

Scheduler: EDF (proof-of-concept)
Mixed-criticality extensions planned.

Experimental Setup & Results

Physical ECU — Avnet Ultra96
Cortex-A53 @ 1.2 GHz · FreeRTOS HW port

Virtual ECU — Cavium ThunderX
aarch64 @ 2 GHz · PREEMPT_RT (Docker)

Network: Ethernet (TSN planned)

Task ID	Load (μs)		Exec (ms)	
	Phys.	Virt.	Phys.	Virt.
1	139	1123	10	6
2	130	1097	54	32
3	137	1077	310	178
4	141	1064	241	163
5	146	1117	84	50
6	135	1075	522	311

Key results:

- **Loading:** 130–146 μs (physical), $\sim$1.1 ms (virtual) — negligible vs. execution time
- **Exec. ratio** $\approx 0.6\times$ — consistent with 1.2 GHz / 2.0 GHz frequency ratio
- **PTP sync:** $\leq$ 10 ms offset
- **Stateful migration** verified: ECU$\rightarrow$ECU *and* ECU$\rightarrow$MEC

vs. stock FreeRTOS

FreeRTOS *only* supports static task compilation, *no* dynamic loading.
130–146 μs is the overhead over standard task creation.

Conclusion & Future Work

Our contribution

- Architecture for breathable distributed real-time systems
- Axcan OS: dynamic ELF loading, checkpointing, PTP coordination
- Single binary executes on physical ECU *and* MEC node
- Sub-ms loading, correct stateful migration

Next steps

- Master node redundancy
- Hard real-time migration strategies
- Reduce migration latency
- TSN + hardware PTP integration
- Security & safety certification

Thank You For Your Attention!

Contact:

delgadillo@fortiss.org bernhard.schwarzberg@tum.de

Backup Slides

For Q&A

Task & Node Characterisation (Detail)

Task model:

$$\tau = (\chi,\ N,\ r,\ C_{\text{wcet}},\ D,\ T)$$

- χ: criticality level
- N: number of jobs
- r: release time
- C_{wcet}: worst-case execution time
- D: relative deadline
- T: period (all times in ms)

All parameters except C_{wcet} are platform-invariant $\rightarrow$ one set of metadata per task, multiple WCET entries.

Node metadata:

- Unique ID, type (physical / virtual)
- Processor frequency, cores, memory
- Available system services (sensors, peripherals, networking)
- Network bandwidth & latency

Placement decision

Master selects target node based on task requirements, node capabilities, and ML-predicted system state [Delgadillo et al., RTNS 2022].

Checkpoint Protocol (Detail)

Periodic task model (sense–think–act):

- State serialised at job end → checkpoint file
- Restored at job start on target node
- Long jobs: sub-step flags for intermediate checkpoints

Checkpoint file format:

- Header: task ID + state size + checksum
- Binary-serialised state payload
- Typical size: **1–10 KB**
- Target validates size & checksum

Design trade-off

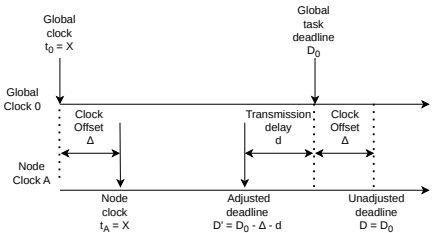
No arbitrary PC restoration
(no stack/heap capture)
⇒ lower complexity & overhead
⇒ well-suited to periodic tasks

No rollback needed

Source continues until target confirms takeover. On failure: source stays active, reports to master for retry.

Deadline Compensation — Formal Argument

$$D' = D - \Delta_{\max} - d$$



- $\Delta_{\max}$: maximum clock offset (SW PTP: 10 ms, HW PTP: $< 1\,\mu$s)
- d: bounded one-way network latency
- D': node-local deadline used by EDF

Correctness: task meeting D' locally guarantees global D.

Cost: reduces schedulable utilisation by $(\Delta_{\max} + d)/T$ per task.

Implication: $D \leq \Delta_{\max} + d \Rightarrow$ not schedulable; hardware PTP + TSN essential for hard real-time.

Current Limitations & Scope

Known limitations

- **Processor architecture constraint**
  ARM aarch64 in our PoC
- **Soft real-time only**
  Migration latency incompatible with hard RT deadlines for npw
- **Task independence assumed**
  No data dependencies
- **Single master** (SPOF)
  Redundancy deferred
- **Standard Ethernet**
  No deterministic guarantees
- **No security mechanisms**

Why these are acceptable now

- Fixed processor architecture is a design decision to avoid abstraction overhead
- **Work-in-progress:** goal is to validate *core feasibility*
- Evaluated workloads (10 ms – 522 ms) representative of non-safety-critical comfort & infotainment functions
- Each limitation is a clearly scoped future work item
- Modular design allows extensions without architectural rework

Related Work — Positioning

Area	Existing work	Gap / our contribution
Task scheduling	Delgadillo et al., Mittal et al.: optim. across vehicular nodes	Assume runtime environment exists; no OS-level migration support
MEC offloading	Mach et al., Liu et al., et al.: latency-sensitive offload	Application-level; no embedded OS support; no hard RT guarantees
Virtualisation	Reinhardt et al., Ayres et al.: AUTOSAR-compliant isolation	Static allocation, heavy overhead; we target lightweight dynamic migration
RTOS extensions	FreeRTOS, RTEMS; Golchin et al., Pourmohseni et al.	No dynamic loading; Axcan OS fills this gap at OS level