EMPYA: Saving Energy in the Face of Varying Workloads

Christopher Eibel, Thao-Nguyen Do, Robert Meißner,
and Tobias Distler

System Software Group
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

The 2018 IEEE International Conference on Cloud Engineering (IC2E '18)

April 20, 2018

Introduction & Motivation

- Execution platforms vs. energy demand in data centers
  - Programming and execution platforms are generally not energy aware
  - Dynamic applications in data centers are faced with varying workloads
  - Resources are often statically assigned

➔ **Consequence:** Lots of energy is being wasted

Introduction & Motivation

- Execution platforms vs. energy demand in data centers
  - Programming and execution platforms are generally not energy aware
  - Dynamic applications in data centers are faced with varying workloads
  - Resources are often statically assigned

➔ **Consequence:** Lots of energy is being wasted

- **Example application: key–value store**
  - Receives and processes user requests with basic operations (e.g., `get(key)`)
  - Programmers may choose between two configuration options:

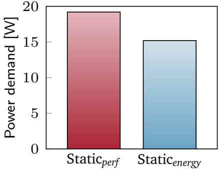
    ➔ Static$_{energy}$: Lower performance when load is high

    ➔ Static$_{perf}$: Wasting energy when load is low

Introduction & Motivation

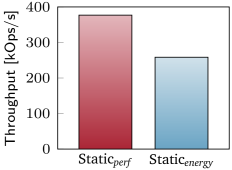
- Execution platforms vs. energy demand in data centers
  - Programming and execution platforms are generally not energy aware
  - Dynamic applications in data centers are faced with varying workloads
  - Resources are often statically assigned

➔ **Consequence:** Lots of energy is being wasted



Power demand at 70 kOps/s

- **Example application: key–value store**
  - Receives and processes user requests with basic operations (e.g., `get(key)`)
  - Programmers may choose between two configuration options:
    - ➔ Static$_{energy}$: Lower performance when load is high
    - ➔ Static$_{perf}$: Wasting energy when load is low



Maximum throughput

Introduction & Motivation

Execution platforms vs. energy demand in data centers

- Programming and execution platforms are generally not energy aware
- Dynamic applications in data centers are faced with varying workloads
- Resources are often statically assigned

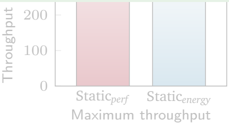
→ C

Goals

- Control the **energy–performance tradeoff**
- Propose **platform** that
  1. frees programmers from taking care of energy optimizations
  2. uses available techniques at hardware and software level
  3. adapts dynamically to varying workloads

Exa

- R
  w
- Programmers may choose between two configuration options:
  → Static_{energy}: Lower performance when load is high
  → Static_{perf}: Wasting energy when load is low



General Approach

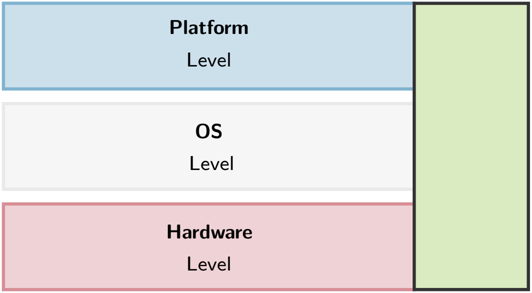
- EMPYA: **e**nergy-aware **m**iddleware **p**latform for d**y**namic **a**pplications
- Key design principles for EMPYA

❶ **Energy-efficiency awareness**
  - Avoid high CPU utilization because of disproportionate power-to-performance ratio
  - Not necessarily select configuration with full resource allocation

❷ **Multi-level awareness**
  - Exploit available techniques at multiple levels
  - Coordinate techniques: Best energy efficiency with respect to required performance

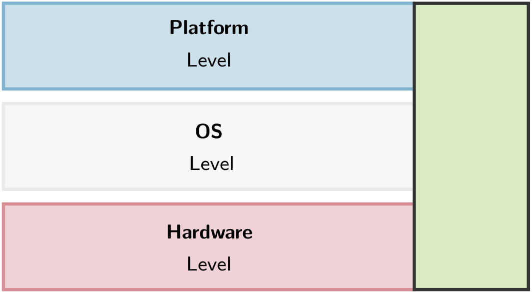
❸ **Energy awareness**
  - Integrated regulator making energy-aware reconfigurations
  - No additional services

EMPYA – Exploiting Techniques at Different Levels



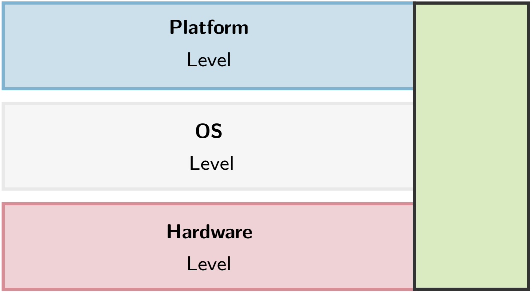
EMPYA – Exploiting Techniques at Different Levels

- ➔ Vary #threads
- ➔ Mapping of application components to threads



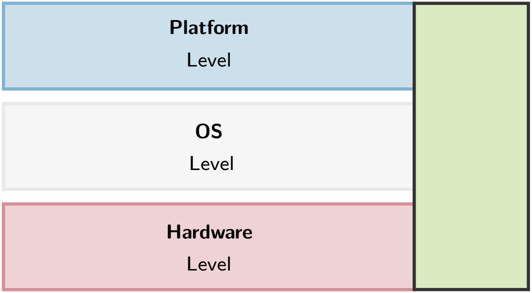
EMPYA – Exploiting Techniques at Different Levels

- ➔ Vary #threads
- ➔ Mapping of application components to threads
- ➔ Vary #(un)active cores
- ➔ Mapping of application threads to active cores



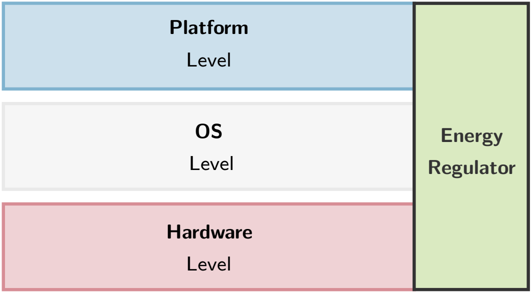
EMPYA – Exploiting Techniques at Different Levels

- ➔ Vary #threads
- ➔ Mapping of application components to threads
- ➔ Vary #(un)active cores
- ➔ Mapping of application threads to active cores
- ➔ Vary upper power limits
- ➔ Instruct hardware to enforce them

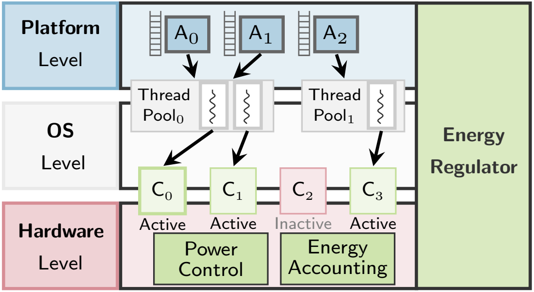


EMPYA – Exploiting Techniques at Different Levels

- ➔ Vary #threads
- ➔ Mapping of application components to threads
- ➔ Vary #(un)active cores
- ➔ Mapping of application threads to active cores
- ➔ Vary upper power limits
- ➔ Instruct hardware to enforce them



EMPYA – Exploiting Techniques at Different Levels



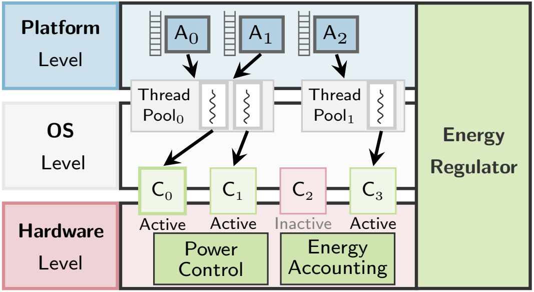
Empya – Exploiting Techniques at Different Levels

❶ Actors

- Each actor maintains its own state
- Communication via message passing
- Actor is independent of executing thread

➔ Implementation: **Akka toolkit**

akka

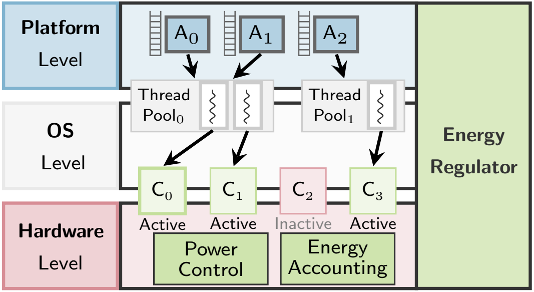


EMPYA – Exploiting Techniques at Different Levels

❶ Actors

- Each actor maintains its own state
- Communication via message passing
- Actor is independent of executing thread

➔ Implementation: **Akka toolkit**

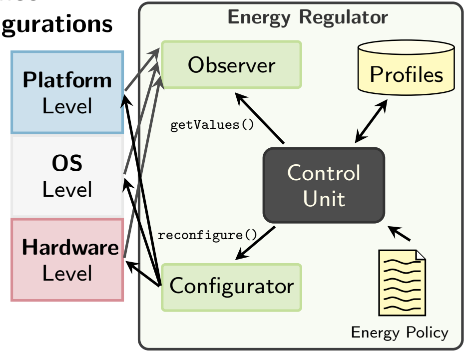


❷ Power limiting

- Running average power limit (RAPL)
- Originally developed for power limiting (e.g., temperature issues)
- Enables power and energy measurements
- Power capping very powerful for reducing the energy demand

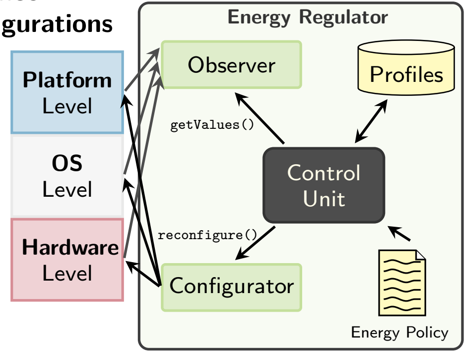
EMPYA – Energy Regulator

- Self-adapting system with continuous feedback loop
  - Monitor **application performance**
  - Emit dynamic HW/SW **reconfigurations**
- Energy-profile database
  - Configuration characteristics
  - Workload-specific power values
- Energy policies
  - Primary performance goal (e.g., throughput)
  - Secondary performance goal (e.g., latency)



EMPYA – Energy Regulator

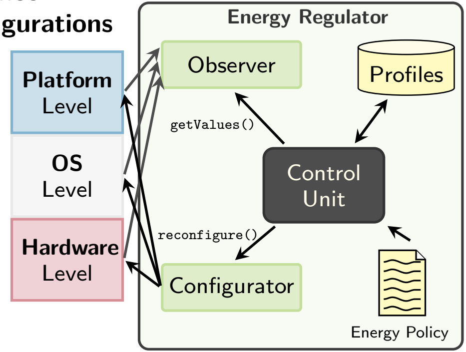
- Self-adapting system with continuous feedback loop
  - Monitor **application performance**
  - Emit dynamic HW/SW **reconfigurations**
- Energy-profile database
  - Configuration characteristics
  - Workload-specific power values
- Energy policies
  - Primary performance goal (e.g., throughput)
  - Secondary performance goal (e.g., latency)



ID	Configuration #Threads	Configuration #Cores	Configuration Cap	Performance Throughput	Performance Latency	Power usage
α	24	8	None	390.5 kOps/s	0.42 ms	51.2 W
				⋮	⋮	⋮
				70.4 kOps/s	0.37 ms	19.3 W
⋮	⋮	⋮	⋮	⋮	⋮	⋮
λ	12	6	22 W	224.8 kOps/s	0.62 ms	22.0 W
				⋮	⋮	⋮
				50.5 kOps/s	0.25 ms	15.3 W
⋮	⋮	⋮	⋮	⋮	⋮	⋮
ω	1	1	10 W	20.6 kOps/s	0.22 ms	10.0 W
				15.1 kOps/s	0.21 ms	9.7 W

EMPYA – Energy Regulator

- Self-adapting system with continuous feedback loop
  - Monitor **application performance**
  - Emit dynamic HW/SW **reconfigurations**
- Energy-profile database
  - Configuration characteristics
  - Workload-specific power values
- Energy policies
  - Primary performance goal (e.g., throughput)
  - Secondary performance goal (e.g., latency)



ID	Configuration			Performance		Power
	#Threads	#Cores	Cap	Throughput	Latency	usage
α	24	8	None	390.5 kOps/s	0.42 ms	51.2 W
				⋮		
				70.4 kOps/s	0.37 ms	19.3 W
⋮						
λ	12	6	22 W	224.8 kOps/s	0.62 ms	22.0 W
				⋮		
				50.5 kOps/s	0.25 ms	15.3 W
⋮						
ω	1	1	10 W	20.6 kOps/s	0.22 ms	10.0 W
				15.1 kOps/s	0.21 ms	9.7 W

```
energy policy {
    application = key-value-store;
    throughput_min_ops_per_sec = 10k;
    throughput_priority = pri;
    latency_max_msec = 0.5;
    latency_priority = sec;
}
```

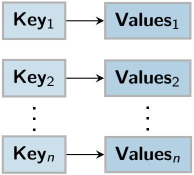
Evaluation – Evaluation Setup

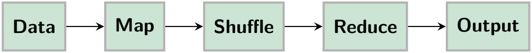
Hardware

- Client and server machines, switched 1 Gbps Ethernet
- Intel Xeon E3-1245 v3 & Xeon E3-1275 v5 processors
- 8 cores with Hyper-Threading enabled, 3.40 GHz
- Speed Step and TurboBoost enabled

Application classes

- **Use case A:** Key–value store with mixed operations (`get`, `set`, `exists`)
- **Use case B:** MapReduce running single, different jobs

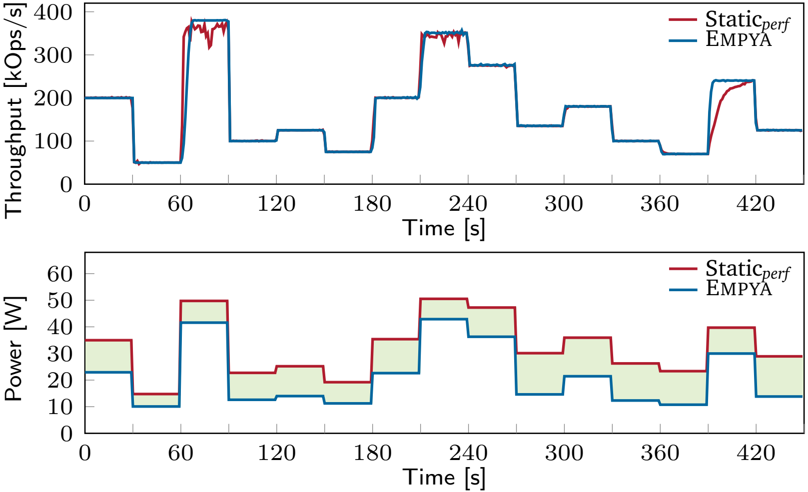




Evaluation – Use Case A: Key–Value Store

Static$_{perf}$ vs. EMPYA

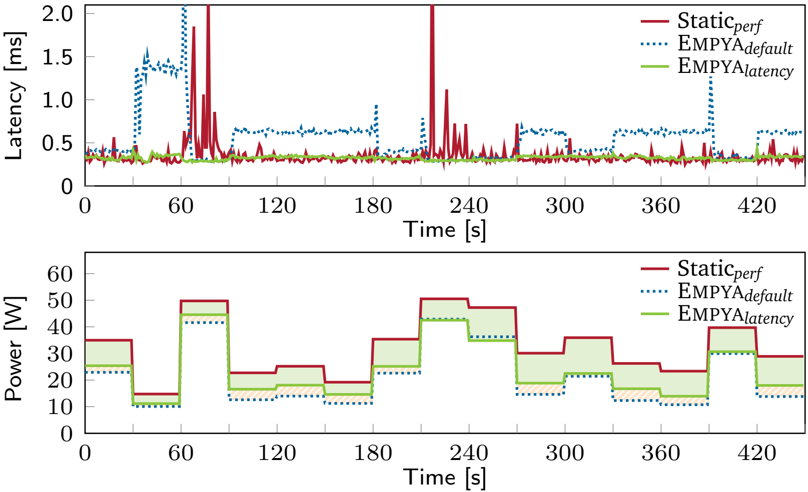
Throughput as primary performance goal



Evaluation – Use Case A: Key–Value Store

Static$_{perf}$ vs. EMPYA$_{latency}$

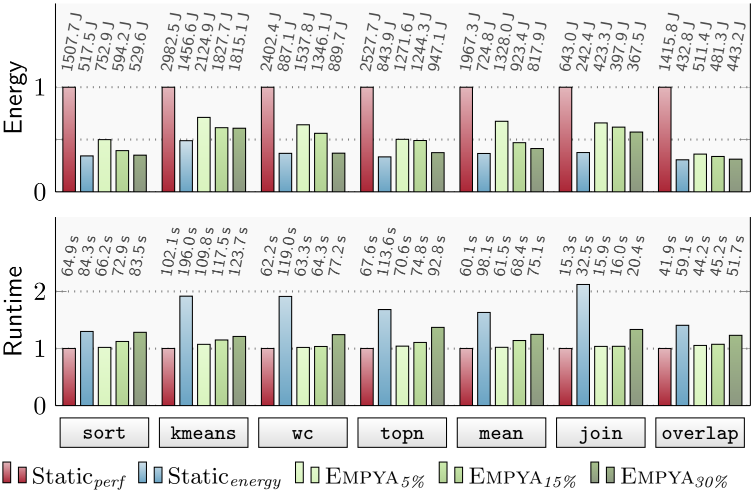
Throughput as primary and latency as secondary performance goal



Evaluation – Use Case B: MapReduce

Static$_{energy}$/Static$_{perf}$ vs. EMPYA

Performance goal: Specifying maximum execution-time penalties



Conclusion

- **EMPYA**
  - Self-adaptive middleware platform enforcing HW and SW reconfigurations
  - Exploiting actors and operating-system functionality
  - Power capping as an effective power- and energy-reduction measure
    - Key–value store: Up to 34 % less power demand
    - MapReduce: Energy savings of 22–64 %
- **Future and ongoing work**
  - Making decisions in a distributed manner for multiple machines
  - Carefully increasing the configuration space → heterogeneity

C. Eibel, C. Gulden, W. Schröder-Preikschat, and T. Distler
Strome: Energy-Aware Data-Stream Processing
In Proceedings of the 18th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2018), 2018.

Conclusion

- **EMPYA**
  - Self-adaptive middleware platform enforcing HW and SW reconfigurations
  - Exploiting actors and operating-system functionality
  - Power capping as an effective power- and energy-reduction measure
    - Key–value store: Up to 34 % less power demand
    - MapReduce: Energy savings of 22–64 %
- **Future and ongoing work**
  - Making decisions in a distributed manner for multiple machines
  - Carefully increasing the configuration space $\rightarrow$ heterogeneity

Thank you for your attention.

Questions?

C. Eibel, C. Gulden, W. Schröder-Preikschat, and T. Distler
Strome: Energy-Aware Data-Stream Processing
In Proceedings of the 18th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2018), 2018.