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
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
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
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
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\text{\text{energy}}: Lower performance when load is high.
- Static\text{\text{perf}}: Wasting energy when load is low.

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.
General Approach

EMPYA: energy-aware middleware platform for dynamic applications

Key design principles for EMPYA

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

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

3. Energy awareness
   - Integrated regulator making energy-aware reconfigurations
   - No additional services
EMPYA – Exploiting Techniques at Different Levels

- **Platform**
  - Level

- **OS**
  - Level

- **Hardware**
  - Level

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
→ Vary #threads
→ Mapping of application components to threads

<table>
<thead>
<tr>
<th>Platform Level</th>
<th>OS Level</th>
<th>Hardware Level</th>
</tr>
</thead>
</table>

IC2E ’18  Empya: Saving Energy in the Face of Varying Workloads  Approach
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

Platform Level
OS Level
Hardware Level
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

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

Implementation:
Akka toolkit

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

Platform Level

OS Level

Hardware Level

Energy Regulator

Thread Pool

C

Active

Inactive

Power Control

Energy Accounting

Thread Pool

C

Active

Active

Inactive

Active

Platform

OS

Hardware

Energy Regulator

A0 A1 A2
ThreadPool0 ThreadPool1

C0 C1 C2 C3
Active Active Inactive Active

Power Control

Energy Accounting
EMPYA – Exploiting Techniques at Different Levels

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

→ Implementation: **Akka toolkit**

![Diagram](image_url)
1. **Actors**
   - Each actor maintains its own state
   - Communication via message passing
   - Actor is independent of executing thread

   → Implementation: *Akka toolkit*

2. **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
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)
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)

<table>
<thead>
<tr>
<th>ID</th>
<th>Configuration</th>
<th>Performance</th>
<th>Power usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Threads</td>
<td>#Cores</td>
<td>Cap</td>
</tr>
<tr>
<td>α</td>
<td>24</td>
<td>8</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>λ</td>
<td>12</td>
<td>6</td>
<td>22 W</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ω</td>
<td>1</td>
<td>1</td>
<td>10 W</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>ID</th>
<th>Configuration</th>
<th>Performance</th>
<th>Power usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Threads</td>
<td>#Cores</td>
<td>Cap</td>
</tr>
<tr>
<td>α</td>
<td>24</td>
<td>8</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>λ</td>
<td>12</td>
<td>6</td>
<td>22W</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ω</td>
<td>1</td>
<td>1</td>
<td>10W</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

![Diagram of Key-Value Store](Image)
Evaluation – Use Case A: Key–Value Store

**Static\textsubscript{perf} vs. EMPYA**

*Throughput as primary performance goal*

![Graph showing throughput comparison between Static\textsubscript{perf} and EMPYA over time.]

![Graph showing power consumption comparison between Static\textsubscript{perf} and EMPYA over time.]

IC2E ’18  EMPYA: Saving Energy in the Face of Varying Workloads  Evaluation 8
Evaluation – Use Case A: Key–Value Store

**Static_{perf} vs. EMPYA_{latency}**

Throughput as primary and latency as secondary performance goal

![Graph showing comparison of Static_{perf} and EMPYA_{latency}](image)

![Graph showing comparison of Static_{perf} and EMPYA_{latency}](image)
Evaluation – Use Case B: MapReduce

**Static\textsubscript{energy}/Static\textsubscript{perf} vs. EMPYA**

**Performance goal:** Specifying maximum execution-time penalties

![Graph showing performance comparison](image-url)
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**

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

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._