Monitoring Energy Consumption With SIOX
Autonomous Monitoring Triggered by
Abnormal Energy Consumption

Julian M. Kunkel\textsuperscript{1}  Alvaro Aguilera\textsuperscript{2}  Nathanael Hübbe\textsuperscript{3}
Marc Wiedemann\textsuperscript{3}  Michaela Zimmer\textsuperscript{3}

\textsuperscript{1} DKRZ
\textsuperscript{2} ZIH Dresden
\textsuperscript{3} University of Hamburg

September 2\textsuperscript{nd} 2014, EnA-HPC
Outline

1. Introduction
2. Architecture
3. Analyzing Energy Consumption
4. Intelligent Monitoring
5. Summary
SIOX will

- collect and analyse
  - activity patterns and
  - performance metrics

in order to

- assess system performance
- locate and diagnose problem
- learn & apply optimizations
- intelligently steer monitoring
1 Tool for tracking and analysis of energy consumption
   - For (potentially) every application
   - System-wide

2 Intelligent monitoring: Restrict collection of events
   - Trigger upon “abnormal” energy consumption
   - Evaluate some strategies
Partners and Funding

- Funded by the BMBF
  - Grant No.: 01 IH 11008 B
- Start: Juli 1st, 2011
- End: September 30, 2013
Outline

1. Introduction
2. Architecture
   - Modularity of SIOX
   - Example Configuration
3. Analyzing Energy Consumption
4. Intelligent Monitoring
5. Summary
Modularity of SIOX

- The SIOX architecture is flexible and developed in C++ components
- License: LGPL, vendor friendly
- Upon start-up of (instrumented) applications, modules are loaded
- Configuration file defines modules and options
  - Choose advantageous plug-ins
  - Regulate overhead
- For debugging, reports are output at application termination
  - SIOX may gather statistics of (application) behavior / activity
  - Provide (internal) module statistics
Example Configuration

- **Monitoring Path**
  - Activities: I/O operations
  - Node statistics: CPU ...
  - Data stored in files or DB

- **Knowledge Path**
  - Propagate/use knowledge
  - Reasoner: correlates issues and trigger monitoring

- **System Statistics**
  - Support hardware counters

- **Reporting**
  - Provide debug statistics
Features of the Working Prototype

- Monitoring
  - Application (activity) behavior
  - Ontology and system information
  - Data can be stored in files or Postgres database
  - Trace reader

- Daemon
  - Applications forward activities to the daemon
  - Node statistics are captured
  - Energy consumption (RAPL) can be captured

- Activity plug-ins
  - GenericHistory plug-in tracks performance, proposes MPI hints
  - Fadvise (ReadAhead) injector
  - FileSurveyor prototype – Darshan-like

- Reporting of statistics on console or file (independent and MPI-aware)
Analyzing Energy Consumption

Outline

1. Introduction

2. Architecture

3. Analyzing Energy Consumption
   - System-Level Analysis: Database GUI
   - User-Level Analysis: Reporter
   - Example Use-Case

4. Intelligent Monitoring

5. Summary
A PHP GUI provides access to the Postgres DB
Overview of applications, activities, chain-of-effects
Statistics of individual applications

Detail of Activity 19693

Causal Chain

```
#19691: MPI_Init()

#19690: fopen()

#19693: fread()
```

Attribute List

<table>
<thead>
<tr>
<th>name</th>
<th>fread()</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique_id</td>
<td>19693</td>
</tr>
<tr>
<td>ucaaid</td>
<td>200</td>
</tr>
<tr>
<td>id</td>
<td>5</td>
</tr>
<tr>
<td>thread_id</td>
<td>1</td>
</tr>
<tr>
<td>cid_pid_nid</td>
<td>103</td>
</tr>
<tr>
<td>cid_pid_time</td>
<td>7 d 7 h 48 m 19 s</td>
</tr>
<tr>
<td>cid_id</td>
<td>0</td>
</tr>
<tr>
<td>time_start</td>
<td>27 03 2014 17 47 16 940989834</td>
</tr>
<tr>
<td>time_stop</td>
<td>27 03 2014 17 47 16 9410272243</td>
</tr>
<tr>
<td>duration</td>
<td>37.409 µs</td>
</tr>
<tr>
<td>attributes</td>
<td>quantity(BytesToRead = 8192</td>
</tr>
<tr>
<td>remote_calls</td>
<td>MPI_Init(), fopen()</td>
</tr>
<tr>
<td>parents</td>
<td></td>
</tr>
<tr>
<td>error_value</td>
<td></td>
</tr>
</tbody>
</table>

Detailed view of an activity showing the causal chain and list of attributes.
Statistics for instrumented applications
- Average value for complete run
- Diagrams for each statistic
- Sampling interval: 100 ms

<table>
<thead>
<tr>
<th>ID</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>3666</td>
<td>./parabench -e -D pattern0.pbl</td>
</tr>
<tr>
<td>3828</td>
<td>./parabench -e -D pattern1.pbl</td>
</tr>
<tr>
<td>3979</td>
<td>./parabench -e -D pattern2.pbl</td>
</tr>
<tr>
<td>4156</td>
<td>./parabench -e -D pattern3.pbl</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Duration</th>
<th>Socket/RAPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>3666</td>
<td>377 s</td>
<td>23605 J</td>
</tr>
<tr>
<td>3828</td>
<td>133 s</td>
<td>8279 J</td>
</tr>
<tr>
<td>3979</td>
<td>119 s</td>
<td>7453 J</td>
</tr>
<tr>
<td>4156</td>
<td>120 s</td>
<td>7541 J</td>
</tr>
</tbody>
</table>
User-Level Analysis: Reporter

- Executed upon application termination
- Outputs statistics reported by reasoner
- MPI reporter aggregates statistics among all processes

Example output reported by the Reasoner

```
1 CONSUMED_CPU_SECONDS = 2.285407
2 CONSUMED_ENERGY_JOULE = 46.924286
3 ACCESSED_IO_BYTES = 23068672
4 TRANSFERRED_NETWORK_BYTES = 6336953
5 OBSERVED_RUNTIME_MS = 2600
```
Energy Consumption of MPI-IO

Comparing the 4 levels of access
- Independent vs. Collective
- Contiguous vs. Noncontiguous
- 8 processes writing/reading 6400 blocks of 100 KiB

Parabench runtime and energy consumption per process.

<table>
<thead>
<tr>
<th>Level</th>
<th>Runtime [s]</th>
<th>Energy [J]</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>377</td>
<td>23605</td>
<td>21335</td>
</tr>
<tr>
<td>1</td>
<td>133</td>
<td>8279</td>
<td>17390</td>
</tr>
<tr>
<td>2</td>
<td>119</td>
<td>7453</td>
<td>11704</td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>7541</td>
<td>2826</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Architecture
3. Analyzing Energy Consumption
4. Intelligent Monitoring
   - Workflow
   - Strategies
   - Evaluation
5. Summary
Workflow & Configuration

- **ANetFWClient**: buffers observed activities
- **Likwid plugin**: provides RAPL statistics
- **StatisticsHealthADPI**: identifies abnormal states
- **Reasoners**: correlate issues and trigger monitoring
  1. Processes query node information from daemon
  2. Client reasoner triggers local monitoring
Reasoning Rules (on the Node-Reasoner)

```java
query_currentAnomalies(nBad, nGood, nOther)
if (nBad + nGood + nOther > 0)
    if (nBad > kThreshold && nBad > kRatio * nGood)
        // mostly bad issues
        state = ABNORMAL_BAD
    else if (nGood > kThreshold && nGood > kRatio * nBad)
        // mostly good issues
        state = ABNORMAL_GOOD
    else
        // good & bad at the same time
        state = ABNORMAL_OTHER
...
if (state != GOOD && state != ABNORMAL_GOOD)
    // try to explain the CAUSE
    for (c in categories)
        if (utilization[c] > kMaxLoad)
            raise_issue("Overloaded", c)
```

Node status is forwarded and used by process reasoner.
Excerpt of Strategies

What are “abnormal”/interesting energy states to monitor?

- Interesting statistics:
  - Best, worst 5% energy consumption (Strategy: “StatisticsADPI”)
  - Sudden change in statistic behavior
- System does not behave as it should:
  - Energy consumption not as predicted by a model
  - Example: CPU utilization (Strategy: “EnergyEfficiencyADPI”)

Pseudocode for “EnergyEfficiencyADPI” algorithm

```
1 query_current(cpuConsumed)
2 query_current(energyConsumed)
3 currentEfficiency = cpuConsumed / energyConsumed
4 nValues = nValues + 1
5 update_distribution_estimate(currentEfficiency)
6 query_estimated_distribution(mean, stddev)
7 if (nValues > nStabilizationLimit)
8   if (currentEfficiency > mean + stddev)
9     flag_anomaly(ABNORMAL_GOOD)
10   if (currentEfficiency < mean - stddev)
11     flag_anomaly(ABNORMAL_BAD)
```
Evaluation

- Evaluated on the ICON climate model
- Writes variables as 2D/3D arrays into a shared file
- 4 repeats with one plugin, 5 with both, nStabilizationLimit=25

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Abnormal phases</th>
<th>Activities</th>
<th>Activities stored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>S</td>
<td>E</td>
<td>S&amp;E</td>
</tr>
<tr>
<td>Avg.</td>
<td>553 s</td>
<td>0.3 %</td>
<td>5.8 %</td>
<td>39.4 %</td>
</tr>
<tr>
<td>Min</td>
<td>552 s</td>
<td>0.2 %</td>
<td>4.6 %</td>
<td>3.0 %</td>
</tr>
<tr>
<td>Max</td>
<td>553 s</td>
<td>0.4 %</td>
<td>7.2 %</td>
<td>87.7 %</td>
</tr>
</tbody>
</table>

S: Statistics ADPI only, E: Energy Efficiency ADPI only, S&E: both.
SIOX aims to monitor and optimize I/O

We are building a modular and open system

SIOX can capture and analyze (energy) statistics
  - System-wide database
  - Application-specific reports

We can restrict monitoring to “relevant“ phases
  - Trigger monitoring on abnormal energy consumption
  - Reduces activities to 6-10%.

More research needed to take full benefit.
Module Interactions of an Example Configuration

SIOX within the Process

MPI
- MPIReporter
- AMux
- AMux Plug-Ins
  - AForwarder
  - FileSurveyor
  - GenericHistory

POSIX
- AMux Plug-Ins
  - FileSurveyor
  - Fadvise read-ahead
  - AForwarder

- AMux
- Reasoner

- ConsoleReporter
  - Histogram ADPI
  - ANetFWClient

SIOX Daemon

- ConsoleReporter
- ANetFWServer
- StatisticsCollector
- SMux
- Reasoner

- AMux
- FileWriter
- PostgresWriter

Statistics Provider Plugins
- CPU
- Network
- Memory
- QualitativeUtilization

Global
- Reasoner
- Statistics File
- Activity File
- Transaction System
  - Activity Store

Julian M. Kunkel

Autonomous Monitoring Triggered by Abnormal Energy Consumption