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

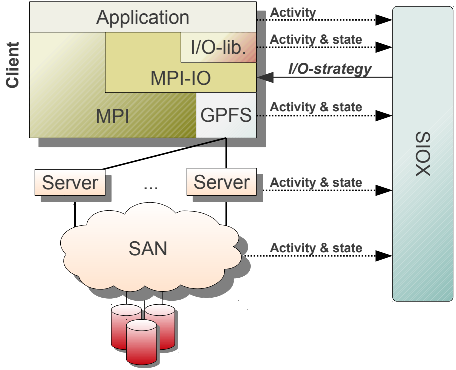
Julian M. Kunkel[1] Alvaro Aguilera[2] Nathanael Hübbe[3]
Marc Wiedemann[3] Michaela Zimmer[3]

1 DKRZ
2 ZIH Dresden
3 University of Hamburg

September 2^{nd} 2014, EnA-HPC

Outline

Project Goals



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

Goals & Contributions

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
   - Evalute basic strategies

Partners and Funding

Bundesministerium
für Bildung
und Forschung

- Funded by the BMBF
  Grant No.: 01 IH 11008 B
- Start: Juli 1st, 2011
- End: September 30, 2014

DKRZ
DEUTSCHES
KLIMARECHENZENTRUM

HLRS

IBM

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

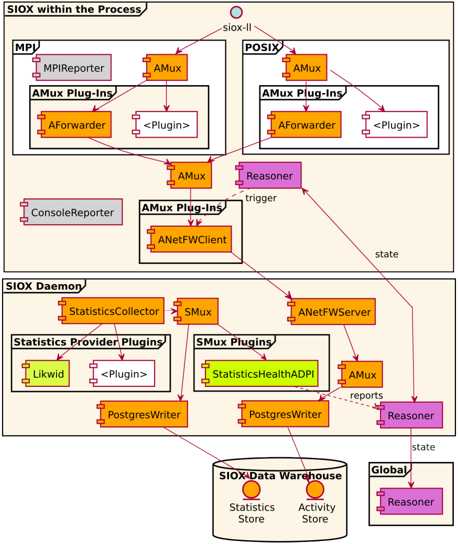
ZIH
Zentrum für Informationsdienste
und Hochleistungsrechnen

Outline

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)

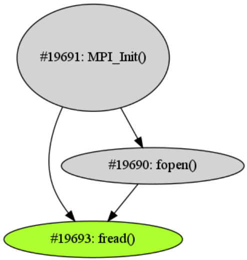
Outline

System-Level Analysis: Database GUI

- 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



Attribute List

name	fread()
unique_id	19693
ucaid	200
id	5
thread_id	1
cid_pid_nid	103
cid_pid_time	7 d 7 h 48 m 19 s
cid_id	0
time_start	27.03.2014 17:47:16.**940989834**
time_stop	27.03.2014 17:47:16.**941027243**
duration	37.409 µs
attributes	quantity/BytesToRead = 8192
remote_calls	
parents	MPI_Init(), fopen()
error_value	

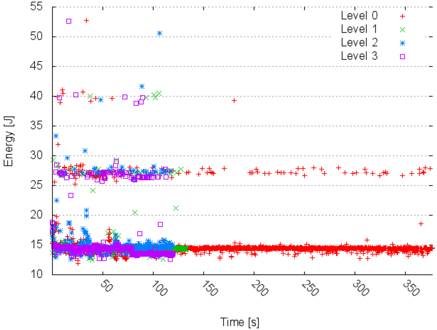
Detailed view of an activity showing the causal chain and list of attributes.

System-Level Analysis: Database

- Statistics for instrumented applications
    - Average value for complete run
- Diagrams for each statistic
    - Sampling interval: 100 ms

```
ID       Command
[3666]:  ./parabench -e -D pattern0.pbl
[3828]:  ./parabench -e -D pattern1.pbl
[3979]:  ./parabench -e -D pattern2.pbl
[4156]:  ./parabench -e -D pattern3.pbl

ID    Duration    Socket/RAPL [...]
3666  377 s       23605 J
3828  133 s       8279 J
3979  119 s       7453 J
4156  120 s       7541 J
```



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

```
CONSUMED_CPU_SECONDS = 2.285407
CONSUMED_ENERGY_JOULE = 46.924286
ACCESSED_IO_BYTES = 23068672
TRANSFERRED_NETWORK_BYTES = 6336953
OBSERVED_RUNTIME_MS = 2600
```

Energy Consumption of MPI-IO

Comparing the 4 levels of access

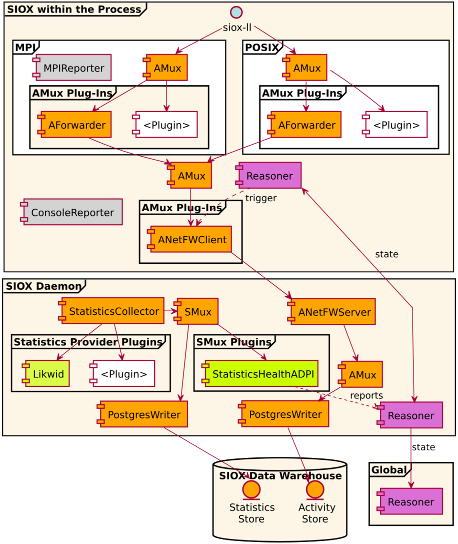
- Independent vs. Collective
- Contiguous vs. Noncontiguous
- 8 processes writing/reading 6400 blocks of 100 KiB

Parabench runtime and energy consumption per process.

Level	Runtime [s]	Energy [J]	Activities
0	377	23605	21335
1	133	8279	17390
2	119	7453	11704
3	120	7541	2826

Outline

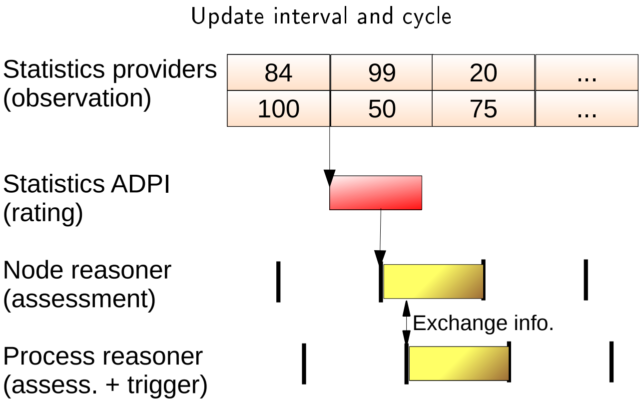
Workflow & Configuration



- 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

Information Updates and Timing

- Statistics and reasoner update every 100 ms
- No synchronization between reasoner(s) and statistics



Reasoning Rules (on the Node-Reasoner)

```
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

```
query_current(cpuConsumed)
query_current(energyConsumed)
currentEfficiency = cpuConsumed / energyConsumed
nValues = nValues + 1
update_distribution_estimate(currentEfficiency)
query_estimated_distribution(mean, stddev)
if (nValues > nStabilizationLimit)
   if (currentEfficiency > mean + stddev)
      flag_anomaly(ABNORMAL_GOOD)
   if (currentEfficiency < mean - stddev)
      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

S: StatisticsADPI only, E: EnergyEfficiencyADPI only, S&E: both.

	Time	**Abnormal phases**			Activities	**Activities stored**		
		S	E	S&E		S	E	S&E
Avg.	553 s	0.3 %	5.8 %	39.4 %	15,297	6.9 %	12.7 %	11.3 %
Min	552 s	0.2 %	4.6 %	3.0 %	15,297	6.7 %	6.5 %	6.5 %
Max	553 s	0.4 %	7.2 %	87.7 %	15,297	7.4 %	18.4 %	18.4 %

Summary

- 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