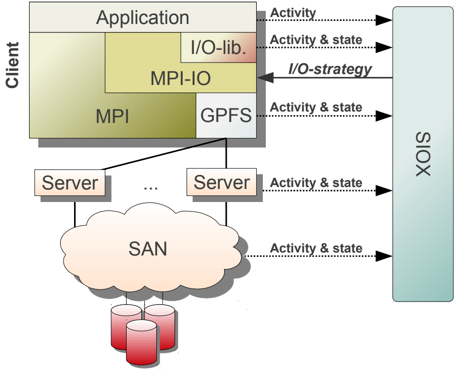
Monitoring and Optimization of I/O performance with SIOX

Julian Kunkel[1] Michaela Zimmer[1] Marc Wiedemann[1]
Alvaro Aguilera[2] Holger Mickler[2]
Xuan Wang[3] Andriy Chut[3] Thomas Bönisch[3]
Jakob Lüttgau[1] Roman Michel[1] Johann Weging[1] Daniela Koudela[2]

1 University of Hamburg
2 ZIH Dresden
3 HLRS Stuttgart

Sept. 5 – 2013, Gauß-Allianz

Project Goals



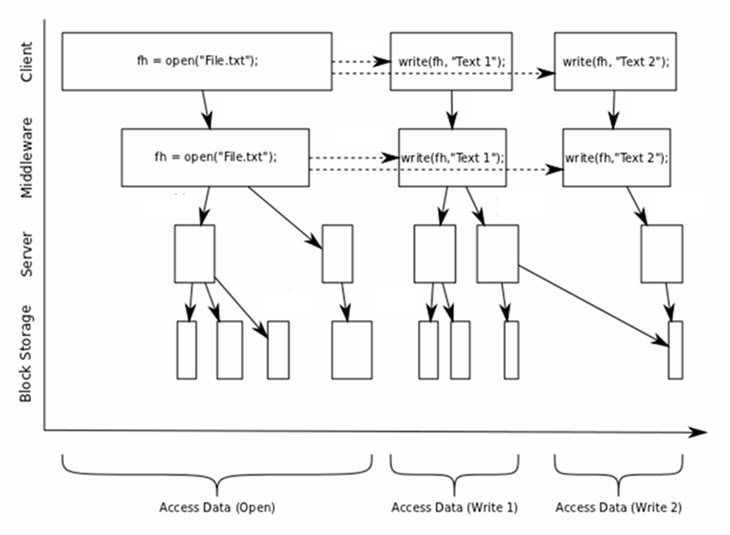
SIOX will

- collect and analyse
  - activity patterns and
  - performance metrics

in order to

- assess system performance
- locate and diagnose problem
- learn optimizations

Example Cause-and-Effect Chain



Partners and Funding

DKRZ
DEUTSCHES KLIMARECHENZENTRUM

Bundesministerium
für Bildung
und Forschung

H L R | S

IBM

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Funded by the BMBF
  Grant No.: 01 IH 11008 B
- Start: Juli 1st, 2011
- Duration: 36 Months

ZIH
Zentrum für Informationsdienste
und Hochleistungsrechnen

SIOX

Increasing SIOX's Capabilites

The project proposal targets instrumentation of

Selected applications, MPI-IO, POSIX and GPFS.

We designed a flexible modular architecture and API.

This allows us to integrate arbitrary:

- Libraries
- File systems
- Hardware information and statistics

An individual instantiation of modules is also possible:

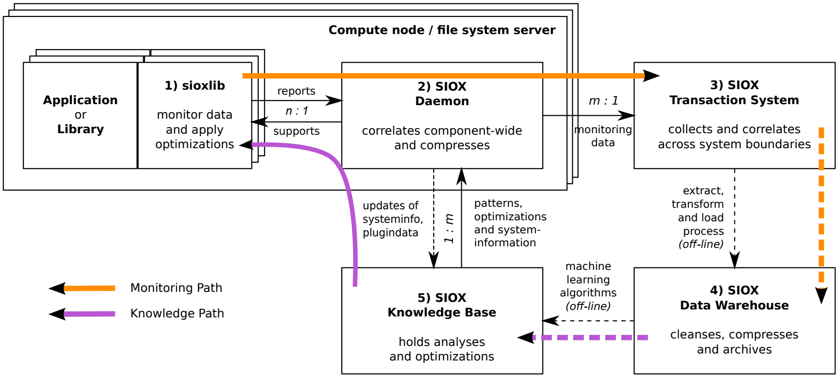
- Decide which modules are configured:
  - For each layer, process, compute and server nodes
- Each module has an XML configuration

Industry Collaboration

- We collaborate to develop industry-relevant software
  - Collaborating companies: Xyratex, Netapp
- We aim to deliver components to the Exascale10 (EIOW) middleware
  - Monitoring, machine learning, automatic optimization

- SIOX is licensed under LGPL3

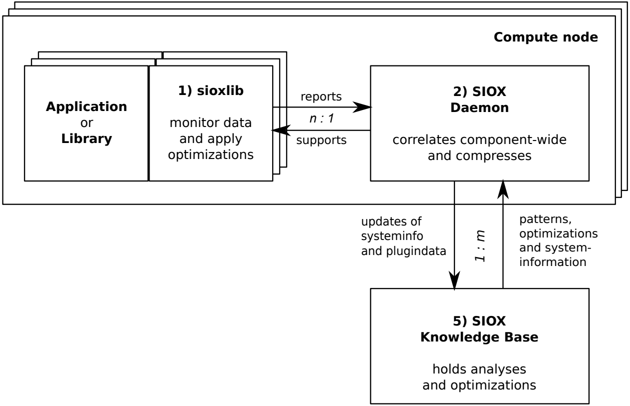
Faces of SIOX (1): General System Architecture



- Data gathered is stored via the *monitoring path*.
- Components receive the knowledge gleaned via the *knowledge path*.

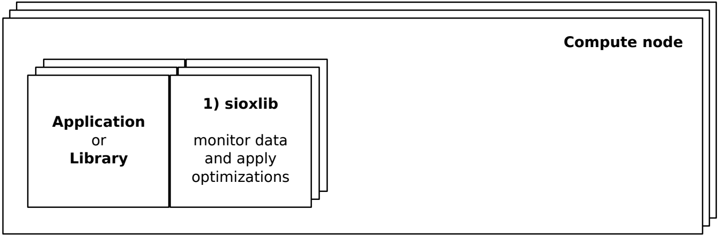
Faces of SIOX (2): Configuration for Online Mode

No pattern recording, optimization without machine learning



Faces of SIOX (3): Configuration for Static Knowledge

Apply static best-practices with low overhead



Overview of Concepts and Mechanisms

- User-level monitoring API
  - "Wrapper" to ease instrumentation of software layers
- Relation of activities
  - Implicit linking of process-internal activities
  - Explicit linking to remote activities
  - Explicit links are created during the ETL into the warehouse
- Analysis of observed activities and statistics by plug-ins
  - Synchronous and/or asynchronous
  - Activities can be handled stateful (within a process) or stateless
  - May use (static) system information/knowledge
- Incorporation of system knowledge
  - One database entry per node, file system, storage device
  - Plugins may create their own node/fs/device specific entries
  - Detect hardware changes (upon startup)
- Local and global "reasoning" to assess system state

Semi-Automatic Instrumentation of Software-Layers

Workflow

1. Save relevant function prototypes in a header file
2. Annotate functions in the header
3. Tool parses header and creates either
   - a shared library for `LD_PRELOAD`
   - a library to use with `ld --wrap`

Instrumentation can be done incrementally

Example Header for POSIX

```
//@component "POSIX"

//@register_metric fileName "File Name"
    ↪ SIOX_STORAGE_STRING
/////// END GLOBAL SECTION ////////////////

//@activity
//@activity_attribute fileName pathname
//@horizontal_map_put_int ret
//@error ''ret < 0'' errno
int open(const char *pathname, int flags, mode_t mode);

//@activity
//@activity_attribute bytesToWrite count
//@activity_link_int fd
//@error ''ret < 0'' errno
ssize_t write(int fd, const void *buf, size_t count);
```

Putting the Knowledge to Good Use

SIOX will exploit the knowledge gleaned to

- control available I/O optimizations

Internally, we will use it to

- adapt own level of activity to the host system's state
- reduce the amount of data logged
- focus and guide its acquisition of new data (active learning)

Putting the Knowledge to Good Use

SIOX will exploit the knowledge gleaned to

- control available I/O optimizations

Internally, we will use it to

- adapt own level of activity to the host system's state
- reduce the amount of data logged
- focus and guide its acquisition of new data (active learning)

Why can't we capture all events?

The Data Deluge – A Numerical Example

Assume program writes a 1 GB file to a parallel file system. . .

- . . . of 100 I/O servers managing 5,000 storage devices
- ⇒ 200 KB per device to write. . .
- . . . writing 4 KB per block on device
- ⇒ 250,000 blocks to write. . .
- . . . logging 20 B per block written
- ⇒ 5 MB logging data
- ⇒ *0.5 % logging overhead.*

The Data Deluge – A Numerical Example, Continued

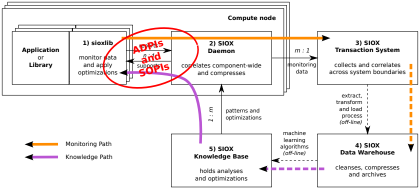
The HPC Cluster *Blizzard* at DKRZ reads and writes...

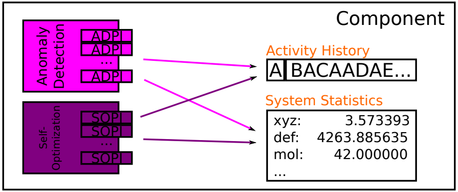
- ...10 GB/s, 24/7, 365 days a year
- $\Rightarrow$ 50 MB/s to log for SIOX
- $\Rightarrow$ 1.6 PB/a...
- ...at a very conservative estimate!

Intelligent Components

Each layer and daemon may use:

- Plug-ins to detect exceptional behavior and steer logging
- Plug-ins to suggest possible optimizations





Building SIOX's Brain

To harness the data gathered, SIOX uses *Knowledge Packages*.

A Knowledge Package. . .

consists of

- a Machine Learning Plug-In

and corresponding plugins

- Anomaly Detection Plug-In
- Self-Optimization Plug-In

Knowledge Package may use private *Action Tables* in the Knowledge Base.

The MLPI will create (and possibly update) the action table, which may also be done manually.

SOPI Example for `posix_fadvise()`

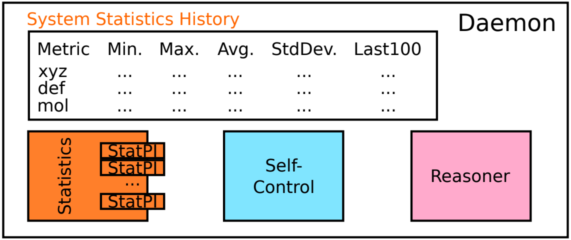
A more complex Action Table: Injecting non-functional calls

Action table

Pattern	Response
SequentialRead() SequentialRead() SequentialRead()	seq & willneed(size)
Open(ext = "nc")	willneed(0, 20 KiB)
Open(ext = "dat")	noReuse & random
RandomWrite(size < 4K){5x}	noReuse & random

A plug-in may use a state machine to track monitored activities

The SIOX Daemon

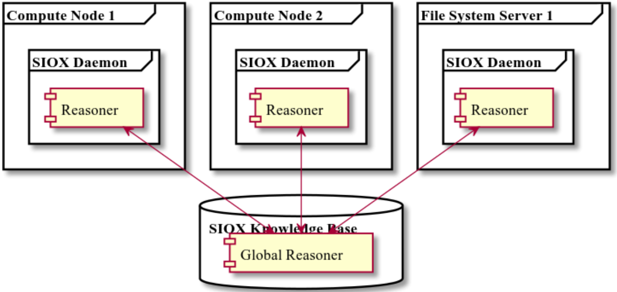


A physical node's daemon holds:

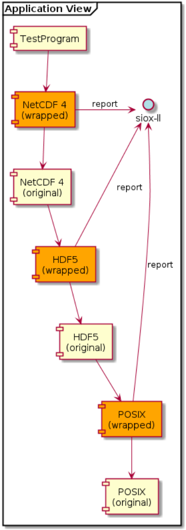
- Recent node-local system statistics, updated regularly
- A module with plug-ins to provide node-local system statistics
- A rule-based reasoner classifying system-state and bottlenecks
- A module with plug-ins to control SIOX's behavior

Reasoning

- Node-local reasoner decides when and how long to log
- System-state, detected bottlenecks and reasons are communicated
  - E.g. "Server overloaded", "Bad I/O pattern"
  - Refined knowledge is transferred to a global reasoner
  - Overview is communicated to all daemons
- Each reasoner maintains statistics for later investigation
- Feedback to user upon application termination



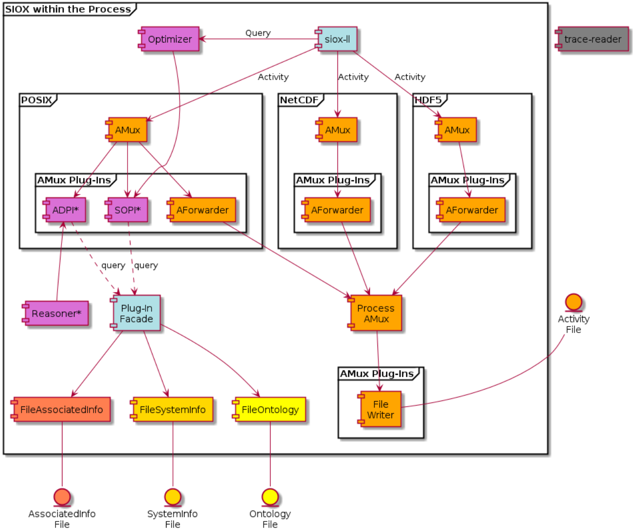
Currently Working Prototype



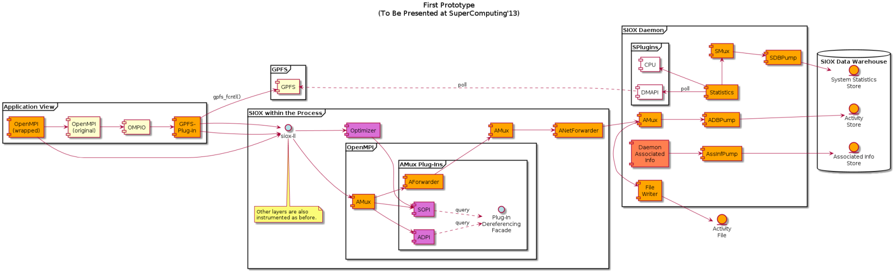
- SIOX manages data within a single process
- Full instrumentation for POSIX
- Partial instrumentations for NetCDF and HDF5

- Application behavior can be recorded in files
- Ontology and system information is stored in files
- Trace-reader parses all files

Currently Working Prototype – Internals



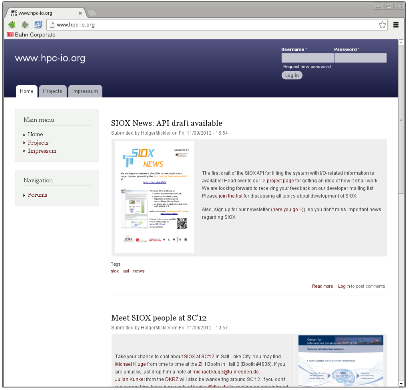
Towards a Prototype for SC'13



Summary

- SIOX aims to capture and optimize I/O
  - on all layers and filesystems
- Intelligent filtering reduces log size
- Integrated reasoning tries to localize causes and bottlenecks
- **We are building a flexible and open system**

Finally: SIOX and You



- Think we missed a problem?
- Think you could solve one?
- Like to see SIOX on your favourite file system?

We cordially invite you to become involved at

http://www.HPC-IO.org

Backupslides

SOPI Example 1

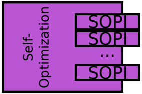
A simple Action Table: Adjusting a system parameter

Action table for an SOPI write-behind plug-in

Pattern	Buffer Size
Open()	4 MiB
Write(size $<$ 2 KiB){5x}	1 MiB
Write(size $<$ 4 MiB) Write(size $<$ 4 MiB)	20 MiB
Write(size $\geq$ 100 MiB)	direct-write

Self-Optimization Plug-Ins

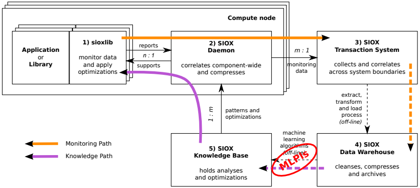
What SOPIs can do:



- Take any action available via any interface accessible, e.g.:
- Adjust system parameters (cache size, MPU size,. . .)
- Inject non-functional calls (`fadvise()`, MPI hints,. . .)
- Inject "housekeeping" calls (`flush()`, refresh page,. . .)
- Adjust parameters of functional calls (access mode, optional caching,. . .)
- Select between alternative modules to employ (MPICH2 vs. OpenMPI, shmem vs. TCP,. . .)

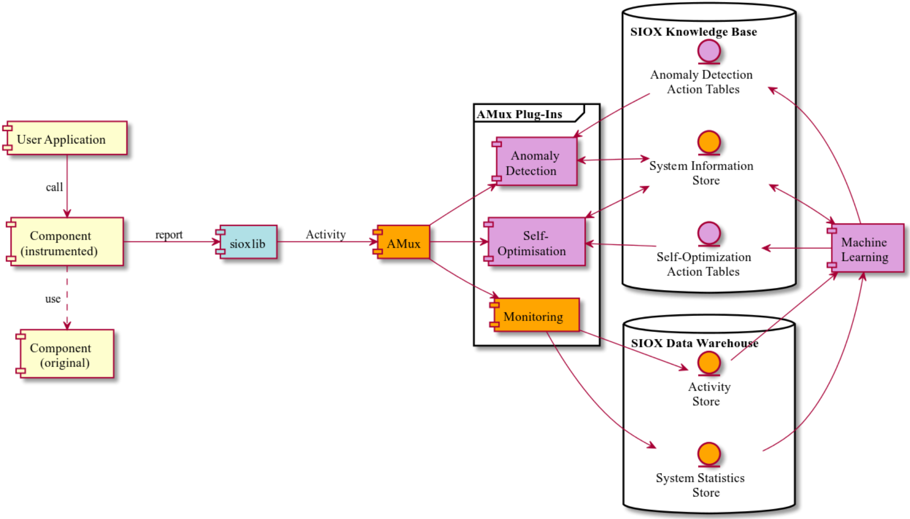
SOPIs are controlled via *Action Tables* in the Knowledge Base.

Machine Learning Plug-Ins (MLPIs)

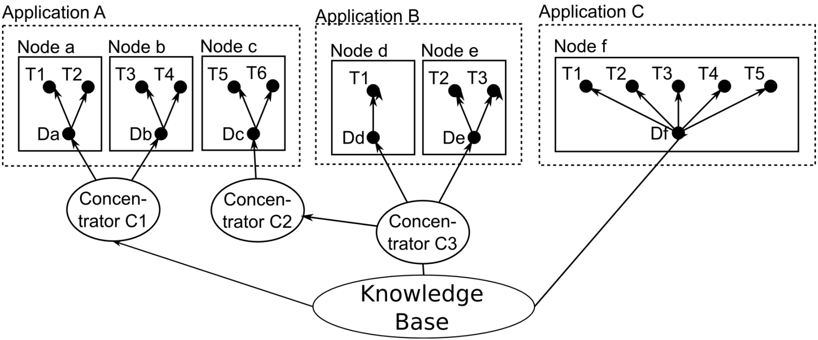


- Run off-line, on-demand
- Analyse data archived in the Data Warehouse with machine learning algorithms or heuristics
- Refine their findings into instructions for their ADPI and/or SOPI
- Store these in a plug-in-specific "action table" in the Knowledge Base
- Action tables may also be built manually (usually to implement common heuristics)

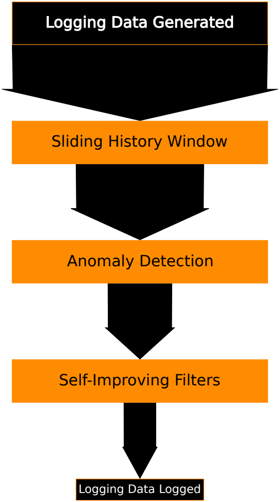
Interplay Between Monitoring and Knowledge Path (1)



Scalability through Hierarchical Data Transport



Damming in the Data Deluge



Logging

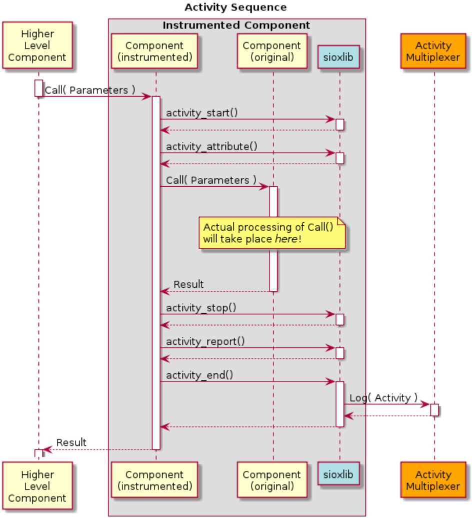
- *every* Activity
- on *every* Level
- *all* of the time

will cause prohibitive overhead.

The SIOX solution:

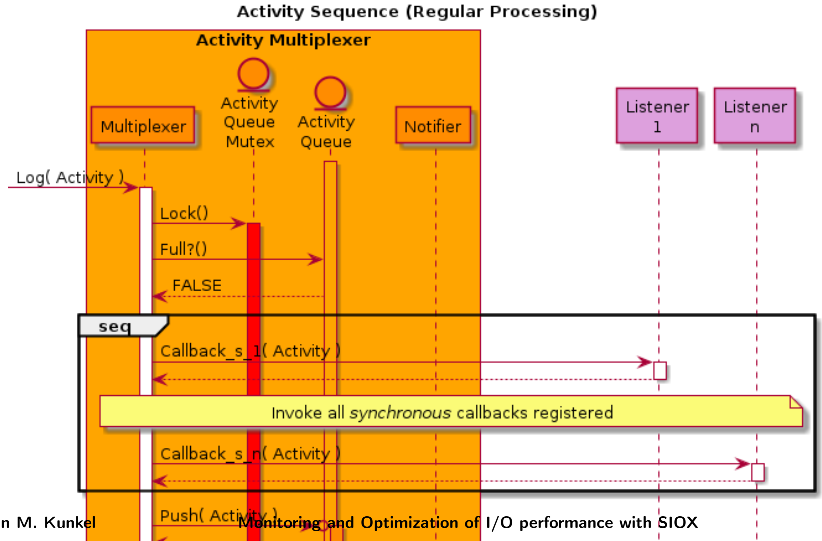
- Log to local window, discard the uneventful
- Focus on anomalies (good & bad)
- Learn from logs to improve filters

Instrumentation and the Activity Multiplexer



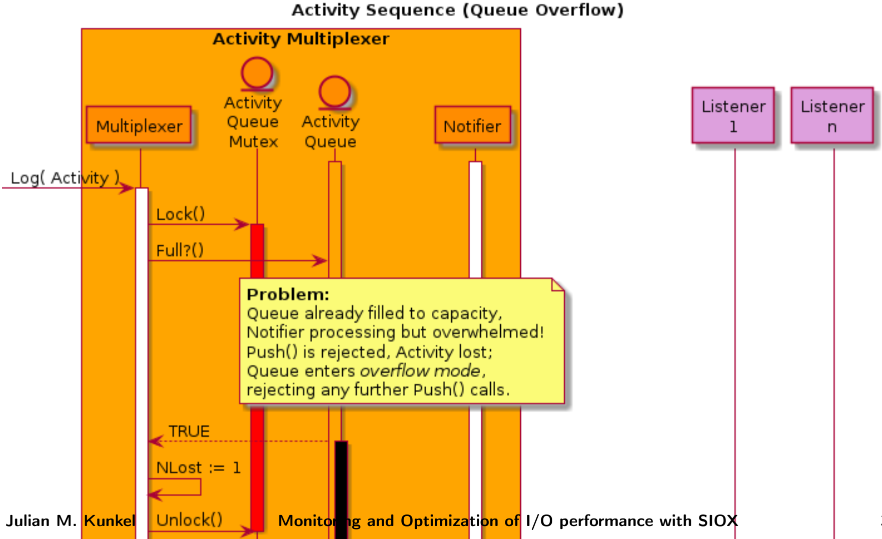
Activity Multiplexer Normal Behavior

Click to access the PNG of the design



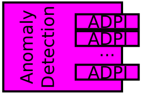
Activity Multiplexer Throttling (Overflow) Behavior

Click to access the PNG of the design



Anomaly Detection Plug-Ins

What ADPIs can do:



- Detect anomalies (exceptional system behaviour)
- Adjust log-levels
- Report recent activity history for analysis
- Mark recent activity sequence for future "watchlist":
  $\Rightarrow$ Timely adjustments to log-levels *before* anomaly recurs
- Make every byte logged count!

ADPIs are controlled via *Action Tables* in the Knowledge Base.

ADPI Example 1

A simple rule-based and stateless plug-in detecting exceptional performance

Mathematical model and Action Table

$$f_{\text{Utilization}}(\text{Component}, \text{Activity}) = \frac{t_{\text{possible}}(\text{Component}, \text{Activity})}{\text{Time}(\text{Activity})}$$

$$t_{\text{possible}} = \frac{\text{Size}(\text{Activity})}{\text{SequentialTransferRate}(\text{Component})} + \text{Latency}(\text{Component})$$

Result	Action
$f_{\text{Utilization}} < 0.10$	Report("Exceptionally low")
$0.10 < f_{\text{Utilization}} < 0.95$	No Action
$0.95 < f_{\text{Utilization}}$	Report("Exceptionally high")

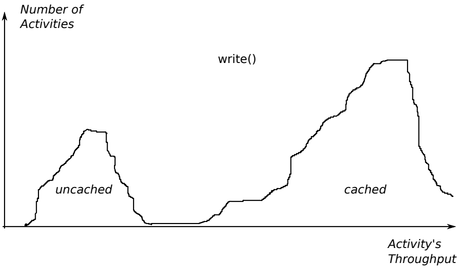
"Component" can be a software layer, a compute node, or a file system.

ADPI Example 2

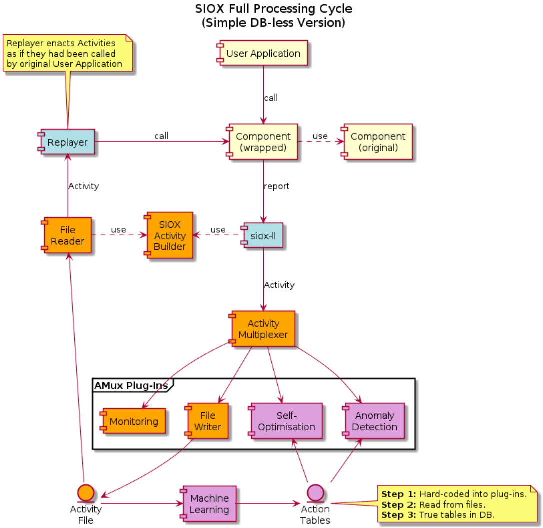
A very simple Action Table implementing a "watchlist"

Action table to detect a write cache filling up

Pattern	Buffer Size
Write(fast) Write(slow)	Report("Cache full")



Prototype (2)



- Application behavior can be recorded in files
- Activities and their metrics are read from files
- Replayer to mimic program behavior
- Machine learning restricted to parameters in heuristics