Towards Intelligent Self-Optimisation in HPC I/O

Julian Kunkel  Michaela Zimmer  Marc Wiedemann

University of Hamburg

June 20, ISC ’13
1 Introduction

2 Architecture

3 Intelligent I/O-Handling

4 Summary
SIOX will

- collect and analyse
  - activity patterns and
  - performance metrics

in order to

- assess system performance
- locate and diagnose problems
- learn optimizations
Bundesministerium für Bildung und Forschung

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

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

1) sioxlib
monitor data and apply optimizations
Architecture

1) sioxlib
   - monitor data
   - apply optimizations

2) SIOX Daemon
   - correlates component-wide
   - compresses

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

Julian Kunkel (Univ. of Hamburg)  Self-Optimisation in HPC I/O  June 20, ISC ’13
2) SIOX Daemon correlates component-wide and compresses.

3) SIOX Transaction System collects and correlates across system boundaries.

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

2) SIOX Daemon correlates component-wide and compresses.

3) SIOX Transaction System collects and correlates across system boundaries.

4) SIOX Data Warehouse cleanses, compresses and archives.

Compute node

1) sioxlib
monitor data and apply optimizations

Application or Library

reports

supports

monitoring data

extract, transform and load process (off-line)

Data gathered is stored via the monitoring path.

Components receive the knowledge gleaned via the knowledge path.
- Data gathered is stored via the *monitoring path*. 
Data gathered is stored via the *monitoring path*.
Data gathered is stored via the *monitoring path*.
Components receive the knowledge gleaned via the *knowledge path*.
Alternative Architecture Configuration: Online-Mode

1) sioxlib
   - Monitor data and apply optimizations

2) SIOX Daemon
   - Correlates component-wide and compresses

5) SIOX Knowledge Base
   - Holds analyses and optimizations

Configuration is loaded upon startup and initializes modules
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 between remote activities
  - Link is created while transferring data to the warehouse
- Observed activities and statistics are processed by multiple plugins
  - Synchronous and/or asynchronous
  - Activities can be handled statefull (within a process) or stateless
  - May use (static) system information/knowledge
  - Usage: Learning of optimizations, intelligent logging, own overhead
- 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. Saving 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

```c
// @component "POSIX"

// @register_descriptor 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, ...);

// @activity
// @activity_attribute bytesToWrite count
// @activity_link_int fd
// @error "ret < 0" errno
ssize_t write(int fd, const void *buf, size_t count);
```
Intelligent I/O-Handling

Logical View of the Monitoring Path

Physical Node

Layer 3
- Component 3 (original)
- Component 3 (instrumented)
  - use
  - report
  - sioxlib
  - Activity
  - AMux

Layer 2
- Component 2 (original)
- Component 2 (instrumented)
  - use
  - report
  - sioxlib
  - Activity
  - AMux

Layer 1
- Component 1 (original)
- Component 1 (instrumented)
  - use
  - report
  - sioxlib
  - Activity
  - AMux

SIOX Daemon
- Statistics
- SMux

SIOX Data Warehouse
- System Statistics Store
- Activity Store
Intelligent I/O-Handling

Intelligent Components

Each component/layer holds:
- Plug-ins to detect exceptional behaviour
- Plug-ins to suggest possible optimizations

Additionally, a daemon holds:
- Recent system statistics, updated regularly
- Statistics plug-ins
- A plugin to control SIOX behavior
- A rule-based reasoner classifies system-state and bottlenecks

Julian Kunkel (Univ. of Hamburg)  Self-Optimisation in HPC I/O  June 20, ISC ’13
To harness the data gathered, SIOX uses *Knowledge Packages*.

A Knowledge Package…

contains of
- a Machine Learning Plug-In
- 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.
Intelligent I/O-Handling

Interplay Between Monitoring and Knowledge Path (1)

Julian Kunkel (Univ. of Hamburg)

Self-Optimisation in HPC I/O

June 20, ISC ’13
Intelligent I/O-Handling

Interplay Between Monitoring and Knowledge Path (2)

Click to access the fullpage PDF of the design
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“
  - All knowledge to global reasoner
  - Overview is communicated to all daemons
- Global reasoner maintains statistics for later investigation
Anomaly-Detection Plugin Example 1

A simple rule-based and stateless plugin detecting exceptional performance

**Mathematical model and Action Table**

\[
\begin{align*}
    f_{\text{Utilization}}(\text{Component, Activity}) &= \frac{\text{Time(Activity)}}{t_{\text{expected}}(\text{Component, Activity})} \\
    t_{\text{expected}} &= \frac{\text{Size(Activity)}}{\text{SequentialTransferRate(Component)}} + \text{latency(Component)}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Result</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f_{\text{Utilization}} &lt; 0.10)</td>
<td>Report( &quot;Exceptionally low&quot; )</td>
</tr>
<tr>
<td>0.10 &lt; (f_{\text{Utilization}} &lt; 0.95)</td>
<td>No Action</td>
</tr>
<tr>
<td>0.95 &lt; (f_{\text{Utilization}})</td>
<td>Report( &quot;Exceptionally high&quot; )</td>
</tr>
</tbody>
</table>

Component can be a subset of \{current software layer, compute node, file system\}
A simple Action Table: Adjusting a system parameter

### Action table for an SOPI write-behind plug-in

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Buffer Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Open()</code></td>
<td>4 MiB</td>
</tr>
<tr>
<td><code>Write(size &lt; 2 KiB){5x}</code></td>
<td>1 MiB</td>
</tr>
<tr>
<td><code>Write(size &lt; 4 MiB)</code></td>
<td>20 MiB</td>
</tr>
<tr>
<td><code>Write(size &lt; 4 MiB)</code></td>
<td></td>
</tr>
<tr>
<td><code>Write(size ≥ 100 MiB)</code></td>
<td>direct-write</td>
</tr>
</tbody>
</table>
A more complex Action Table: Injecting bespoke non-functional calls

### Action table for an SOPI `fadvise()` plug-in

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Advice</th>
</tr>
</thead>
<tbody>
<tr>
<td>SequentialRead() SequentialRead() SequentialRead()</td>
<td>seq &amp; willneed(size)</td>
</tr>
<tr>
<td>Open(ext = &quot;nc&quot;)</td>
<td>willneed(0, 20KiB)</td>
</tr>
<tr>
<td>Open(ext = &quot;dat&quot;)</td>
<td>noReuse &amp; random</td>
</tr>
<tr>
<td>RandomWrite(size &lt; 4K){5x}</td>
<td>noReuse &amp; random</td>
</tr>
</tbody>
</table>
Application behavior can be recorded in files
Activities and their metrics read from files
Replayer to mimic program behavior
Machine learning restricted to parameters in heuristics
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
Backupsides
Scalability through Hierarchical Data Transport
A program writes a 1 GiB file to a parallel file system...

- ...of 100 I/O servers managing 5,000 storage devices
- \Rightarrow 200 \text{ KiB} per device to write...
- ...writing 4 KiB per block on device
- \Rightarrow 250,000 blocks to write...
- ...logging 20 B per block written
- \Rightarrow 5 \text{ MiB} logging data
- \Rightarrow 0.5 \% logging overhead...
The HPC Cluster *Blizzard* at DKRZ reads and writes…

- 10 GiB/s, 24/7, 365 days a year
- ⇒ 50 MiB/s to log for SIOX
- ⇒ 1,576 PiB/a logging information
Instrumentation and the Activity Multiplexer
Activity Multiplexer Normal Behavior

Click to access the PNG of the design

Activity Sequence (Regular Processing)

Activity Multiplexer

Listener 1
Listener n

Log( Activity )
Lock()
Full?()
FALSE

seq
Callback_s_1( Activity )

Callback_s_n( Activity )
Push( Activity )

Invoke all **synchronous** callbacks registered
Activity Multiplexer Throttling (Overflow) Behavior

Click to access the PNG of the design

Activity Sequence (Queue Overflow)

Activity Multiplexer

Log( Activity )

Lock()

Full?()

TRUE

NLost := 1

Problem:
Queue already filled to capacity, Notifier processing but overwhelmed!
Push() is rejected, Activity lost;
Queue enters overflow mode, rejecting any further Push() calls.

Listener 1

Listener n