The SIOX Architecture – Coupling Automatic Monitoring and Optimization of Parallel I/O

Julian Kunkel[1] Michaela Zimmer[1] Nathanael Hübbe[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]

1 University of Hamburg
2 ZIH Dresden
3 HLRS Stuttgart

June 26th – 2014, ISC

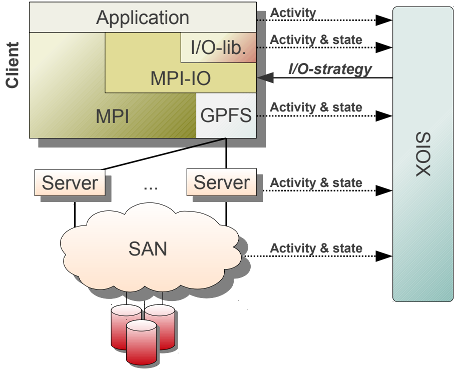
Outline

1. Introduction
2. The Modular Architecture of SIOX
3. Analysis and Visualization of I/O
4. Experiments
5. Summary

Outline

1. Introduction
2. The Modular Architecture of SIOX
3. Analysis and Visualization of I/O
4. Experiments
5. Summary

Project Goals



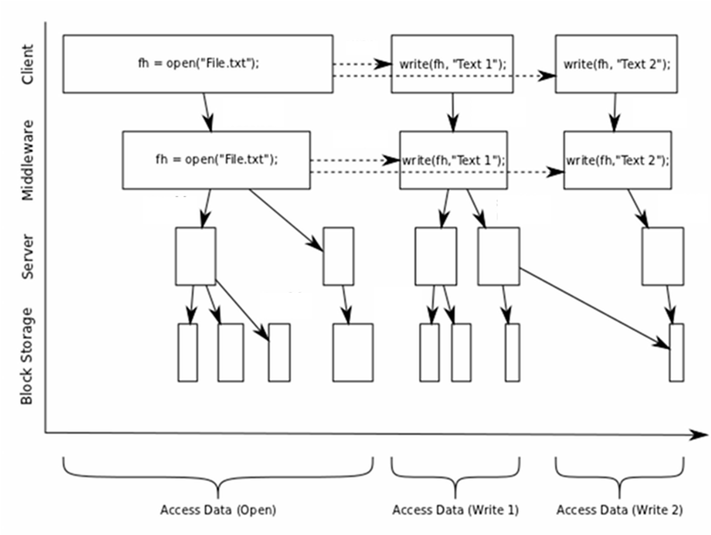
SIOX will

- collect and analyse
  - activity patterns and
  - performance metrics

in order to

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

Activity Patterns: Example Cause-and-Effect Chain



Partners and Funding

Bundesministerium
für Bildung
und Forschung

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

DKRZ
DEUTSCHES
KLIMARECHENZENTRUM

HLRS

IBM

UHH Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

ZIH
Zentrum für Informationsdienste
und Hochleistungsrechnen

Outline

1. Introduction

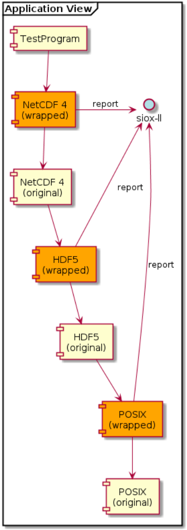
2. The Modular Architecture of SIOX
   - Low-Level API
   - Instrumentation
   - Faces of SIOX
   - Modules

3. Analysis and Visualization of I/O

4. Experiments

5. Summary

Low-Level API – Overview and Instrumentation



- C-Interface for monitoring / analysis
  - Monitor activities and system statistics
  - Query suitable optimization
- Relies on modules to
  - store activities
  - store and query (ontology and system) information
- Instrumentation uses low-level-API
  - A tool and workflow is provided; already instrumented:
  - POSIX (stdio and low-level) and MPIIO
  - Initial instrumentations of NetCDF and HDF5

Modularity of SIOX

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

Instrumentation

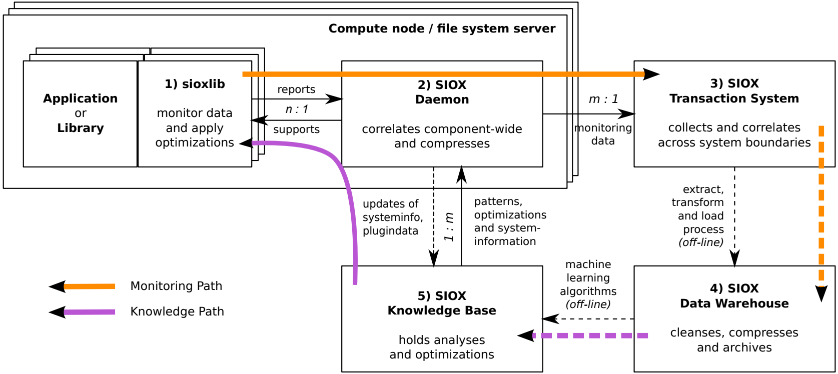
Workflow

- Annotation of header file
- Tool `siox-wrapper-generator` creates libraries
  - Run-time instrumentation with LD_PRELOAD
  - Compile-time instrumentation using `ld -wrap`
- `siox-inst` tool simplifies instrumentation

Header annotations for `MPI_File_write_at()`

```
//@activity
//@activity_link_size fh
//@activity_attribute filePosition offset
//@splice_before ’’int intSize; MPI_Type_size(datatype, &intSize);
                   uint64_t size=(uint64_t)intSize*(uint64_t)count;’’
//@activity_attribute bytesToWrite size
//@error ’’ret!=MPI_SUCCESS’’ ret
int MPI_File_write_at(MPI_File fh, MPI_Offset offset, void * buf, int count,
                      MPI_Datatype datatype, MPI_Status * status);
```

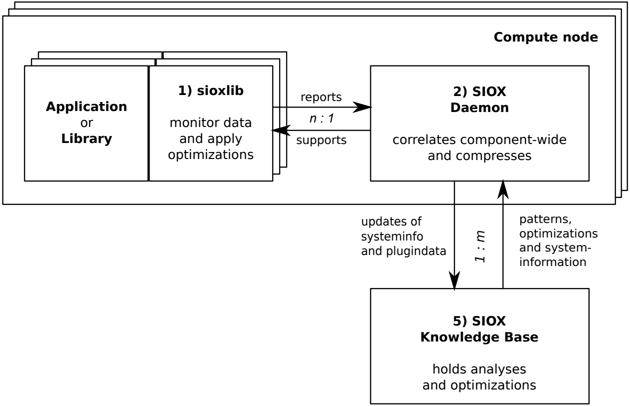
Faces of SIOX (1): General System Architecture



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

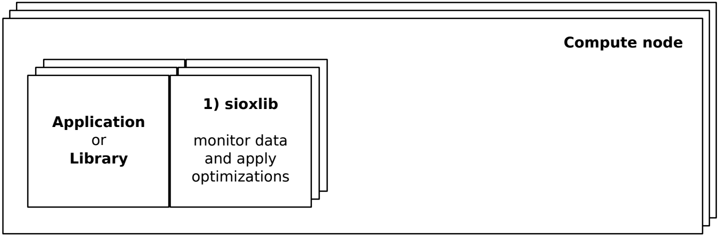
Faces of SIOX (2): Configuration for Online Mode

No pattern recording, optimization without machine learning



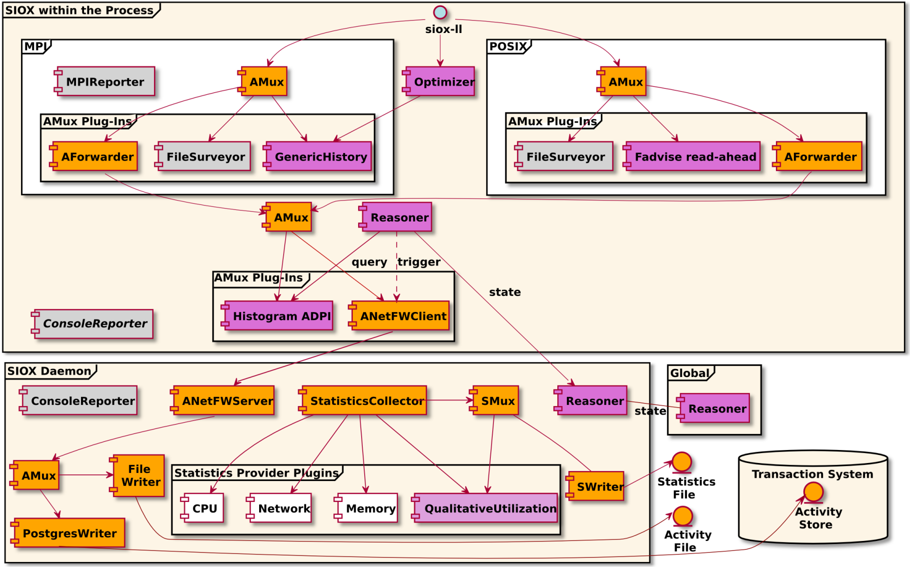
Faces of SIOX (3): Configuration for Static Knowledge

Apply static best-practices with low overhead.



A configuration with a node-global daemon is also possible

Module Interactions of an Example Configuration



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 plugins
  - *GenericHistory* plugin tracks performance, proposes MPI hints
  - Fadvise (ReadAhead) injector
  - *FileSurveyor* prototype – Darshan like
- Reasoner component (with simple decision engine)
  - Intelligent monitoring: trigger monitoring on abnormal behavior
- Reporting of statistics on console or file (independent and MPI aware)

Outline

1. Introduction
2. The Modular Architecture of SIOX
3. Analysis and Visualization of I/O
   - Trace Reader
   - Database GUI
   - Reporting
4. Experiments
5. Summary

Trace Reader

Concepts

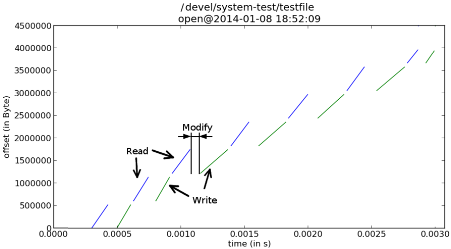
- Supports different file and database back-ends
- Plugin based
  - Text output
  - time-offset plots for files

Example text output created by the trace-reader

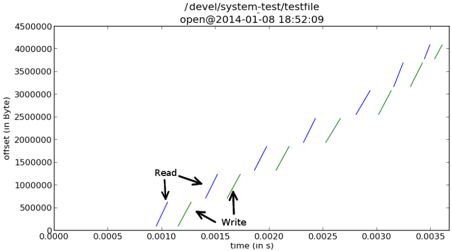
```
0.0006299 ID1 POSIX open(POSIX/descriptor/filename="testfile",
  POSIX/descriptor/filehandle=4) = 0
0.0036336 ID2 POSIX write(POSIX/quantity/BytesToWrite=10240,
  POSIX/quantity/BytesWritten=10240, POSIX/descriptor/filehandle=4,
  POSIX/file/position=10229760) = 0 ID1
0.0283800 ID3 POSIX close(POSIX/descriptor/filehandle=4) = 0 ID1
```

Trace Reader Plugin: AccessInfoPlotter

- Plot for each file and rank information about accessed data
- Example: non-contiguous MPI I/O to a shared file by 2 processes
  - Reveal underlying POSIX access pattern
  - Read-Modify-Write cycle of data-sieving



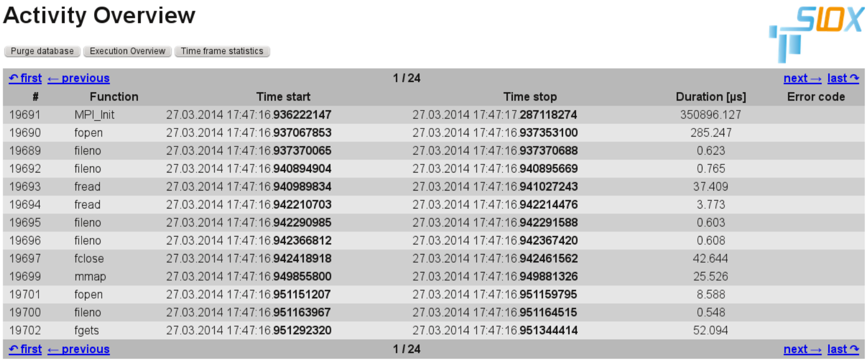
(a) Rank 0



(b) Rank 1

Database GUI

- A PHP GUI provides access to the Postgres DB
- Overview of applications, activities, chain-of-effects



Activity Overview

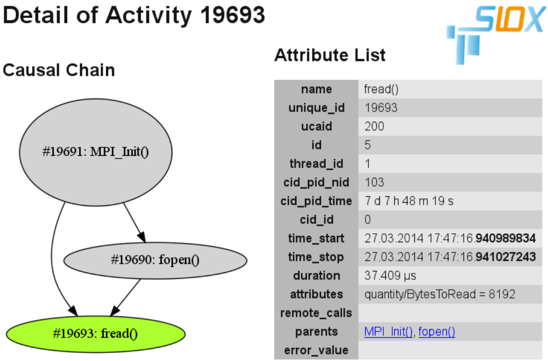
↶ first ← previous   1 / 24   next → last ↷

#	Function	Time start	Time stop	Duration [µs]	Error code
19691	MPI_Init	27.03.2014 17:47:16.**936222147**	27.03.2014 17:47:17.**287118274**	350896.127	
19690	fopen	27.03.2014 17:47:16.**937067853**	27.03.2014 17:47:16.**937353100**	285.247	
19689	fileno	27.03.2014 17:47:16.**937370065**	27.03.2014 17:47:16.**937370688**	0.623	
19692	fileno	27.03.2014 17:47:16.**940894904**	27.03.2014 17:47:16.**940895669**	0.765	
19693	fread	27.03.2014 17:47:16.**940989834**	27.03.2014 17:47:16.**941027243**	37.409	
19694	fread	27.03.2014 17:47:16.**942210703**	27.03.2014 17:47:16.**942214476**	3.773	
19695	fileno	27.03.2014 17:47:16.**942290985**	27.03.2014 17:47:16.**942291588**	0.603	
19696	fileno	27.03.2014 17:47:16.**942366812**	27.03.2014 17:47:16.**942367420**	0.608	
19697	fclose	27.03.2014 17:47:16.**942418918**	27.03.2014 17:47:16.**942461562**	42.644	
19699	mmap	27.03.2014 17:47:16.**949855800**	27.03.2014 17:47:16.**949881326**	25.526	
19701	fopen	27.03.2014 17:47:16.**951151207**	27.03.2014 17:47:16.**951159795**	8.588	
19700	fileno	27.03.2014 17:47:16.**951163967**	27.03.2014 17:47:16.**951164515**	0.548	
19702	fgets	27.03.2014 17:47:16.**951292320**	27.03.2014 17:47:16.**951344414**	52.094	

↶ first ← previous   1 / 24   next → last ↷

Activity list showing I/O function and timestamps.

Database GUI



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 activity showing the causal chain and list of attributes.

Reporting: FileSurveyor

- Easy to collect and track relevant application statistics
- FileSurveyor prototype collects POSIX/MPI access statistics
- 1000 LoC
- ... *Yes we'll pretty print things at some point* ...

```
[...] "(Aggregated over all files)"/Accesses = (40964,40964,40964)
...
[...] "/mnt/lustre/file.dat"/Accesses = (40964,40964,40964)
[...] "/mnt/lustre/file.dat"/Accesses/Reading/Random, long seek = (20481.8,20480,20482)
[...] "/mnt/lustre/file.dat"/Accesses/Reading/Random, short seek = (0,0,0)
[...] "/mnt/lustre/file.dat"/Accesses/Reading/Sequential = (0.2,0,2)
[...] "/mnt/lustre/file.dat"/Bytes = (8.38861e+09,8.38861e+09,8.38861e+09)
[...] "/mnt/lustre/file.dat"/Bytes/Read per access = (204780,204780,204780)
[...] "/mnt/lustre/file.dat"/Bytes/Total read = (4.1943e+09,4.1943e+09,4.1943e+09)
[...] "/mnt/lustre/file.dat"/Seek Distance/Average writing = (1.0238e+06,1.0238e+06,1.02382e+06)
[...] "/mnt/lustre/file.dat"/Time/Total for opening = (3.9504e+08,3.66264e+08,4.38975e+08)
[...] "/mnt/lustre/file.dat"/Time/Total for reading = (1.47169e+11,1.0968e+11,1.76617e+11)
[...] "/mnt/lustre/file.dat"/Time/Total for writing = (1.08783e+12,1.03317e+12,1.16192e+12)
[...] "/mnt/lustre/file.dat"/Time/Total for closing = (1.0856e+11,6.11782e+10,1.46834e+11)
[...] "/mnt/lustre/file.dat"/Time/Total surveyed = (1.34568e+12,1.34568e+12,1.3457e+12)
```

Example report created by FileSurveyor and aggregated by MPIReporter (shortened excerpt). The number format is (average, minimum, maximum).

Outline

1. Introduction

2. The Modular Architecture of SIOX

3. Analysis and Visualization of I/O

4. Experiments
   - System Configuration
   - Overhead
   - Parabench I/O Benchmark
   - Injection of “I/O-Hints”

5. Summary

System Configuration

Test system

- 10 compute nodes
- 10 I/O nodes with Lustre

Compute Nodes

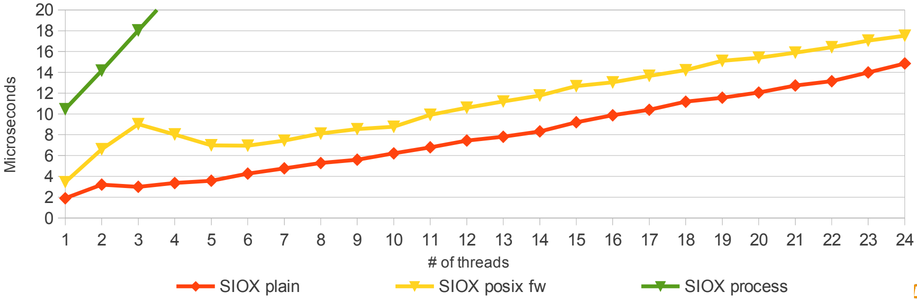
- Dual-socket Intel Xeon X5650@2.67 GHz
- Ubuntu 12.04
- Applications are compiled with: GCC 4.7.2, OpenMPI 1.6.5

I/O Nodes

- Intel Xeon E3-1275@3.4 GHz, 16 GByte RAM
- Seagate Barracuda 7200.12 (ca. 100 MiB/s)
- CentOS 6.5, Lustre 2.5

Overhead

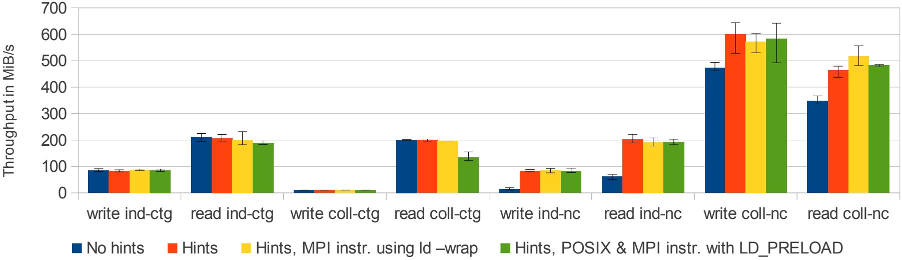
- Due to asynchronous handling applications are never stalled
- A call to SIOX in the order of several μs
  - We see a potential for improving!
- Initialization of SIOX with fixed costs
- SIOX IPC handles 90,000 (1 KiB) msgs per second
- PostgreSQL only 3,000 activities (we'll need to invest more time)



Overhead per thread due to critical regions in the modules.

MPI 4-levels-of Access

- Each process accesses 10240 blocks of 100 KiB
- Several hint sets are evaluated



Performance comparison of the 4-levels of access on our Lustre file system. The hints increase the collective buffer size to 200 MB and disables data sieving.

Observations

- Note: SIOX could inject the proper hints (for nc) for performance
- Overhead in read coll-ctg due to instrumentation of network!

Optimization Plugin: Read-Ahead with Fadvise

- Plugin injects `posix_fadvise()` for strided access
- Compute-“Benchmark” reads data, then sleeps
  - 100μs and 10 ms for 20 KiB and 1000 KiB stride, respectively

Results

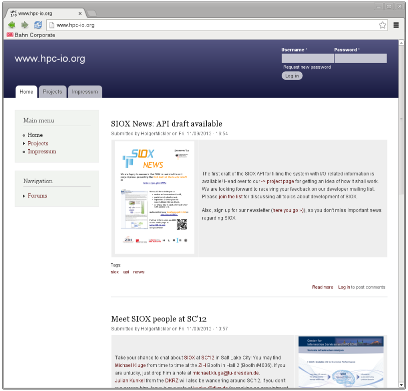
Experiment	20 KiB stride	1000 KiB stride
Regular execution	97.1 μs	7855.7 μs
Embedded fadvise	38.7 μs	45.1 μs
SIOX fadvise read-ahead	52.1 μs	95.4 μs

Time needed to read one 1 KiB data block in a strided access pattern.

Summary

- SIOX aims to capture and optimize I/O
  - on all layers and filesystems
- We analyzed the overhead of the prototype
  - Remembers best MPI hints and sets them
  - Bearable monitoring overhead
  - Flexible configuration
- **We are building a modular and open system**
- **We are looking forward to contributing components to E10**

Finally: SIOX and You



SIOX News: API draft available

Submitted by HolgerMickler on Fri, 11/09/2012 - 16:54

The first draft of the SIOX API for filling the system with I/O-related information is available! Head over to our -> project page for getting an idea of how it shall work. We are looking forward to receiving your feedback on our developer mailing list. Please join the list for discussing all topics about development of SIOX.

Also, sign up for our newsletter (here you go :-)), so you don't miss important news regarding SIOX.

Meet SIOX people at SC'12

Submitted by HolgerMickler on Fri, 11/09/2012 - 10:57

Take your chance to chat about SIOX at SC'12 in Salt Lake City! You may find Michael Kluge from time to time at the ZIH Booth in Hall 2 (Booth #4036). If you are unlucky, just drop him a note at michael.kluge@tu-dresden.de. Julian Kunkel from the DKRZ will also be wandering around SC'12. If you don't run across him, leave him a note at kunkel@dkrz.de for making an appointment

- 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