SIOX: A Flexible Approach

Julian Kunkel Jakob Lüttgau[1]

German Climate Computing Center

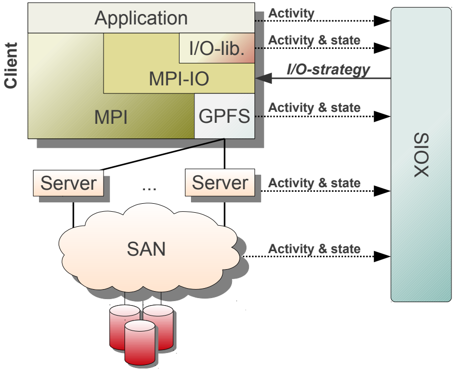
Analyzing Parallel I/O BoF
SC 2015

[1]University of Hamburg

Outline

1. Introduction
2. The Modular Architecture of SIOX
3. Analysis and Visualization of I/O
4. Experiments
5. News and Ongoing R&D
6. Outlook & Summary

Project Goals



SIOX is a flexible prototype for

- collecting and analyzing
  - activity patterns and
  - performance metrics

in order to

- assess system performance
- locate and diagnose problem
- **learn & apply optimizations**
- intelligently steer monitoring

Extensibility for Alternate APIs

Workflow

1. Annotate a header file
2. Tool `siox-wrapper-generator` creates intercepting libraries
   - Run-time instrumentation with LD_PRELOAD
   - Compile-time instrumentation using `ld -wrap`
3. `siox-inst` tool simplifies instrumentation

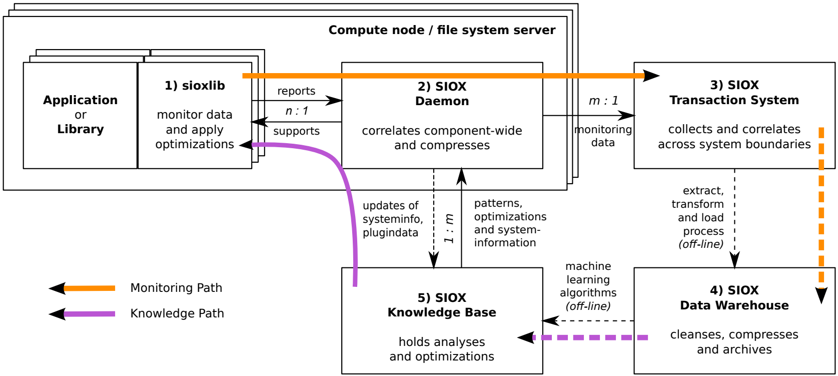
Header annotations for `MPI_File_write_at()`

```
//@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);
```

Modularity of SIOX

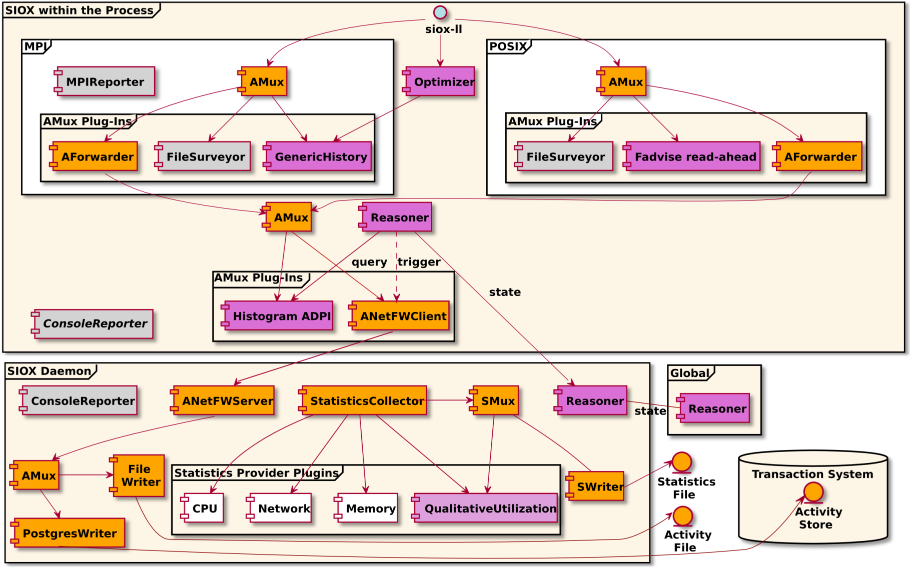
- The SIOX architecture is flexible and developed in C++ components
- License: LGPL, vendor friendly
- Upon start-up of (instrumented) applications, modules are loaded
- Configuration file defines modules and options
  - Choose advantageous plug-ins
  - Regulate overhead
- For debugging, **reports**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

Example Workflow (many are possible)



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

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 plug-ins
  - *GenericHistory* plug-in 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)

Trace Reader

Concepts

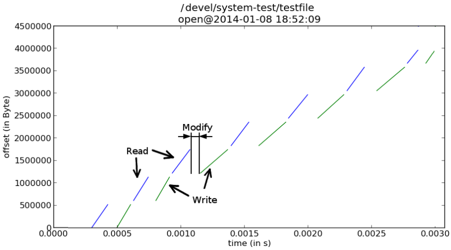
- Supports different file and database back-ends
- Plug-in based
  - Text output
  - Time-offset plots for files

Example text output created by the trace-reader

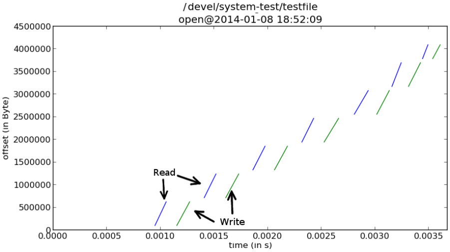
```
0.0006299 ID1 POSIX open(POSIX/descriptor/filename="testfile",
  POSIX/descriptor/filehandle=4) = 0
0.0036336 ID2 POSIX write(POSIX/quantity/BytesToWrite=10240,
  POSIX/quantity/BytesWritten=10240, POSIX/descriptor/filehandle=4,
  POSIX/file/position=10229760) = 0 ID1
0.0283800 ID3 POSIX close(POSIX/descriptor/filehandle=4)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

Trace Reader Plug-in: AccessInfoPlotter

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



(a) Rank 0



(b) Rank 1

Reporting: FileSurveyor

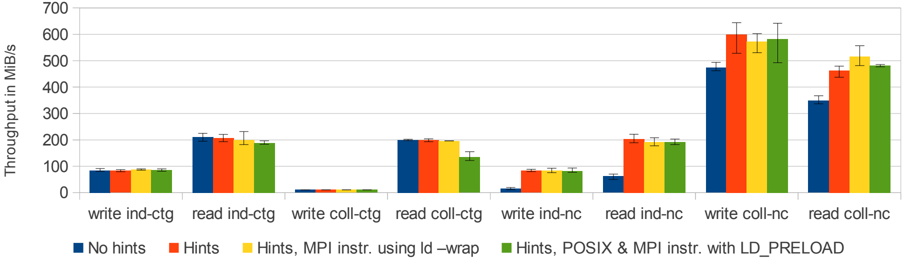
- Easy to collect and track relevant application statistics
- FileSurveyor prototype collects POSIX/MPI access statistics
- Only 1000 LoC
- ... *Yes we'll pretty print things at some point ...*

```
[...] "(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).

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 disable data sieving.

Observations

- GenericHistory could inject the hints automatically for ind-nc cases
- Overhead in read coll-ctg due to instrumentation of network!

Optimization Plug-in: Read-Ahead with Fadvise

- Plug-in injects `posix_fadvise()` for strided access
- vs. no prefetching vs. in code embedded execution
- Compute "Benchmark" reads data, then sleeps
  - 100 μs 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.

Changing I/O Behavior on the Fly

Motivation

- What is the benefit of implementing an I/O optimization in the code?
  - Traditional methodology: (estimate), implement, evaluate

  ⇒ Time consuming!

Alternative strategies

- Trace and record application I/O, then alter and replay I/O
- Intercept I/O and manipulate directly

Pro/Cons

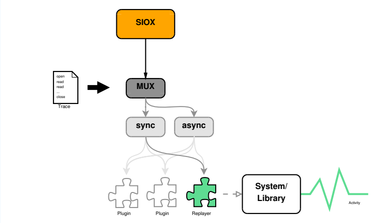
\+ Implement an optimization once, test/run with many applications

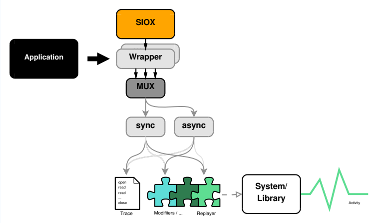
\- Looses some performance due to interception

Modifications to SIOX

- The proposed strategies have been implemented (for a subset of POSIX)
- We extracted the execution of calls from the monitoring path
  - Now, a playback plugin executes calls
  - The same plugins can be used in trace/replay scenarios
  - + This also reduced the complexity of the interception layer

Modification during trace-replay and online-playback





Ongoing Works

- Apply SIOX to more applications on DKRZ's Mistral supercomputer
  - 1.4 (phase 1), 3 PetaFlop/s
  - 45 PetaByte storage
- Improve intelligence
  - Information about predicted storage class (e.g. cached, uncached)
  - Performance predictors for anomaly detection
  - Machine learning plug-ins
  - Online optimization
  - System-wide reasoning logic
- Stretch monitoring annotations to also create replay plugins
- Act as source for DKRZ system-wide monitoring system
  - Will integrate statistics e.g. knowledge/assessments for jobs
  - Optional to run applications with SIOX

Assessing Storage Class and Performance

Simplified output of an application run could be

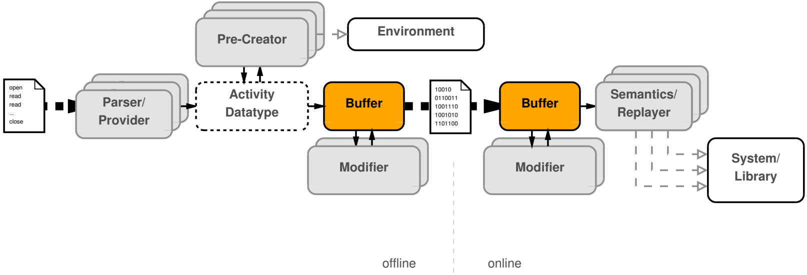
```
Read I/O
Total: 200 calls/100 MiB in 5.1s

These operations are presumably in the following classes:
Cached in the page cache: 10 calls/10 MiB
Cached on the server's cache: 10 calls/20 MiB
Average disk seek time: 100 calls/40 MiB (0.4s time loss)
Unexpected slow: 5 calls/100 KiB (1.5s time loss)
```

Summary

- SIOX aims to capture and optimize I/O
- We can change behavior without modifying code!
  - Design the optimization once, apply on many applications
  - Useful to evaluate strategies without implementing them (again)
- The goal of SIOX is a modular and open system

A complete environment for testing alternative optimizations



Database GUI

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

Activity Overview

Purge database | Execution Overview | Time frame statistics

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	

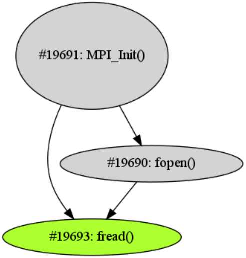
first ← previous 1 / 24 next → last

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.

Optimization Plug-in: GenericHistory

- Plug-in remembers hints and observable I/O performance
  - Does not store hints – tracks them for application life
  - Pre-defined
- Proposes MPI hints based on historic knowledge

System Configuration

Test system

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

Compute Nodes

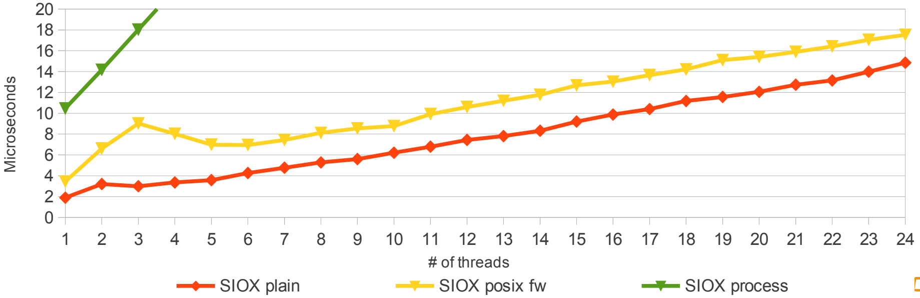
- Dual-socket Intel Xeon X5650@2.67 GHz
- Ubuntu 12.04
- Applications are compiled with: GCC 4.7.2, OpenMPI 1.6.5

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 room for improvement, and have some solutions in mind!
- 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.

Observable Performance – Discussion

Bad news

- For fast I/O operations several μs is expensive
- Additionally, locks protect several modules

⇒ I/O calls are synchronized (max. 100K Ops/s)

Good news

- We are already monitoring overhead

⇒ We will integrate methods to control the overhead

- Flexible and easy configuration can strip costly calls

Application runs?

For the ICON climate model, only initialization overhead is measurable

- A DB cache module reducing overhead