BoF: Analyzing Parallel I/O

Julian Kunkel[1] Philip Carns[2] Michael Kluge[3]
(Michaela Zimmer[4] Alvaro Aguilera[3])

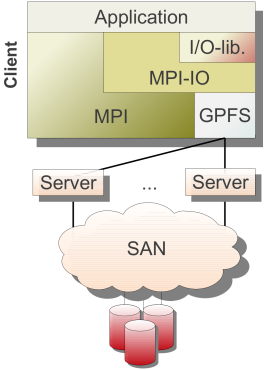
1 German Climate Computing Center
2 Argonne National Laboratory
3 ZIH, TU Dresden
4 University of Hamburg

Supercomputing 2014

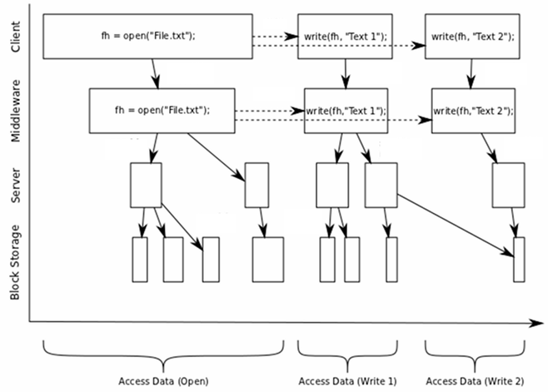
Outline

Introduction

A typical HPC I/O stack



I/O and its cause and effect chain



Tools Overview

Facets of monitoring and analysis

- Scope: Client side, middleware, kernel-space, file system, block devices
- Process centric vs. application centric vs. system-wide
- Profile/statistics vs. Tracing
- Analysis: textual, plots, group by signatures, bottlenecks
- Online, upon application termination, periodically updated

Not covered in this BoF explicitly

- Benchmarking, but (workload) replay tools are correlated
- Optimizing I/O

Existing Tool Landscape

- Local tools: blktrace, /*proc*, ...
- Cluster wide tools: Ganglia, Lustre Monitoring Tool (LMT), ...
- Tracing tools for HPC
  - Vampir (Score-P)
  - TAU (Score-P)
  - IPM
  - LANL-Trace
  - IOSIG
  - PAS2P-I/O
  - RIOT
  - ScalaIOTrace
  - SIOX

Wishlist for I/O Analysis Tools

- Ease of use
- Low overhead
- On demand application instrumentation
  - comprehensive information
  - but focusing on abnormal behavior
- System-wide instrumentation
- Portability
- Guided or automatic tuning
- Support for non-HPC environments
- Extensibility (support for new I/O libraries)

Discussion

- Requirements for future tools?
- How much overhead is acceptable?
- What environments/applications/platforms are most important to the community?
- What kind of information about I/O accesses is of interest for users?

Talk to us!

Darshan Philip Carns <carns@mcs.anl.gov>
SIOX Julian Kunkel <kunkel@dkrz.de>
Vampir Michael Kluge <Michael.Kluge@tu-dresden.de>

TECHNISCHE UNIVERSITÄT DRESDEN

Center for Information Services and High Performance Computing (ZIH)

Analyzing Parallel I/O BOF:

Vampir(Trace)

Zellescher Weg 12

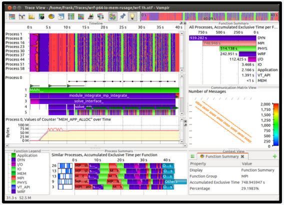
Willers-Bau A 208

Tel. +49 351 - 463 – 34217

Michael Kluge (michael.kluge@tu-dresden.de)

Center for Information Services & High Performance Computing

Vampir Mission



Visualization of dynamics
of complex parallel processes

Requires two components

- Monitor/Collector (VampirTrace or Score-P)
- Charts/Browser (Vampir)

Typical questions that Vampir helps to answer:

- What happens in my application execution during a given time in a given process or thread?
- How do the communication patterns of my application execute on a real system?
- Are there any imbalances in computation, I/O or memory usage and how do they affect the parallel execution of my application?

- Community efforts to make tools more versatile

Measurement: **Score-P**

Partners:

Technische Universität München

gns

JÜLICH FORSCHUNGSZENTRUM

TECHNISCHE UNIVERSITÄT DRESDEN

RWTH AACHEN UNIVERSITY

German Research School for Simulation Sciences

UNIVERSITY OF OREGON

Analysis:

- TAU

Tuning and Analysis Utilities

- **Vampir**

VAMPIR

- Scalasca

scalasca

Tracing – Monitors

- Currently for I/O VampirTrace is still the better tracing tool

Measurement: **Score-P**

Partners:

Technische Universität München

gns

JÜLICH FORSCHUNGSZENTRUM

TECHNISCHE UNIVERSITÄT DRESDEN

RWTH AACHEN UNIVERSITY

German Research School for Simulation Sciences

UNIVERSITY OF OREGON

Measurement:
VampirTrace
(Legacy)
Just for Vampir

Analysis:

- TAU
  Tuning and Analysis Utilities
- **Vampir**
  VAMPIR
- Scalasca
  scalasca

Roadmap

November 2014

Vampir:

- Collapsing timelines

Score-P:

- Sampling+tracing
- Xeon Phi (Native+Offload)
- Full SIONlib support (multi-flush, MPI+OpenMP)
- Memory tracing
- CPU ID
- 3rd party library wrapping
- OpenCL support

June 2015

Score-P:

- Cobi (Dyninst) instrumentation
- **I/O tracing**
- More MPI-3 support

Longer term

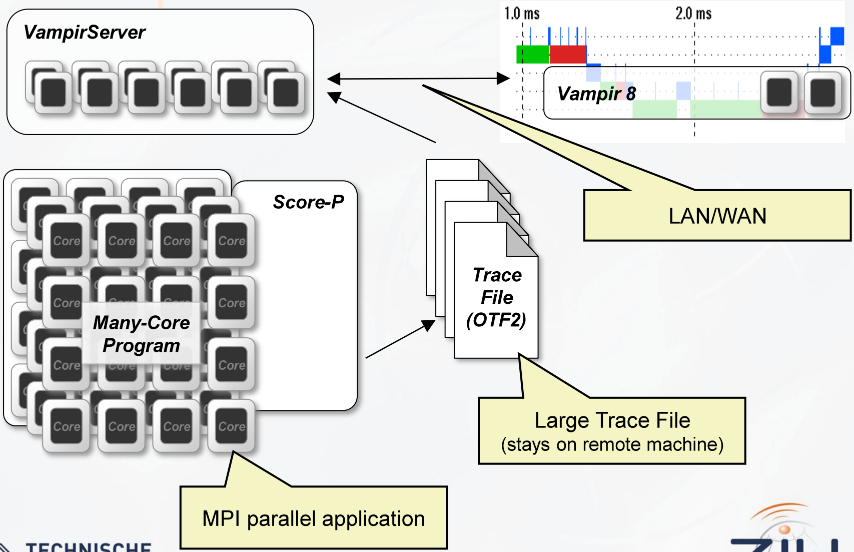
Vampir:

- Evaluation of traces with semantic compression

Score-P:

- Generation of traces with semantic compression

Vampir Workflow

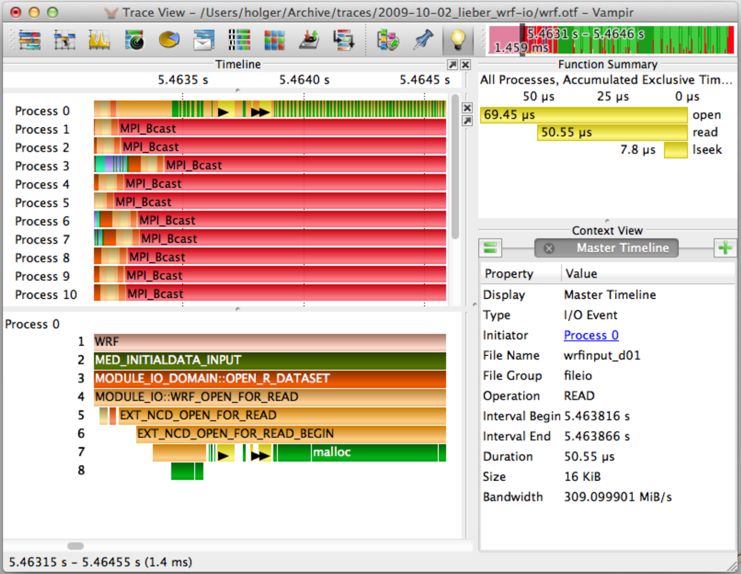


Vampir I/O Analysis

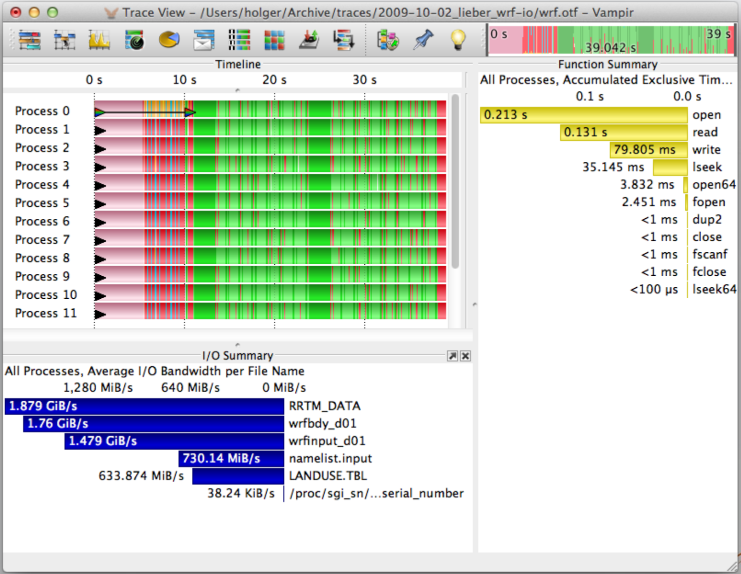
- Load the trace file -> Information about all I/O accesses in memory

- Analyze via the GUI:
  - take: number of operations, operation size, operation time, I/O bandwidth (min/max/avg)
  - group by file name, access size or operation type
  - filter by process id or I/O operation
  - sort
  - graph

Vampir & I/O

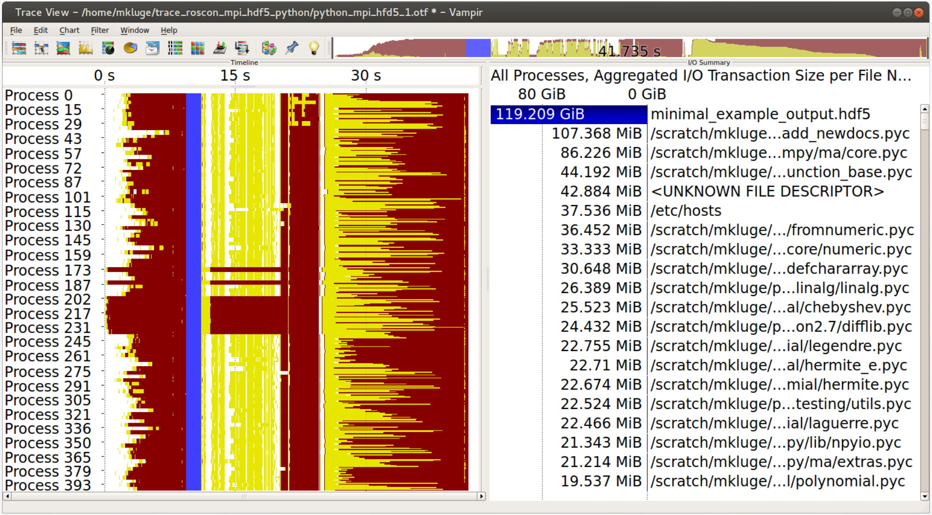

Trace View – /Users/holger/Archive/traces/2009-10-02_lieber_wrf-io/wrf.otf – Vampir
5.4631 s – 5.4646 s
1.459 ms
Timeline
5.4635 s
5.4640 s
5.4645 s
Process 0
Process 1
Process 2
Process 3
Process 4
Process 5
Process 6
Process 7
Process 8
Process 9
Process 10
MPI_Bcast
Process 0
1 WRF
2 MED_INITIALDATA_INPUT
3 MODULE_IO_DOMAIN::OPEN_R_DATASET
4 MODULE_IO::WRF_OPEN_FOR_READ
5 EXT_NCD_OPEN_FOR_READ
6 EXT_NCD_OPEN_FOR_READ_BEGIN
7 malloc
8
Function Summary
All Processes, Accumulated Exclusive Tim...
50 µs
25 µs
0 µs
69.45 µs open
50.55 µs read
7.8 µs lseek
Context View
Master Timeline
Property Value
Display Master Timeline
Type I/O Event
Initiator Process 0
File Name wrfinput_d01
File Group fileio
Operation READ
Interval Begin 5.463816 s
Interval End 5.463866 s
Duration 50.55 µs
Size 16 KiB
Bandwidth 309.099901 MiB/s
5.46315 s – 5.46455 s (1.4 ms)

Vampir & I/O

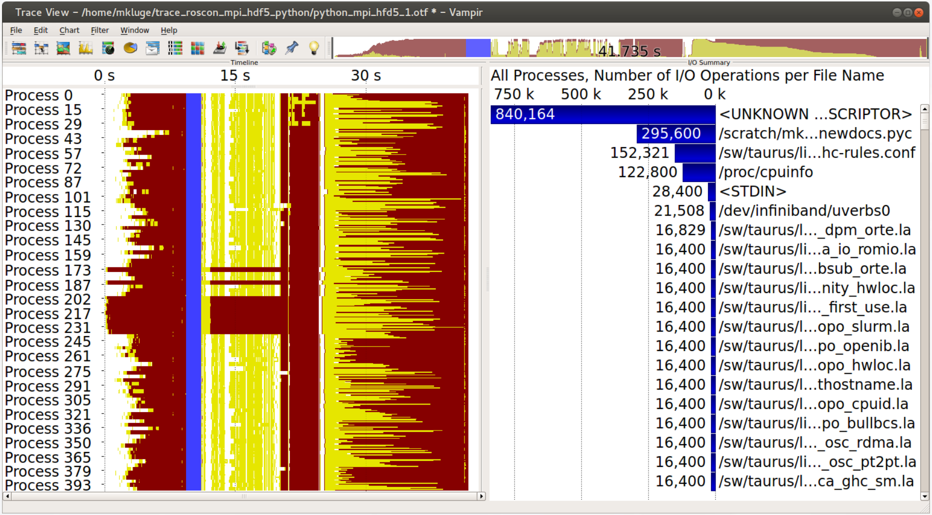


Trace View – /Users/holger/Archive/traces/2009-10-02_lieber_wrf-io/wrf.otf – Vampir
0 s
39.042 s
39 s
Timeline
0 s
10 s
20 s
30 s
Process 0
Process 1
Process 2
Process 3
Process 4
Process 5
Process 6
Process 7
Process 8
Process 9
Process 10
Process 11
Function Summary
All Processes, Accumulated Exclusive Tim...
0.1 s
0.0 s
0.213 s open
0.131 s read
79.805 ms write
35.145 ms lseek
3.832 ms open64
2.451 ms fopen
<1 ms dup2
<1 ms close
<1 ms fscanf
<1 ms fclose
<100 µs lseek64
I/O Summary
All Processes, Average I/O Bandwidth per File Name
1,280 MiB/s
640 MiB/s
0 MiB/s
1.879 GiB/s RRTM_DATA
1.76 GiB/s wrfbdy_d01
1.479 GiB/s wrfinput_d01
730.14 MiB/s namelist.input
633.874 MiB/s LANDUSE.TBL
38.24 KiB/s /proc/sgi_sn/...serial_number

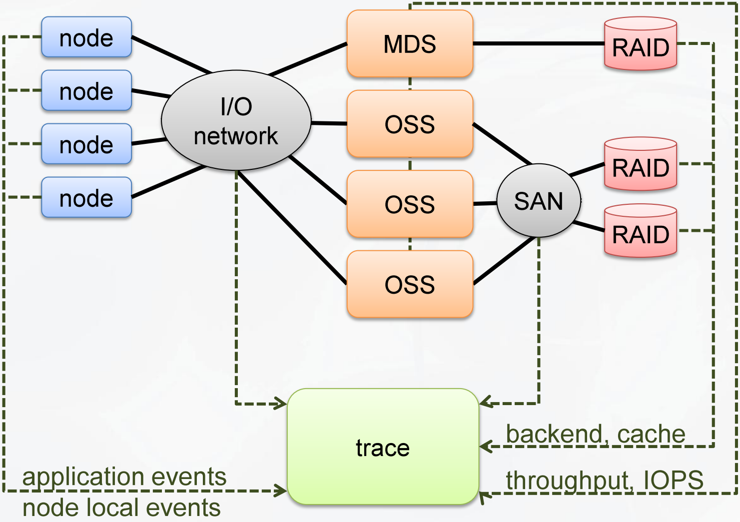
Vampir I/O Analysis



Vampir I/O Analysis



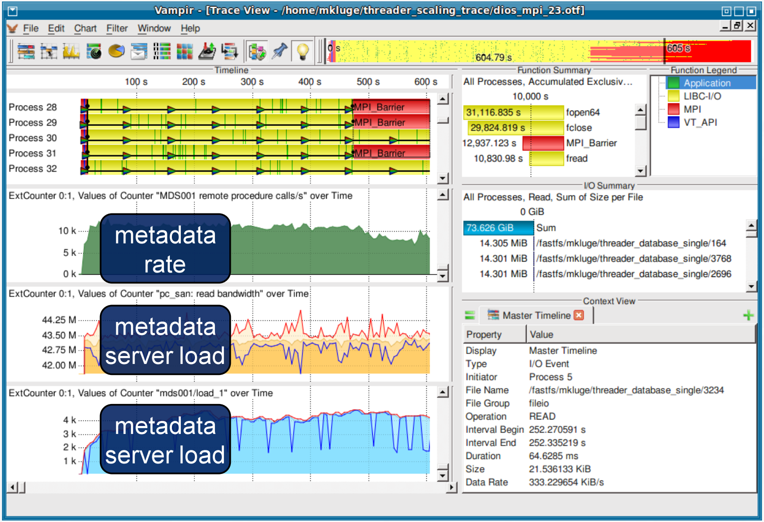
System Interaction (1)



Interaction:

- MPI
- I/O calls
- I/O system

System Interaction (2)

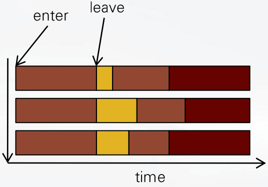


Interaction:

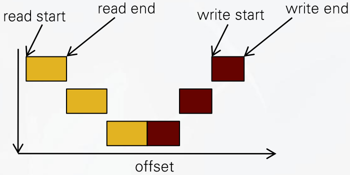
- MPI
- I/O calls
- I/O system

File Access Pattern Visualization

- Standard Timeline shows:
  - X axis: time
  - Y axis: processes

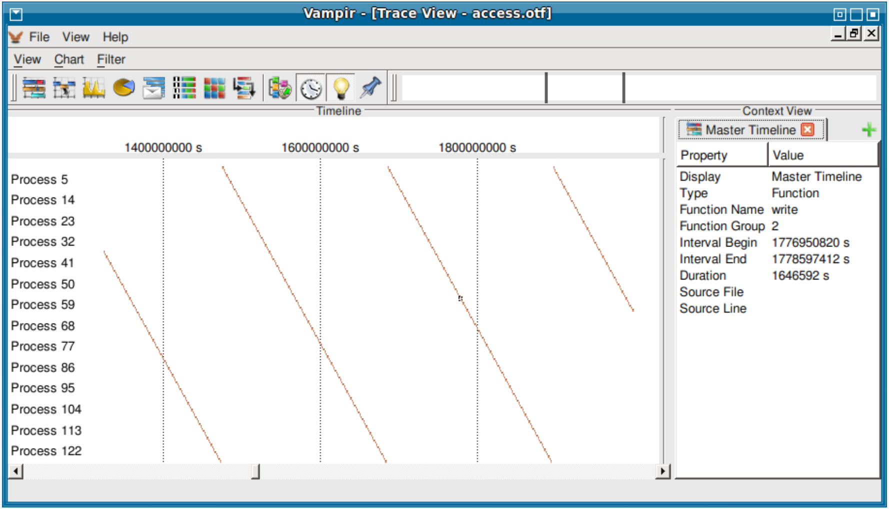


- What is needed for file access patterns:
  - X axis: byte range within a single file
  - Y axis: processes

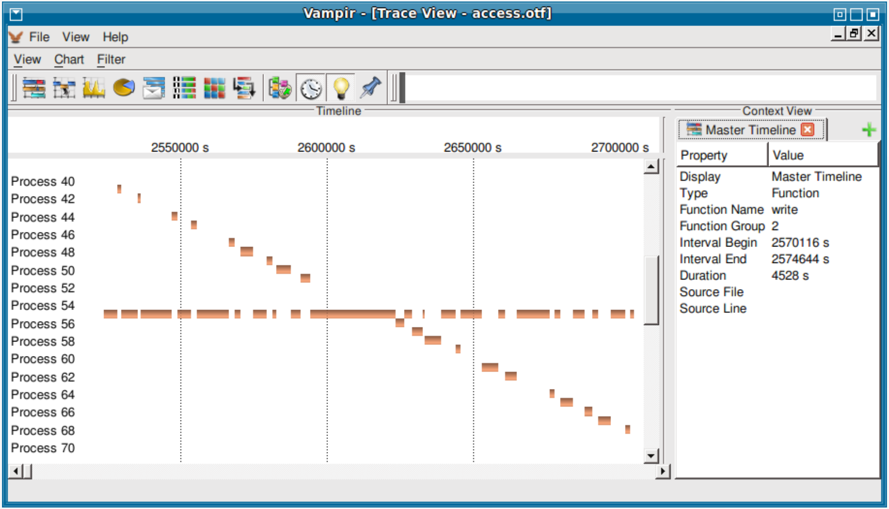


- Possibilities:
  - mark multiple writes, read-after-write conditions, etc.

Access Pattern Visualization



Access Pattern Visualization



Analyzing Parallel I/O BOF:

Darshan

Phil Carns
Mathematics and Computer Science Division
Argonne National Laboratory

Darshan Overview

Darshan is a lightweight, scalable I/O characterization tool that captures I/O access pattern information from production applications.

Key design principle: ***don't perturb application performance or behavior***

- Low (fixed) memory consumption
- Wait until MPI_Finalize() to aggregate, compress, and store data
- No source code or makefile changes

Typical use cases:

- Performance tuning
- Analysis of system-wide I/O trends

What is it?

- Runtime library:
  - Instruments POSIX, MPI-IO, and some HDF5 and PnetCDF functions
  - No kernel modifications or persistent services
- Command-line utilities:
  - Post-processing of Darshan logs
  - Includes tools to produce graphical I/O summaries

- Portability
  - Works on IBM Blue Gene, Cray, and Linux environments
  - Compatible with all popular compilers
  - Compatible with all popular MPI implementations
- Low barrier to entry
  - Depending on the platform, Darshan may be enabled by default by your administrator. Otherwise Darshan is enabled by loading a software module or by compiling with a Darshan-enabled compiler script.

A typical Darshan workflow

- Compile an MPI program (C, C++, or FORTRAN)
- Run the application
- Look for the ***Darshan log file***
  - This will be in a particular directory (depending on your system's configuration)
  - <dir>/<year>/<month>/<day>/<username>_<appname>*.darshan.gz
- Use Darshan command line tools to analyze the log file

- **Darshan does not capture a trace of all I/O operations**: it instead reports key statistics, counters, and timing information for each file accessed by the application.
- **Darshan does not provide real-time monitoring**: the application must run to completion and call MPI_Finalize() before producing a log file.

Darshan analysis tool example

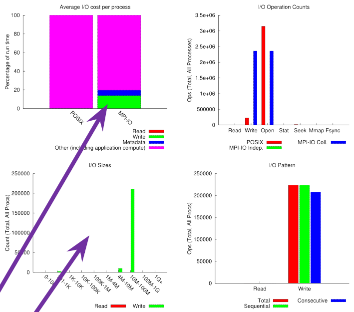
- Example: Darshan-job-summary.pl produces a 3-page PDF file summarizing various aspects of I/O performance
- This figure shows the I/O behavior of a 786,432 process turbulence simulation (production run) on the Mira system at ANL
- This particular application is write intensive and benefits greatly from collective buffering, no obvious tuning needed.

Example measurements: % of runtime in I/O

access size histogram

(9/25/2013) 1 of 3

jobid: 149563	uid: 6729	nprocs: 786432	runtime: 2751 seconds



Most Common Access Sizes

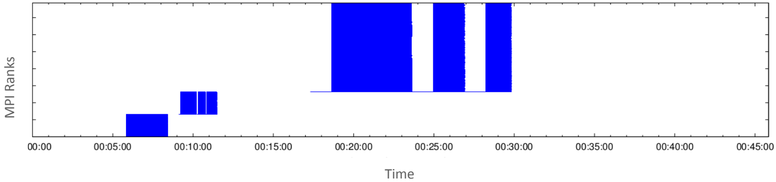
access size	count
16777216	210977
8388608	9866
256	2598
68	9

File Count Summary
(estimated by I/O access offsets)

type	number of files	avg. size	max size
total opened	17	199G	1.6T
read-only files	1	2.0K	2.0K
write-only files	13	260G	1.6T
read/write files	0	0	0
created files	13	260G	1.6T

/gpfs/mira-fs0/projects/WallModJet/cwhamman/wavelength/./rbturb.x 11 specs.in

Darshan analysis tool example

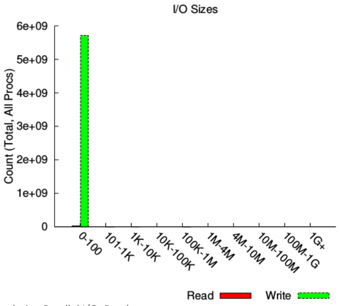


The darshan-job-summary.pl output also shows intervals of I/O activity. This example shows bursts of write activity from different subsets of MPI ranks at different points in the job execution.

Example: investigating I/O performance

- **Scenario:** Small writes can contribute to poor performance
  - Particularly when writing to shared files
  - Candidates for collective I/O or batching/buffering of write operations

- **Example:**
  - Issued 5.7 billion writes to shared files, each less than 100 bytes in size
  - Averaged just over 1 MiB/s per process during shared write phase



Most Common Access Sizes

access size	count
1	3418409696
15	2275400442
24	42289948
12	14725053

Example: finding superfluous I/O activity

- **Scenario:** Applications that read more bytes of data from the file system than were present in the file (redundant read traffic)
  - Disruptive to the network, even with caching
  - Candidates for aggregation or collective I/O
- **Example:**
  - Scale: 6,138 processes
  - Run time: 6.5 hours
  - Avg. I/O time per process: 27 minutes

File Count Summary
(estimated by I/O access offsets)

type	number of files	avg. size	max size
total opened	1299	1.1G	8.0G
read-only files	1187	1.1G	8.0G
write-only files	112	418M	2.6G
read/write files	0	0	0
created files	112	418M	2.6G

1.3 TiB of file data available

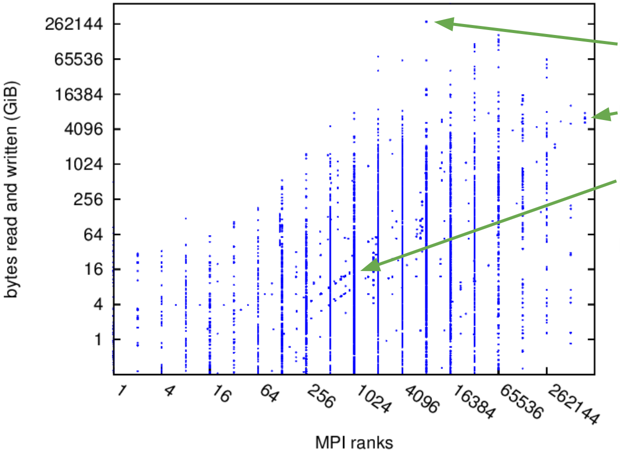
500+ TiB read from file system

Data Transfer Per Filesystem

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/	47161.47354	1.00000	575224145.24837	1.00000

Example: system-wide analysis

Job size vs. data volume for Mira BG/Q system in 2014
(~128,000 logs as of October, ~8 PiB of traffic)



- Biggest by volume: ~300 TiB
- Biggest by scale: 768K processes
- Probably some scaling experiments?
- Most jobs use power of 2 numbers of processes on Mira

Status and future work

- Darshan is available (open source) at http://www.mcs.anl.gov/research/projects/darshan
- Anonymized logs are also available for download and analysis
- Actively maintained by ANL/ALCF and LBL/NERSC
  - Also enabled automatically for all users at those facilities
  - Available as an optional package at many other sites

- Ongoing maintenance to insure portability and performance
- Current major development effort is to modularize Darshan so that it can be more easily extended to cover additional I/O libraries or other use cases.

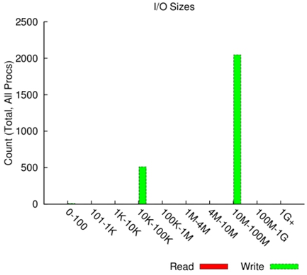
We are always looking for feedback!

This work was supported by Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract Nos. DE-AC02-06CH11357 and DE-AC02-05CH11231 including through the Scientific Discovery through Advanced Computing (SciDAC) Institute for Scalable Data Management, Analysis, and Visualization.

Extra material

Example: Checking I/O expectations

- Darshan summaries are often helpful in confirming the expected behavior of applications.



- In this case, there were 512 relatively small writes of 40 KiB each
- That size corresponds to the file header size of the application (as expected)
- But there are only 129 files, why are there 512 headers?

Most Common Access Sizes

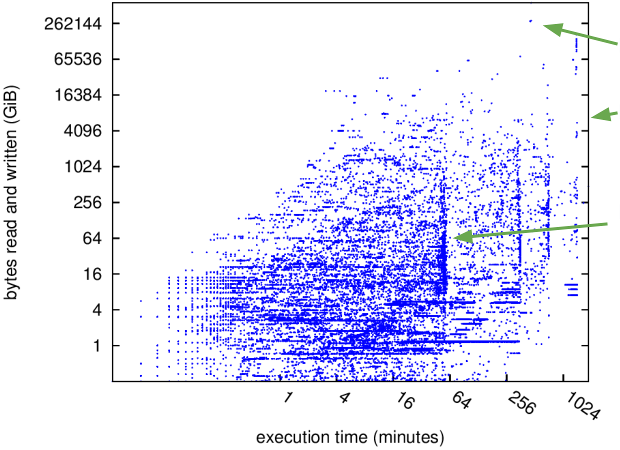
access size	count
67108864	2048
41120	512
8	4
4	3

File Count Summary

type	number of files	avg. size	max size
total opened	129	1017M	1.1G
read-only files	0	0	0
write-only files	129	1017M	1.1G
read/write files	0	0	0
created files	129	1017M	1.1G

Example: system-wide analysis (2)

Job duration vs. data volume for Mira BG/Q system in 2014
(~128,000 logs as of October, ~8 PiB of traffic)



- Largest job by volume ran for about 6 hours
- Longest jobs ran for 24 hours (maximum wall time according to Mira policy)
- Most popular execution time was roughly 1 hour

SIOX: Scalable I/O for Extreme Performance

Julian Kunkel[1] Michaela Zimmer[2]

1 German Climate Computing Center
2 University of Hamburg

Analyzing Parallel I/O BoF
SC 2014

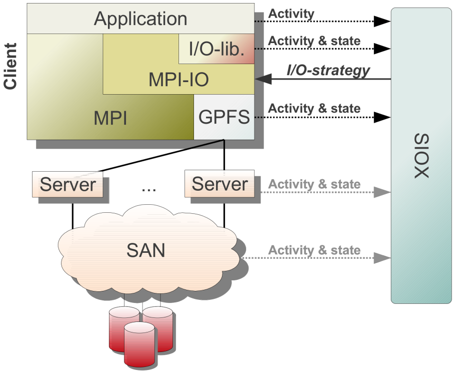
Outline

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

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

Extensibility for Alternate APIs

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

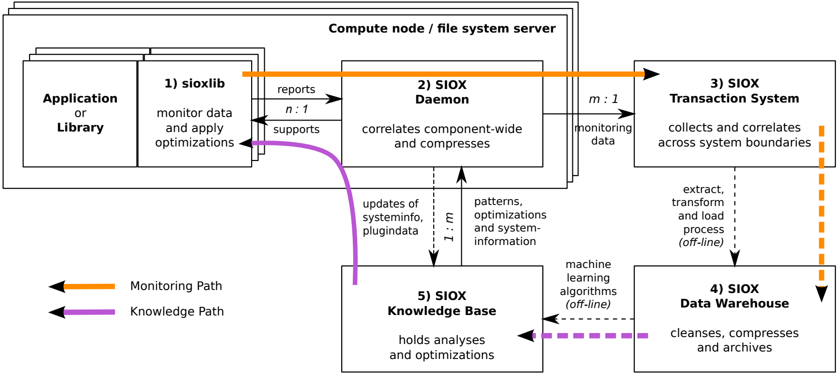
Header annotations for `MPI_File_write_at()`

```
//@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);
```

Modularity of SIOX

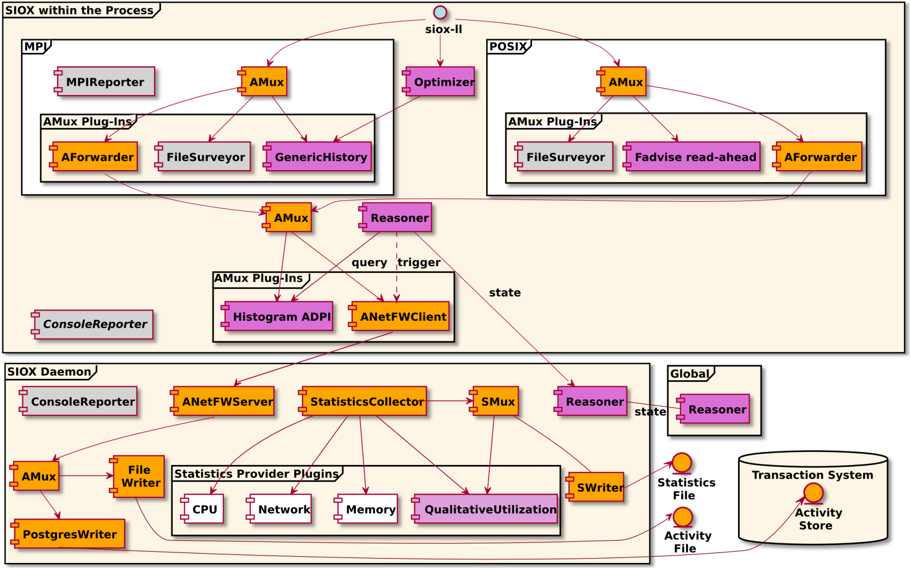
- The SIOX architecture is flexible and developed in C++ components
- License: LGPL, vendor friendly
- Upon start-up of (instrumented) applications, modules are loaded
- Configuration file defines modules and options
  - Choose advantageous plug-ins
  - Regulate overhead
- For debugging, **reports**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

Proposed Workflow



- 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

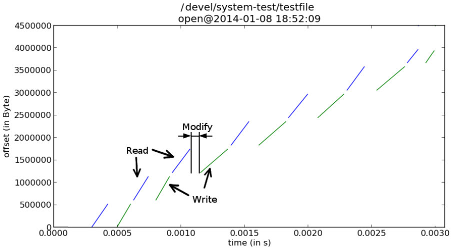
- Supports different file and database back-ends
- Plug-in based
  - Text output
  - Time-offset plots for files

Example text output created by the trace-reader

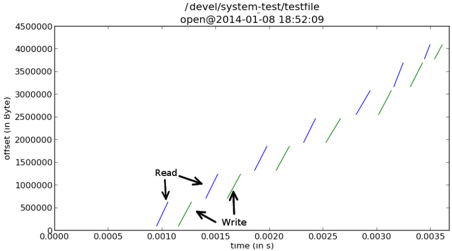
```
0.0006299 ID1 POSIX open(POSIX/descriptor/filename="testfile",
  POSIX/descriptor/filehandle=4) = 0
0.0036336 ID2 POSIX write(POSIX/quantity/BytesToWrite=10240,
  POSIX/quantity/BytesWritten=10240, POSIX/descriptor/filehandle=4,
  POSIX/file/position=10229760) = 0 ID1
0.0283800 ID3 POSIX close(POSIX/descriptor/filehandle=4)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

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	

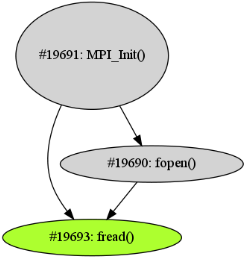
↶ first ← previous   1 / 24   next → last ↷

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.

Reporting: FileSurveyor

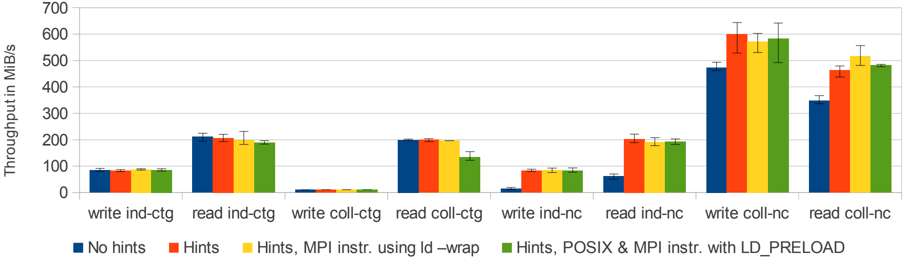
- Easy to collect and track relevant application statistics
- FileSurveyor prototype collects POSIX/MPI access statistics
- Only 1000 LoC
- ... *Yes we'll pretty print things at some point ...*

```
[...] "(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

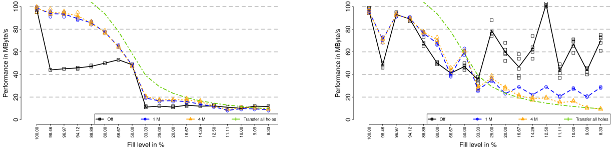
Results

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

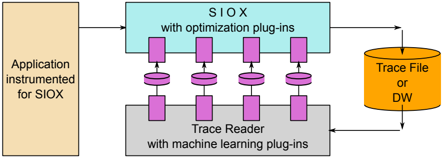
Time needed to read one 1 KiB data block in a strided access pattern.

To Sieve, or Not to Sieve?

Strided data – choice of best parameters highly non-trivial:

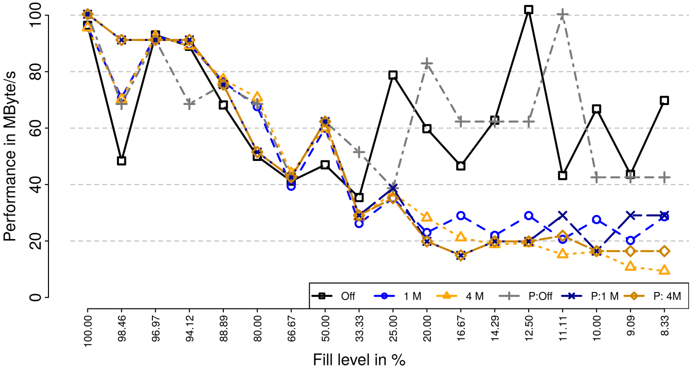


Left: d_{data} =16 KiB; Right: d_{data} =256 KiB



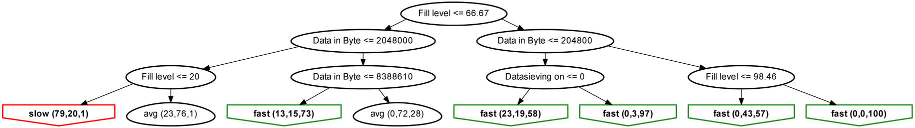
Apply full workflow cycle:
Learning with Classification and Regression Trees (*CART*, library: Shark)

Lessons Learned (1): A Performance Predictor



True and predicted data for $pdata = 256$ KiB;
387 instances used for training

Lessons Learned (2): New Heuristics from Decision Rules



First 3 levels of the CART classifier tree for classes slow, avg, fast ($[0;25]$, $]25;75]$, > 75 MiB/s). Leaf nodes show dominant labels and class probabilities

Lessons Learned (3): Performance Gain!

Default Choice	CART, 387 Instances	Best Choice
Off	4.2 MiB/s	9.6 MiB/s
1 MiB/s	1.9 MiB/s	7.6 MiB/s
4 MiB/s	6.9 MiB/s	12.2 MiB/s
100 MiB/s	6.9 MiB/s	12.2 MiB/s

Average performance improvements compared to a user's default choice

Automatically closes the gap to optimum performance by 25-50%!

Future Work

- More instrumentations
  - Partners working on GPFS
  - Expressed 3rd party interest in Panasas, BeeGFS/FhGFS, Lustre
- More optimizations
  - Machine Learning plug-ins
  - Performance predictors
  - Reasoner logic
- Will be used on the DKRZ's next HPC machine, HLRE3
  - 3 PetaFlop/s
  - 45 PetaByte HDD/SSD storage
- Part of the E10 initiative, providing functionality and API

Summary

- SIOX aims to capture and optimize I/O
  - on all layers and file systems
- We analyzed the overhead of the prototype
  - Learns best MPI hints and data sieving parameters and sets them
  - Bearable monitoring overhead
  - Flexible configuration
- We demonstrate how we change behavior without modifying code!
  - Design the optimization once, apply on many applications

- **We are building a modular and open system**
- **We are looking forward to contributing components to E10**

System Configuration

Test system

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

Compute Nodes

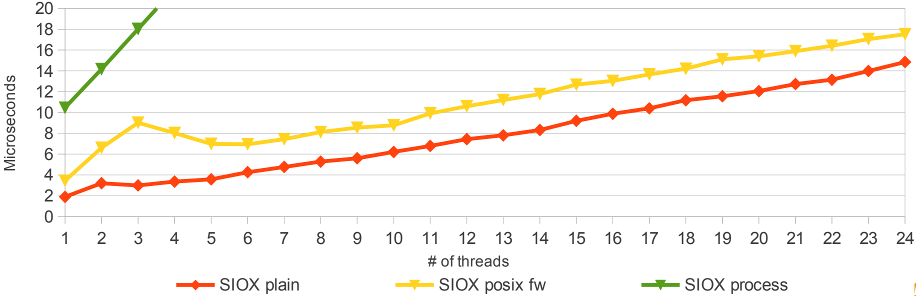
- Dual-socket Intel Xeon X5650@2.67 GHz
- Ubuntu 12.04
- Applications are compiled with: GCC 4.7.2, OpenMPI 1.6.5

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 initalization overhead is measurable

- A DB cache module reducing overhead

Discussion

- Requirements for future tools?
- How much overhead is acceptable?
- What environments/applications/platforms are most important to the community?
- What kind of information about I/O accesses is of interest for users?

Talk to us!

Darshan Philip Carns <carns@mcs.anl.gov>
SIOX Julian Kunkel <kunkel@dkrz.de>
Vampir Michael Kluge <Michael.Kluge@tu-dresden.de>