Challenges in HPC I/O

Universität Basel

Julian M. Kunkel

German Climate Computing Center / Universität Hamburg

10. October 2014

Outline

High-Performance Computing – Motivation

Scientific applications have a high demand of

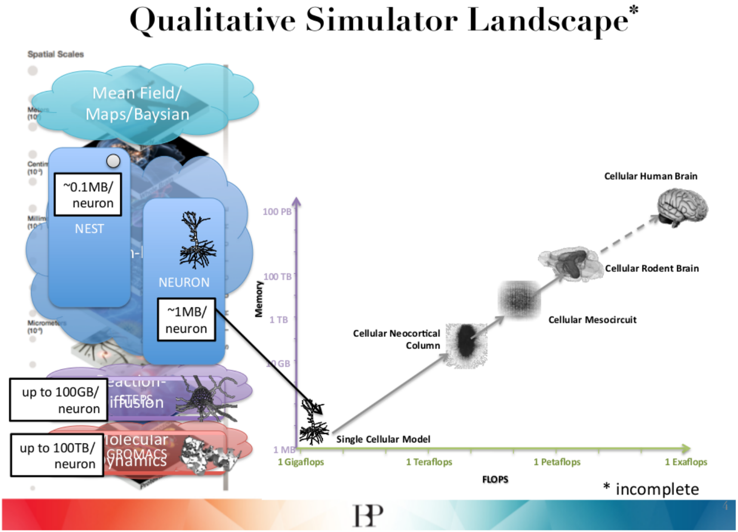
- Computing time
- Memory
- Storage

A common PC/server is not able to compute solution (in time)

$\Rightarrow$ Parallel usage of many CPUs and servers

- Moore's „law" increases parallelism and not (any more) frequency

Relevance illustrated on the Human Brain project



Source: „Simulation Codes in the Human Brain Project“, Prof. Felix Schürmann, 2014

Titan: Second Fastest Supercomputer

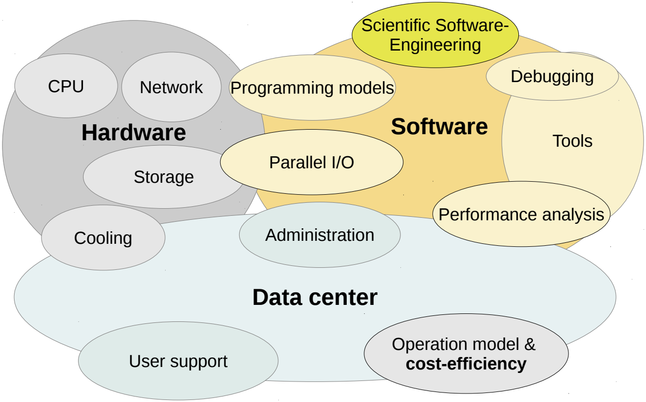
Titan – front view [*Source: Wikipedia, Titan_(supercomputer)*]

- 200 racks on 400 m^2
- 18.688 nodes:
  - 1 AMD Opteron 16-core CPU
  - 1 Nvidia Tesla GPU
  - 32 GByte RAM
- Peak performance: 27 PFlop/s
- Main memory: 710 TByte
- Storage capacity: 40 PByte
- Costs: 100 million $
- Energy consumption: 8.2 MW

Definition: High-Performance Computing

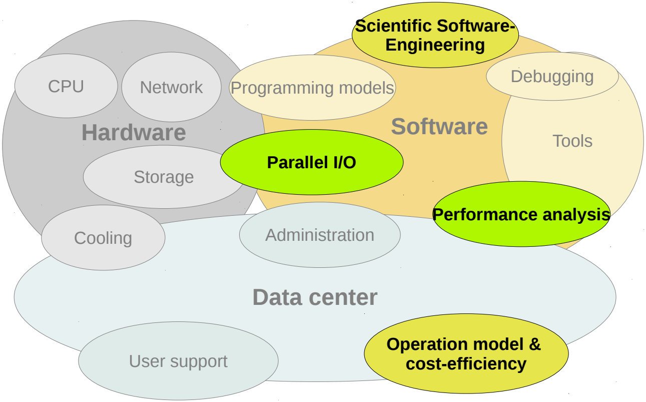
1. Programming of applications with high resource demand
2. Construction and usage of supercomputers

Selected subdisciplines



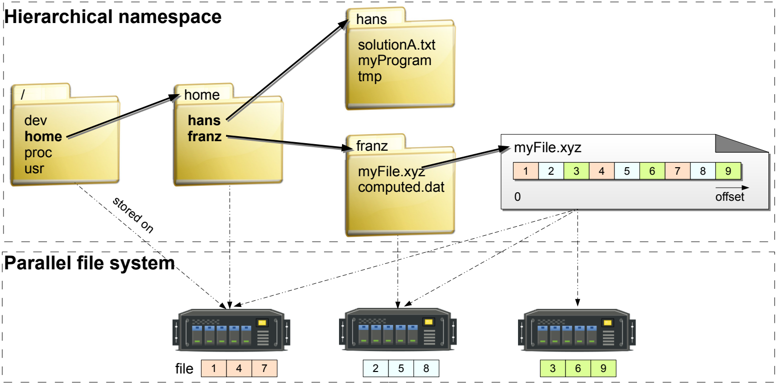
Research Profile

- Parallel I/O and performance analysis
- Scientific Software-Engineering and cost-efficiency



Parallel File Systems

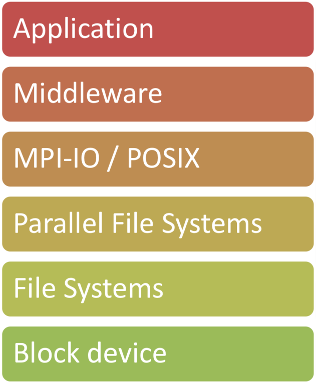
- Distribute data (and metadata) among many servers
- Goal: Aggregated performance of all servers
- Parallel: Concurrent access of processes to a shared file



Example hierarchical namespace

Challenges of the HPC-I/O Stack

- Co-existence of different access paradigms
  - File (POSIX, ADIOS, HDF5), SQL, NoSQL
- File system: Level of abstraction is very low
  - A file is an array of bytes
  - ⇒ Scientific domains re-implement similar features
- Re-implementation of features across the stack
  - ⇒ Unpredictable interactions and resource waste
- Loss of semantical (application) knowledge
  - ⇒ Suboptimal steering and performance loss
- Insufficient performance portability
  - Each layer needs to be optimized for all systems
- Handling Petabyte of data?



Example I/O stack

Semantical Gap of File Access

Information hidden from file systems

- Data types
- Data semantics
- Required synchronization
- Value of data
- Data lifecycle: production, usage, deletion

Characteristics can even vary within a file

Storage systems could use this information

- Improving performance: Automatic tiering, caching, replication
- Simplifying management: ILM, offering alternative data views
- Correctness: Ensuring data consistency

Tuning Performance

- There are many options to tune the I/O-stack
  - API: `posix_fadvise()`, HDF5 properties, open flags, cache size
  - Via command line: `lfs setstripe`
  - Setup/initialization of a storage system
- Most options are of technical nature
  - Example: read-ahead window of 256 KiB
  - Many types of hints/tweaks are not portable
  - ⇒ Performance gain/loss is system (and application) specific
- Performance loss forces us to use these optimization
  - Are 50% performance loss acceptable if one option is wrong?

BMBF-Project: SIOX

Motivation

- Lack of tools for assessing application's I/O performance
- Existing optimizations are hard to parameterize

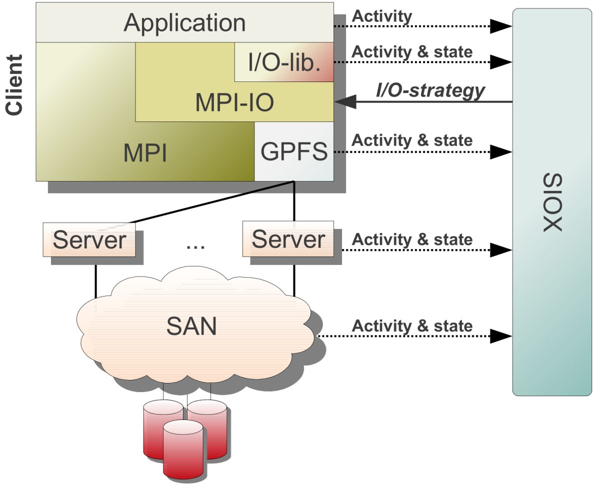
Goals of SIOX

- Monitoring of I/O behavior and performance
- Automatic assessment of observed activity
- Extracting of new knowledge
- Learn (and apply) controllable optimizations (using ML)

Cooperation

- Universität Hamburg, HLRS Stuttgart, ZIH Dresden, Exascale10

High-Level Architectural View

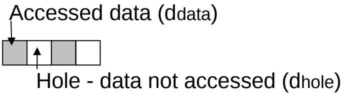


Integration of SIOX in a typical HPC I/O stack

Example Study: Learning Parameters for MPI-I/O

Background

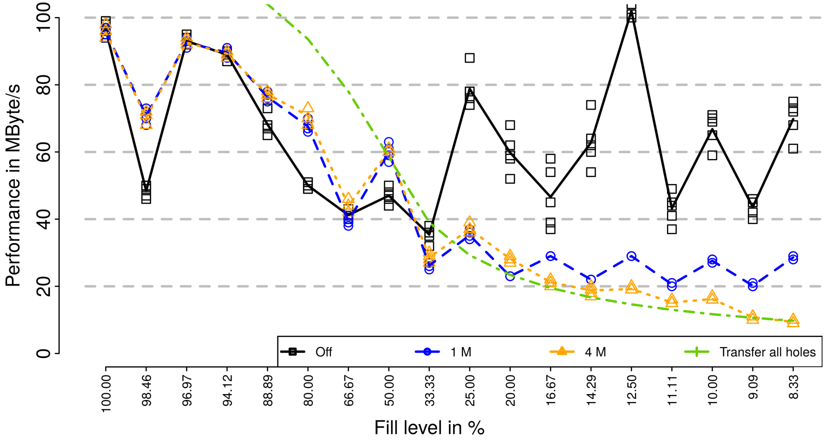
- Non-Contiguous I/O accesses multiple file regions by one call
- The data-sieving optimization accesses whole datatype
- Optimization parameters (on file level): on/off, buffer size



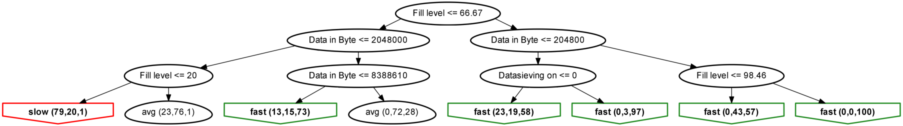
Approach

- Measure a set of similar patterns by one process
- Vary: block size, gap, optimization (> 700 configurations)
- Use ML to
  - predict performance
  - choose best optimization

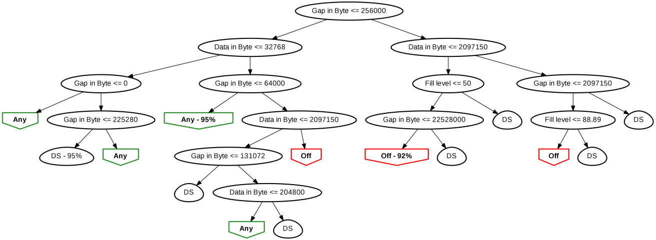
Performance for ddata = 256 KiB



Decision Trees



Performance prediction, first 3 levels, classes slow, avg, fast ([0; 25],]25; 75], > 75 MiB/s). Dominant label in the leaf nodes – class probability is provided in ()



Optimization to choose

Rules can be extracted to create (new) knowledge

Example-Optimization: Pre-fetching

- Pre-fetching: Read data from slower medium before needed
- Use `posix_fadvise()` to tell OS to read data
- Benchmark reads data and simulates compute time
  - 100 μs and 10 ms for 20 KiB and 1000 KiB stride
  - Program: Manual pre-fetching vs. extended version
  - SIOX plug-in injects advise without code modification

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 injects fadvise	52.1 μs	95.4 μs

Average time to read 1 KiB of data

Critical Discussion

Questions from the users' perspective

- Why do I have to organize the file format?
  - *It's like taking care of the memory layout of C-structs*
- Why must I provide system-specific performance hints?
  - *It's like telling the compiler to unroll a loop exactly 4 times*
- Why do I have to convert data between storage paradigms?
  - *I want to access data of my experiments based on their properties*
- Why is a file system not offering the consistency model I need?
  - *My application knows the required level of synchronization*

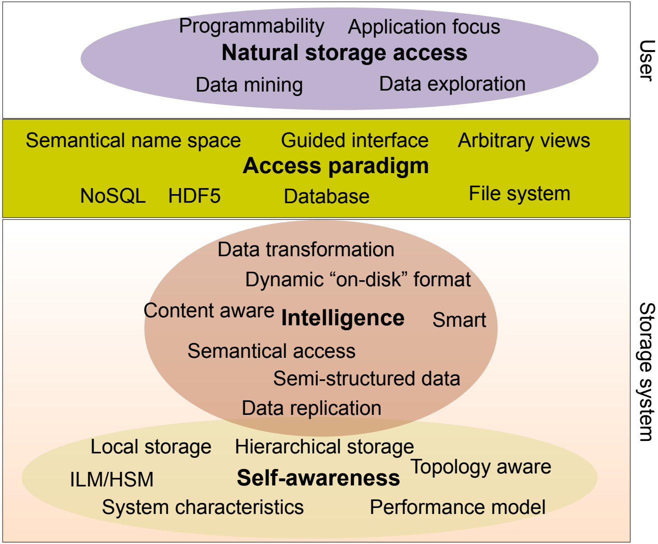
User View on Controlling File Systems' Performance

Source: Elya: „Antonow an24 cockpit“, Wikipedia commons, CC BY-SA 3.0

Do you believe this complexity is needed in storage systems?

Would you rather like to code your actual application?

Personal Vision of Future Storage Systems



The Exascale I/O Workgroup (Exascale10)

- Goal: Development of a middleware with advanced features
  - Data-type aware storage
  - Multiple "views" to the same data (BigData)
  - Guided interfaces instead of technical hints
  - Data "format" handled by storage system
  - Multi-tiering support
  - Intelligent monitoring
  - Feedback to optimization and "what-if" analysis
  - Integrates active storage concept
  - Post-processing is handled by "file" system
- First design documents have been published

Background

- E10 is a workgroup of EOFS (Lustre community)
- International and open initiative

EXASCALE IO

http://www.exascale10.com

Summary & Conclusions

- High-Performance Computing is a fascinating area
- Efficiency of money spent can be improved
  - Requires holistic view of hardware, sw, middleware & data center
- Demand for semi-automatic optimization tools
  - SIOX combines measurement, analysis and optimization
- Storage access paradigms need to change
  - Flexible, intelligent and self-aware storage
  - Guided interfaces vs. technical hints
  - Exalscale10 is an enabler for data intensive science

Backup Folien

Expected Input from Users: Guided Interfaces

Guiding vs. automatism vs. technical hints

Users provide additional information to guide an intelligent system.
The I/O stack exploits this information.

Information which could be provided by users

- Data types
- Semantics
- Relations between data
- Lifecycle (especially usage)

Several issues have been addressed in different access paradigms.
Also some behavioral hints exist: open() flags, fadvise(), ...

Parallel Programming

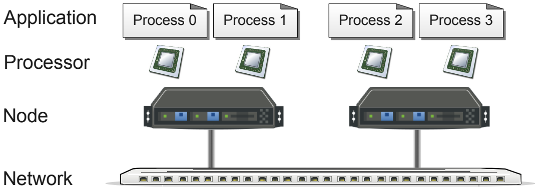
Goal

Many CPUs cooperatively compute the solution

Approach

Parallel processing of data and/or tasks

- Cooperation requires coordination and communication
- Programming model defines formal specification



Application with four processes on two nodes