I/O Interfaces Will Change

Chance or Curse for Programmers?

Julian M. Kunkel

German Climate Computing Center (DKRZ)

15-05-2014

Outline

1 Motivation

- 2 State of the Art
- **3** Ongoing Projects

4 Summary

Semantical Gap of File Access (1)

Applications work with (semi)structured data

Vectors, matrices, n-Dimensional data

A file is just a sequence of bytes!



Applications/Programmers must serialize data into a flat namespace

- Uneasy handling of complex data types
- Mapping is performance-critical (on HDDs)
- Vertical data access unpractical (e.g. to to pick a slice of multiple files)

Semantical Gap of File Access (2)

Information hidden from file systems

- Data types
- Data semantics
- Value of data
- Type: Checkpoint, computed, original, logfile
- Data lifecycle: production, usage, deletion

Characteristics can even vary within a file, e.g. for metadata

Storage systems could use this information for

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

Peeking at the Current I/O Stack – System Perspective

Coexistence of access paradigms

- File (POSIX, ADIOS, HDF5), SQL, NoSQL
- Semantical information is lost through layers
 - Suboptimal performance
- Reimplementation of features across stack
 - Unpredictable interactions
 - Wasted ressources
- Restricted (performance) portability
 - Optimizing each layer for each system?



User Perspective: Accessing Data

Multitude of data models

- POSIX File: Array of bytes
- HDF5: Container like a file system
 - Dataset: N-D array of a (derived) datatype
 - Rich metadata, different APIs (tables)
- Database: structured (+arrays)
- NoSQL: document, key-value, graph, tuple

Choosing the right interface is difficult. A workflow could involve different data models.

Properties / qualities

- Namespace: Hierarchical, flat, relational
- Access: Imperative, declarative, implicit (mmap())
- Concurrency: Blocking vs. non-blocking
- Consistency semantics: Visibility and durability of modifications

Consistency Semantics

Example: Two processes accessing one file ("data", offset, size)

- P1: write("1", 0, 1024) write("1", 1024, 1024) read(0, 2048)
- P2: write("2", 0, 1024) write("2", 1024, 1024) read(0, 2048)

Which data is stored and read depends on the execution sequence AND the consistency semantics.

Aspects of consistency

- Visibility to the modifying processes vs. other processes
 - Distributed system makes consistency expensive
 - Delay before modifications become visible Inconsistency window
- Granularity in which modifications are atomic
 - No guarantee, single operation, batch or transactions

Consistency Models (Selection)

- Strict (linear) consistency (POSIX)
 - Modifications made to NFS if accessed by only one node
- Sequential consistency
 - Any possible sequential execution possible
 - Processes have the same view always
 - Atomic-Mode for MPI-IO (applicable for collective file access)
- Weak consistency
 - Inconsistency "window"
- Eventual consistency (DNS, Amazon S3)
 - Inconsistency window can be estimated
 - Especially for replicated services
- Read-after-write consistency (does not include data updates)
 - Amazon S3 rolling upgrade in US between 2009 and 2012: Now all clients see new data
- Release consistency (like the session model of NFS)

Performance Tweaks

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
- Many options are of technical nature
 - Performance gain/loss depend on hardware, software
 - Specific to file system, API (MPI, POSIX, HDF5)
 - Many types of hints/tweaks are not portable
- Performance loss forces us to use these optimization

Performance Tweaks

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
- Many options are of technical nature
 - Performance gain/loss depend on hardware, software
 - Specific to file system, API (MPI, POSIX, HDF5)
 - Many types of hints/tweaks are not portable
- Performance loss forces us to use these optimization

Usually we are losing system performance!

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 do I have to convert data between storage paradigms?
- Why must I provide system specific performance hints?
 - It's like telling the compiler to unroll a loop exactly 4 times
- Why can't I rely on a correct implementation of the (POSIX) consistency model?
 - Parallel file systems have their issues with most models
- Why is a file system not offering the consistency model I need?
 - My application knows the required level of synchronization

Would you rather like to code your actual application?

Newer, current and future projects aim to

- Converge / unify the I/O stack
- Abstract from existing solutions, e.g. by a middleware
- Offer new ways of exploiting user information / semantics

Let's peek at

- ADIOS
- Fast Forward Storage & IO
- Exascale10

ADIOS

Adaptable IO System

- Alternative high-level I/O interface
 - Annotations of variables similar to HDF5
- Offers various back-ends: POSIX, MPI-IO, NULL or in-situ vis.
- Own file format (BP)
 - Throughput oriented, avoids synchronization
 - An ADIOS file may be represented by one or multiple objects
 - Easy conversion of BP files into NetCDF or HDF5
- XML specification of variables and run-time parameters
 - Adapt programs to the site's file system without code adjustment
 - Translate XML into C or Fortran code to read/write data

```
Motivation
00
```

Example code using ADIOS

```
int NX = 10. NY = 10. NZ = 100: double matrix[NX][NY][NZ]:
      MPI_Comm comm = MPI_COMM_WORLD: int64_t adios_handle:
      int adios_err; uint64_t adios_groupsize, adios_totalsize;
      MPI_Init(&argc, &argv); MPI_Comm_rank(comm, &rank);
      adios_init("example.xml"):
6
7
      for (t = 0; t < 10; t++) {
8
        adios_start_calculation():
9
        /* computation */
10
        adios_stop_calculation();
11
12
        /* MPI communication */
        adios_open(&adios_handle, "fullData", "testfile.bp", t == 0
13
             \hookrightarrow ? "w": "a". &comm):
14
  #include "gwrite_fullData.ch"
        adios close(adios handle):
15
        /* indicate progress for write-behind */
16
        adios_end_iteration():
17
      }
18
19
      adios_finalize(rank): MPI_Finalize(): return 0:
20
```

Motivation 00

OOOOOO

Ongoing Projects

gwrite_fullData.ch

Efficient I/O

Caching

- ADIOS aggressively caches data
- Write-behind during compute phases
- Iterative programs can indicate pace by calling a function

User controls runtime behavior via XML

- Choose the back-end for a supercomputer and task
- Set optimal parameters such as the cache size
- Instruct to create derived data (histograms)

```
Ongoing Projects
                                       ADIOS XMI code
<adios-config host-language="C">
 <adios-group name="fullData" coordination-communicator="comm"</pre>
    time-index="iteration">
    <attribute name="description" path="/fullData"</pre>
      value="Global array of memory data" type="string"/>
    <var name="NX" type="integer"/>
    <var name="NY" type="integer"/>
    <var name="NZ" type="integer"/>
    <var name="matrix data" gwrite="matrix" type="double"</pre>
      dimensions="iteration,NX,NY,NZ"/>
 </adios-group>
 <analysis adios-group="fullData" var="matrix data"</pre>
   min="0" max="3000000" count="30"/>
```

```
<method group="fullData" method="MPI"/>
<buffer size-MB="80" allocate-time="now"/>
```

```
</adios-config>
```

Fast Forward Program: Storage & IO

US Department of Energy; 2-year funding Collaboration: Whamcloud/Intel, HDF5 group, Cray, EMC

Goals

- Exascale storage for scientists
- Support complex analysis, increase scalability
- Fault-tolerance, data consistency and integrity

A completely redesigned IO stack for Exascale

- · Objects instead of files
 - Array objects for semantic storage of multi-dimensional data
 - Blob objects for traditional sequences of bytes
 - Key-value stores for smaller get/put operations
- Containers instead of directories
 - Snapshots for efficient COW across sets of objects
 - Transactions for atomic operations across sets of objects
- · List IO all the way through the stack
 - Reduce trips across network
- Everything fully asynchronous
 - · Reads, writes, commits, unlink, etc
- · Explicit Burst Buffer management exposed to app
 - Migrate, purge, pre-stage, multi-format replicas, semantic resharding
- End-to-end data integrity
 - Checksums stored with data, app can detect silent data corruption

Fast Forward I/O and Storage

High Performance Data Division



Source: Presentation DOE Storage Fast Forward Quick Overview and

Programming API's/Vignettes by Gary Grider

Julian M. Kunkel

I/O Interfaces Will Change - Chance or Curse?

Fast Forward I/O Architecture



Fast Forward I/O and Storage

High Performance Data Division



Source: Presentation DOE Storage Fast Forward Quick Overview and

Programming API's/Vignettes by Gary Grider

I/O stack

Applications and tools

- · Query, search and analysis
 - Index maintenance
- · Data browsers, visualizers, editors
- · Analysis shipping
 - Move I/O intensive operations to data

Application I/O

- Non-blocking APIs
- Function shipping CN/ION
- · End-to-end application data/metadata integrity
- · Domain-specific API styles
 - HDFS, Posix, ...
 - OODB, HDF5, ...
 - Complex data models

Fast Forward I/O and Storage







Source: Presentation Fast Forward I/O & Storage by Eric Barton

New HDF5 Capabilities

- Asynchronous Operations
 - All HDF5 routines that touch the file add event to an "event queue" object
 - Event queues have test/wait routines that operate on all events in queue, etc.
- Transactions
 - New "transaction" API in HDF5: open, commit, abort, etc.
 - · Explicitly bundle HDF5 operations into a transaction
 - · Explicitly push/pull data between flash and disk storage
- End-to-End Integrity
 - Checksums applied to all data on CN, stored all the way to disk, verified on reads

Fast Forward I/O and Storage

High Performance Data Division

(intel)

Source: Presentation DOE Storage Fast Forward Quick Overview and

Programming API's/Vignettes by Gary Grider



- You can even start a new transaction to do metadata or data ops with trans3++ and overlap as much IO and computation, including abort.
- You can't be sure anything made it to storage until H5AOtest/wait says that transaction is secure.
- · You can control structure, async behavior, rollback, etc.

Fast Forward I/O and Storage

High Performance Data Division

(intel)

Source: Presentation DOE Storage Fast Forward Quick Overview and

Programming API's/Vignettes by Gary Grider

Exascale10

- The Exascale I/O Initiative (former EIOW)
- Goal: Development of a Middleware with advanced features
 - Complete redesign of the I/O system
 - Different back-ends (hardware, file systems)
 - Arbitrary schemas (POSIX, HDF5, Flatland, ...)
 - Abandon restrictions of POSIX in the long run
 - Guided interfaces / Behavior indicators
 - Embedded monitoring & performance optimization
- International and open initiative
 - Collaboration: Xyratex, BSC, JGU Mainz, UHH, ...
 - Driven by the needs of the community (e.g. in requirement workshops)
 - Work-in-progress
- We will prepare a white-paper for ISC

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(), ...



Current architecture



Component decomposition (source: http://eiow.org)

Summary & Conclusions

File access paradigm will change

- Transactions
- Different namespace
- Away from explicit technical hints
- Applications have to realize their consistency model
- Guided interfaces provide insight into intended behavior
- Let the storage system and infrastructure take care of
 - Data conversion
 - Data arrangement & "file" format
 - Performance optimization
 - HSM / ILM
- Take the chance to influence upcoming "standards"
 - Make sure your requirements are heard/handled

References

- Swidler, Shlomo: Read-After-Write Consistency in Amazon S3 2009, http://shlomoswidler.com/2009/12/ read-after-write-consistency-in-amazon.html
- Vogels, Werner: Eventually Consistent Communications of the ACM, 2009, http://doi.acm.org/10.1145/1435417.1435432
- DeCandia, Giuseppe et.al.: Dynamo: Amazon's Highly Available Key-value Store 2007, Amazon.com http://www.allthingsdistributed.com/ files/amazon-dynamo-sosp2007.pdf
- Brueckner, Rich: Slidecast: Eric Barton Updates Progress on Fast Forward Storage & IO Program
 2013, http://www.whamcloud.com/news/eric-barton-updatesprogress-on-fast-forward-storage-io-program
- - Barton, Eric: Fast Forward I/O & Storage 2013, Intel High Performance Data Division (Presentation)