A Semantics-Aware I/O Interface for High Performance Computing

Michael Kuhn

Scientific Computing
Department of Informatics
University of Hamburg

2013-06-18

Introduction

- Parallel Application
  - Exhibits particular access pattern
  - Usually different for every application
  - Examples: Earth system models

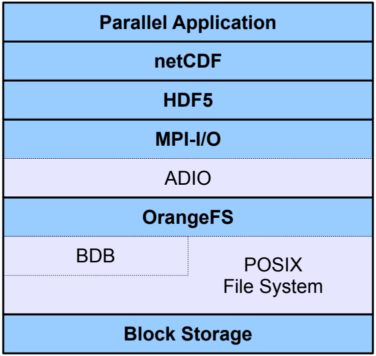
Introduction

- Parallel Application
  - Exhibits particular access pattern
  - Usually different for every application
  - Examples: Earth system models
- File system
  - Actually performs the I/O operations
  - Examples: OrangeFS, Lustre, GPFS
  - Usually optimized for specific use cases

Introduction

- Parallel Application
  - Exhibits particular access pattern
  - Usually different for every application
  - Examples: Earth system models
- File system
  - Actually performs the I/O operations
  - Examples: OrangeFS, Lustre, GPFS
  - Usually optimized for specific use cases
- Interface
  - Defines *which* accesses are possible
  - Examples: POSIX, MPI-I/O, HDF5, netCDF, ADIOS
- Semantics
  - Defines *how* accesses are handled
  - Examples: POSIX, Session, MPI-I/O
  - Sometimes single aspects are changeable (e. g. atomicity)

Motivation



- Local POSIX file system for data and metadata
  - Introduces overhead: path lookup, permissions, . . .
- Lower layers do not have information about upper ones
  - Different optimizations on each layer to use full potential

Motivation

- Semantical information about the data cannot be specified
  - Examples: “This file is accessed concurrently.” or “This is a checkpoint.”
- Information cannot be handed down in the I/O stack
  - Lower layers do not support it or it is lost through the layers
  - Optimizations have to be implemented within the upper layers

Motivation

- Semantical information about the data cannot be specified
  - Examples: “This file is accessed concurrently.” or “This is a checkpoint.”
- Information cannot be handed down in the I/O stack
  - Lower layers do not support it or it is lost through the layers
  - Optimizations have to be implemented within the upper layers
- Goal: Providing a semantics-aware I/O interface and file system prototype
  - Enough information to perform meaningful optimizations
  - Can adapt to I/O requirements of applications

Design

- Minimize the overhead during normal operation
  - Avoid POSIX file systems, which perform potentially redundant operations
  - Path lookup: Reading metadata, checking permissions, etc.
- Limited file system hierarchy
  - Divided into *stores*, *collections*, and *items*
  - Stores include collections, collections include items
- Accesses performed via so-called *batches*
  - Each one can consist of multiple I/O operations
  - Example: Create multiple items in one batch
  - Knowledge about all operations can be used for optimizations

Design

- Specify the semantics of file system operations at runtime
  - Atomicity, concurrency, consistency, persistency and safety
  - Can be changed on a per-batch basis

Design

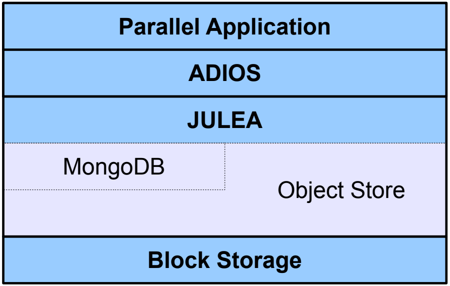
- Specify the semantics of file system operations at runtime
  - Atomicity, concurrency, consistency, persistency and safety
  - Can be changed on a per-batch basis

```
batch = new Batch(POSIX_SEMANTICS);

for (i = 0; i < 1000; i++)
{
    item = new Item("Test" + i);
    batch.add(collection.add(item));
}

batch.execute();
```

JULEA Architecture



- Less layers and less duplication of functionality
- ADIOS allows describing application I/O via an XML file
  - Code is generated automatically
  - Could be extended to specify additional semantical information

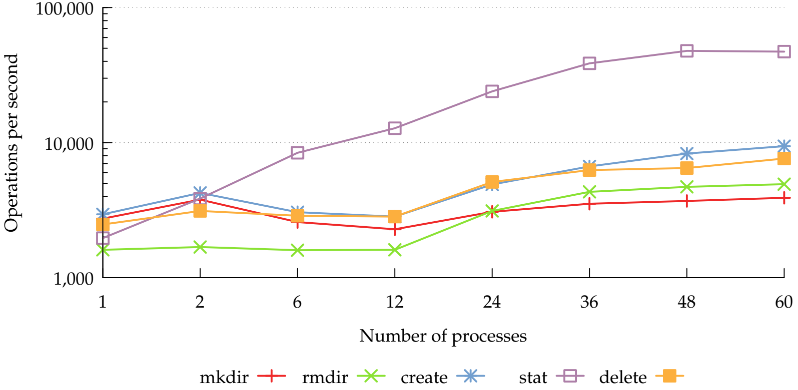
Implementation

- Metadata stored in MongoDB
  - NoSQL database systems usually scale well
  - Still depends on an underlying POSIX file system
- Support for multiple data back ends
  - POSIX, GIO and NULL are implemented
  - Object store back end is in progress
- Built-in support for tracing client and server activities
  - OTF and HDTrace are implemented
  - Analyze and visualize the inner workings
- Run-time statistics are collected and exported

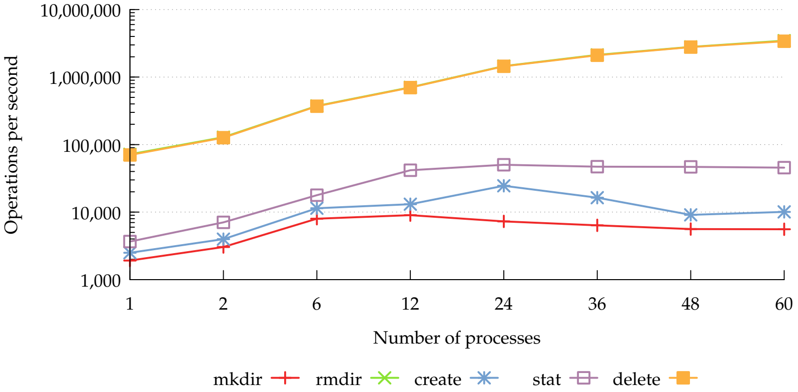
Setup

- Metadata benchmark using `fileop`
  - Part of IOzone
  - Extended to support the native file system interfaces
- Only most interesting metadata-heavy operations
  - `mkdir`, `rmdir`, `create`, `stat` and `delete`

Lustre

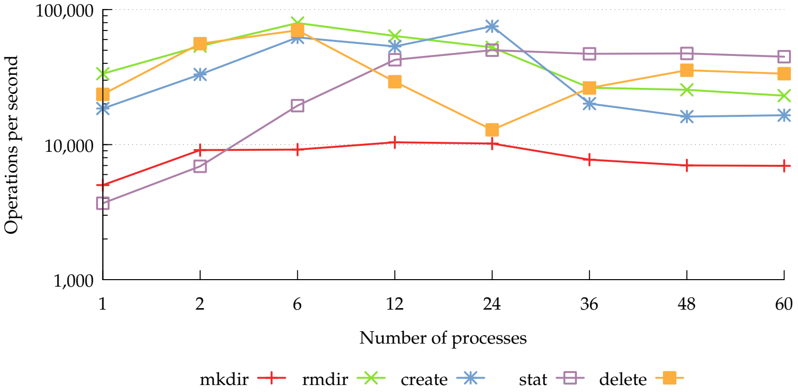

100,000
10,000
1,000
Operations per second
1
2
6
12
24
36
48
60
Number of processes
mkdir
rmdir
create
stat
delete

JULEA



10,000,000
1,000,000
100,000
10,000
1,000
Operations per second
1
2
6
12
24
36
48
60
Number of processes
mkdir
rmdir
create
stat
delete

JULEA (Batch)


100,000
10,000
1,000
Operations per second
1
2
6
12
24
36
48
60
Number of processes
mkdir
rmdir
create
stat
delete

Conclusion

- Current interfaces do not provide ways to specify the I/O requirements of individual applications
  - Semantical information could be used for optimizations
- New semantics-aware I/O interface and file system prototype
  - Allows application-specific semantics to be specified
  - Multiple operations can be aggregated in batches
- Specify *what* to do and *how* to behave
  - Leave the actual realization to the I/O system
  - Provide sane default semantics and templates

Future Work

- Look at requirements of more real-world applications
  - Evaluate potential benefits
- Extend ADIOS to support the new I/O interface
  - Allow semantical information to be specified
- Investigate transaction support
  - Batches might be a suitable granularity
- Allow semantics to be implemented in plug-ins
  - Optimize different aspects for specific hardware/software environments