Performance evaluation of the PVFS-2 architecture

Julian M. Kunkel

Institute for Computer Science
University Heidelberg

PDP 2007

Outline

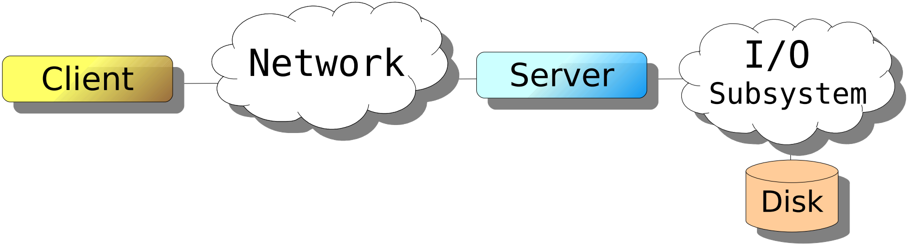
1. Introduction
2. Overview of PVFS2
3. Performance Limitations
4. Realization with TAS
5. Results
6. Summary

Outline

1. Introduction
   - Motivation
   - Idea
2. Overview of PVFS2
   - Overview
   - Architecture
3. Performance Limitations
4. Realization with TAS
5. Results
   - Metadata
   - Small Contiguous I/O Requests
   - Large Contiguous I/O Requests
6. Summary

Motivation

- Performance of a parallel file system depends on capabilities of participating components
  - Network
  - I/O subsystem
  - Client and server machines
  - The parallel file system



- Performance characteristics and performance optimizations of components interfere with each other

Motivation

Example:

- Performance of the I/O subsystem depends on multiple factors:
  - Location of the file's blocks on disk and the disk's current head position
  - File system fragmentation
  - Caching strategies
  - RAID level if appropriate
  - ...
- Typically the I/O subsystem greatly limits the aggregated performance

As a consequence it is non-trivial to:

- develop meaningful benchmarks
- assess measured performance
- tune a parallel file system
- discover of performance bottlenecks (and bugs) in the architecture

It is important to reduce the complexity for these tasks!

Idea

Replace layers of a parallel file system with efficient stubs

- Upper bound for throughput of replaced layers
- Measure overhead of upper layers
- Evaluate lower layers
- Performance regression tests to show impact of modifications
- Persistency layer is a good candidate
  - replace it by a dummy which pretends to manage data correctly
  - most benchmarks can be applied
  - answer the questions:
    - Is the parallel file system able to saturate the network ?
    - Does a modification of a specific network parameter increase the throughput for a test-case ?
    - Is a bottleneck in the persistency layer ?
    - ...

Idea

- Why not run on a in-memory file system like tmpfs ?
  - Memory is limited
  - The persistency layer itself might be complex, thus locating the reason for a discovered bottleneck in the code might be not easy
  - The persistency layer may incorporate caching mechanisms
  - For analysis it might be easier to replace the handling in the layer with algorithms and data structures which have a well known complexity
  - But, tmpfs is a good candidate for comparison with original and replaced layer

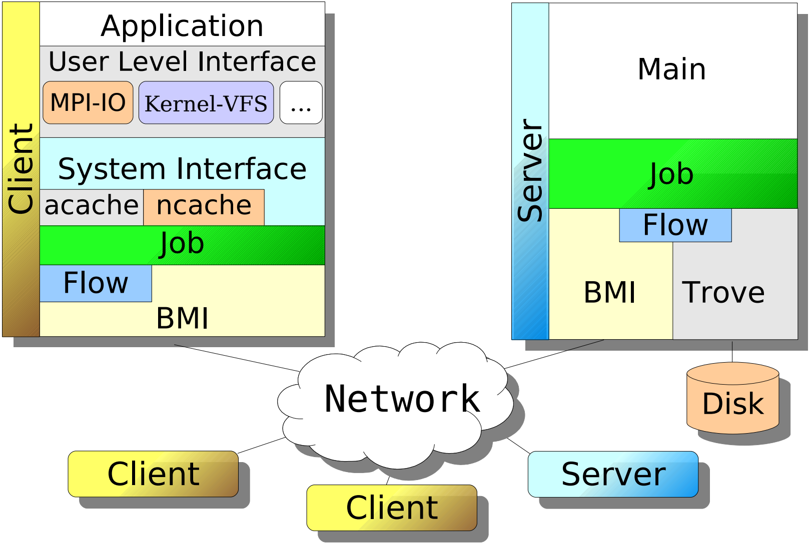
Outline

1. Introduction
   - Motivation
   - Idea
2. Overview of PVFS2
   - Overview
   - Architecture
3. Performance Limitations
4. Realization with TAS
5. Results
   - Metadata
   - Small Contiguous I/O Requests
   - Large Contiguous I/O Requests
6. Summary

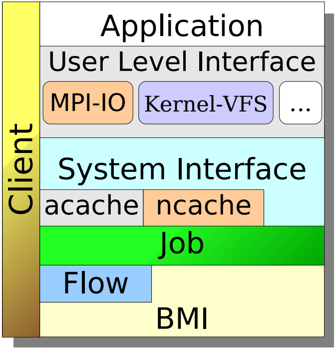
Overview of PVFS2

- Open source
- Redevelopment of the Parallel Virtual File System
- Developed and maintained at Argonne National Lab and Clemson University
- Tightly integrated into MPI-IO (drivers available in ROMIO/MPICH-2)
- Servers can be configured to be data and/or metadata servers
- Data is typically striped over the data servers in 64 KByte chunks (RAID-0)

Architecture of PVFS2



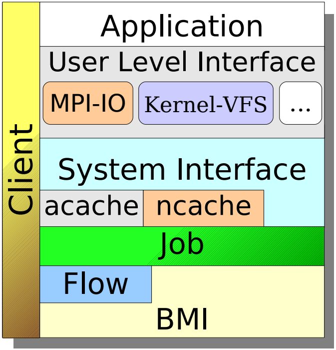
Architecture of PVFS2 - Client



Description of the layers

- User-level-interface
  - Integration into linux VFS for POSIX access
  - ROMIO module available in MPICH2
- System Interface
  - Provides API for manipulation of file system objects
  - Contains caches for directory hierarchy and object attributes
- Job
  - Thin layer, combines and controls lower layers

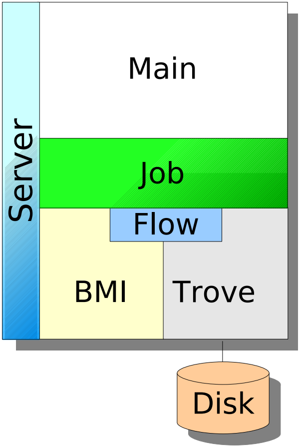
Architecture of PVFS2 - Client



Description of the layers

- Flow
  - Reliable transfer of data between two endpoints
  - Defines data flow policy e.g. parallel streams
- BMI
  - Network interface
  - TCP, Myrinet, IB, ...

Architecture of PVFS2 - Server



Description of the layers

- Main process
  - Accepts new requests
  - Starts server statemachines
- Trove
  - Persistency layer
  - Implementation uses Berkeley DB and local file system.

Outline

1. Introduction
   - Motivation
   - Idea
2. Overview of PVFS2
   - Overview
   - Architecture
3. Performance Limitations
4. Realization with TAS
5. Results
   - Metadata
   - Small Contiguous I/O Requests
   - Large Contiguous I/O Requests
6. Summary

Simple Model

Performance limitations

- CPU
  - Use hash tables etc. $\Rightarrow$ constant time needed per request
  - Limits the number of requests
- Input/Output subsystem
  - Access time
  - Throughput
- Network
  - Latency
  - Bandwidth > Throughput

- Estimate and compare performance with measured throughput

Performance implications of the PVFS2 architecture

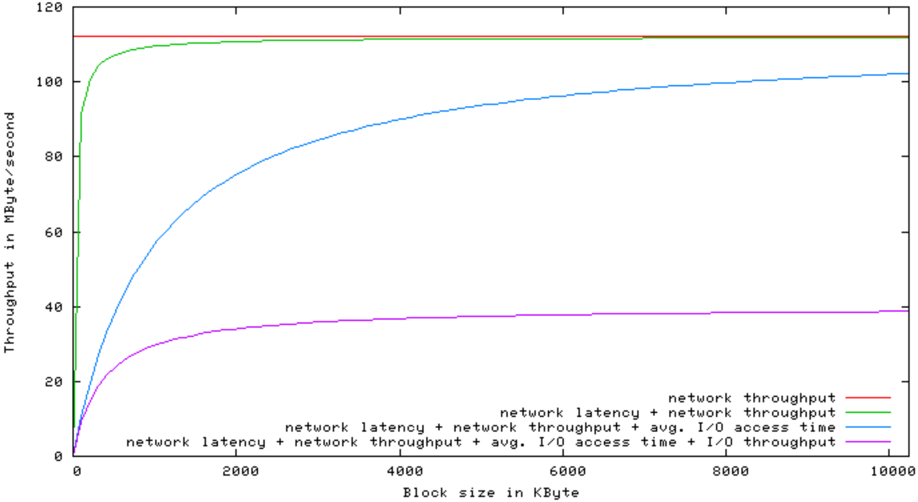
I/O

- No client side cache for data $\Rightarrow$ each I/O operation requires at least one message exchange
- Small I/O requests are transfered with initial requests (Read) or response (Writes)
- Larger I/O requests require rendezvous protocol $\Rightarrow$ extra round-trip for writes

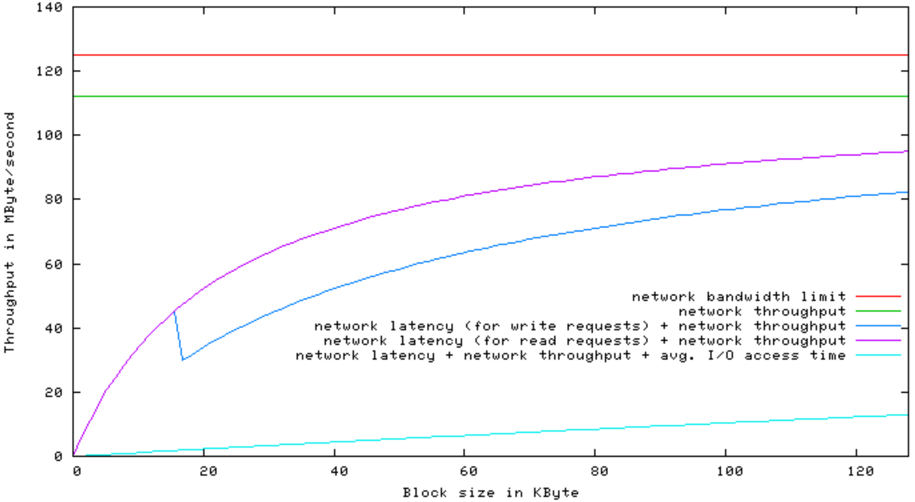
Metadata

- Read-only operations are cached for a small time frame
- Modifying operations typically consist of multiple requests mostly processed in serial e.g. creation with MPI needs 4 requests to metadata servers and one for each participating data server
- Each request requires one message exchange

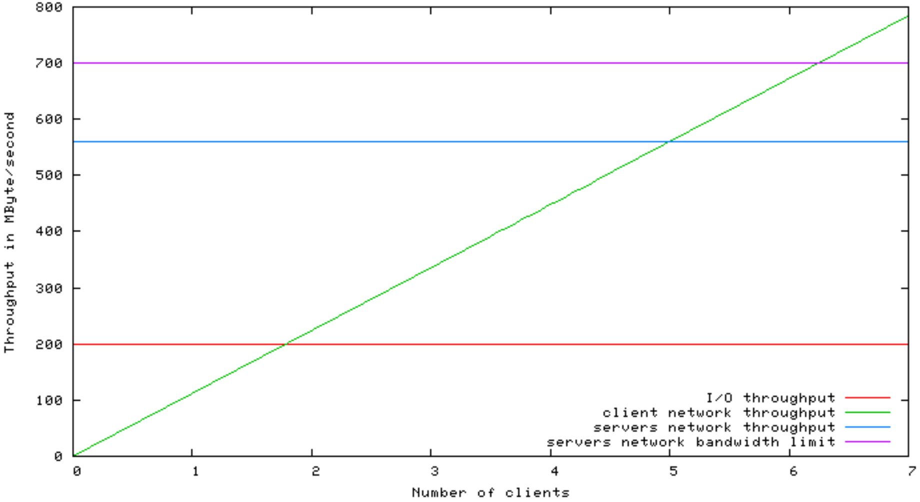
Estimated performance for small contiguous I/O requests



Estimated performance for small contiguous I/O requests



Estimated performance for large contiguous I/O requests



Outline

1 Introduction
- Motivation
- Idea

2 Overview of PVFS2
- Overview
- Architecture

3 Performance Limitations

4 Realization with TAS

5 Results
- Metadata
- Small Contiguous I/O Requests
- Large Contiguous I/O Requests

6 Summary

Realization with TAS

- Replaces persistency layer (TROVE) with stub
- Handle metadata correctly
- Use a red-black-tree as basic data structure
- Discard data from I/O requests and signal completion
- Return immediately from requests (No sperate threads)

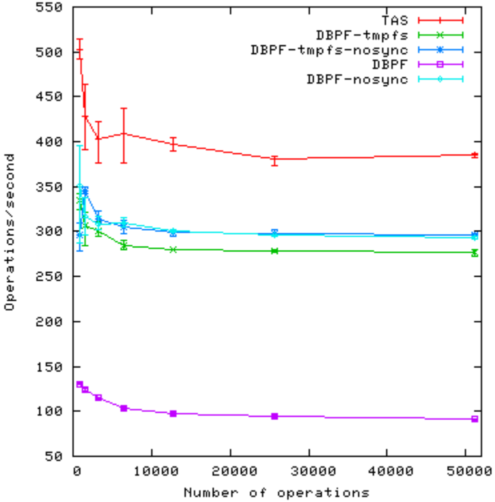
Outline

1. Introduction
   - Motivation
   - Idea
2. Overview of PVFS2
   - Overview
   - Architecture
3. Performance Limitations
4. Realization with TAS
5. Results
   - Metadata
   - Small Contiguous I/O Requests
   - Large Contiguous I/O Requests
6. Summary

Benchmarking Program for Metadata

- MPI program
- Operates in one directory
- Each client creates a disjoint set of files with MPI_MODE_CREATE
- Runtime is measured on each process and maximum time used to calculate metadata throughput in operations/second.

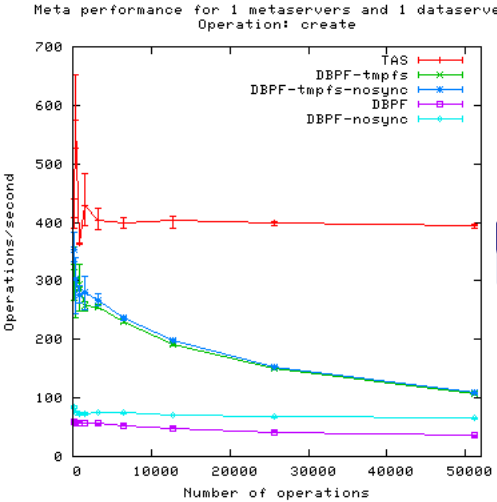
One data server, one metadata server, one client



Notes

- Data is from July 06.
- DBPF is the current persistency implementation
- DBPF normally synchronizes modifications to disk, thus mainly limited by disk latency (and throughput)
- Nosync refers to a variant where metadata synchronization with the I/O subsystem is omitted

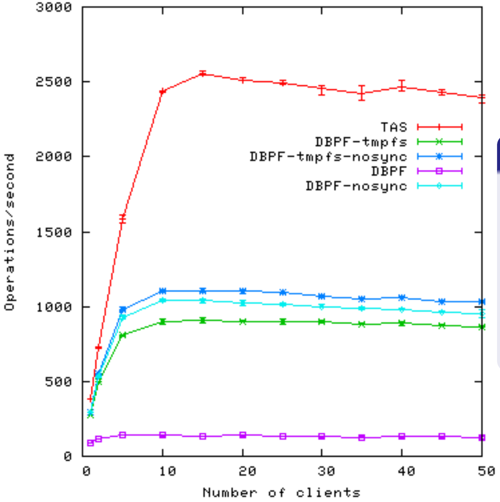
One data server, one metadata server, one client



Notes

- Data is from December 05.

One data server, one metadata server



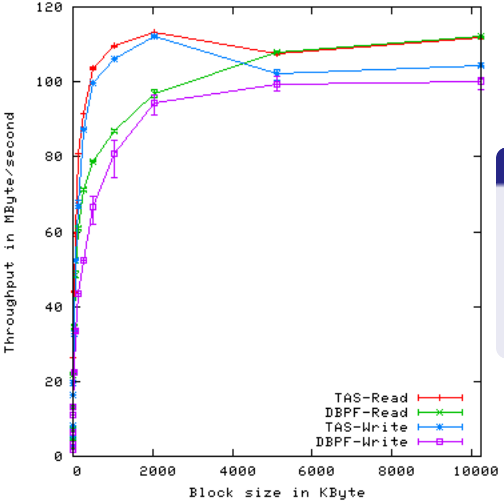
Notes

- Multiple clients creating a total of 51200 files
- DBPF tmpfs and nosync results are close together like expected

Benchmarking Program for I/O Throughput

- MPI program (mpi-io-test)
- Operates on one file
- Each client
  - opens the file individually
  - writes a number of blocks of the same size with MPI_File_write
  - opens the file again
  - reads the data in chunks back
- The processes synchronize between two I/O operations
- Time is measured for each I/O operation and maximum taken for calculation

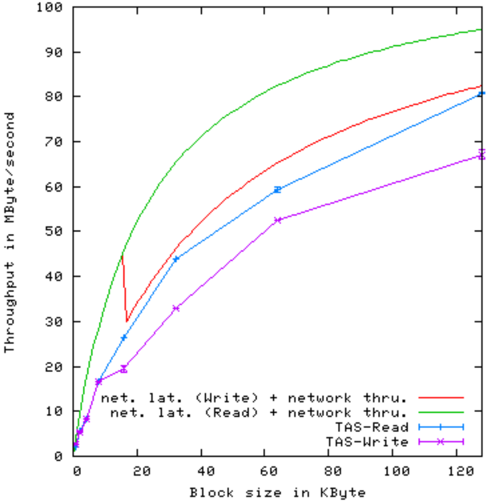
1 Client, 1 Data servers



Notes

- Total file size: 100 MByte
- Should be cached well by OS
- TAS forms a upper bound
- Close to network bandwidth

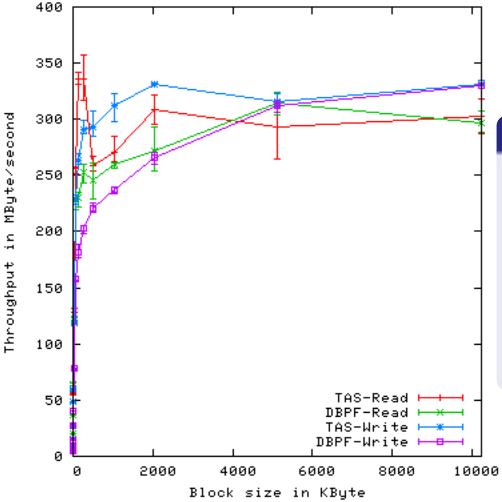
1 Clients, 1 Data server



Notes

- Estimated performance limits asymptoticly close to measured performance
- Drop due to handshake can be seen for write
- Caching strategies could boost throughput

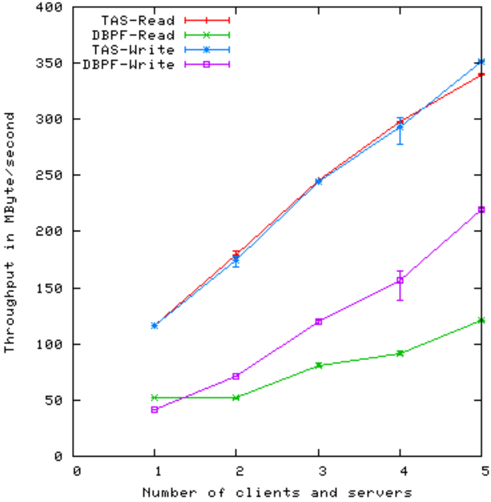
5 Clients, 5 Data servers



Notes

- TAS write performance > read performance
- Can be seen for DBPF with 10 MByte blocks
- About 3 times performance of 1 Client and 1 Data server

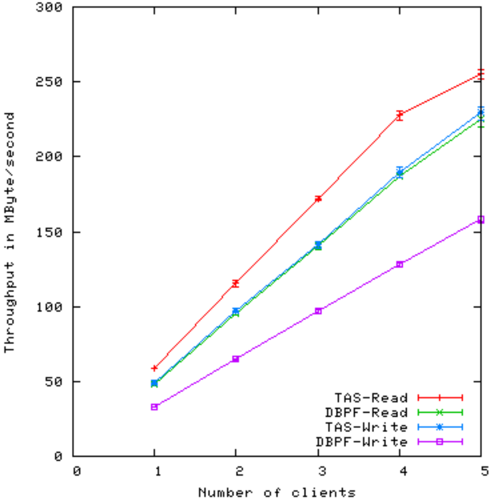
Variable number of clients and data servers



Notes

- 12800 MByte accessed in 10 MByte chunks
- Disjoint clients and servers
- Does not scale linear, network technology (Ethernet, star topology) ?
- I/O throughput does not scale for reads

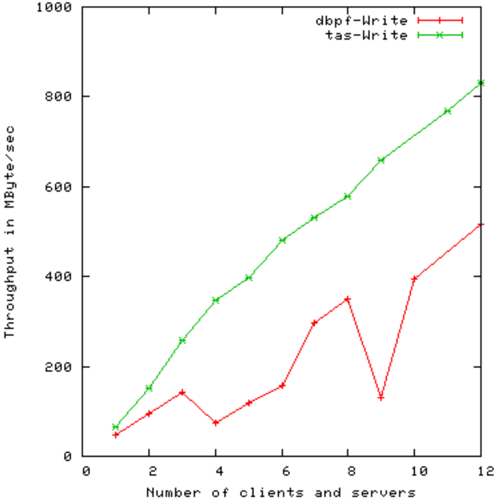
Variable number of clients and data servers



Notes

- Disjoint clients and servers
- 100 MByte accessed in 64 KByte chunks
- Good caching behavior expected
- Performance scales linear

Chiba 150 MByte per client



Notes

- Dual PIII 500 MHz, 512 MByte memory, Myrinet 2000
- Effective throughput measured between two nodes: 90 Mbytes
- Chiba hardware is old, hardware problems with myrinet interconnection, high packet loss
- Only qualitative analysis

Outline

1. Introduction
   - Motivation
   - Idea
2. Overview of PVFS2
   - Overview
   - Architecture
3. Performance Limitations
4. Realization with TAS
5. Results
   - Metadata
   - Small Contiguous I/O Requests
   - Large Contiguous I/O Requests
6. Summary

Summary

- A performance reference is useful for comparison and evaluation
- Analysis stubs can be used to evaluate performance
- Method allows to reduce complexity of analysis
- Regression tests with analysis stubs reveal performance differences more detailed