Towards a Grid File System Based on a Large-Scale BLOB Management Service

Viet-Trung Tran[1], Gabriel Antoniu[2], Bogdan Nicolae[3], Luc Bougé[1], Osamu Tatebe[4]

[1] ENS Cachan - Brittany, France
[2] INRIA Centre Rennes - Bretagne-Atlantique, France
[3] University of Rennes 1, France
[4] University of Tsukuba, Japan

New Challenges for Large-scale Data Storage

Scalable storage management for new-generation, data-oriented high-performance applications

- Massive, unstructured data objects (Terabytes)
- Many data objects (10^3)
- High concurrency (10^3 concurrent clients)
- Fine-grain access (Megabytes)
- Large-scale distributed platform: large clusters, grids, clouds, desktop grids

Applications: distributed, with high throughput under concurrency

- E-science Data-centric applications
- Storage for cloud services
- Map-Reduce-based data mining
- Checkpointing on desktop grids

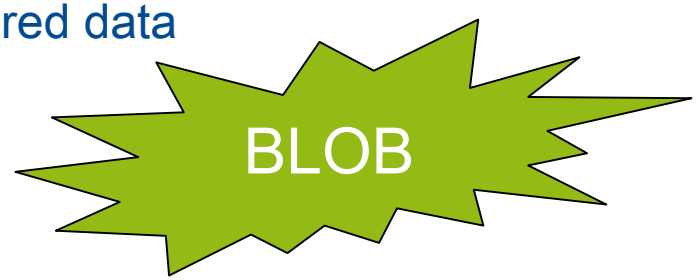
BlobSeer: a BLOB-based Approach

Developed by the KerData team at INRIA Rennes

- Recently created from the PARIS project-team

Generic data-management platform for huge, unstructured data

- Huge data (TB)
- Highly concurrent, fine-grain access (MB): R/W/A
- Prototype available



Key design features

- Decentralized metadata management
- Beyond MVCC: multiversioning exposed to the user
- Lock-free write access through versioning
  - Write-once pages

A back-end for higher-level, sophisticated data management systems

- Short term: highly scalable distributed file systems
- Middle term: storage for cloud services
- Long term: extremely large distributed databases

http://blobseer.gforge.inria.fr/

BlobSeer: Design

http://blobseer.gforge.inria.fr

Each blob is fragmented into equally-sized “pages”

- Allows huge data amounts to be distributed all over the peers
- Avoids contention for simultaneous accesses to disjoint parts of the data block

Metadata : locate pages that make up a given blob

- Fine-grained and distributed
- Efficiently managed through a DHT

Versioning

- Update/append: generate new pages rather than overwrite
- Metadata is extended to incorporate the update
- Both the old and the new version of the blob are accessible

BlobSeer: Architecture

Clients

- Perform fine grain blob accesses

Providers

- Store the pages of the blob

Provider manager

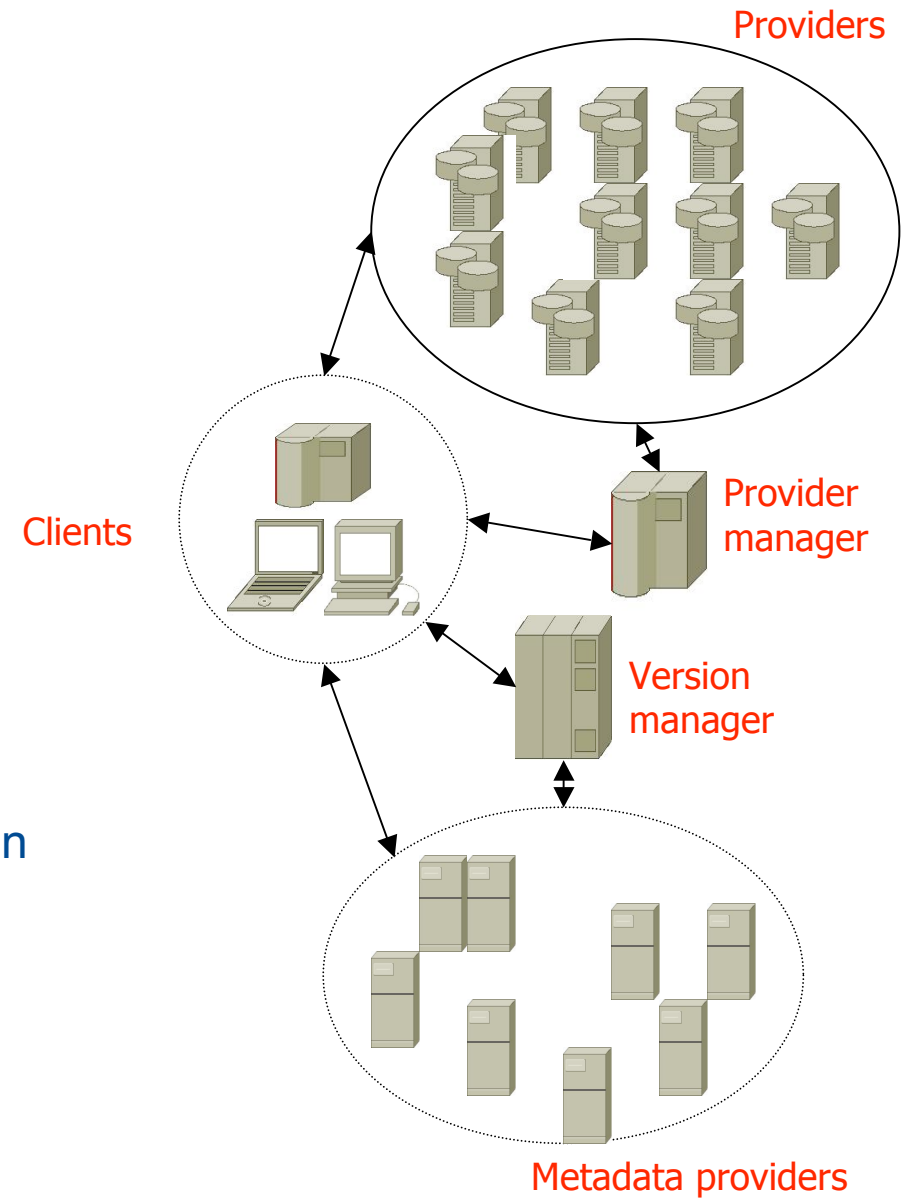
- Monitors the providers
- Favors data load balancing

Metadata providers

- Store information about page location

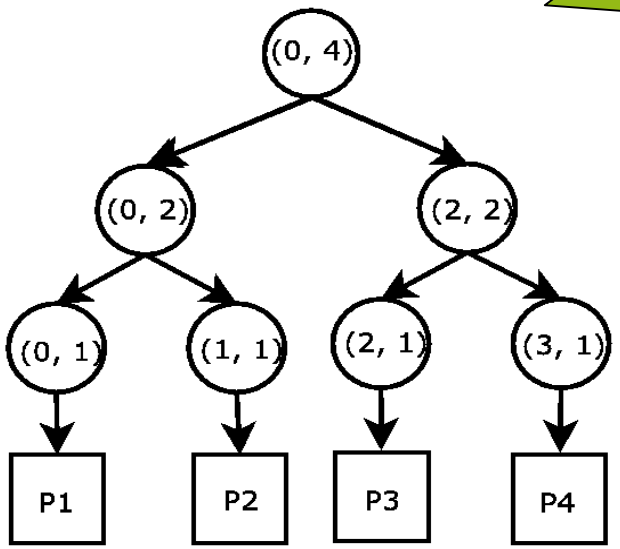
Version manager

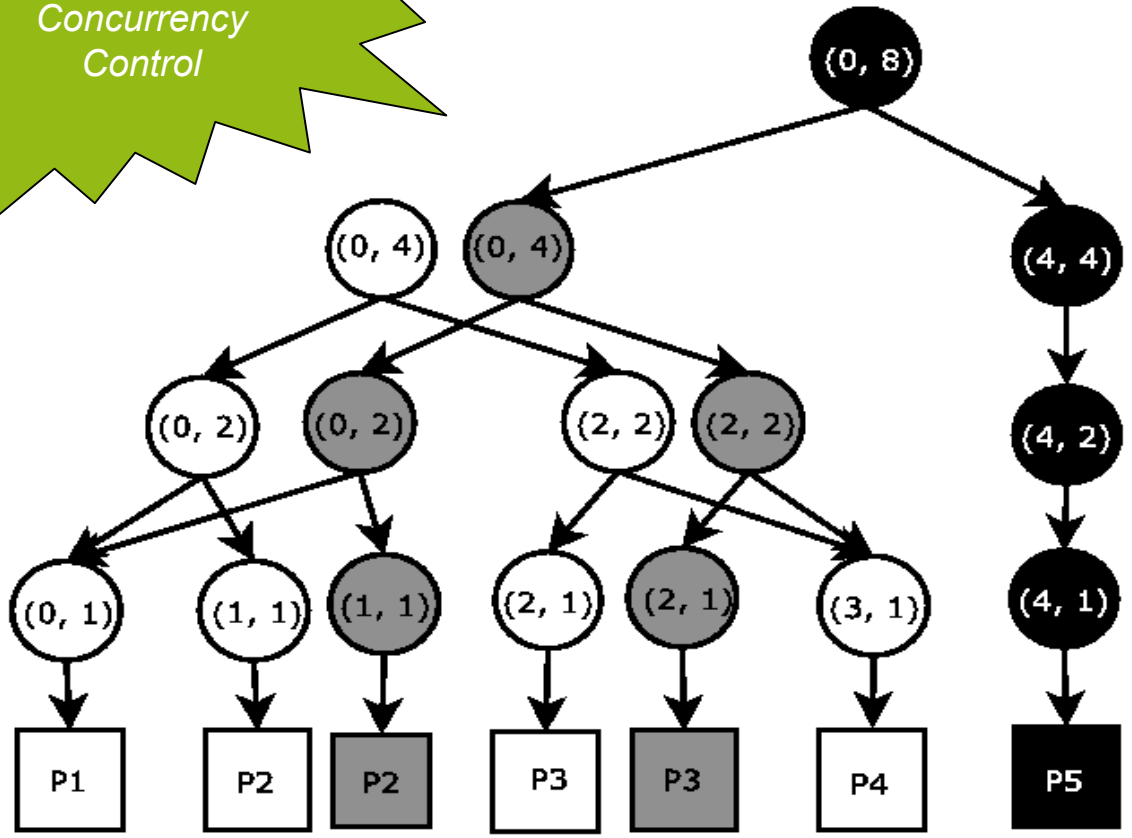
- Ensures concurrency control



Version Management in BlobSeer

Beyond Multiversion Concurrency Control





Background: Object-based File Systems

From block-based to object-based file systems

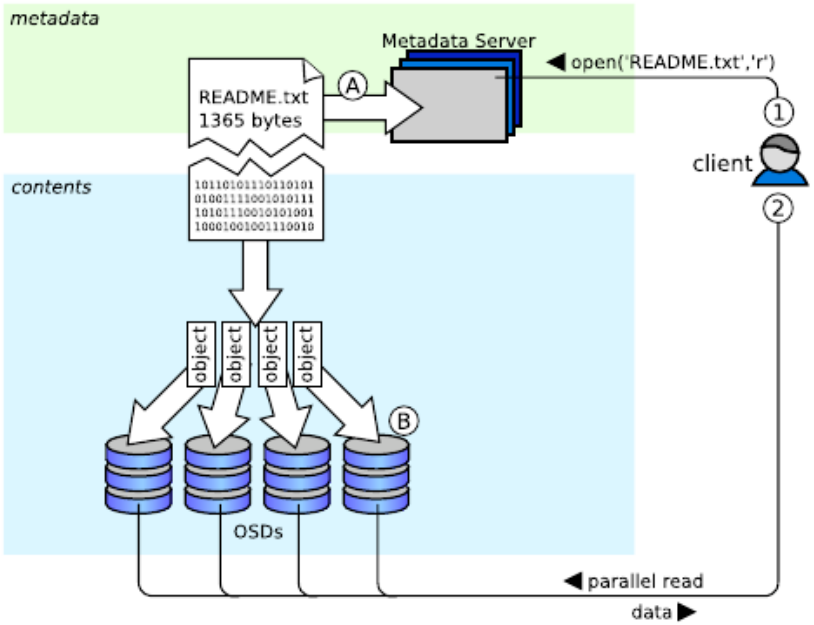
- Block management at the file server
- Object management is delegated to object-based storage devices (OSDs) or storage servers

Advantages

- Scalability: offload ~90% of workload from the metadata server
- OSDs and storage servers are more autonomous, self-managed and easier to share

Examples of object-based file systems

- Lustre (Schwan, 2003)
- Ceph (Weil et al., 2006)
- GoogleFS (Ghemawat et al., 2003)
- XtreemFS (Hupfeld et al., 2008)

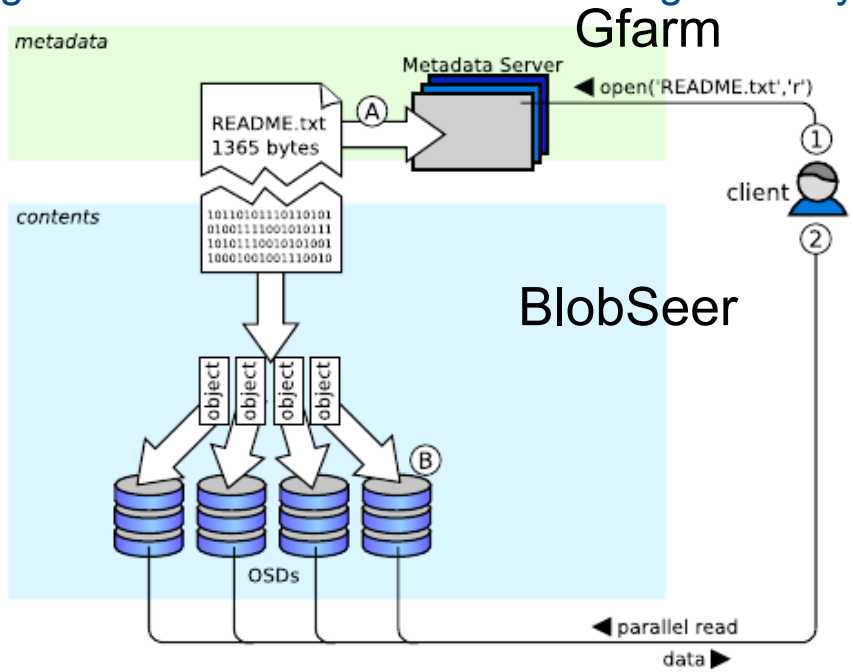


Towards a BLOB-based File System

Goal: Build a BLOB-based file system, able to cope with huge data and heavy access concurrency in a large-scale distributed environment

Hierarchical approach

- High-level file system metadata management: the Gfarm grid file system
- Low-level object management: the BlobSeer BLOB management system



The Gfarm Grid File System

The Gfarm file system [University of Tsukuba, Japan]

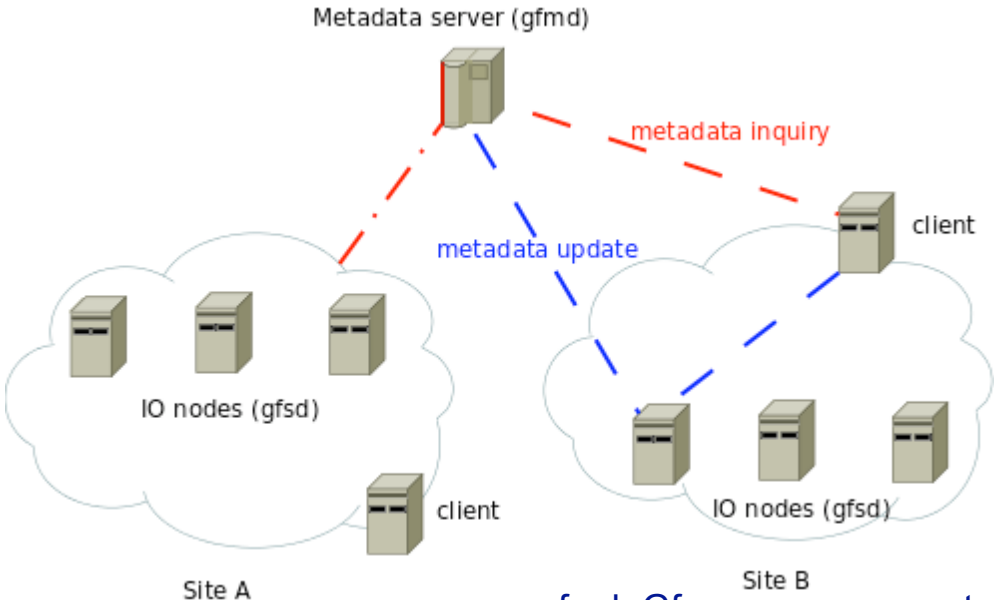
A distributed file system designed for working at the Grid scale

File can be shared among all nodes and clients

Applications can access files using the same path regardless of the file location

Main components

- Gfarm's metadata server
- File system nodes
- Gfarm clients



gfmd: Gfarm management daemon
gfsd : Gfarm storage daemon

The Gfarm Grid File System [2]

Advanced features

- User management
- Authentication and single sign-on based on Grid Security Infrastructure (GSI)
- POSIX file system API (gfarm2fs) and Gfarm API

Limitations

- No file striping, thus file size is limited
- No access concurrency
- No versioning capability

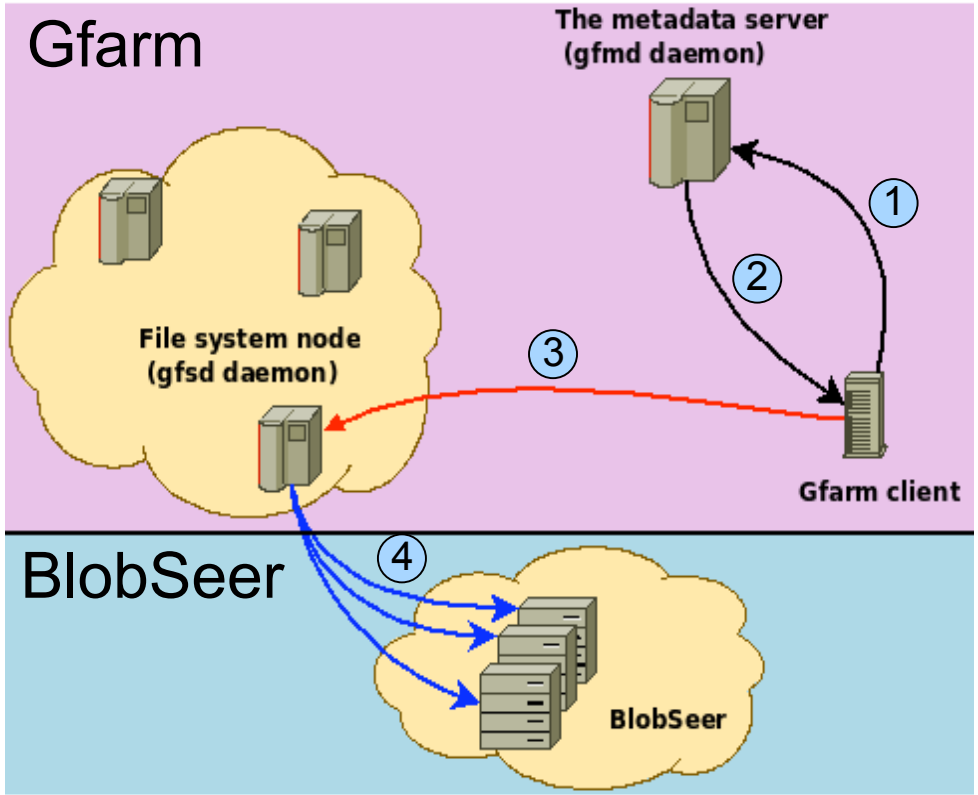
Why combine Gfarm and BlobSeer?

Gfarm	BlobSeer	Gfarm/BlobSeer
POSIX interface User management Support GSI	Lack of POSIX file system interface	POSIX interface User management Support GSI
File sizes are limited Not suitable for concurrent access No versioning	Access concurrency Fine-grain access Versioning	Access concurrency Huge file sizes Fine-grain access Versioning

General idea: Gfarm handles file metadata, BlobSeer handles file data

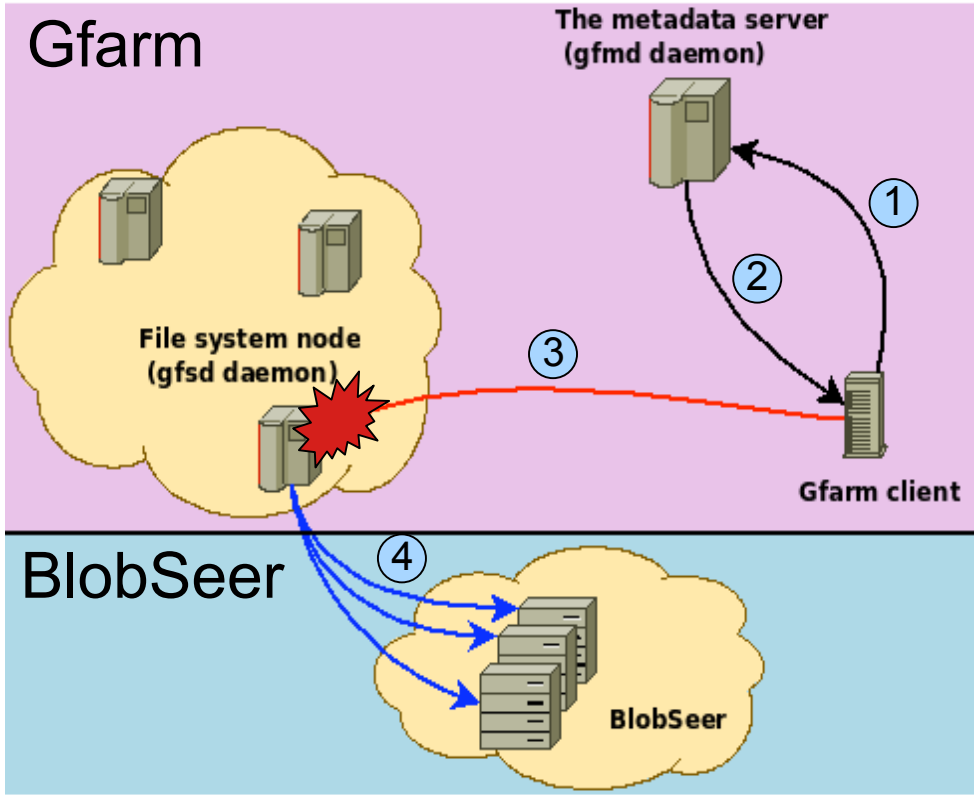
Coupling Gfarm and BlobSeer [1]

- The first approach
  - Each file system node (gfsd) connects to BlobSeer to store/get Gfarm file data
  - It must manage the mapping from Gfarm files to BLOBs
  - It always acts as an intermediary for data transfer
  - Bottleneck at the file system node



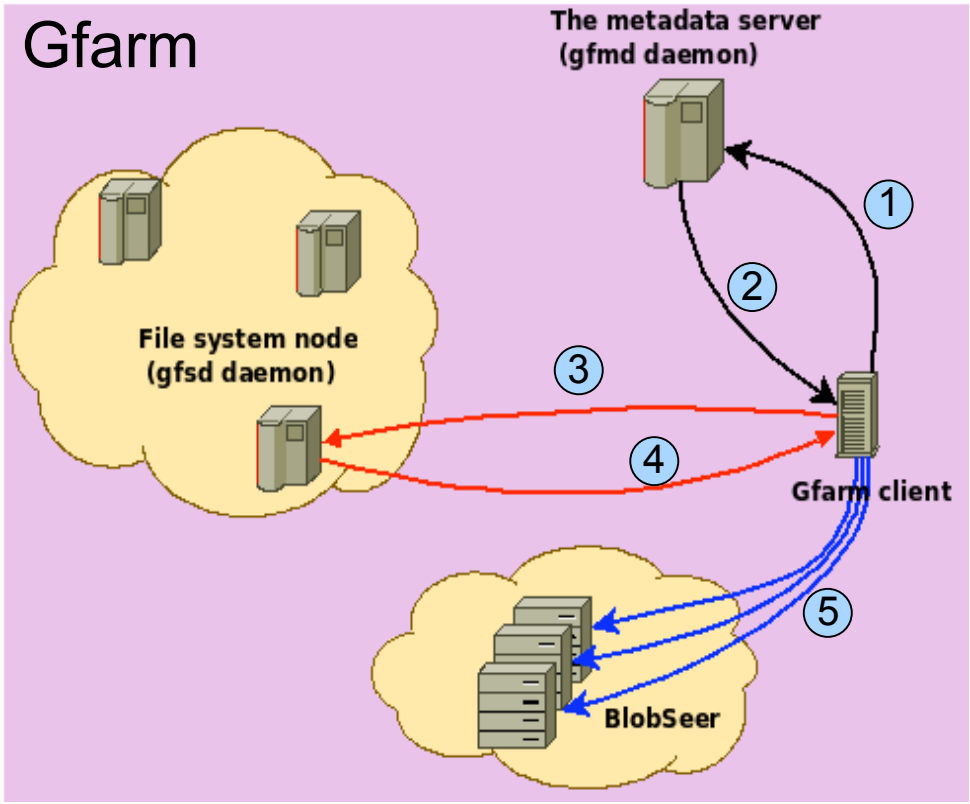
Coupling Gfarm and BlobSeer [1]

- The first approach
  - Each file system node (gfsd) connects to BlobSeer to store/get Gfarm file data
  - It must manage the mapping from Gfarm files to BLOBs
  - It always acts as an intermediary for data transfer
  - Bottleneck at the file system node



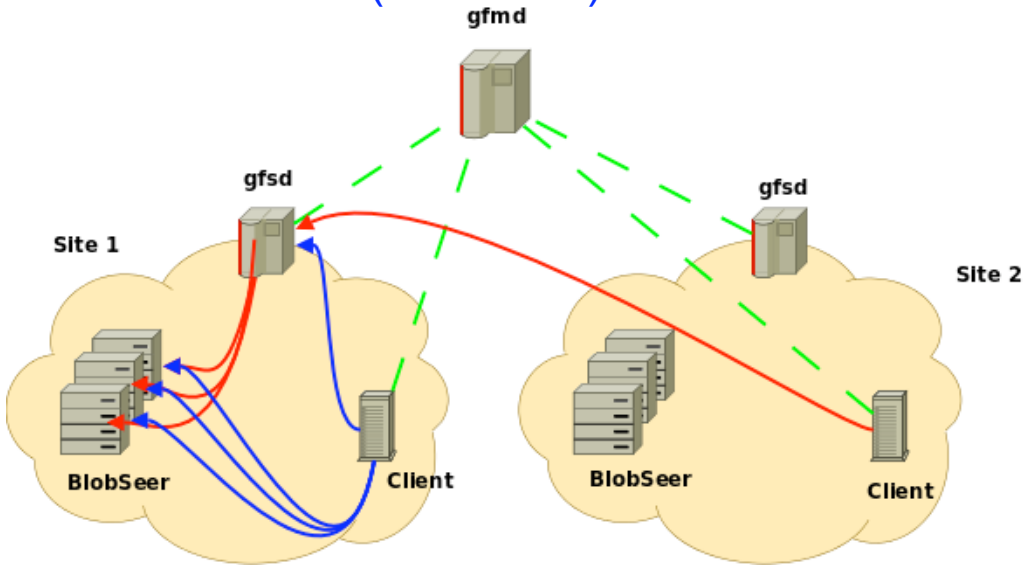
Coupling Gfarm and BlobSeer [2]

- Second approach
  - The gfsd maps Gfarm files to BLOBs, and responds the client's request within the BLOB ID
  - Then, the client directly access data in BlobSeer



Gfarm/BlobSeer: Design Considerations

- The whole system is dimensioned for a multi-site Grid
  - One BlobSeer instance at each site
- The client can **dynamically** switch between these two well defined access modes
  - Remote access mode (red lines): if the client cannot directly access BlobSeer
  - BlobSeer direct access mode (blue lines): the clients can directly access BlobSeer

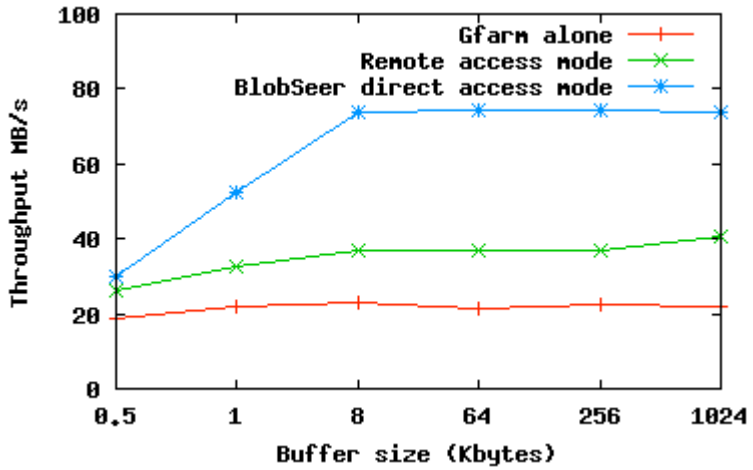


Discussion

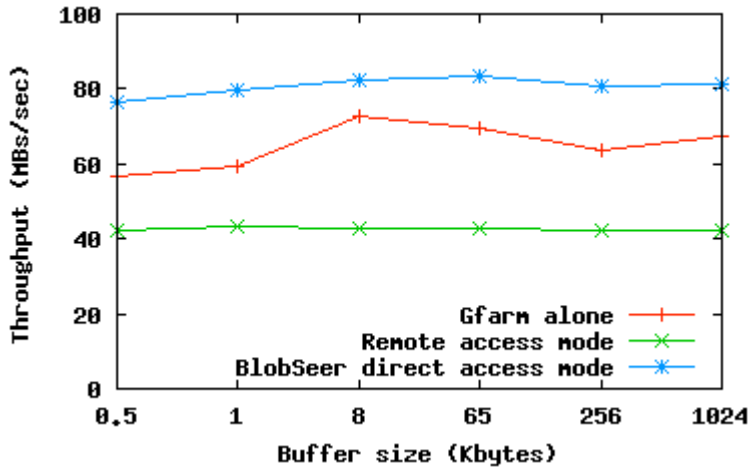
- The integrated system can support huge file sizes since Gfarm file is now transparently striped over BlobSeer data providers.
- Remote access mode
  - Only required modifying the source code of the gfsd daemon
  - Could not resolve the current bottleneck problem on the gfsd daemon
- BlobSeer direct access mode
  - Supports concurrent accesses
  - Was more complicated in implementation, a new access mode had to be introduced into Gfarm

Experimental Evaluation on Grid'5000 [1]

- Access throughput with no concurrency
- Gfarm benchmark measures throughput when writing/reading
- Configuration: 1 gfmd, 1 gfsd, 1 client, 9 data providers, 8MB page size
- BlobSeer direct access mode provides a higher throughput



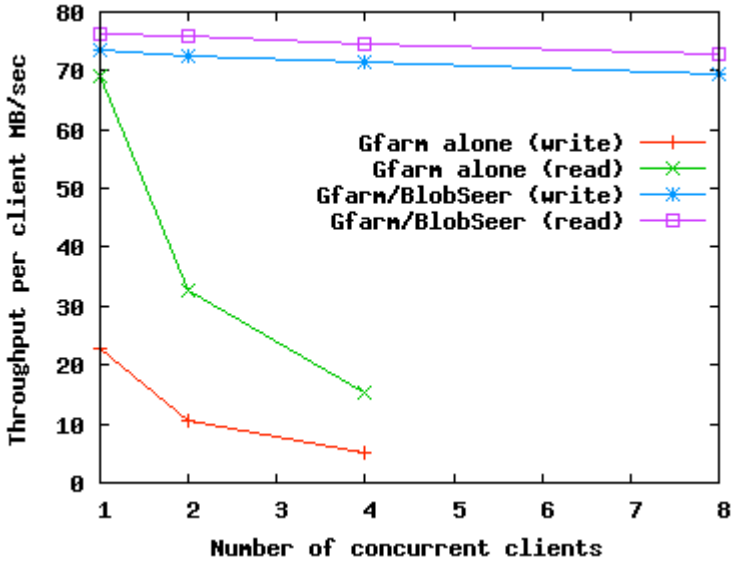
Writing



Reading

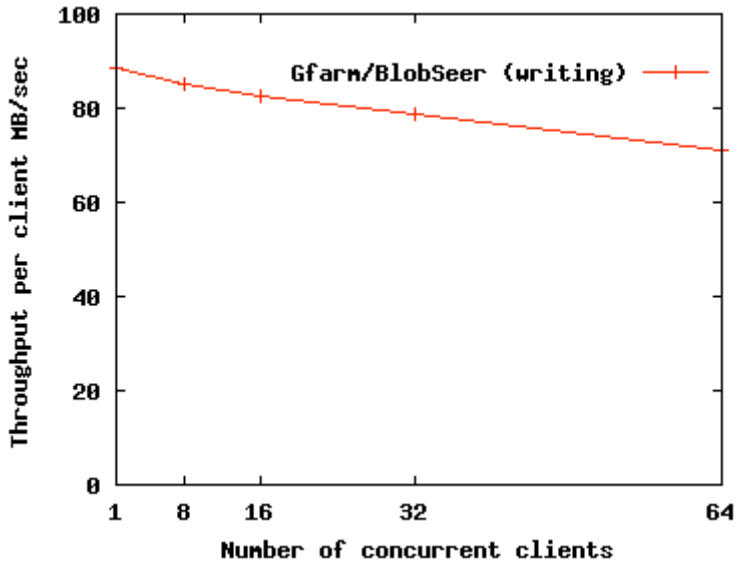
Experimental Evaluation on Grid'5000 [2]

- Access throughput under concurrency
- Configuration
  - 1 gfmd
  - 1 gfsd
  - 24 data providers
  - Each client accesses 1GB of a 10GB file
  - Page size 8MB
- Gfarm sequentializes concurrent accesses



Experimental evaluation on Grid'5000 [3]

- **Access throughput under heavy concurrency**
- Configuration (deployed on 157 nodes of Rennes site)
  - 1 gfmd
  - 1 gfsd
  - Each client accesses 1GB of a 64GB file
  - Page size 8MB
  - Up to 64 concurrent clients
  - 64 data providers
  - 24 metadata providers
  - 1 version manager
  - 1 page manager



Introducting Versioning in Gfarm/BlobSeer

- Versioning is the capability of accessing data of a specified file version
- Not only to rollback data when desired, but also to access different file versions within the same computation
- Favors efficient access concurrency
- Approach
  - Delegate versioning management to BlobSeer
  - A Gfarm file is mapped to a single BLOB
  - A file version is mapped to the corresponding version of the BLOB

Difficulties

Gfarm does not support versioning

Some issues we had to deal with:

New extended API for clients

A coordination between clients and gfsd daemons via new RPC calls

Modification of the inner data structures of Gfarm in order to handle file versions

Versioning is not a standard feature of POSIX file systems

Versioning interface

- Versioning capability was fully implemented
- At Gfarm API level
  - gfs_get_current_version(GFS_File gf,size_t *nversion)
  - gfs_get_latest_version(GFS_File gf,size_t *nversion)
  - gfs_set_version(GFS_File gf,size_t version)
  - gfs_pio_vread(size_t nversion,GFS_File gf, void *buffer, int size, int *np)
- At POSIX file system level
  - Defined some ioctl commands

```
fd = open(argv[1], 0_RDWR);

np = pwrite(fd, buffer_w, BUFFER_SIZE,0);
ioctl(fd, BLOB_GET_LATEST_VERSION, &nversion);

ioctl(fd, BLOB_SET_ACTIVE_VERSION, &nversion);
np = pread(fd, buffer_r, BUFFER_SIZE,0);

ioctl(fd, BLOB_GET_ACTIVE_VERSION, &nversion);
close(fd);
```

Conclusion

- The experimental results suggest that our prototype well exploited the combined advantages of Gfarm and BlobSeer.
  - A file system API
  - Concurrent accesses
  - Huge file sizes (tested up to 64GB file)
  - Versioning
- Future work
  - Experiment the system on a more complex topology as several sites of Grid'5000, and with a WAN network
  - Ensure the semantic of consistency since Gfarm has not yet maintained cache coherence between buffers on clients
  - Compare our prototype to other Grid file systems

Application scenarios

Grid and cloud storage for data-mining applications with massive data

Distributed storage for large-scale Petascale computing applications

Storage for desktop grid applications with high write-throughput requirements

Storage support for extremely large databases