The ExaStore™ File System: Overview

WHITE PAPER

FEBRUARY 2007

EXANET

BREAK THE STORAGE LIMITS

1. Introduction

The major key to the ExaStore architecture is its distributed and highly scaleable file system. The system provides an abstracted unified view to a collection of multiple, possibly heterogeneous, RAIDs. This unified view can be accessed through various Remote File Protocols. Internally the system manages files data and metadata in a fully distributed manner. ExaStore system is designed with the following major goals in mind:

- Linear scalability
- High Availability
- Parallel access
- Adaptive behavior
- Automatic configuration

2. Modules

The distributed file system consists of a collection of multiple instances of loosely coupled modules, each carrying its own functionally: the Front End service is responsible for the client communication, the File System Daemon (FSD) issues the client request and the Store Agent handles the RAID I/O.

2.1 Front End

The ExaStore Front End functions as a protocol converter, translating between client-side protocol requests and internal file system RPC requests. Dedicated, protocol-specific front ends are provided for NFS, CIFS, and AFP.

2.2 FSD Service

The FSD implements all the file system modules on a single process with internal, non-preemptive threads. Local inter-modules calls within the same FSD are handled as local procedure calls with minimal overhead. RPC encoding/decoding takes place only for remote calls. Non-preemptive threads simplify the synchronization considerably, and enhance the performance by reducing context switch overhead to a minimum. To take advantage of SMP architecture, multiple instances of FSD are created in each node, so that each one serves a different sub-domain, and runs almost exclusively on a single CPU. As a result, there's almost no overhead on kernel context switching. Figure 1 describes the FSD modules and how they interact with each other.

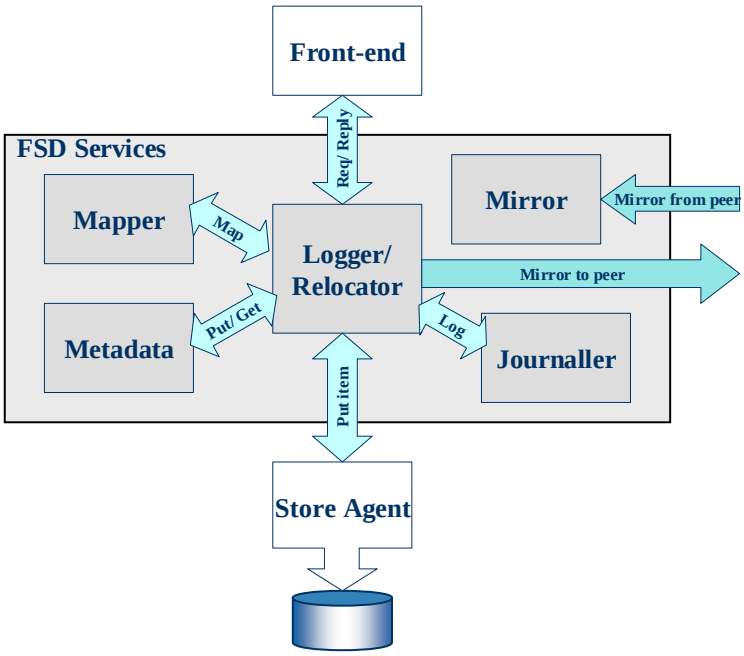

Figure 1: The FSD services

Logger – The logger performs two major tasks, managing the read/ write cache and data relocation. The cache manages all current active data and meta-data. Newly written data resides on mirrored, UPS backed memory.

Relocator – The Relocator is a background relocation process that sends the written data to the RAID volumes. As read operations are also passed through the cache, synchronization overhead is minimal. The relocator applies coalescing algorithms to accelerate disk I/O.

Mapper - The mapper serves as a pointer repository whose main purpose is to keep track of the location of data, which is distributed on all system volumes. Since all data and metadata accesses are indirect (that is, coordinated by the mapper), data relocation occurs transparently to the clients.

Metadata Service - The Metadata server is a plain repository for file metadata, such as ownership, permission, and size.

Mirror – The Mirror, running on the peer node, maintains an image of the log of the peer Logger.

Jornaller –The Journaller, activated when peer node is down (degraded mode) and on power failure. In degraded mode, when mirroring the log to a peer node is not availale, the Journaller maintains an image of the log on the Raid. On power failure, the Journaller dumps the log contents to local disk.

2.3 Store Agent

The store agent is responsible to read/write data from a RAID volume. The agent interface is low level (read/write at certain block) since all details of allocation and mappings are handled by the mapper in a higher level. Volumes are shared, and may be accessed by different nodes in different times as a result of takeover. The agent service runs always on the node that currently accesses the volume.

2.4 All Together

Figure 2 provides a general overview of the file system modules in a four-node system. The FSD services running on different nodes communicate using the internode communication (For more information, see the Exanet white paper Networking in ExaStore™). Figure 2 describes the FSD modules and how they interact with each other.

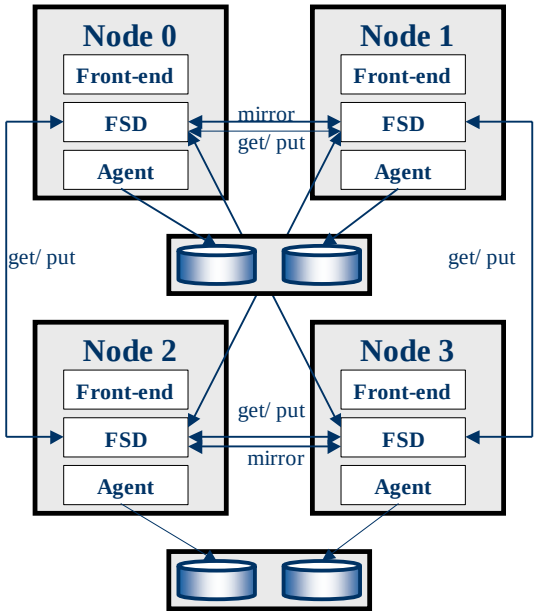

Figure 2: File system modules in a four-node system

2.5 Data Flow

2.5.1 Read Access

When a client issues a data read request, it is first handled by the front end. The front end queries the metadata server for the attributes and validity of the object being accessed and, if successful, fetches the data from the read cache and responds to the client. If there is a cache miss, the cache uses the store agent to fetch the data from the RAID.

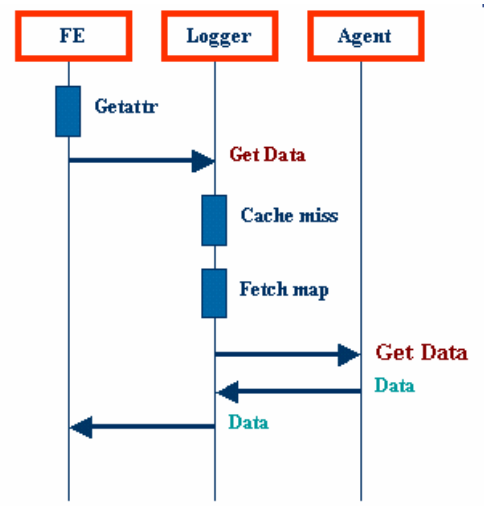

Figure 3: Read data flow

2.5.2 Write access

Data write requests are also handled by the front end, which similarly queries the metadata server before processing the request. A successful metadata query is followed by a write operation to the cache (with mirroring) After the write has been acknowledged, the relocator moves the object to a disk store agent, which then stores the data to the RAID. Having received its acknowledgement, the client is free to perform other operations. In the meantime, the data is stored to disk at a slower pace, a process that hides disk access latency from the client.

Note that there are circumstances in which the data is not sent to the RAID. If, for example, a second write operation modifies the data object again before the results of the first write are moved to the RAID, the store agent knows to store the second, newer version of the object rather than the first.

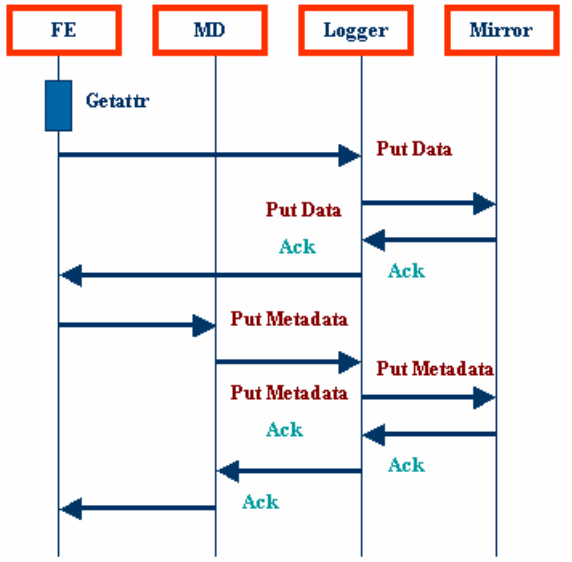

Figure 4: Write data flow

2.6 Data placement

The ExaStore System distributes data evenly across RAIDs. This even distribution improves the balance among system components, hence improving performance and resource utilization, which leads to better ROI and TCO. Uniform distribution can also accelerate backup and restore times substantially. Contrast this with traditional systems, in which some RAIDs may be assigned much more data than others. In such cases, the RAIDs with the most data become the bottleneck, delaying completion of backup and restore operations. Exanet's balanced distribution, on the other hand, ensures that I/O load is distributed evenly and thus shortens the overall backup time.

2.7 Journaling and Recovery

2.7.1 Degraded Mode Operation

When a node fails, all the services it carries are moved to a peer node. The data that has not yet relocated to the RAID is written sequentially to a journal on one of the RAID volumes, and operation can continue, journalled to a stable storage. In this way, newly written data is protected from another failure.

When the failed node become available again, the two peer nodes synchronize and resume normal operation.

2.7.2 UPS Power Failure

When a node senses a power failure, it starts immediately to dump all the write cache content (Data and meta-data items that were modified by user request, and not reached the disk yet) to a local disk.

The dump process is internal to the node, and does not depend on the availability of any external resources, in particular the interconnect, the FC fabric and the RAID are not needed when dumping.

When power is restored, the node is restarted, to enable again all network and FC services, and the dump is restored from latest version, and mirrored in a peer node, so that normal operation mode can start.

2.7.3 Summary

The ExaStore system implements a distributed and highly scaleable File System. The system provides an abstracted unified view to a collection of multiple RAIDs (potentially of different flavors). This unified view can be accessed through various Remote File Protocols (NFS, CIFS, AFP). Internally the system manages files data and metadata in a fully distributed manner. Designed with high availability in mind, the file system is also highly scalable and massively parallel. The file system is adaptive and automatically re-configurable.