Newest Trends in High Performance File Systems

Elena Bergmann

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg
Betreuer Julian Kunkel

2015-11-23

Agenda

Introduction

- Current situation:
  - Fundamental changes in hardware
    - Core counts are increasing
    - Performance improvement of storage devices is much slower
  - Bigger system, more hardware, more failure probabilities
    - System is in a state of failure at all times
- And exascale systems?
  - Gap between produced data and storage performance (20 GB/s to 4 GB/s)
  - I/O bandwidth requirement is high
  - Metadata server often bottleneck
  - Scalability not given

Upcoming technologies until 2020

- Deeper storage hierarchy (tapes, disc, NVRAM . . .)
- Is traditional input/output technology enough?
- Will POSIX (Portable Operating System Interface) I/O scale?
- Non-volatile memory
- Storage technologies (NVRAM)
- Location across the hierarchy
- Node local storage
- Burst buffers
- New programming abstractions and workflows
- New generation of I/O ware and service

File Systems

File Systems

- Store data in logical “files” that are just byte arrays
- File system objects:
  - Files, directories and metadata
- Organize files in an hierarchical namespace
  - Dir\subdir\file
- File system: structure and logic rules to manage the data
- User perspective
  - User has to map data structures to array of bytes
  - User also has to map in the files

File Systems – Classification according to [6]

- Generation 0:
  - No system, stream of data (punchcards, audio cassette)
- Generation 1:
  - Multiple named files on one device
- Generation 2:
  - Files and directories on one device
- Generation 3:
  - Metadata for more information about files
  - Access control (read, write, admin ...)
- Generation 4:
  - Journaling file system for consistency
  - System knows what should happen during a failure and can restore the status

File Systems – Generation 5 – ZFS created by Sun Microsystem

- Built in volume management
  - Dynamically edit partitions
- Per block checksumming
  - Every block of data has a associated checksum to verify it
- Self healing redundant arrays
  - Correction of bytes is possible
- Atomic COW snapshots
  - Backup point of the current state of the system
- Asynchronous replication
  - Use a snapshot of the system on another machine
- Far future scalability
  - Supports up to 16 exbibyte (2^{60} byte)

File Systems

- Space management: Where is the file? Where is free place?
- Providing interface, utilities and access control for the user
  - Read, write, delete or copy files or directories
  - Readable, writable attributes
- Consistency:
  - Stored data stay in its storage location
    - Empty places stay empty
  - Transactional file system specially invented, but not practicable
- Limitations
  - Converting the type
    - E.g. FAT to NTFS for 4GB+ files
  - Loooong file paths and names
    - Path name is too long, cannot change the files inside a directory anymore

File Systems – Types (1)

- Device file systems are optimized for the characteristics of their storage media
- Disk file systems
  - E.g. FAT, exFAT, ext4...
  - Optical disks like DVD
- Flash file systems
  - Optimized for solid state media
  - Erasing, access, wear levelling
- Tape file systems
  - LTFS with self-describing form
- Minimal file systems
  - Audio cassette tape
- Flat file systems
  - Floppy disk

File Systems – Types (2)

- Special file systems
  - E.g. device file systems, path name can include a device prefix
- Database file systems
  - Files are identified by their characteristics (metadata)
- Shared disk file systems
  - A number of machines have access to the same external disk subsystem
- Clustered file systems
  - Access to the file system in one client
- Network file systems
  - E.g. NFS, AFS
  - Access to a client through remote access in one network

Parallel File Systems

- Motivation
  - Parallel access for a number of machines to all files
  - Increasing the performance

Aspects

Access: multiple user, access control, feels like “local file access”
Location: finding files, filename does not reveal location
Concurrency: shared files, like one user view, but for multiple user
Failure: hardware, system
Replication: prevents failures, provides scalability
Migration: files move without client’s knowledge
Heterogeneity: different hardware and o. s. support
Scalability: scale up – faster systems

Scale up

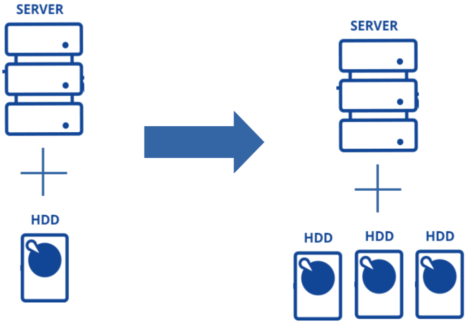

Figure: Figure based on: http://itknowledgeexchange.techtarget.com/storage-soup/scale-out-vs-scale-up-the-basics/

Scale out

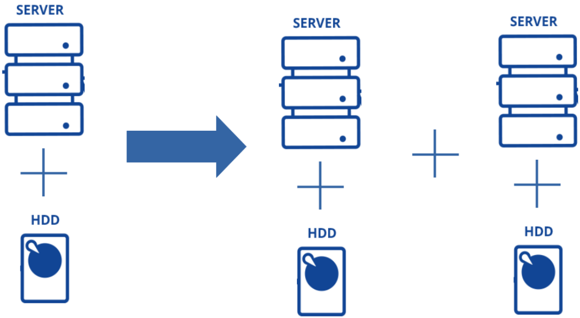

Figure: Figure based on: http://itknowledgeexchange.techtarget.com/storage-soup/scale-out-vs-scale-up-the-basics/

Scale out and scale up

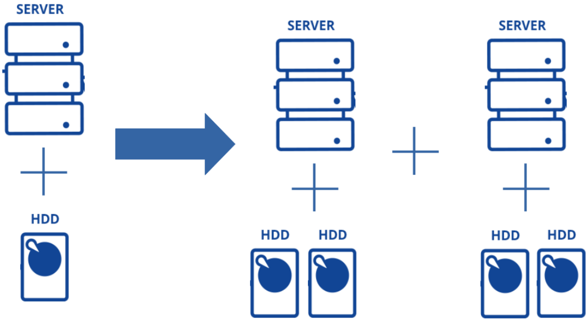

Figure: Figure based on: http://itknowledgeexchange.techtarget.com/storage-soup/scale-out-vs-scale-up-the-basics/

Distributed file systems/Network File System (NFS)

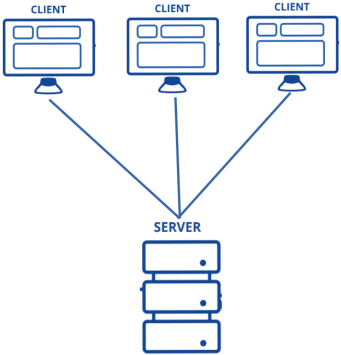

Figure: NFS, figure based on:
`http://gerska-consulting.de/wp-content/uploads/2014/11/SAN.png`

Shared-disk file system/Storage area network (SAN)

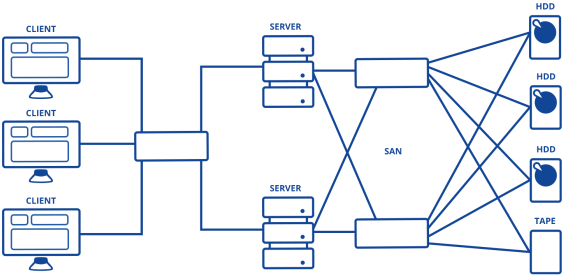

Figure: SAN, source:
http://gerska-consulting.de/wp-content/uploads/2014/11/SAN.png

Parallel File Systems

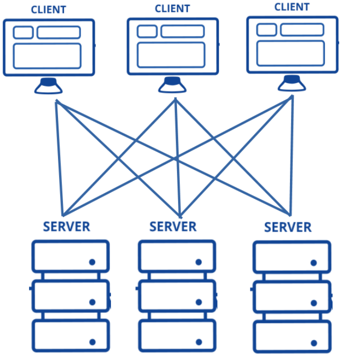

Figure: Parallel file system, figure based on:
http://gerska-consulting.de/wp-content/uploads/2014/11/SAN.png

Current solutions for network file systems

- Distributed file systems/Network File System (NFS)
  - Uses network protocol (easy)
  - Failure of file server stops the entire network
  - Scalability is limited (until NFSv4)
- Shared disk file system/Storage area network (SAN)
  - Fibre channel storage used
  - Costly: n-connections needed
  - Each node needs fibre channel host bus adapter
- Parallel File Systems
  - Metadata server on I/O nodes or server
  - Global name space for all files
  - Scalability for more servers
  - Distribute large tiles across multiple nodes, can use subsystems

File Systems – Disadvantages and Development

- Hardware failures cause data loss
  - File system needs to contain its health autonomously
  - E.g. replicate servers, moving data
- Performance is slow:
  - Disk access, CPU processing and messages need time
  - Minimize communication between clients and servers
- Concurrency problems:
  - Two clients change one file at the same time
- New file system projects:
  - E.g. PVFS2, Lustre, GPFS, BeeGFS, PanFS

File System PanFS

- Parallel file system
- For ActiveStor hybrid scale-out NAS appliance
- Survives triple failure
- RAID 6+ for data protection
- RAID per file makes it scalable
- Small files mirrored with flash speeds
- System wide parallel rebuild avoiding degradation at scale

Sirocco File System

Sirocco File System – Project

- Project lead:
  Sandia National Laboratories
  Scalable System Software Department
- Published documents:
  `http://www.cs.sandia.gov/Scalable_IO/sirocco`
- The goal: parallel, high performance exascale storage system
- Inspired by peer-to-peer systems
- Clients can chose the best storage server without regarding the storage location
- Server uses local decisions for independent tasks to minimize interference

Sirocco File System – Design Principles

1. No central index for locations
   - Free seating! Every data from every client can be stored everywhere
   - But: extensive search needed to find one particular file
2. Continually moving data between the storage devices
   - Ensure longevity, integrity and system health
   - Variable store size is possible
   - Clients are not notified of the events
3. Emphasize scalability over legacy support
   - Support for legacy storage system semantics POSIX is required
   - Scalability is not harmed by POSIX and benefits from Sirocco

Sirocco File System – Design Principles

4. Heterogeneous media support
   - All and future available media types should be supported
   - Application Programming Interface (API) should be symmetric
5. Various resilience guarantee for different forms of data
   - Some data need more resilience guarantee than others
   - Clients can determine the level of protection
6. Scalable as possible server-side operations
   - No coupling during operations
   - No reliability on other servers

Sirocco File System – Network

- Self aware peer to peer overlay (SAP2P)
  - Using distance optimized links, chosing the lowest known latency links for connections
  - Small number of random links for connectivity
- Locality and node needs
  - Nodes with necessary resources are preferred
  - Closest available nodes are used as storage targets or for location requests

Sirocco File System – SAP2P

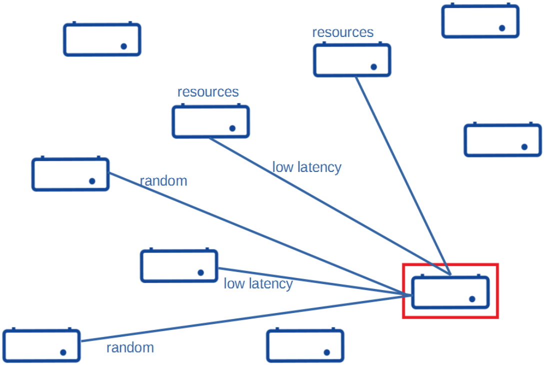

Figure: Nodes form connections with distance optimized links and locality, figure based on: [1]

Sirocco File System – Namespace

- ID-based namespace, not human friendly
- Advanced Storage Group (ASG) interface implemented

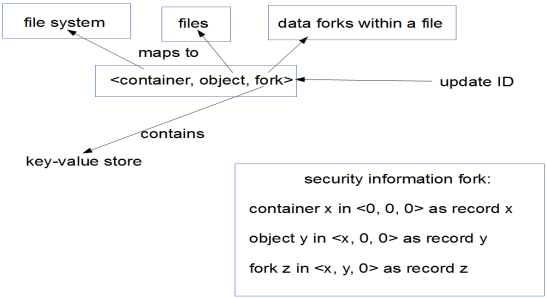

Figure: Namespace description, figure based on: [1]

Sirocco File System – Data interface

- Provided operations:
  - Write
    (data buffer, container ID, object ID, fork ID,
    start record, number of records, record length, update ID)
  - Read
    (container ID, object ID, fork ID,
    start record, number of records, map buffer)
- Indirect provided operations:
  - Writing a range of records creates objects
  - Resetting the records to their default state deletes an object
- Read/Location Protocols
  - Unpredicted reading of data requires extensive search
  - Predicted reading possible through authoritative server for a range – gets all write requests of his area

Sirocco File System – Data Durability

- Sirocco stores data fast on free space
- But: resilience guarantee not checked
- User wants particular guarentee for his data
- Each server has a durability attribute for a fork
- Durability is achieved with:
  - A set of disks in RAID
  - Replication of unsafe servers
- Sync command forces data to safe locations
- Another server accepts writing of data
  - Gives resilience guarantee
- Data is temporary in unsafe location until sync command

Sirocco File System – Optimistic Concurrency Control

- Locking is an overhead comparing to operation
- Possibility of conflicting operations is low
- Server compares incoming update IDs with present ones
- Overwrites only when the incoming update ID is higher
- Works in transactional batches, enables rollback for failures

Sirocco File System – Batches

- Batched updates:
  - Sending several commands at once is possible
  - Batches can be specified transactional, enabling ACID updates
- Triggered batch (TB) with leased locks
  1. TB is condition to batch – executed once the condition is met
  2. TB stays queued until false condition is true, then executed
  3. TB can only be used on a single record

Sirocco File System – Triggered batch routine

1. Lock requester (user) submits TB
2. Operation proceeds when update ID states “unused”
3. TB sets update ID to estimated used time and notifies user
4. Once TB is at the head of the queue, server notifies user of status and start time of the last lock
5. User requests time stamps of the lock and start time to calculate the waiting time
6. Client modifies record and obtains the lock
7. Requester finishes and releases the lock, resets the update ID

Sirocco File System – Pessimistic Concurrency Control

- User ist notified:
  - Once his TB is queued
  - Second his TB is on top of the queue
- User can detect unreleased locks and starts recovery protocol
- Non responsive locker failure:
  - Next requester takes the batch and forces him out of the queue
  - Non responsive locker receives a cancel signal
- Non responsive requester failure:
  - Simply removed from the queue

Sirocco File System – First Results (2012)
Introduction

- Deployed Sirocco storage servers on nodes alongside a compute job running CTH application
- CTH application simulates strong shock waves on solid mechanics, producing large stream of information
- CTH application writes checkpoint data to Sirocco
- Sirocco takes the data as fast first tier and transfers it asynchronously to slower disk-based storage (PanFS)

Sirocco File System – First Results (2012)
Server and Clients

- Sirocco storage server provides an object based storage API
- API is based on remote direct memory access (RDMA)
  - One client accesses memory of another
  - Independent of operating systems
- Source specific multicast used an MPI-based transport layer
  - Receives data from one specific source
  - Message Passing Interface: parallel
- Custom POSIX like client was crafted
  - Stores checkpoint data with a file per object
  - One megabyte local buffer ensures that larger messages are sent to the storage server as needed

Sirocco File System – First Results (2012)
Instrumenting CTH for Sirocco

- Extra processes inside CTH allocation for Sirocco access
- MPI_Init extended with own initialization routine with global communicator
- 4 nodes CTH and 1 node Sirocco in a group
- Sirocco process satisfy storage requests per event loop
- CTH node computes normally
- I/O infrastructure Syncio was replaced with own routines to interact directly with the Sirocco storage servers

Sirocco File System – First Results (2012)
Hardware and Results

- Tests were conducted on a Cielo/Cray XE6
  (used as a capability platform)
- Shaped charge example problem
  with varying scales from 256 to 32,768 cores as test job
- Baseline runs wrote checkpoints directly to 10PB panasas storage (PanFS)
- Each Sirocco run needed 25% more nodes
  than the CTH job required

Sirocco File System – First Results (2012)

- 10x to 60x performance increase
- PanFS varies caused by general use of the machine

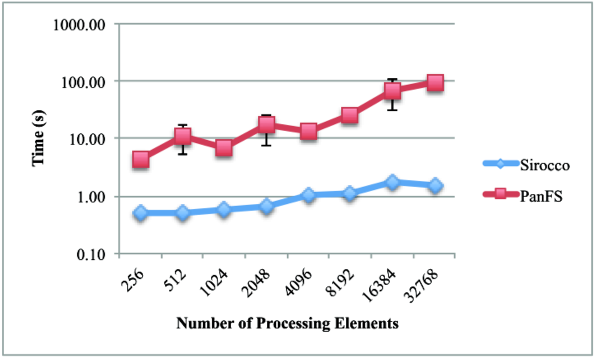

Figure: Average checkpoint time, Source: [2]

Summary

- Hardware changes rapidly, file systems have to catch up
- Newest generation of file systems are better with concurrency and with consistency problems
- NFS is easy but can be a single point of failure, SAN is expensive, real parallel file systems are needed
- Sirocco uses a peer-to-peer concept with free data movement and placement
- Supports all media types, scalability should be high
- First results show performance increase up to 60%, but 25% more nodes were needed

Further information

- File System Projects
  http://filesystems.org/all-projects.html
- Evolution of File Systems by Christian Bandulet
  http://www.snia-europe.org/objects_store/Christian_Bandulet_SNIATutorialBasics_EvolutionFileSystems.pdf
  "At the end, everything is saved and stored as blocks on tape, magnetic disk, optical disk or flash memory."

Burst buffers

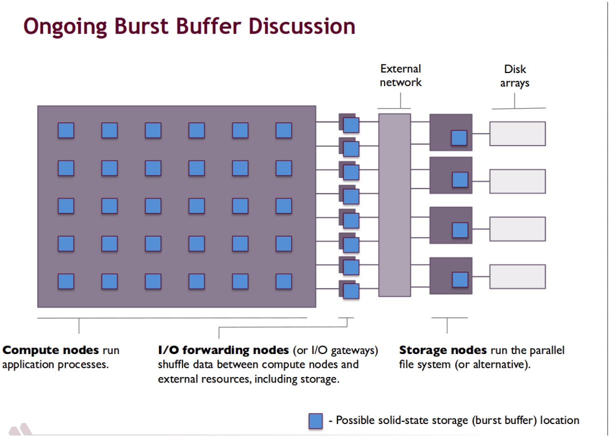

Figure: Burst buffer can be at the blue locations, source: advisor

Literature I

[1] Matthew L. Curry, H. Lee Ward, Geoff Danielson *Motivation and Design of the Sirocco Storage System, Version 1.0, Sandia Report* 2015.

[2] Matthew L. Curry, Ruth Klundt, H. Lee Ward *Using the Sirocco File System for High-Bandwidth Checkpoints, Sandia Report* 2012.

[3] Amina Saify, Garima Kochhar, Jenwei Hsieh, Ph. D., Onur Celebioglu *Enhancing High-Performance Computing Clusters with Parallel File Systems* 2005.

[4] Wikipedia File System "https://en.wikipedia.org/wiki/File_system", 2015, accessed 31-October-2015.

Literature II

[5] Wikipedia Clustered File System, "https://en.wikipedia.org/wiki/Clustered_file_system", 2015, accessed 31-October-2015

[6] Bitrot and atomic COWs: Inside “next-gen” filesystems, "http://arstechnica.com/information-technology/2014/01/bitrot-and-atomic-cows-inside-next-gen-filesystems/", 2015, accessed 03-November-201

[7] CTH Shock Physics, "http://www.sandia.gov/CTH/", 2015, accessed 06-November-2015

[8] PanFS Storage Operating System, "http://www.panasas.com/products/panfs", 2015, accessed 06-November-2015