Newest Trends in High Performance File Systems

— Seminarbericht —

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

Vorgelegt von:	Elena Bergmann
E-Mail-Adresse:	elena.bergmann@posteo.de
Matrikelnummer:	6525540
Studiengang:	Informatik
Betreuer:	Julian Kunkel

Hamburg, den 30.03.2016

Abstract

File Systems are important for storage devices and therefore also important for high computing. This paper describes file systems and parallel file systems in general and the Sirocco File System in detail. The development of Sirocco shows how it could handle the big gap between produced data from the processor and the data rate that can be stored by current storage systems. Sirocco is inspired by loose coupling of peer-to-peer systems, where clients can join and exit the network any time without endangering the stability of the network. The user gets the ability to chose their storage location and the system replicates files and servers to maintain its health.

Contents

1. Introduction

This chapter describes the current situation of file systems in high computing and some technologies that will be released by 2020.

The current development is driven by fundamental changes in hardware, caused by the rapidly increasing core counts, which are leaving the storage devices far behind. The performance improvement of storage devices proceeds much slower and the gap between produced data and stored data is large. Disk-based systems cannot compete with the I/O bandwidth required by checkpoints. The amount of parts in a system is also growing and this amount of hardware gives a lot of probabilities for failures. If the system is too big, the probability that some part of the hardware failed increases and the system can be in a state of failure at all times. Exascale systems have a gap between produced data with 20 GB/s and storage performance with 4 GB/s. The Input/Output bandwidth requirement is high and often the metadata server is the bootleneck of the system, because all requests of metadata have to go through it. Scalability is not guaranteed for exascale systems. Heterogeneous storage technologies are required.

By 2020 there will be a deeper storage hierarchy with tapes, disc, NVRAM, so there will be a wide range of variety of storage devices used for one system. Non-volatile memory and storage technologies like NVRAM are also upcoming. Node local storage and burst buffers are new technologies. It is possible, that the traditional input/output technology could be replaced with a new technology. Also unsure is the scalability for POSIX (Portable Operating System Interface) for input/output. There will be new programming abstractions and workflows and a new generation of input/output ware and service.

2. File Systems

Definition of file systems and examples.

2.1. File Systems in General

File systems store data in logical "files", that just consist of byte arrays. Files, directories and metadata are file system objects. They are organized in a hierarchical namespace with directories and files. The file system consists of the structure of files and directories and logic rules to manage the data. From the user perspective the user has to map their data structures to the array of bytes and in the files.

According to [6] the file systems can be classified in five generations. Generation 0 has no system and is only a stream of data. These file systems are on punchcards and audio cassettes. The first generation has multiple named files on one devices, and the second generation has additional to the first generation directories for storing. Generation 3 consists of metadata for more information about the files and access control. Access control for a file allows or denies the user to read or write the file. For better consistency generation 4 is a journaling file system and knows what should happen during a failure and restores the status, because it can look up in the journal what was supposed to happen during the failure and restores the outcoming status.

Generation 5 is the newest and the author of [6] explains it by means of ZFS created by Sun Microsystem. This file system has a built in volume management to dynamically edit partitions and a per block checksumming. So every block of data has an associated checksum to verify it. The redundant arrays are self healing and therefore a correction of the bytes is possible. ZFS has atomic COW snapshots to make backup points of the current state of the system. An asynchronous replication is possible by using the snapshot of a running system on another machine. The scalability supports up to 16 exbibyte (2^{60} byte).

File systems organize the space management, so they can tell you the exact location of a file or free place. They provide the interface, utilities and access control for the user. The main functions are reading, writing, deleting or copying files and directories. The files and directories have attributes for their access level (readable, writable). A certain consistency is required, so the stored data stays in the specified storage location and empty places stay empty. Transactional file systems were specially invented for better consistency, but they were not practicable in life systems and weren't further developed.

There are two limitations in file systems: converting a type of one file system to another and long pathnames in file systems. E.g. converting a file system from FAT

to NTFS for the support of 4GB+ files, the storage device has to be empty to format another file system type on it. Every file systems supports a different number of path lengths and the files and directories are not editable, when the path name is too long and out of reach for the file system. The user can manually edit dir names on the beginning of the path name to shorten the whole path name, but reducing the names can cause data and information loss (e.g. dir name "project" to "p").

There are different types of file systems. Each device file system is optimized for the characteristics of their storage media. E.g. a flash file system has to support much more read and write accesses than a DVD, which is mostly written once. There are disk file systems like FAT, exFAT, ext4 for optical disks like CD, DVD etc. The flash file systems are optimized for solid state media and have a different care about erasing files, access and wear leveling. Wear leveling extends the life of a solid state drives and flash memory. The tape file systems contain often a self-describing form like LTFS. An audio cassette tape has a minimal file system and a floppy disk has a flat file system.

There are more specialized file systems like device file systems, in which the path name can include a device prefix, or the database file systems, in which the files are identified through their characteristics provided by the metadata. A shared disk file system supports a number of machines with access to the same external disk subsystem. Similar to that a clustered file system gives access to the file system in one client. NFS and AFS are network file systems, with whom a client has access through remote protocol in one network.

2.2. Parallel File Systems

We need parallel file system for real parallel access with a number of machines to all files in a system and to increase the performance that transfers the data to the storage devices.

There are more aspects for parallel file systems to consider: We have access for multiple users and have to control the access, but the access should feel like the files are locally on the same device and not splitted on multiple storage devices. A filename does not reveal the location, but files have to be found on a widespread system. Also shared files can be accessed by more than one user at the same time and consistency should be given if different users change the same part of the data. The user should not notice that there are more users in the system. Hardwares and systems can fail and there are more parts to fail than in a small system. Replication prevents failure and provides scalability. Files move without the client's knowledge (migration aspect). Heterogeneity should be given, so that different hardware and operating systems are supported. Scalability for faster systems should be possible.

There are three possibilities to scale a system for more performance. The following three figures show the layouts of the types.

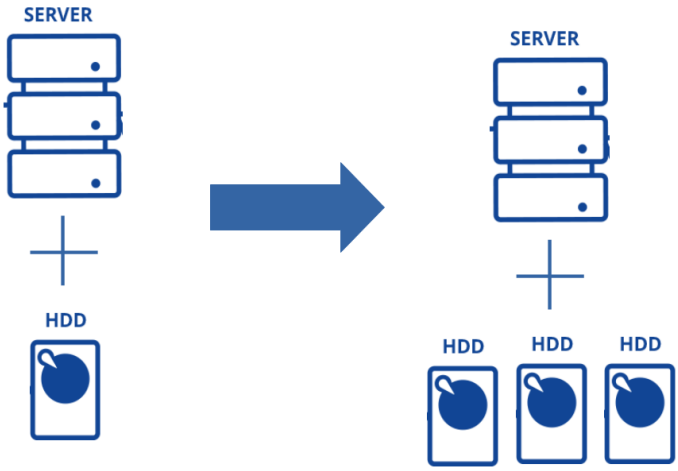

Figure 2.1.: Scale up means putting more storage devices to one system. Figure based on: [9]

One server gets more storage devices, this is scale up shown in figure 2.1.

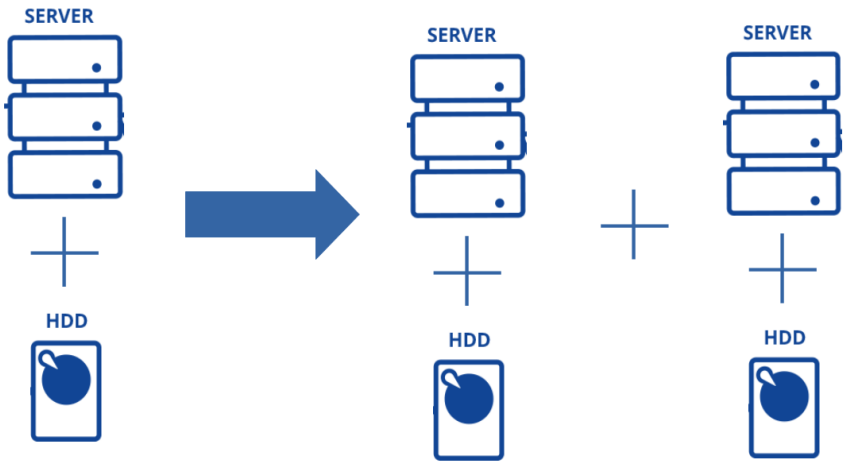

Figure 2.2.: Scale out describes enlarging a system with a second system. Figure based on: [9]

In figure 2.2 there was one server and they added a second one with the same storage devices, this is called scale out.

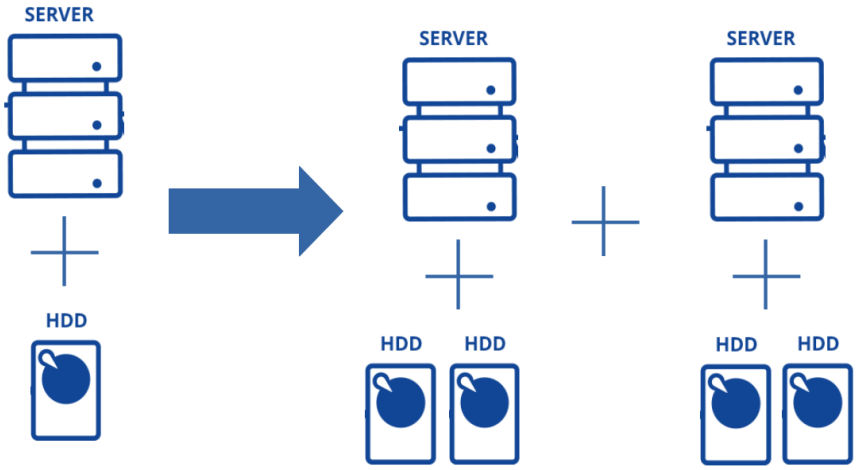

Figure 2.3.: Scale out and up at the same time, another system is added and also more storage devices to the current ones. Figure based on: [9]

Scaling out and up can also be done at the same time, so that all servers get more storage devices and there are new servers, shown in figure 2.3.

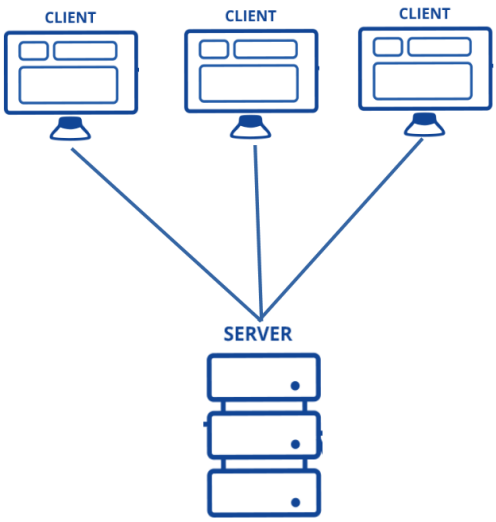

Figure 2.4.: NFS, figure based on: [10]

A distributed file system or Network File System (NFS) uses an easy to use network protocol, but a failure of the file server stops the entire network. The scalability was limited up to NFSv4. Figure 2.4 shows the layout of a NFS, where multiple clients communicate through a network protocol with the other clients and the server. The server holds all functions of the network and his failure can stop the whole network.

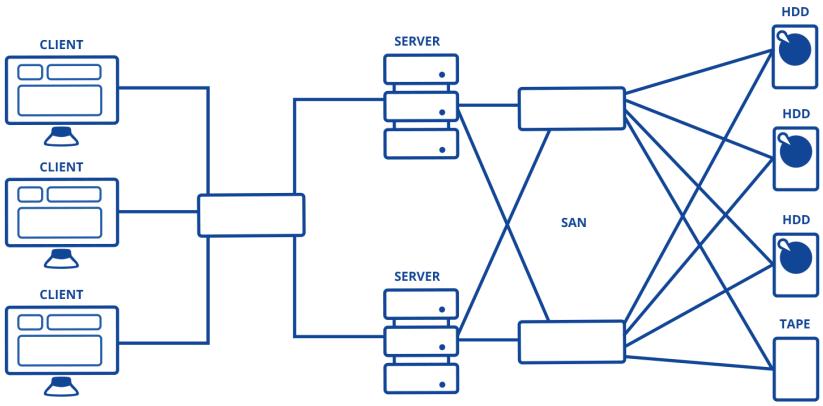

Figure 2.5.: SAN, source [10]

A shared disk file system or storage area network (SAN) uses fiber channel storage and requires n-connections of the fiber channel cables. Every node needs a fiber channel host bus adapter. It is fast, but costs much. Figure 2.5 shows the layout with clients

on the left side, which communicates through a switch with the servers in the middle. These servers have to communicate with the SAN own network, on which the storage devices are located.

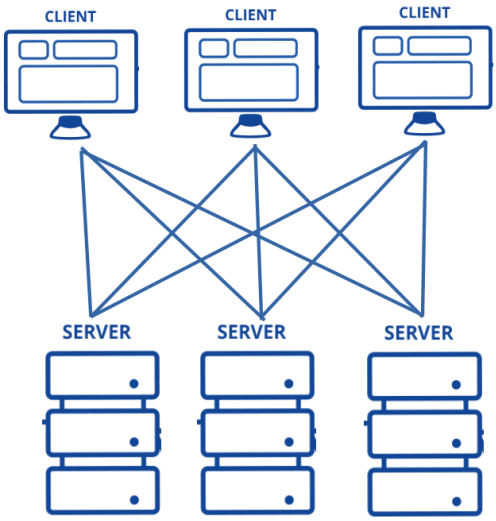

Figure 2.6.: Parallel file system, figure based on [10]

A Parallel File System has a metadata server on I/O nodes or on a server. It has a global name space for all files and scalability for more servers, because adding new servers scales the performance. Distributing large tiles across multiple nodes is possible, so one can use subsystems. In figure 2.6 there are multiple clients on the top and multiple servers on the bottom. Each client communicates with the other clients and every server. Also the server are connected with all clients and servers.

A hardware failure causes data loss and a file system needs to contain its health autonomously to prevent this. Replicating servers and moving data increases the data safety. Parallel file systems have a slower performance, because additionally to disk access and CPU processing time the sending and receiving of the messages between the clients also need time. There are concurrency problems to prevent, if two clients change the same file simultaneously. There are more new file system projects in developing e.g. PVFS2, Lustre, GPFS, BeeGFS and PanFS.

PanFS will be explained shortly, because it is used in the tests for Sirocco. It is a parallel file system for ActiveStor hybrid scale-out NAS appliance and can survive a triple failure. It has RAID 6+ for its data protection and RAID per files for scalability. Small files are mirrored with flash speeds and a system wide parallel rebuild avoids degradation at scale.

3. Sirocco File System

Concept of the Sirocco File System and test results from 2012.

The project lead of the Sirocco file system is Sandia National Laboratories – Scalable System Software Department. Published documents are found: `http://www.cs.sandia.gov/Scalable_IO/sirocco`. The goal of Sirocco is a parallel, high performance exascale storage system. It is inspired by peer-to-peer systems. The clients can chose the best storage server without regarding the storage location and the server uses local decisions for independent tasks to minimize the interference. The asynchronous management of heterogeneous storage devices increases the speed and safety of the file system.

3.1. Design Principles

Sirocco has no central index for locations and files are stored anywhere in the first moment. This adds speed to the system, but an extensive search could be needed to find particular files. The data is continually moving between the storage devices and the files are replicated, copied and destroyed to ensure longevity, integrity and system health. This makes also a variable store size possible. The clients are not notified of the events. The design of Sirocco wants to emphasize scalability over legacy support. The support for the legacy storage system semantics POSIX is required, although the scalability is not harmed by POSIX and benefits from Sirocco.

With a heterogeneous media support Sirocco should be functioning on all available and future media types. The Application Programming Interface (API) should be symmetric for easy use on different types. There are various resilience guarantees for different forms of data, because some data need a greater resilience guarantee than others. The clients can determine the level of protection for their data. Sirocco has scalable as possible server-side operations, so there is no coupling during operations and no reliability on other servers.

3.2. System

3.2.1. Network

The storage swarm of Sirocco servers uses a two-stage method for building a network. In the first stage, Sirocco uses a self aware peer to peer overlay (SAP2P), which uses distance optimized links and choses the lowest known latency links for connections. There is a small number of random links for connectivity. For the second stage, nodes with

necessary resources are preferred and the closest available nodes are used as storage targets or for location requests. This is shown in figure 3.1. Sirocco uses this concept for data migration and replication. The SAP2P is stable to massive changes in the network caused by a high amount of members leaving or joining the network. Partitioning is avoided through the overlay. This churn resilience guarantees that Sirocco is used to write back caches for checkpoint workloads and handles joining and leaving of thousands of servers in short time periods.

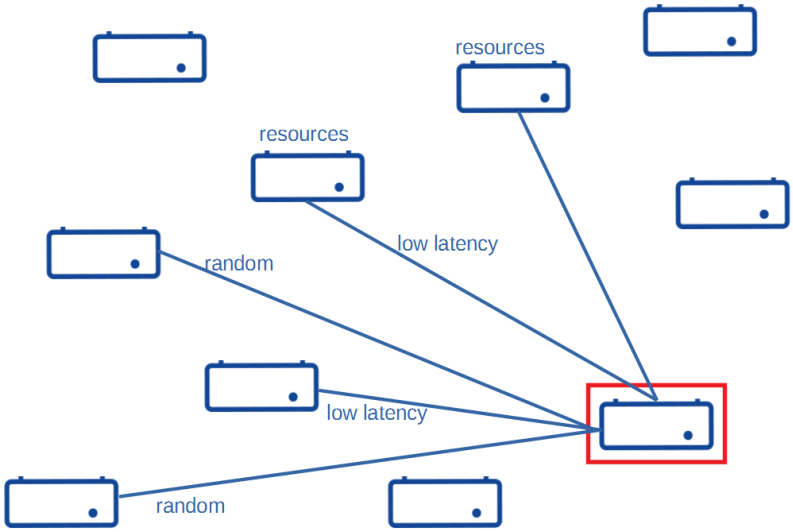

Figure 3.1.: Nodes form connections with distance optimized links and locality, figure based on: [1]

3.2.2. Namespace

Sirocco uses a not human friendly ID-based namespace with an implemented Advanced Storage Group (ASG) interface, so it can provide primitives for higher level file systems.

Three 64-bit values make up the name space, described as container ID, object ID and fork ID, also written <container, object, fork>. The container ID maps to the file system, the object ID maps to the files and the fork ID maps to data forks within the file. These values are static and cannot move between objects. The fork also contains a key-value store (key is 64 bits) and a value up to 2^{32} bytes. The key-value is only updated as a whole and not only the record. The security information in the fork has a container x in <0, 0, 0> as record x, an object y in container x in <x, 0, 0> as record y and a fork z in object y in container x in <x, y, 0> as record z (shown in figure 3.2).
The security attributes of the fork give access control for the records and are inherited from the object or container if not present.
The update ID of each record is changeable by the user and a logical clock. Each copy of

a record as its own update ID, which is increasing after each write. The highest update ID specifies the current version of the record.

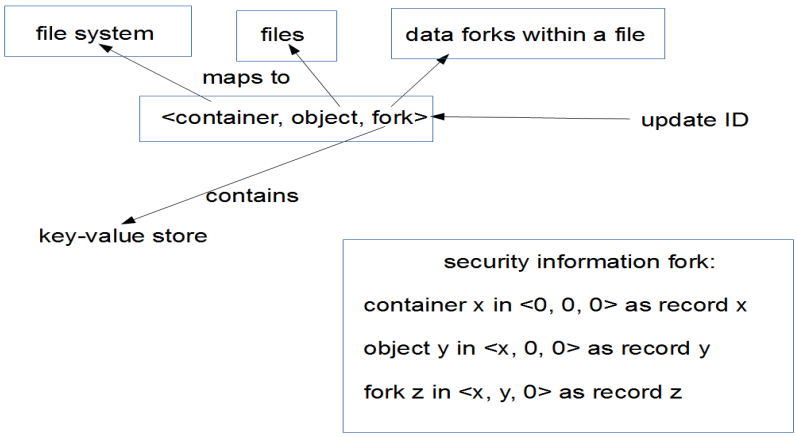

Figure 3.2.: Namespace description, figure based on: [1]

3.2.3. Data Operations and Search

Writing and reading are provided operations for the file system, so the writing operation contains the data buffer, container ID, object ID, fork ID, start record, number of records, record length and the update ID. The reading operations contains the container ID, object ID, fork ID, start record, number of records and the map buffer. For each record the map of a range gives the size and update ID to the user. Creating and deleting objects is indirectly provided, because the writing of a range of records creates an object and the resetting to the default states deletes the object. As Sirocco stores multiple versions of records, the missing operations prevent tidying up the whole system to find all copies of a file to delete them or create them on multiple sources. Deleting one object on a server leaves its copies on the other servers, but they are supposed to be empty records.
The system has read and location protocols. A predicted reading is possible through an authoritative server for a range, which gets all write requests in his area. Initially all servers are non-authoritative and do not know the locations of the requests data. An unpredicted reading requires an extensive search over all storage servers and is performed on behalf of the user.

3.2.4. Storage and Updates

Sirocco stores data fast on free space, but the resilience guarantee is not automatically checked. If users want a particular guarantee for their data, they have to check with the durability attribute for the forks of the server. The durability is achieved with a set of disks in RAID or the replication of unsafe servers. These options create the abstract

value for the durability attribute. The sync command forces the data to safe locations and when another server accepts to write the data, it has to give the wanted resilience guarantee for the user. So the data is temporary in an unsafe location until the sync command is committed by the user.

Sirocco does not lock an update, when the operation has a low possibility of conflicting with others and the locking would be an overhead compared to the operation. This is called optimistic concurrency control. For a simple update the server compares incoming update IDs with the present ones and overwrites the files only when the incoming IDs are higher. This works also in transactional batches and gives the possibility of rollbacks for failures.

For a batched update it is possible to send several commands at once and to specify as transactional update for ACID updates. The pessimistic concurrency control uses leased locks and the basic mechanim is the triggered batch. For triggered batches (TB) Sirocco uses leased locks with three rules. First, the TB is a condition to a batch and executed once this condition is met. Second, the TB stays in the queue until all false conditions are true and is then executed. Last, the TB can only be used on a single record. To perform a TB, the user as lock requester submits a TB. The operation proceeds when the update ID states "unused". The TB sets the update ID to estimate the used time and notifies the user of it. Once the TB is at the head of the queue, the server notifies the user of status and start time of the last lock, so the user can request the time stamps of the lock and calculates the waiting time. The lock requester could abort the operation at this point, elsewhere the file system client modifies the record and obtains the lock. After the requester finishes, the requester releases the lock and resets the update ID. The TB notifies the user twice: first, when his TB is queued and second, when their TB is on top of the queue. So the user can detect unreleased locks and starts a recovery protocol. The procedure of the TB has two failure points: If the locker is non responsive, the next requester takes the batch and forces the non responsive locker out of the queue. The non responsive locker also gets a cancel signal, so if it gets active again, it quits and does not perform the TB. If a requester is non responsive, the requester gets removed from the queue.

3.3. First Results

2012 a team from the project lead deployed Sirocco storage servers on nodes alongside a compute job running the CTH application. The CTH application simulates strong shock waves on solid mechanics, and produces a large stream of information, which is stored on storage servers. In this test the CTH application writes the checkpoint data to Sirocco and Sirocco takes the data as fast first tier and transfers it asynchronously to the slower disk-based storage with PanFS as parallel file system. The Sirocco storage server provides an object based storage API, which is based on the remote direct memory access (RDMA). The objects have a 128-bit identifier and a 64-bit address space. With RDMA one client accesses directly and independently the memory of another operating system. The source specific multicast (SSM) uses a MPI-based transport layer. With the

SSM the receiving of data is precised to one specific source and MPI uses messages for parallel processing. For the test the team crafted a custom POSIX like client to store the checkpoint data with one file per object and to have one megabyte local buffer to ensure that larger messages are sent to the storage server as needed. They also made extra processes inside the CTH allocation to access Sirocco. The MPI_Init was extended with its own initialization routine with a global communicator. They divided the nodes in group of five, with four nodes for the normal CTH computing and one node for Sirocco. The Sirocco process satisfies the storage requests per event loop, so the process waits in an active mode for a request. CTH processes were not changed. Each node had 16 ranks. The I/O infrastructure Syncio was replaced with own routines to interact directly with the Sirocco storage servers. Syncio coordinates the requests for storage to prevent overloads. The standard tasks were reading, writing, opening and closing, so the team could easily replace the tasks with their own routines.

The tests were conducted on a Cielo/Cray XE6 which is used as a capability platform at the labors. They used the shaped charge example problem with varying scales from 256 to 32,768 cores as test job. The baseline runs wrote checkpoint data directly to the 10 PB panasas storage with PanFS as file system. Each Sirocco run needed 25% more nodes than the CTH job alone required, because they made theses group of five nodes with an extra Sirocco node along the CTH nodes. The average of the first four checkpoints was recorded. With this setting they noticed a 10 to 60x performance increase between Sirocco and PanFS (see on 3.3). The varying of the PanFS was caused by the general use of the machine for others jobs, so they did not isolate the job on the servers and accepted influence on the test results for the PanFS. The diagram shows the measured number of processing elements over time in an average checkpoint time.

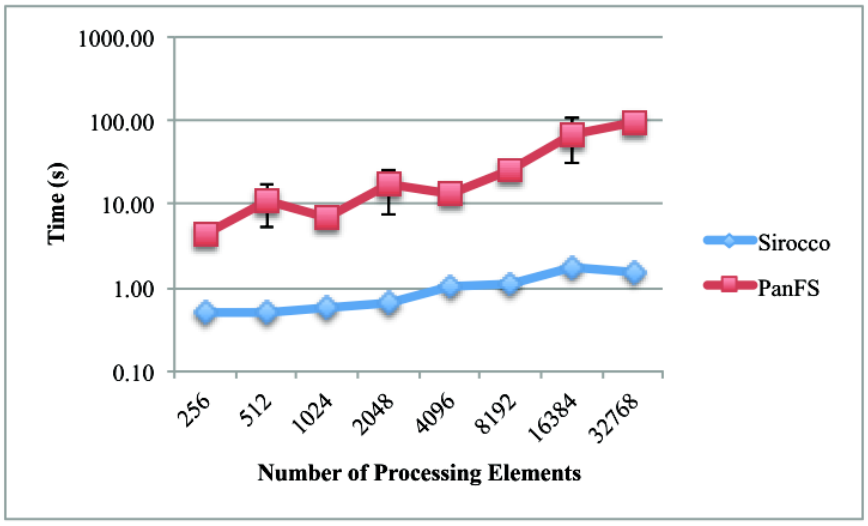

Figure 3.3.: Average checkpoint time, Source: [2]

4. Summary

The hardware changes rapidly and the processed data is much larger than the stored data. So the file systems have to catch up with speed. The newest generation of file systems handles issues with concurrency and consistency better. A network file system is easy to use, but a hardware failure crashes the whole system and makes it a single point of failure. SAN is expensive, because every connection needs a fiber channel cable and adapter. The Sirocco File System uses a peer-to-peer concept with free data movement and placement. It supports all media types and the scalability should be high. First results show a performance increase up to 60%, but 25% more nodes were needed to manage the data movement.

Further information for
File System Projects on
`http://filesystems.org/all-projects.html`

"At the end, everything is saved and stored as blocks on tape, magnetic disk, optical disk or flash memory." (from [6]).

Bibliography

[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.

[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

[9] Scale out vs scale up the basics, `http://itknowledgeexchange.techtarget.com/storage-soup/scale-out-vs-scale-up-the-basics/`, 2015, accessed 03-November-2015

[10] SAN image, `http://gerska-consulting.de/wp-content/uploads/2014/11/SAN.png`, 2015, accessed 06-November-2015

List of Figures

Appendices

A. Burst buffers

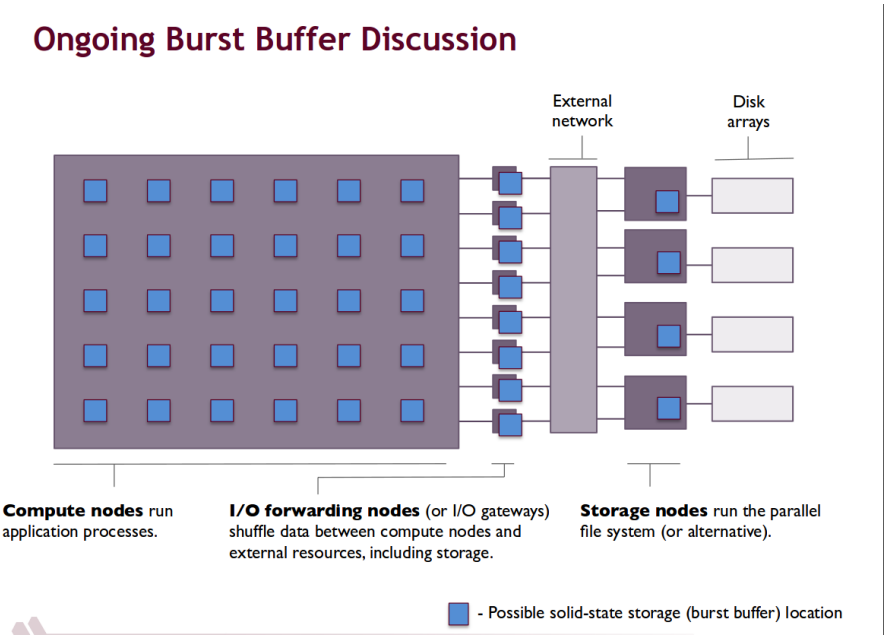

Figure A.1.: Burst buffer can be at the blue locations, source: advisor