University POLITEHNICA of Bucharest
Faculty of Automatic Control and Computer Science

TEZĂ DE DOCTORAT

Optimizări și soluții de stocare distribuită

Distributed Storage Solutions and Optimizations

Autor: Ing. Sorin-Andrei Pîștirică

COMISIE DOCTORAT

Preşedinte	**Prof. Dr. Ing. Adina Magda Florea**	**Universitatea POLITEHNICA din Bucureşti**
Coordonator ştiinţific	**Prof. Dr. Ing. Florica Moldoveanu**	**Universitatea POLITEHNICA din Bucureşti**
Referent	**Prof. Dr. Ing. Nicolae Ţăpuş**	**Universitatea POLITEHNICA din Bucureşti**
Referent	**Prof. PhD. Eng. Alexandru Soceanu**	**Munich University of Applied Sciences**
Referent	**Prof. Dr. Ing. Ştefan Pentiuc**	**Universitatea „Ştefan cel Mare” din Suceava**

București, 2015

București 2015

TABLE OF CONTENTS

1 INTRODUCTION 10

1.1 SCIENTIFIC PUBLICATIONS IN CONNECTION WITH THIS THESIS 12
1.2 THESIS OUTLINE 13

2 STORAGE SYSTEMS: STATE-OF-THE-ART 14

2.1 STORAGES CLASSIFICATION 14
2.2 REQUIREMENTS FOR DISTRIBUTED STORAGES 15
2.3 STORAGE SYSTEMS SCALING METHODS 17
2.4 SHARING SEMANTICS 20
2.5 LOW LEVEL DATA ORGANIZATION 20
2.5.1 Data organization at device level 20
2.5.2 Networked storage data organization 22
2.5.3 Generalization of data organization 23
2.5.4 Storage units 23
2.5.5 Trends towards object based storage systems 24
2.6 DISTRIBUTED STORAGES ARCHITECTURES 25
2.6.1 Andrew File System 25
2.6.2 Google File System 26
2.6.3 General Parallel File System 28
2.6.4 Lustre File System 29
2.6.5 Ceph File System 30
2.6.5.1 Data storage: RADOS 32
2.6.5.2 Namespace management 33
2.6.5.3 Decentralized data distribution 33
2.6.5.4 Security in Ceph 35
2.6.6 Comparison 35
2.7 NETWORK TECHNOLOGIES FOR DISTRIBUTED STORAGES 36
2.7.1 The topology 36
2.7.2 Protocol stack 39
2.7.3 Real world considerations 40
2.8 INTERFACING WITH STORAGE SYSTEMS (CASE STUDY: CLOUDS) 41

3 CASE STUDY: INFRASTRUCTURE FOR ELEARNING ENVIRONMENTS 44

3.1 MOTIVATION 44
3.2 CLOUDS STORAGE SYSTEMS AND ENHANCED NETWORKS FOR ELEARNING ENVIRONMENTS 44
3.2.1 Data distribution for eLearning environments 45
3.2.2 Clouds network 46
3.3 CLOUDS AND ELEARNING ENVIRONMENT: PROPOSED ARCHITECTURE 47
3.3.1 System administration and eLearning components 48

3.3.2 The storage system *48*
3.3.3 The network profile *48*
3.3.4 Real world considerations *49*

4 OPTIMIZATIONS OF DISTRIBUTED STORAGE SYSTEMS BASED ON HARDWARE ACCELERATORS 50

4.1 MOTIVATION 50
4.2 INTEGRATED PACKET PROCESSING ENGINES 51
4.3 HARDWARE PACKET PROCESSING 52
4.4 HARDWARE ACCELERATED CLUSTER NODES 53
4.5 CASE STUDY: HARDWARE ACCELERATED CEPH WITH QORIQ™ 55
4.5.1 QorIQ™ packet processors *56*
4.5.2 Accelerated Ceph's nodes *57*
4.5.3 Accelerated RADOS based on queue prioritization *58*
4.5.4 Micro-benchmark results *59*

5 NETWORKING CHARACTERISTICS IN CONVERGED INFRASTRUCUTRE 61

5.1 FIBRE CHANNEL 62
5.2 DATA CENTER BRIDGING 63
5.2.1 Priority Flow Control *63*
5.2.2 Enhanced Transmission Selection *63*
5.2.3 Data Center Bridging eXchange *63*
5.2.4 Quantized Congestion Notification *63*
5.3 QCN WEAKNESSES AND IMPROVEMENTS 67
5.3.1 The Fair QCN Solution *69*

6 DYNAMIC LOAD BALANCING ALGORITHM BASED ON QCN NETWORKS 72

6.1 MOTIVATION 72
6.2 ALTERNATIVE SOLUTIONS 72
6.3 QCN WEIGHTED FLOW QUEUE RANKING 73
6.3.1 Defining the topology *74*
6.3.2 QCN-WFQR indicatives *74*
6.3.2.1 Flows 75
6.3.2.2 Ranking 75
6.3.2.3 Flow shares 75
6.3.2.4 Weighting 78
6.3.3 QCN-WFQR algorithm based on standard QCN *80*
6.3.4 Algorithm analysis *81*
6.3.4.1 Flows shares analysis 81
6.3.4.2 Congestion indicatives analysis 82
6.3.4.3 QCN-WFQR simulator 82
6.3.4.4 QCN-WFQR simulations analysis 84
6.3.5 QCN-WFQR applications *87*
6.3.5.1 Elections in distributed systems with homogenous nodes 87
6.3.5.2 Dynamically balanced flows in computer networks 88

7 CONCLUSIONS AND FUTURE WORK 90

7.1 THE ORIGINAL CONTRIBUTIONS OF THE THESIS 90

7.2 FUTURE WORK 93

8 AUTHOR'S PUBLICATIONS 94

8.1 PATENTS 94

8.2 ARTICLES 94

9 BIBLIOGRAPHY 96

10 ACRONYMS AND DEFINITIONS 99

LIST OF FIGURES

Figure 1: Storage systems classification based on four basic characteristics 14
Figure 2: SNIA file system taxonomy 14
Figure 3: Storage systems scale-out diagram 17
Figure 4: Mass storage solution (RAID based) 18
Figure 5: Device: structure based data organization 21
Figure 6: Device: log based data organization 21
Figure 7: Device: tree based data organization (BTRFS leaves) 21
Figure 8: Hybrid storage system: SAN / NAS 21
Figure 9: Generalized data layout 23
Figure 10: Andrew File System architecture 25
Figure 11: Google File System architecture 27
Figure 12: General Parallel File System architecture 28
Figure 13: Lustre File System architecture 29
Figure 14: Lustre file mapping 30
Figure 15: Ceph file system architecture 31
Figure 16: Decentralized data distribution (layout) 34
Figure 17: Network topologies 38
Figure 18: Cloud components 41
Figure 19: Open Stack storage integration 42
Figure 20: eLearning/ cloud – architecture proposal 47
Figure 21: Block diagram of a generic packet processing SoC 51
Figure 22: Typical stages for receiving packets 52
Figure 23: Hardware accelerated models 53
Figure 24: Ingress/Egress flow chart 57
Figure 25: QCN sampling probability 64
Figure 26: Quantized Congestion Notification (QCN) – high level view 65
Figure 27: Reaction point's rate limiter 65
Figure 28: Quantized Congestion Notification – Congestion Notification Domain defense 68
Figure 29: Quantized Congestion Notification – Congestion profiling 71
Figure 30: Port's queue ranking 73
Figure 31: Flow share synthetic tests 81
Figure 32: NS3-QCN-WFQR high-level view 83
Figure 33: QCN-WFQR (NS3): simulation results for variation of flow related indicatives 85
Figure 34: QCN-WFQR (NS3): simulation results for variation of queue related indicatives 86
Figure 35: Elections in distributed systems with homogenous nodes 87

LIST OF TABLES

Table 1: File sharing semantics taxonomy 20
Table 2: Block devices vs. Object Storage Devices 24
Table 3: Google File System semantics 27
Table 4: RADOS main messages 32
Table 5: Storage systems characteristics mapping 35
Table 6: Network topologies scaling properties 38
Table 7: Protocol stacks for storages 39
Table 8. Accelerated MDS micro-benchmark 60
Table 9. Accelerated RADOS micro-benchmark I 60
Table 10. Accelerated RADOS micro-benchmark II 60
Table 11: Congestion System Indicatives 74
Table 12: QCN-WFQR parameters 83

Abstract

The distributed storage systems are one of the most I/O intensive systems, composed of multiple modules each with different demands from the networking subsystem, such as low latency or high throughput.

The present thesis begins with the state-of-the-art of the technology analyzed from different perspectives including: scaling methods, architecture, topologies and protocols used, that underlie a proposal for eLearning infrastructures.

Packet processing at multi-gigabit rates of I/O intensive systems imposes a great pressure especially at receiver side, while at sender side there are multiple techniques to solve any occurring bottleneck. A method of solving receiver's weaknesses and even increasing its performance is the usage of integrated hardware systems of dedicated packet processors with general purpose cores. Therefore, the performance of each node is increased by combining software design with multicore capabilities and with features of hardware engines for classification and distribution of data flows.

When combining applications with specific traffic requirements and general purpose traffic, the result is often inefficiency. In this context, Ethernet has become the primary network protocol used due to its undeniable advantages such as low cost, high speeds or ease of management. Being a best effort protocol, the IEEE Data Center Bridging Task Group developed a series of enhancements for Layer 2, including Quantized Congestion Notification (QCN) and enabled a lossless environment. QCN provides congestion control, but it doesn't solve the fairness of congestion profile within the entire network. The proposed solution aims the usage, cooperatively or automatically, of several congestion indicatives computed by Quantized Congestion Notification – Weighted Flow Queue Ranking (QCN-WFQR) algorithm, to balance the traffic load and increase the system responsiveness.

Acknowledgements

I would like to dedicate my thesis to my wonderful parents, who made all my accomplishments possible and for their patience and trust during all these years.

I would like also to thank to my advisor, Prof. Florica Moldoveanu for her guidance and support during the past several years and for the opportunity to find out what research is. And I gratefully acknowledge my friends PhD. Victor Asavei, which convinced me to start this journey and PhD. Mihai Claudiu Caraman for his excellent suggestions to my work.

Last but not least, I am grateful to have had the opportunity to innovate within IBM Corporation and to the professional development within Freescale Semiconductor Inc.

1 INTRODUCTION

During the last ten years internet traffic volume grew more than 300 times and the pressure on technologies used for network infrastructure increased significantly. Besides internet, there are systems such as clouds or data centers, where the pressure on I/O intensive systems, like storage systems, influenced the evolution of network infrastructure concerning the capital costs without performance penalty and pushed the design of the software towards distribution.

Considering the above, I studied various features of storage systems and propose an infrastructure for eLearning technologies. Since storage systems are very I/O intensive, I propose a solution for increasing their performances based on dedicated hardware for packet processing. Furthermore, considering the trend of converged networks emerged from the need to ease the management and reduce the capital costs, I propose several improvements to Quantized Congestion Notification protocol as well as an algorithm for dynamic balancing of traffic flows: Quantized Congestion Notification – Weighted Flow Queue Ranking.

In context of distributed storage systems, the scaling-out methods based on file system layers are very important. Each layer requires different type of network, thus generates different types of distributed storage systems. It is also important to see how they are characterized based on classifications and what are the main deficiencies of the existing implementations. Therefore, I propose a simple, yet comprehensive classification based on four main characteristics: locality, sharing, distribution level and semantics of concurrent access. In terms of low level organization of data there are several ways, but all follow the same constituent elements, so I propose a suitable generalized data layout regardless the distribution status of the system. Within proposed layout, data is distributed in different units (i.e. files, objects, blocks or data segments) analyzed further, with an increasing trend towards objects, due to autonomous characteristic of object devices. Afterwards, I studied five different architectures of distributed storage systems (Andrew File System, Google File system, General Parallel File System, Lustre File System and Ceph) with more focus on Ceph's particularities, used later as case study and performance measurements for the optimizations proposals. Afterwards, I studied the way that everything is linked together and using graph theory I present several key characteristics that influence properties such as throughput, capital cost, fault tolerance or high availability, followed by studies of protocol stacks used to identify different alternatives suited for converged networks.

I propose several enhancements to solve issues related to system overheads due to processing of flows at multi-gigabit rates. The enhancements are based on integrated hardware systems of dedicated packet processors with general purpose cores. Besides system overhead

issues, another challenge is the enhancement of the network hardware performance to make optimal use of the available network bandwidth. These problems are overcome by combining multi-core processors with parallel processing and multi-function network hardware capabilities. Thus, I propose a generic hardware packet processor emphasizing its components along with processing stages for each frame. I also designed two alternatives for coexistence of different applications with different traffic demands: Per Core Cluster Node and SMP Cluster Node. Using Ceph as case study, I propose two different optimizations: accelerations methods of each node using the proposed models listed above (i.e. A-MDS, A-MON and A-OSD) and a method of decreasing latency of sensitive flows based on queue weighting (A-RADOS). The proposed solutions are supported by several micro-benchmark results based on P2041 QorIQ™ as integrated packet processor.

Furthermore, in the context of network convergence in data centers, I/O protocols (such as SCSI) do not have contention or retransmission support and they require a lossless transmission environment, such as Fibre Channel – a high speed, low latency and lossless network by design. Ethernet by design is a best effort communication environment and with IP protocol it provides an end-to-end network for reliable transport protocols, such as TCP. In absence of reliable protocols, Ethernet has been enriched with a set of protocols which enabled a lossless medium: Priority Flow Control (PFC), Enhanced Transmission Selection (ETS), Data Center Bridging Capabilities exchange (DCBx) and Quantized Congestion Notification (QCN). Despite the main purpose of these enhancements, there are other uses cases, for example “TCP Incast” that can be improved using QCN.

Basically, QCN provides congestion information to the source to avoid packet loss, but it doesn’t solve the congestion fairness within the entire network. For this, I propose QCN Weighted Flow Queue Ranking (QCN-WFQR), an algorithm based on Quantized Congestion Notification for dynamic workload balancing. The algorithm can be used in traditional computer networks or in new Software Defined Networks. It computes a series of congestion indicatives relative to each flow per each congested point and a series of system wide congestion indicatives. These indicatives can be used cooperatively (by all elements in the cluster) or automatically (by a system profiler or SDN controller) to increase the overall system performance. For exemplification, I propose two different applications for the use of these congestion indicatives. Former application proposes a method of choosing replicas in distributed and parallel file systems to achieve a less congested network. The latter application propose a method of distributing traffic workload in the network by migrating already established flows to alternate less congested paths, thus reducing the need to slow down the traffic at the source.

1.1 Scientific publications in connection with this thesis

Patents

[App. #20150023172]. **Sorin A. Pistirica,** Dan A. Calavrezo, Casimer M. DeCusatis, Keshav G. Kamble, "**Congestion Profiling of Computer Network Devices**", Patent Pending,
USPTO: http://patents.justia.com/patent/20150023172, Jul 16 – 2013

[Patent #8891376]. **Sorin A. Pistirica**, Dan A. Calavrezo, Keshav G. Kamble, Mihail-Liviu Manolachi, "**Quantized Congestion Notification—defense mode choice extension for the alternate priority of congestion points**",
USPTO: http://patents.justia.com/patent/8891376, Oct 07 – 2013

Articles

[PIST, 2013] **Pistirica Sorin Andrei**, Caraman Mihai Claudiu, Moldoveanu Florica, Moldoveanu Alin, Asavei Victor, "**Hardware acceleration in CEPH Distributed File System**", ISPDC: IEEE 12th International Symposium on Parallel and Distributed Computing, Bucharest, June 2013, pp. 209-215, **IEEE Indexed**

[PIST, 2014/1] **Pistirica Sorin Andrei,** Victor Asavei, Horia Geanta , Florica Moldoveanu , Alin Moldoveanu , Catalin Negru , Mariana Mocanu, "**Evolution Towards Distributed Storage in a Nutshell**", HPCC: The 16th IEEE International Conference on High Performance Computing and Communications, August 2014, Paris, pp. 1267-1274, **IEEE Indexed**

[PIST, 2014/2] **Pistirica Sorin Andrei**, Asavei Victor, Egner Alexandru, Poncea Ovidiu Mihai, "**Impact of Distributed File Systems and Computer Network Technologies in eLearning environments**", eLSE: Proceedings of the 10th International Scientific Conference "eLearning and Software for Education", Bucharest, April 2014, Volume 1, pp. 85-92, **ISI Indexed**

[PIST, 2015] **Pistirica Sorin Andrei**, Poncea Ovidiu, Caraman Mihai, "**QCN based dynamically load balancing: QCN Weighted Flow Queue Ranking**", CSCS: The 20th International Conference on Control Systems and Computer Science, Bucharest, May 2015, Volume 1, pp. 197-205, **ISI Indexed**

1.2 Thesis outline

- **Chapter 1** – the introduction;
- **Chapter 2** – provides an overview of the state-of-the-art of storage systems dealing with several topics, including: scaling methods, taxonomies, data organization, system architecture and networking;
- **Chapter 3** – presents a proposal of an infrastructure designed for eLearning systems, as case study;
- **Chapter 4** – presents several optimizations proposed for nodes in distributed systems based on hardware packet processing. A case study based on Ceph and QorIQ™ processors is also presented;
- **Chapter 5** – deals with network convergence in data centers emphasizing different weaknesses of QCN protocol. It is presented a proposal for preserving QoS policies from unwanted traffic injections through edge ports in case of automatic QCN configuration;
- **Chapter 6** – presents a novel algorithm for dynamic load balancing in congested aware networks (**QCN-WFQR**), which proposes two sets of congestion indicatives used to balance the traffic load and achieve a more responsive system and an increase in the overall performance;
- **Chapter 7** – concludes my contributions presented in this thesis.

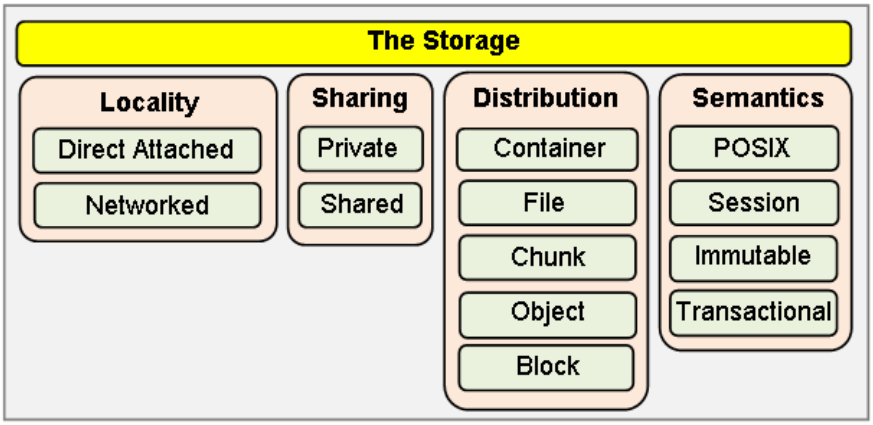


Figure 1: Storage systems classification based on four basic characteristics

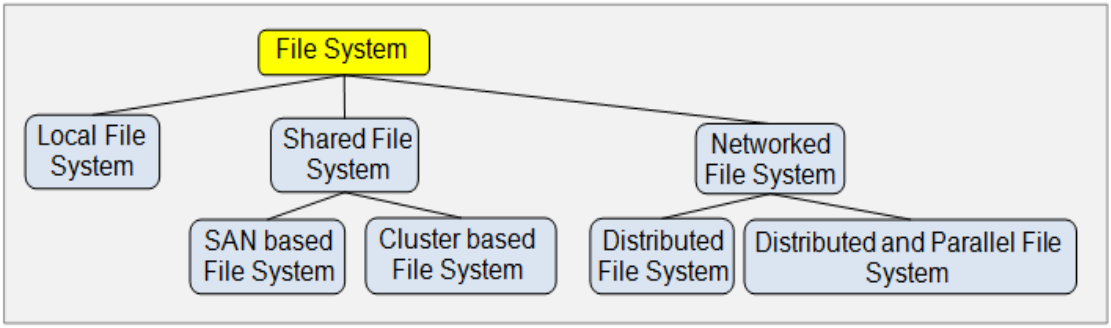


Figure 2: SNIA file system taxonomy

2 STORAGE SYSTEMS: STATE-OF-THE-ART

In general, any **storage system** comprises a set of devices and a collection of software modules for managing data units, such as blocks, objects or files.

Distributed storages are special cases, where storage elements are distributed over a set of nodes within a computer network. There are cases where only storage devices are distributed, but to avoid the single points of failures or to satisfy different requirements (section 2.2), the software modules are distributed as well.

2.1 Storages classification

Storage systems classifications are very important, mostly because it drives the intrinsic characteristics of them and points out the deficiencies of existing implementations. Therefore, I propose a classification of storage systems based on four main characteristics (Figure 1) **[PIST, 2014/1]**, as follows:

- **Locality**: direct attached (i.e. systems that has no network devices involved at any level) and networked (i.e. systems that contains network devices at least at one level);

- **Sharing capability**: private system (i.e. unshared data – dedicated to only one tenant) or shared between multiple tenants;
- **Distribution level**: refers to the data unit used for distribution: containers (i.e. groups of files bound to the same organizational relationship such as trees), files, chunks or segments, objects and blocks;
- **Semantics**: refers to the sharing semantics of concurrent access (section 2.4).

The main purpose of this classification is to include any storage system regardless of its data organization or architecture.

Storage Networking Industry Association (**SNIA**) proposes a different taxonomy [8] that splits file systems in three major categories: **local**, **shared** and **network** (Figure 2). The **local file systems** are co-located on the same machine with client/server, **shared file systems** use blocks shared among clients and **network file systems** use files instead of blocks which are also shared among clients. The two classifications are not disjunctive, but in my opinion SNIA's taxonomy is a mixture of architectural considerations and storage intrinsic characteristics.

2.2 Requirements for distributed storages

In the early 70'es at the beginning of computer networks, the main purpose of distributed storage systems was just to share small pieces of data between nodes. Later on, with the advent of cloud computing, HPC domains or internet applications such as Google or Facebook, the need for storage space increased exponentially, therefore I propose the following list of basic requirements based on articles published in the field:

1) Data Sharing

- Sharing stored data in a multi-tenant system.

2) Storage Scalability

- Increase or decrease storage capacity by adding or removing nodes in the system.

3) Storage Elasticity

- Dynamic adjustment of storage capacity from client point of view without affecting its data or system structure.

4) Transparency

Transparency is a more general term and it refers to the fact that clients are unaware of the system architecture and therefore there are several types of transparency:

a. Access transparency

- *Preserving access interface regardless of system architecture: distributed system or directly attached.*

b. Location transparency

- *Files do not have to contain location information.*

c. Failure transparency

- *Preserve client's data in case of system failures.*

d. Replication transparency

- *Clients should be unaware of data replication.*

e. Migration transparency

- *Clients should be unaware of data migration.*

5) High Availability of data

- Ensure a high degree of data availability in case of partial system failures.

6) Fault Tolerance

- The system continues to work properly in case of partial failure.

7) System recovery

- The system has to be able to recover after unexpected partial or total failures.

8) Data Balanced Distribution

- The system has to be able to distribute data uniformly among a large number of storage nodes.

9) Workload balance

- The system ability to balance access requests among cluster nodes in order to achieve certain performance (e.g. minimizes response time, avoid node overload or optimal resource utilization).

10) Data Migration

- Data migration among storage nodes and balance space utilization in case of storage scaling.

11) Concurrent and Consistent Data Access

- The system ability of multiple tenants to concurrently access a consistent view.

12) Snapshot

- The system ability to save and return to a specific system state.

13) Archive

- The system ability to provide a complete data copy of a snapshot.

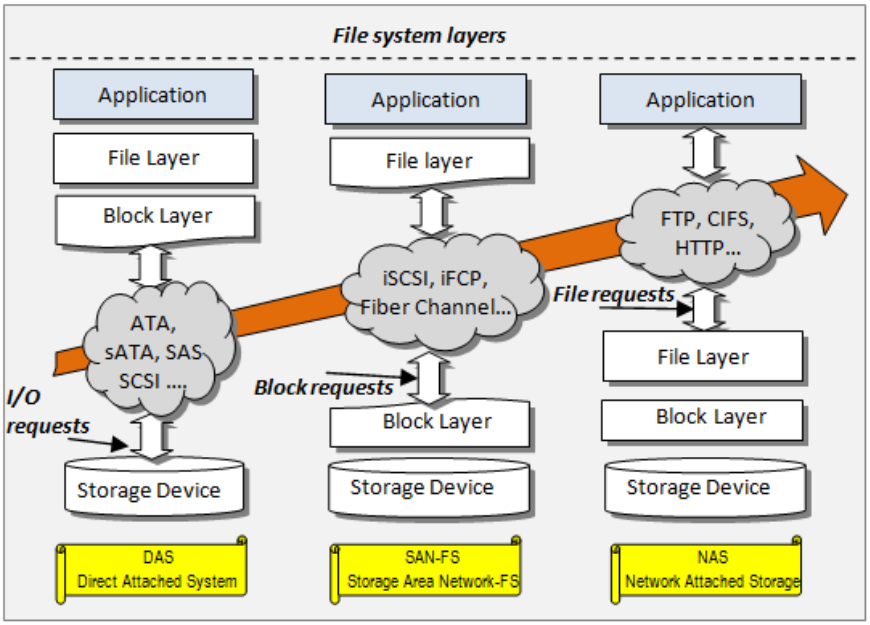


Figure 3: Storage systems scale-out diagram

14) Network infrastructure performance

- The ability of *network infrastructure* to meet a certain performance degree in terms of throughput, latency or congestion.

15) Data Security

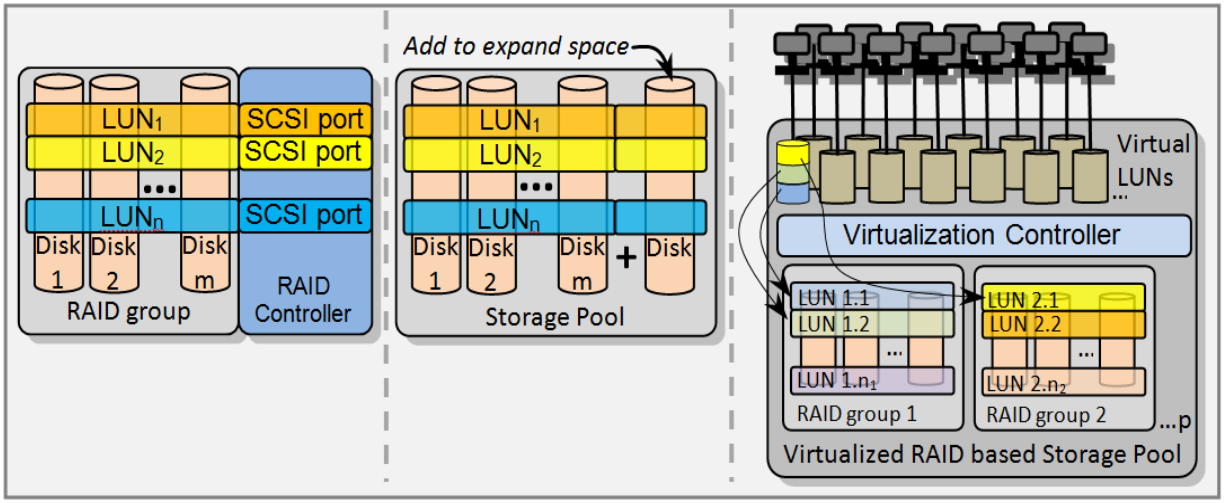
- Mechanisms of data protection from unwanted actions of unauthorized tenants.

2.3 Storage systems scaling methods

To meet nowadays demands of storage space, which is very high reaching at petascale or even exascale size, the number of storage devices is very large (i.e. scale-out) within the cluster **[PIST, 2014/1]**. This can be achieved by distributing data at different file system levels (Figure 3).

File systems, generically are composed by several abstraction layers (presented in Figure 3), as following:

- hardware layer (storage devices);
- block or object layer (sequence of bits stored on storage devices);
- file layer (typed data managed by the underling operating system);
- application layer.



(a). *RAID group* (b). *Storage* Pool (c). Raid *based* Storage Pool

Figure 4: Mass storage solution (RAID based)

Depending on the layer at which the system is decoupled and computer networks are introduced, different types of systems are built:

- SAN based distributed storage systems are built by decoupling the system at block I/O level, where protocols such as SCSI or SATA are carried out by different lossless communication environments, such as Fibre Channel;
- NAS based distributed storage systems are built by decoupling the system at file level, where protocols such us CIFS – Common Internet File System or NFS – Network File System – are used to carry file requests;
- Hybrid systems can be built by mixing these SAN and NAS types (Figure 8);
- Furthermore, *distributed and parallel file systems* (e.g. GFS, Lustre or Ceph) are built by abstracting storage devices using data segments or objects as storage units.

Mass-Storage systems

Redundant Arrays of Independent Disks (**RAID)** are mass-storage systems that can span data within multiple disks from a **RAID group** (i.e. set of storage devices bound to a RAID type), Figure 4 – a. These systems are based on two main ideas: improve performance by accessing multiple disks in parallel (large data chunks stripped over multiple disks or small independent data chunks accessed in parallel) and improve high availability and fault tolerance by duplicating data over multiple devices. There are several RAID types (**RAID0** – to – **RAID6**) differentiated by granularity of stored data and redundancy patterns. Storage systems may also

combine different RAID types obtaining what is called **Nested RAIDs** (e.g. RAID1+0). Also, there are implementations of RAID types other than standard, such as: **Linux MD** or **Hadoop RAID**.

A typical **RAID group** is composed of several disks (16 maximum) with **LUNs** (Logical Unit Numbers also called Volumes) configured (Figure 4 – a) and accessed with an offset (as a direct attached storage) called Logical Block Addressing (**LBA**). The storage space is expanded by adding new disks to the RAID group and extends related LUNs. It is worth to mention that RAID groups remain the only mass storage system that guarantees a specific degree of performance for critical applications, for example in systems such as High Performance Computing.

Furthermore, EMC[1] introduced **Storage Pools** – collections of logical/physical disks or RAIDs managed together (Figure 4 – b), used to simplify space management. To expand a typical Storage Pool a new disk is added and LUNs can be expanded, thus each host has an increased storage space. **RAID based Storage Pools** are constructed by combining RAIDs with storage pools. Usually, in a storage system multiple storage pools are configured based on their characteristics, such as: size, performance, latency and so on. In contrast with standard RAID groups, Storage Pools can expand to hundreds of devices, are more flexible and easy to manage.

Fine grained management space is achieved with virtualization of storage pools into **Virtual LUNs** or **Virtual Volumes** (Figure 4– c) [3] to meet application's requirements in terms of capacity and quality of service, where each virtual volume is usually made of different numbers of equal RAID LUN sizes (equal physical storage slices). The mapping between Virtual LUNs and RAID LUNs are made by a **Virtualization Controller**. Usually when a new RAID group is added to the storage pool, the already stored data is not migrated (to rebalance the system) to all available disks, rather the new data is stored using the new RAID group and the remaining space from old groups.

From architectural point of view, there are three main approaches of mass-storage virtualization:

- Host-based (e.g. *Linux Logical Volume Management*: LVM) [4];
- Device-based (e.g. disk arrays: RAIDS);
- Network-based (e.g. SAN systems) [5] [6].

[1] EMC: corporation that provides IT storage hardware solutions.

Semantics	Description
POSIX Semantics	Every operation on a file is instantly visible to all clients.
Session Semantics	No changes are visible until the file is closed.
Immutable Files	No updates are possible.
Transactional semantics	Based on ACID properties – all or nothing property.

Table 1: File sharing semantics taxonomy

2.4 Sharing semantics

One of the most important characteristics of the distributed storage systems is **sharing semantics**. Sharing semantics refers to the semantics of concurrent access of two or more tenants and it defines the level of data consistency in different situations.

Andrew S. Tanenbaum in his book "Distributed Operating Systems" [7] proposed a simple and comprehensive set of four different types of file sharing semantics listed in Table 1. Depending on the system architecture, its purpose and required level of performance a type of consistency is chosen. In case of distributed systems, POSIX semantics is very difficult to achieve, mainly because it requires complicated synchronization mechanisms.

2.5 Low level data organization

2.5.1 Data organization at device level

In the context of data organization at device level, I consider three main organizational types: structured (Figure 5), log-based (Figure 6) and tree-based (Figure 7). The main difference between structured and log-based organization is that the former organization type has distinct areas for inodes and data blocks, while the later stores them in continuous form (i.e. logs: checkpoint area points to the right version of inode mapping, while data is continuously added to the log) – the advantage is that the reads for both inodes and data blocks are made without moving the head of the device. Tree-based organization (e.g. B-tree File System: BTRFS) uses a copy-on-write (COW) model, where data is organized in a binary tree structure. Basically, when a node is modified a shadow tree is created and pointers are updated accordingly up to the root (and this is how snapshots are created).

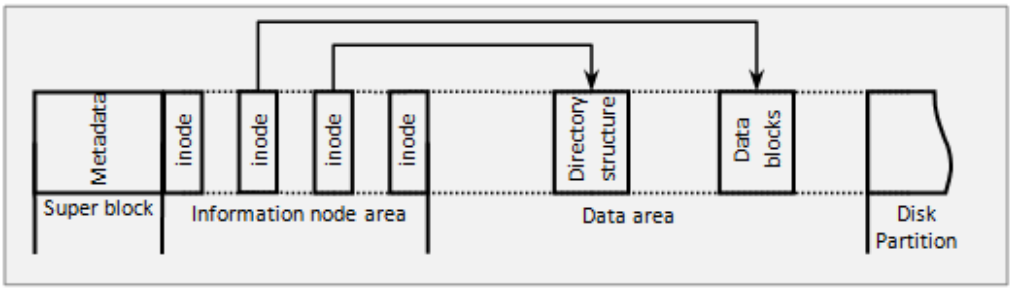


Figure 5: Device: structure based data organization

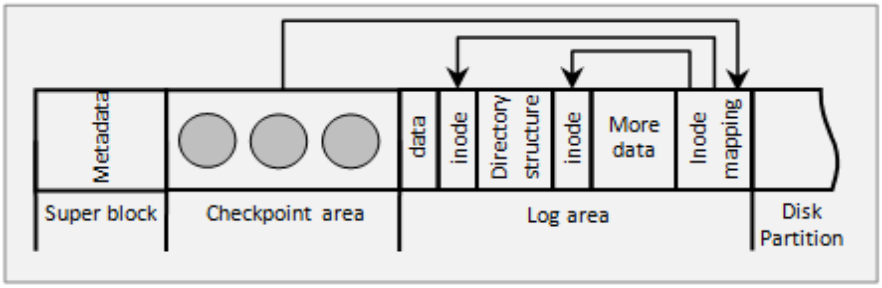


Figure 6: Device: log based data organization

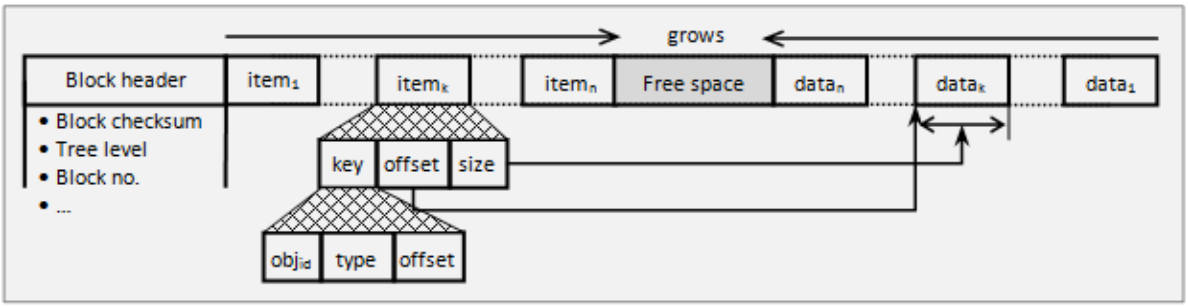


Figure 7: Device: tree based data organization (BTRFS leaves)

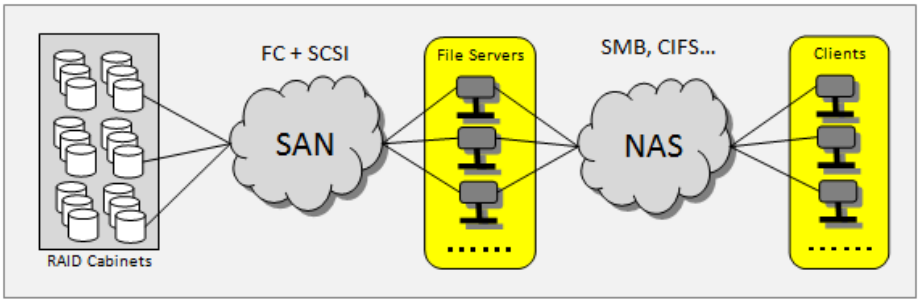


Figure 8: Hybrid storage system: SAN / NAS

On hard drives, also due to the fact that Linux greatly increased in popularity in the last decade, it mostly uses structure based organization of data: Unix File System (UFS) also called Berkeley Fast File System, BSD Fast File System or Fast File System. In Linux, the implementation of UFS is the Extended File System [2]. During time, its design has undergone massive changes to meet different requirements such as logging or increased storage space (i.e. EXT version 4). Briefly, EXT layout [2] is organized into block groups, data management units (i.e. group descriptors and data block bitmap), file management (i.e. inode bitmap and inode table) and data area (i.e. data blocks). Block groups optimize disk seeks and each of these groups contains information about the status of the file system called *superblock*. With following EXT revisions, the superblock become quite large, therefore only several block groups where chose to store it (i.e. sparse superblock).

Following an evolutionarily line, Oracle released in 2007 a file system with a tree based data organization (Binary Tree File System [1]) included in Linux kernel in 2009. In the file system's tree, the leaves contains an array of items (Figure 7) and the inner nodes contains information where to look for an item and where is the next node located. An item can be anything from an inode, file or directory and so on. Related to comparison with UFS, BTRFS scales better than UFS and also showed performance improvements.

2.5.2 Networked storage data organization

Generally speaking, in any distributed storage system, data is divided in different types of units (e.g. files, blocks or objects) distributed across the nodes of a network using different techniques, such as mapping tables or hash functions. As in the case of local file systems, in distributed file systems as well, the organization of data segments is specified in a type of metadata structure, distributed as any regular file or distributed in a private cluster or stored in a centralized way.

Usually, a distributed storage system does not have the concept of partition, but sometimes there are implemented mechanisms that can be considered as similar concepts, for example volumes for Andrew File System or the multiple systems managed by MGS in Lustre.

SAN and **NAS** are two of the most known networked file systems. In terms of architecture, SAN based file systems usually use storage nodes with RAID disks, while NAS based file systems are often used along with SAN environments, were file servers are constructed over NAS and data is stored in a SAN medium (i.e. hybrid file system), Figure 8.

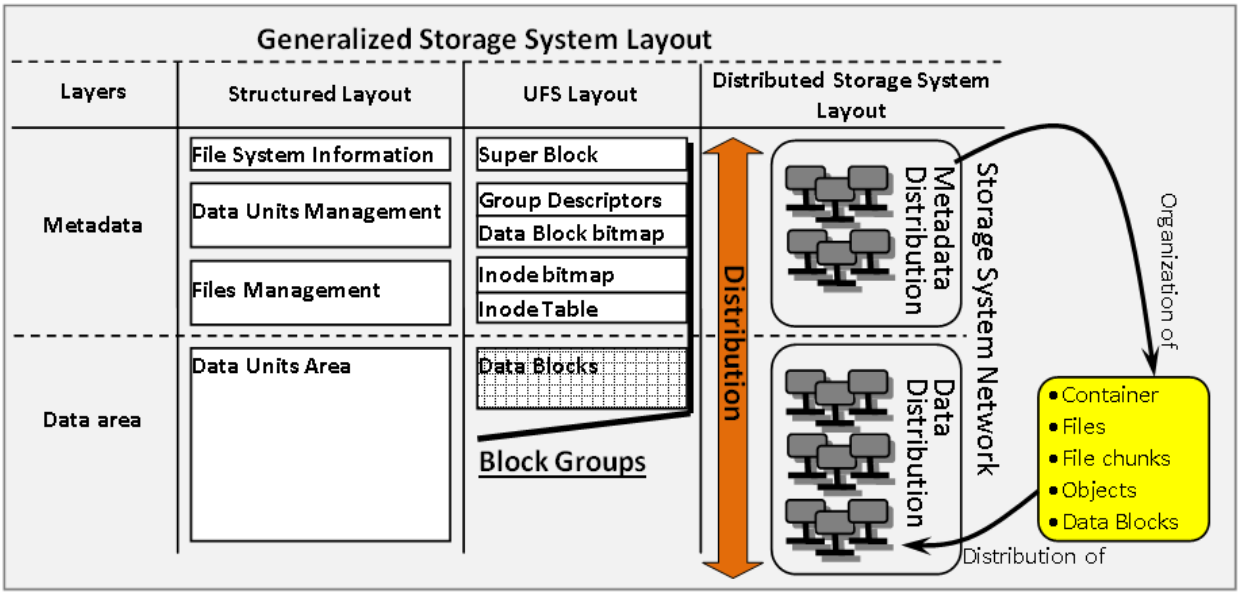


Figure 9: Generalized data layout

2.5.3 Generalization of data organization

Regardless of file semantics (such as hierarchical or semantic file system) or distribution status, any file system has two main layers (Figure 9):

- **Metadata area**: specifications of data organization;
- **Data area**: a flat address space of data units.

2.5.4 Storage units

Stored data may be distributed using different types of units (as shown in Figure 9): containers, files, segments, blocks or objects.

Files are self-contained information, organized (e.g. hierarchically, semantically and so on) and typed (e.g. binary, directory and so on) managed and interpreted by the underlying operating system. Besides the organization method and type, files have also access rights (e.g. ACL: Access Control List) for different operations (e.g. read, write or execute) per groups of users or per user basis – data security.

Data segments (i.e. chunks of data from files) are scarcely used for distribution (e.g. Google File System uses chunks of 64Mb as distribution unit), due to its weaknesses, such as lack of standardized support in the operating systems.

Actions	Block Device	Object Storage Device
Partitions	Available	Available
Space Management	External	Internal
Operations	*N/A*	Create object
		Delete object
	Read block	Read Object
	Write block	Write Object
	N/A	Get Attributes
		Set Attributes
Addressing	Block range	Object/ byte range
Security	Per device	Per object
Recovery	External mechanism	Internal mechanism
Snapshot	*N/A*	Available
Containers	*N/A*	Collection of objects

Table 2: Block devices vs. Object Storage Devices

Blocks and **objects** are the most used data units for distribution and is worth to mention, that nowadays tendencies are to use mostly objects (**Object Storage Devices** – OSDs in composition of **Object Store** – ObS [9]), because OSDs are more autonomous devices with several strength (Table 2), including:

- **OSD v1**: internal space management, object manipulation standard API and object based security [10];
- **OSD v2**: completes the feature list with snapshot capability, object containers and recovery improved mechanism [11].

It is worth to mention that an object may represent an entire file or file fragments or even fragments from multiple files. The representation depends of implementation and requirements of the file system, such as providing multiple file fragments in parallel or improving distribution profile (e.g. uniform distribution).

2.5.5 Trends towards object based storage systems

Currently, the tendency is to use file segments to distribute data, rather the entire file or even groups of files, because for example, this way a higher degree of availability and a better fault tolerance is achieved (two of the most important requirements of distributed systems).

Although, there are several types of data representation used for distribution, objects seems to gain in popularity, since researchers claim that OSDs take advantages from DAS, SAN and NAS architectures grouping them in a single device [12]. Basically, it addresses issues related to **security**, **scalability** and **management**, (Table 2). Therefore, security is improved by adding credentials to each operation, while in contrast a block device has per LUN based

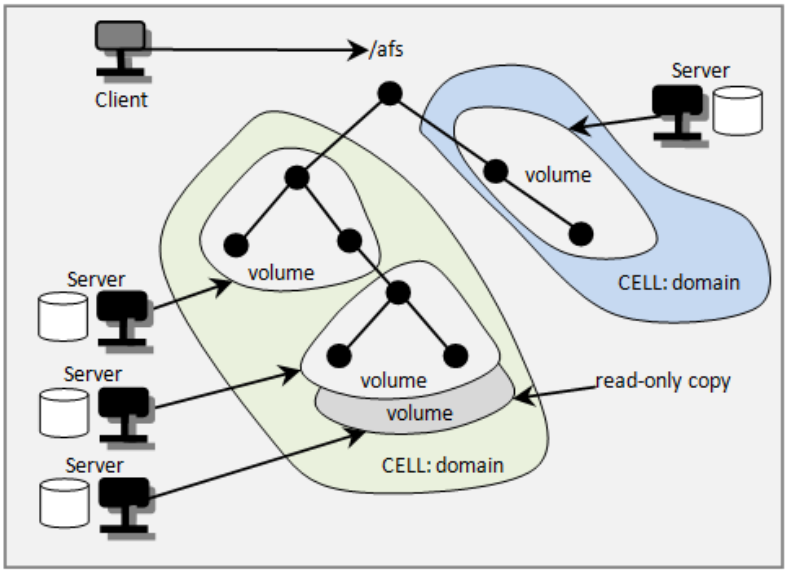


Figure 10: Andrew File System architecture

security. Scalability and management is improved by moving space management from metadata servers to storage controllers. And, one management advantage that emerges from OSDs characteristics is that object based storage systems simplify data layout by replacing large blocks lists with small objects lists and distribute the low-level block allocation problem. Although, it has many advantages object based distribution does not solve the fundamental issue of distributing data among a very large number of devices.

2.6 Distributed storages architectures

There are many distributed storage systems that follows different strategies with advantages and also with disadvantages. In this subsection I analyze the architectures of five storage systems, which I consider representative to cover several highly important topics, such as: low level organization, overall system design or sharing semantics: **AFS** (Andrew File System), **GPFS** (General Parallel File System), **GFS** (Google File System), **Lustre** and **Ceph.** The analysis concludes with a comparison of the characteristics of these systems, highlighting their advantages and disadvantages **[PIST, 2014/1]**.

2.6.1 Andrew File System

Andrew File System (AFS) [13] [14] is one of the pioneers in storage distribution field. The main purpose was to globally share files in a transparent way (e.g. location transparency, access transparency, replication transparency and so on – listed in section 2.2, 4). It has client-server architecture and shares files through NFS protocol and therefore it can be considered a Networked Attached Storage.

In terms of architecture, AFS offers a single global file namespace ("/afs") and organizes data in cells and volumes. Volumes are units smaller than traditional partitions (for example can be a directory such as "/home") and reside on *vice partition* managed by a server (Figure 10). A cell comprises a collection of volumes and represents an entire organization (e.g. "/afs/cs.pub.ro"). The distribution is achieved at volume level – clones. There are two types of clones: read-only copies (for workload balance) and backups (Figure 10).

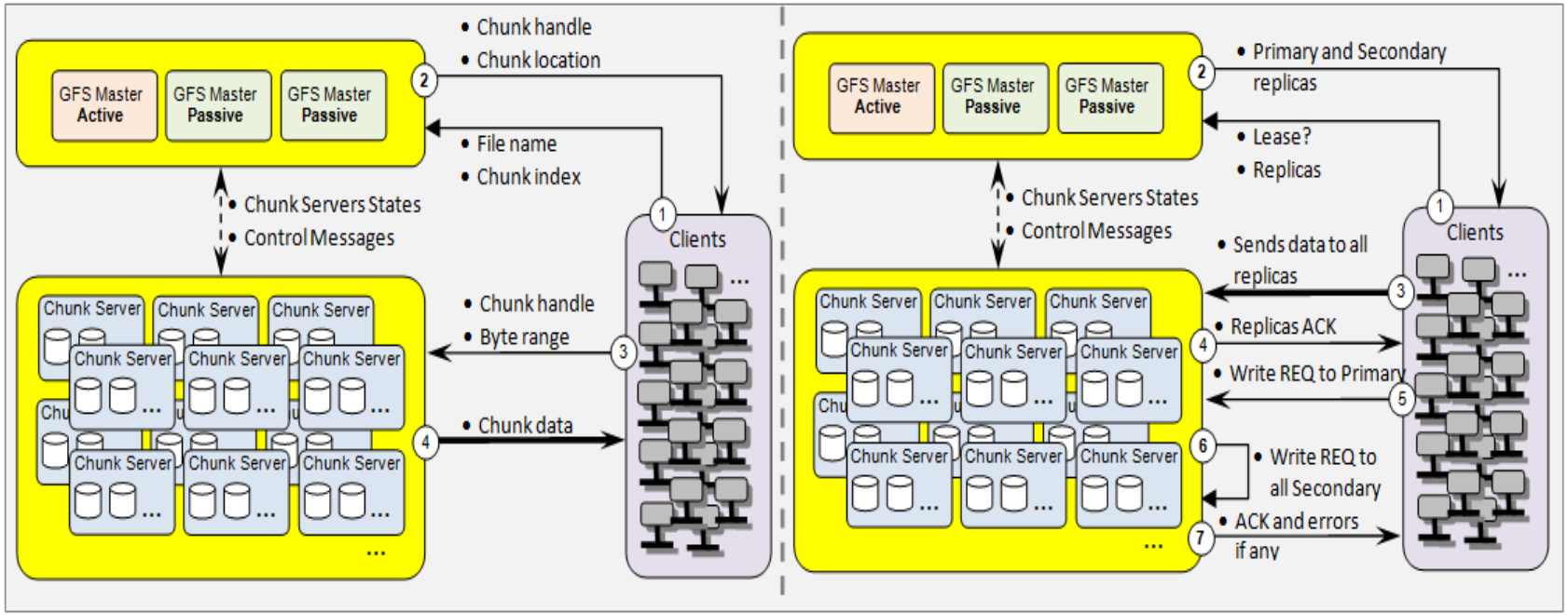
In terms of software design, there is a series of processes that runs on servers for system monitoring and clients interactions: maintains volume location, serves files, monitors volume availability, removes damaged volumes, handle client requests, different tasks related to volumes (e.g. balance data, backups or clones), finds and corrects volume inconsistencies.

2.6.2 Google File System

Google File System (GFS) [15] was built to manage and manipulate huge amount of data capitalizing strength of expensive and reliable servers while compensating cheap hardware weakness. It's not an open source system but serves as model for open source systems like **Hadoop** (HDFS) [16]. From architecture perspective, GFS is composed by one active GFS master server and a few clones (for enabling failover scenarios) to manage the system metadata and hundreds of GFS chunk servers to store chunks of files (i.e. data segments) in a distributed manner. The system implements a series of mechanisms to support a huge number of clients that can access a consistent and shared file system in parallel, also a series of mechanisms to overcome faulty characteristics of commodity hardware (read/write operations are depicted in Figure 11 – a, b).

GFS's main characteristics:

- Files are divided in big data chunks for I/O performance optimizations and distributed across a storage cluster using cheap commodity hardware;
- Uses only one active server for metadata manipulations (atomic metadata mutations) to simplify the system design;
- By profiling, it was observed that files are scarcely modified and appends are more often, therefore the system guarantees a more simplified POSIX sharing semantic model (Table 3);
- Rebalances the data in the cluster periodically;
- Provides logging and checkpointing for chunkservers fault-tolerance;
- Provides snapshotting to create branch copies.

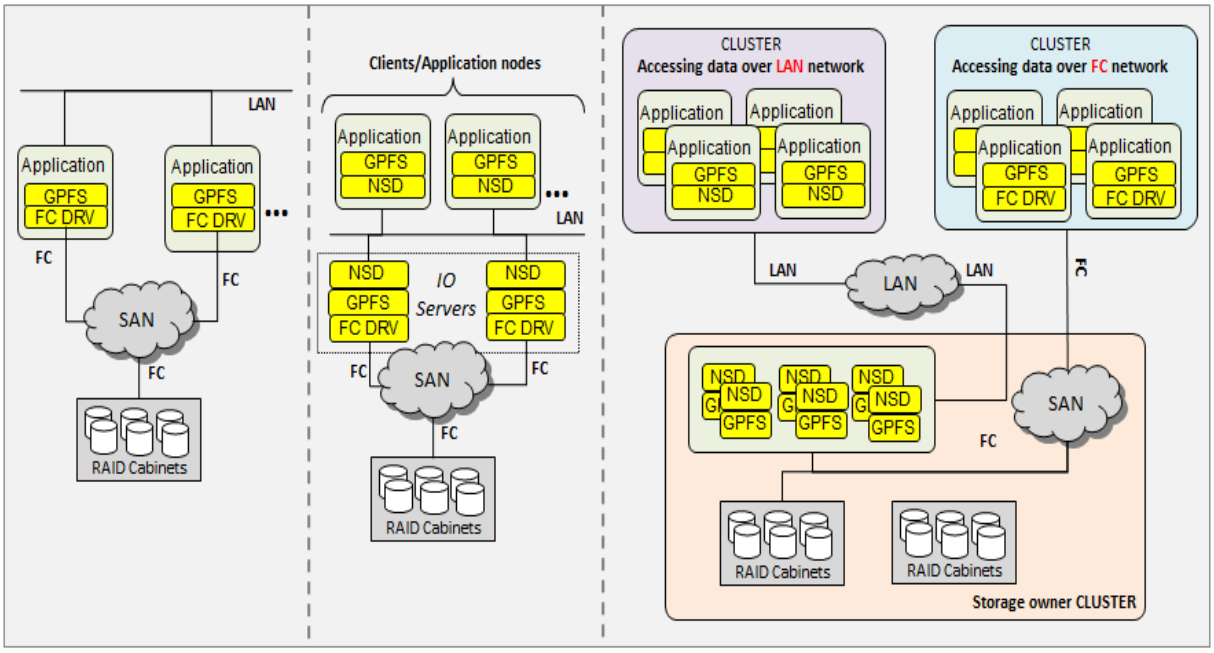


(a). Read flow (b). Write flow

Figure 11: Google File System architecture

Actions	Writes	Appends
Serial success	Defined	Defined intercalated with inconsistent
Concurrent success	Consistent and undefined	
Failures	Inconsistent	

Table 3: Google File System semantics



(a). GPFS Share Disk (b). GPFS Networked I/O (c). GPFS Multi-cluster

Figure 12: General Parallel File System architecture

Hadoop (HDFS) is the open source implementation (by Yahoo) that follows GFS architectural characteristics (since GFS implementation is not open). In HDFS nomenclature the Name Node is equivalent with GFS Master, while Data Nodes are equivalent with GFS Chunk Server.

2.6.3 General Parallel File System

General Parallel File System (GPFS) [17] is the distributed and parallel file system solution from IBM. As GFS, GPFS is closed and there are many unknown mysteries about its architecture. GPFS supports RAID mass-storage systems over Fibre Channel (SAN), but it has also the ability to use LAN networks through Network Shared Disk (NSD) modules which enables block access to data. The choice between these two access modes depends of the system purpose: accessing data through a SAN is much faster, but requires more expensive network equipment while LAN access is less expensive, but slower.

GPFS supports deployments of thousands of storage devices achieving a very good system availability degree. It supports replication, logging and recovery, thus is a fault tolerant system as well. GPFS distributes data by means of two types of policies: *file placement policy* – distributes data to a specific set of disks (disks pool) and *file management policy* – moves or replicates data across system disks preserving files namespace.

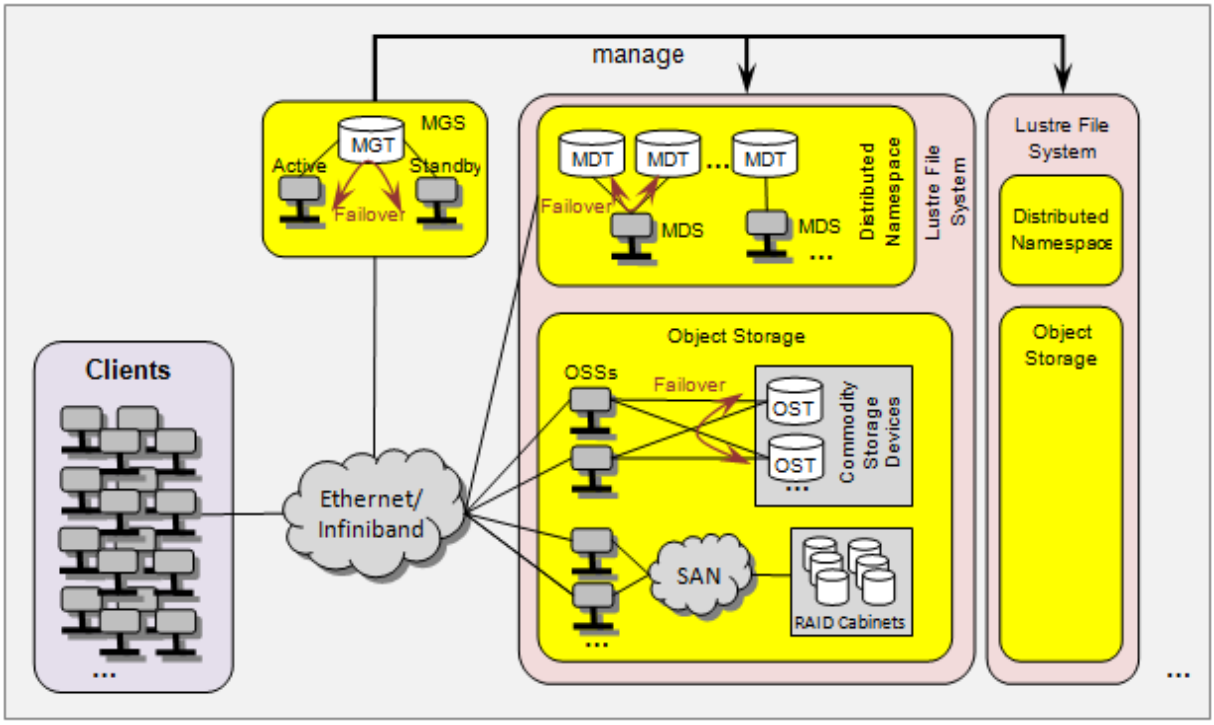


Figure 13: Lustre File System architecture

The files namespaces can be split into small groups by means of *file sets* – achieving management at a smaller granularity. Another GPFS characteristic is that the metadata management is spread across all GPFS clients, thus a single point of failure is avoided and a better availability is achieved. It is also highly configurable, supporting three deployment types: share-disk, networked I/O and multi-cluster (Figure 12).

2.6.4 Lustre File System

Lustre [18] is an open source system that has several unique particularities. Briefly, it is composed by three main entities: metadata servers (for namespaces management), management servers (used to manage all Lustre file systems) and object storage servers (used to manage objects stored on targets), Figure 13. The fault tolerance is achieved by failover configurations (i.e. redundant systems). Each Object Storage Server (OSS) can have a failover configuration achieved by using several targets (Object Store Target – OST). A similar method is used for Metadata Servers (MDS) and Management Servers (MGS). It also distributes namespaces over a cluster of MDSs and MDTs: *Distributed Namespace* (DNE) cluster, former *Clustered Metadata* (CMD) – directories are striped across more MDTs.

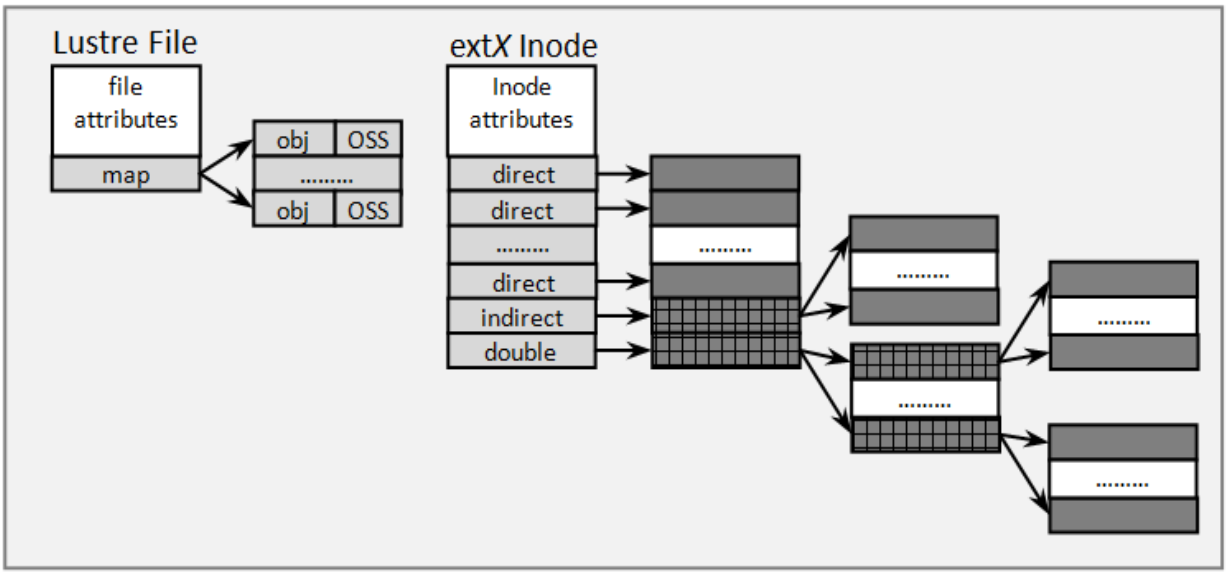


Figure 14: Lustre file mapping

The objects are managed by keeping a map similar to the inode map (as UFS does) and this can be considered a drawback since it puts pressure on Storage Servers (Figure 14). It supports configurations using commodity storage devices, but also SAN based storage.

2.6.5 Ceph File System

Ceph [19] is a new distributed storage system that assembles many advantages from previously existing implementations. It distributes everything: namespaces, monitoring, security and data, therefore it is very scalable, but very complex being considered "*the new dream distributed file system*" (Figure 15).

One of the main characteristics of Ceph is that it completely isolates metadata management from data storage (RADOS – Reliable, Autonomic Distributed Object Store) [20]. Ceph stripes files into objects similar with RAID model, to improve parallel I/O and furthermore RADOS distribute these objects over a cluster of OSDs using *controlled hash function* (CRUSH – Controlled Replication Under Scalable Hashing) [21].

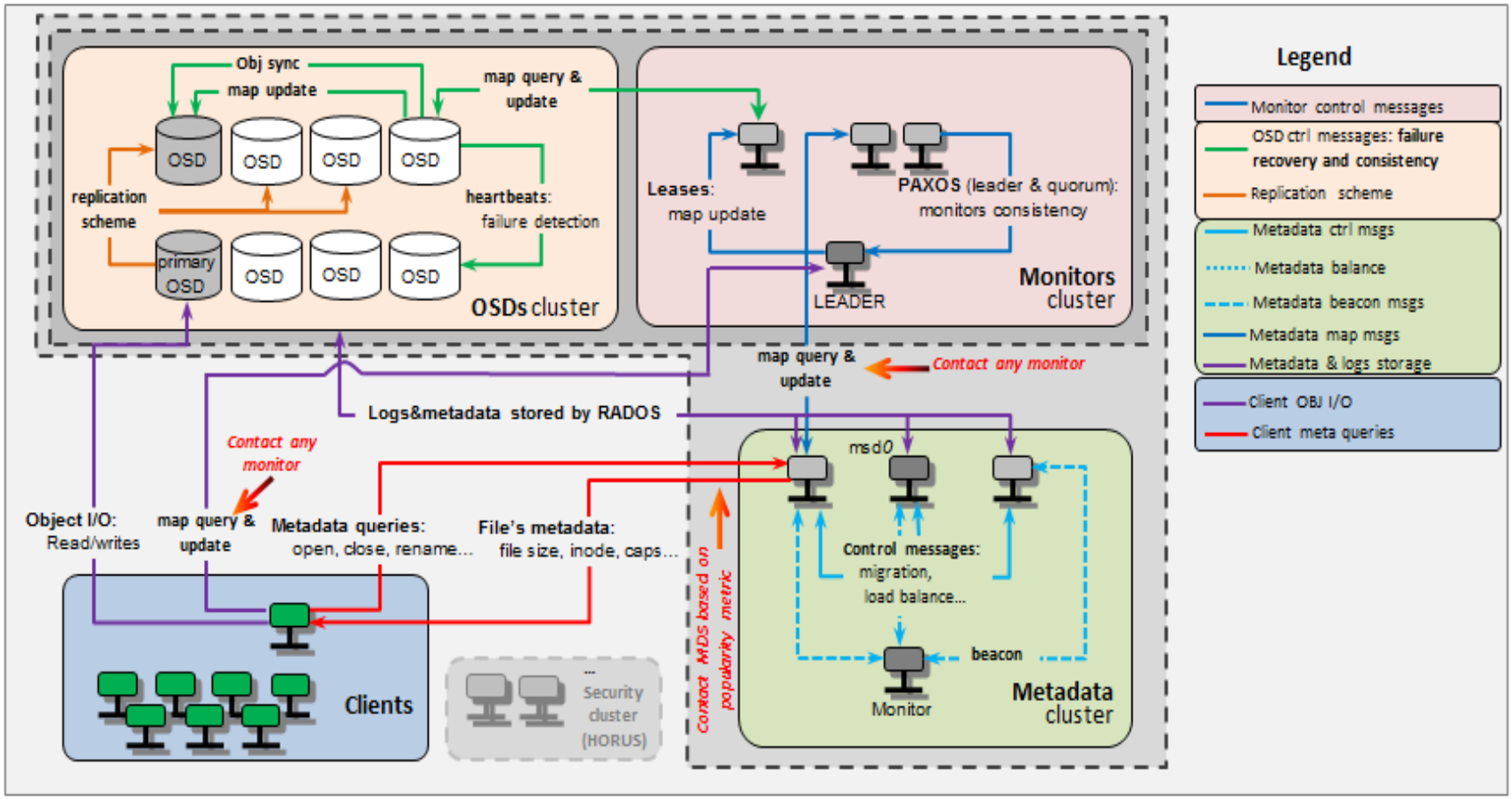


Figure 15: Ceph file system architecture

#	RADOS messages	Description
1	**Leader monitor election**	• Provides consistent cluster-map to all system parties, • Monitors leader election based on simplified PAXOS and quorum for serialization of map updates [22] pp. 127.
2	**Leases**	• Grants short-term leases to active monitors to give them the rights to distribute copies of cluster map to all system parties, [22] pp. 127.
3	**Cluster-map distribution**	• Responses of active monitors (which have granted a lease) to map queries from MSDs, clients or new added OSDs. Since an OSD cluster may have a very large number of devices [22] pp. 129, the monitors don't broadcast map updates, but actually the map updates are exchanged between OSDs from the same placement group.
4	**Heartbeat messages**	• To prevent accessing inconsistent data, timely heartbeat messages are exchanged between OSDs [22] pp. 129.
5	**I/O traffic**	• Objects read/write and data replication (i.e. primary-copy, chain and splay) [22] pp. 131.
6	**Recovery**	• Failure recovery is based on cluster-map epoch and the set of active OSDs in each placement group and it uses a peering algorithm to assure a consistent view of the placement group of all OSDs within [22] pp. 137.

Table 4: RADOS main messages

2.6.5.1 Data storage: RADOS

RADOS is a system composed of a large OSD cluster and a small cluster of monitors used as supervisors. After the files are striped into objects (using RAID based model), RADOS maps these objects to *placement groups* (PGs). Each placement group has a level of replication attached, so it controls the degree of replication declustering. Also, the system has to track a huge number of objects which is more realistic to be done per group, rather than per object. So, based on hashing unique object ids, placement group ids are obtained.

The OSD cluster layout is described by a *cluster map*, containing: map revision for detecting unsynchronized elements in the cluster, the number of placement groups, the status of each OSD in the cluster and an hierarchized description of OSD (e.g. rooms, rows, racks, shelves and devices). By applying sets of *placement policies* on this map, different failure domains can be configured. Furthermore, a deterministic list of OSD is obtained using CRUSH algorithm based on placement group id and a set of placement rules. Only synchronized OSD are used by filtering this list by status of each OSD from cluster map.

The consistency between replicas is assured using a simplified PAXOS algorithm (quorum based election of a leader monitor – it requires odd number of monitors). First, proposes and elects a leader monitor to serialize map updates and manage consistency (Table 4 – 1) then, the monitor leader requests the map epoch from all monitors (while they have a specific time frame to respond) and joins to quorum. The monitor leader must ensure that the majority of monitors have the most recent map epoch. Furthermore, the monitor leader distributes short-term leases to active monitors enabling them to publish copies of cluster map to requesters (Table 4 – 2, 3). If acknowledgements are not received or leases are not renewed, then a new election of a monitor leader is called.

OSD failures detection (based on heartbeat), map synchronization (based on map epoch) and data consistency (based on peering) is achieved by exchanging timely messages between OSD in the same PG (Table 4 – 4). For data recovery, each OSD maintains a history of the placement groups to which it belongs and in case of inconsistency the data is updated through a primary OSD chosen in the replication scheme (Table 4 – 6).

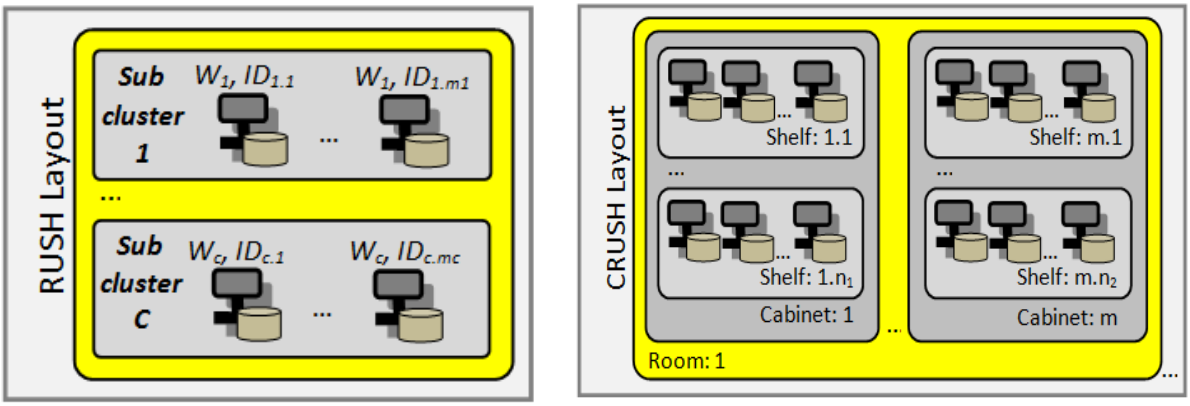
2.6.5.2 Namespace management

The namespace management is distributed across a metadata cluster, which uses a technique named *adaptive workload distribution* (i.e. dynamic sub-tree partition to achieve a scalable performance). Ceph migrates or replicates pieces of files tree at directory fragment level, rather than at the entire directory, thus achieving better performances. The replication and migration decisions are taken based on popularity metric and therefore metadata high availability is achieved.

Despite its advantages, there is an important drawback of distributing namespace management that is given by the cost of management when the number of nodes is very large.

2.6.5.3 Decentralized data distribution

The main purpose of decentralized data distribution algorithms is to replace the lookup servers. In this subsection I briefly present two such algorithms: Replication Under Scalable Hashing (**RUSH**) [23] and Controlled Replication Under Scalable Hashing (**CRUSH)** [21].



(1) RUSH layout example (2) CRUSH layout example

Figure 16: Decentralized data distribution (layout)

RUSH algorithm has several flavors for data placement, such as: using prime numbers, support for removal and a tree-based approach [24]. It has two main principles: identifying which object must be moved to maintain a balanced cluster and decide upon object destination based on hash functions. Also, it implies that devices are replaced in groups (e.g. per shelf basis), rather than one-by-one.

- **RUSH (Figure 16 – a)**
  Briefly, the system is divided into *c* sub-clusters, where each sub-cluster *i* has m_i servers, and each server has W_i weight. The sub-clusters in a system are virtually arranged from the newest added to the oldest. The weights controls which sub-cluster is chosen and how many replicas are stored based on a hyper-geometric distribution (i.e. how many devices are chosen from each sub-cluster).
  The algorithm returns a list of *L* servers based on a pseudo-random hash function, the *x* object, *R* number of replicas of the object and the cluster description (m_i number of server from each sub-cluster with their W_i weights and *C* number of sub-clusters in the system). The algorithm's output is deterministic and consistent if system storage size is changed
- **CRUSH (Figure 16 – b)**
  The system is described by a hierarchical *cluster map* of weighted devices and buckets (e.g. the system is composed by *rooms* with *m cabinets* and each cabinet has n_i number of *shelves* with p_j *storage devices*). Based on *placement rules* CRUSH decide which storage devices are chose from the cluster map.
  The algorithm produces a deterministic ordered list of *L* storage devices for object *x* based on *pseudo-random hash function*, *cluster map* and *placement rules*.

Storage System	Locality	Sharing	Distribution Level	Semantics
AFS	Networked	Shared	File	Session
GFS			Chunk/segment	Mimic POSIX
GPFS			Block	POSIX
Lustre			Object	
Ceph				

Table 5: Storage systems characteristics mapping

Furthermore, **CRUSH** [21] completes with the idea of **controlled distribution** per different failure domains by means of placement policies.

2.6.5.4 Security in Ceph

Maat [25] is a security protocol that uses *Merkle trees* to define **authorized users for a group of files** and proposes few novel techniques to improve performance of petascale file systems and high-performance parallel computing: extended capabilities, automatic revocation and secure delegation.

Horus [26] is a proposed method of **data encryption** that introduces a novel idea to generate different regions based on encryption keys using *Merkle trees*, named **Keyed Hashed Tree** (KHT). KHT allows generation of keys for different ranges of blocks from a file shared by different clients, each with its own permissions over different regions from it. The main advantage of this method is that it offers multiple levels of security, but the KHT height and the region size at each KHT level must be pre-configured – a scalability penalty.

2.6.6 Comparison

The classification of the studied systems based on my proposed taxonomy is listed in Table 5, while below are commented the advantages and disadvantages emerged from their comparison:

- Location of data in the distribution space can be handled in two ways: by using information maps (similar with inode map from UFS) or by decentralized distribution algorithms such as CRUSH or RUSH. The drawback of mapping against decentralized distribution algorithms is the increased pressure on some dedicated lookup servers.
- Distributing metadata management has pros and cons: in essence, a distributed management comes with a better I/O performance, but a more complex architecture. Regarding the above systems, Ceph and GPFS handle metadata in a distributed way, while GFS/HDFS use only shadow servers for high availability, but the actual activities are handled only by one active server.

- The distribution level refers to the data unit used for distribution, such as: data segments, files or objects. The distribution level for each file system is listed in Table 5. There is a global tendency towards OSD usage, due to several strength, including: better data security (per object), flexible size (therefore it can be adjusted to increase I/O performance), standard API and a local space management. In terms of studied systems, Lustre and Ceph uses OSDs and the future leads to OSDs for AFS (OpenAFS) and GPFS as well.
- GPFS is sensitive to devices failures, because usually it uses a declustered RAID approach. In terms of RAIDS, there is also a trend to use Object RAID [27] where PanFS is leading.
- In distributed storages is a trend to add support for commodity hardware, where failures are a common behavior rather than an exception. To overcome the occurring issues, the systems must implement different mechanisms to achieve a specific degree of fault tolerance and high availability.
- The access interface is an important characteristic of storage systems, therefore the distributed storages may be divided in two main categories: systems that have support for conventional API (i.e. Berkeley socket) or using a custom API. A conventional API has the advantage of transparency, but a custom API has the fine tune functionality to increase performance. Ceph mixes the two methods and uses an enriched conventional API with additional features for manipulating different system characteristics.
- Systems that uses pieces of a files to distribute data is less sensitive to failures and have a better degree of high availability compared to AFS, for example, which distributes entire files (volume cloning).
- The file sharing semantics is handled as follows:
  - AFS: Session Semantics (weak);
  - GPFS, Lustre and Ceph: POSIX Semantics;
  - GFS/HDFS: mimic POSIX Semantics (more relaxed semantics).

2.7 Network technologies for distributed storages

The network infrastructure has a great influence in the performance of distributed storage systems **[PIST, 2014/1]** and is divided in two main topics:

- Network topologies and
- Protocol stacks

2.7.1 The topology

The topologies used in distributed systems are defined by graphs used to create paths between storage devices and edges [28]. There are intrinsic similarities between graphs used to connect general purpose cores in multi-core systems and distributed systems in general, since

they aim relatively same characteristics: a perfect interconnection topology should have **smallest latency** possible (round trip time close to zero), **maximized throughput** (line rate using any packet size), **minimum capital cost** and a **perfect fault tolerance** and **high availability** [29]. In data centers (and distributed storage systems) graph diameter influences latency and contention (characteristic more relevant for SAN based systems). Also, a smaller node degree decreases the capital cost of the network's equipment, but it increases the latency since the topology will have more hops, a smaller diameter lowers the latency and round trip time and increases the throughput, a smaller bisection width facilitates the left-to-right traffic (important for data migration and data distribution among storage nodes) and a high number of edges influences the deployment.

Usually, in data centers are used different tree based topologies with alternatives paths, but there are other alternatives with characteristics suitable for different traffic profiles and applications purposes, such as:

1. **Complete graph:** undirected graph where each pair of nodes is connected by one edge (Figure 17 – 1).
2. **2D Torus (k):** undirected graph in form of matrix of nodes, where each node connects with its nearest neighbors plus with wraparound connections (Figure 17 – 2).
3. **Fat-tree (h):** tree where the edges become fatter as one moves towards the root (Figure 17 -3). In industry usually it is used a fat-tree with redundancy to improve system high-availability (Figure 17 – 3').
4. **Hypertree (k, h):** type of 3D tree of height h and degree k. It can be viewed as a complete binary tree of height h from bottom-up and a complete k-ary tree (k≥2) view from top-down (Figure 17 – 4).
5. **Hypercube (k):** graph with 2^k nodes and a total of $k \cdot 2^{k-1}$ edges where each node has exactly *k* neighbors and with mutually perpendicular sides (Figure 17 – 5).
6. **Cube-connected-cycles (k):** a graph obtained by replacing each node of a Hypercube of degree d by a cycle of length k, where k ≥ d (if k = d, then CCC (d, k) →CCC (k), Figure 17 – 6).

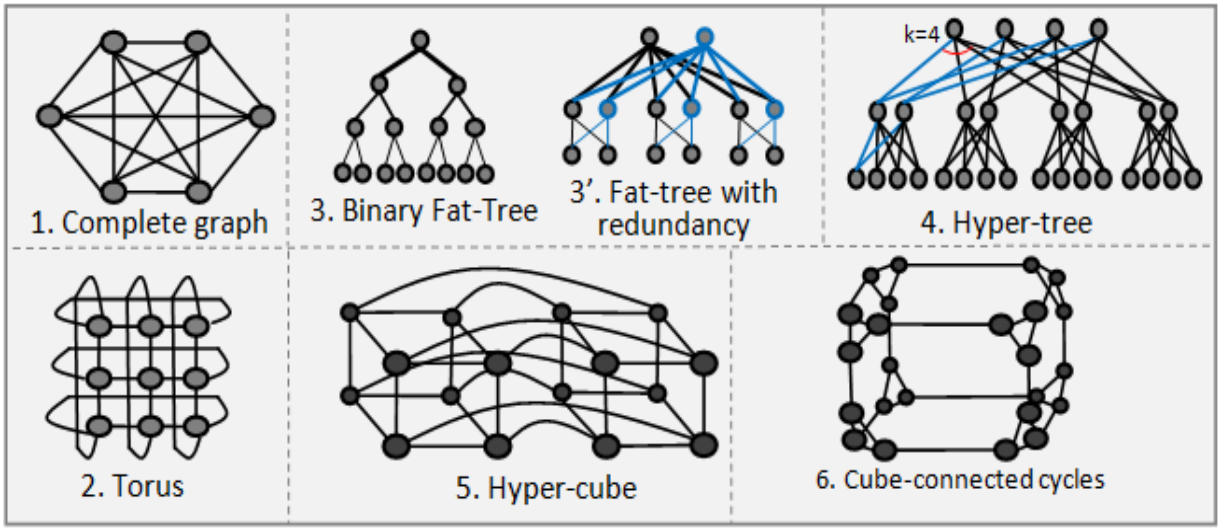
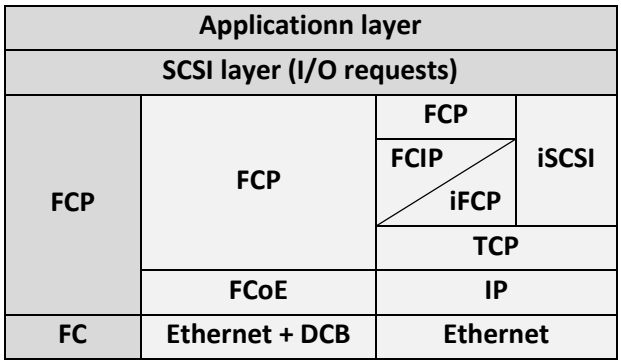


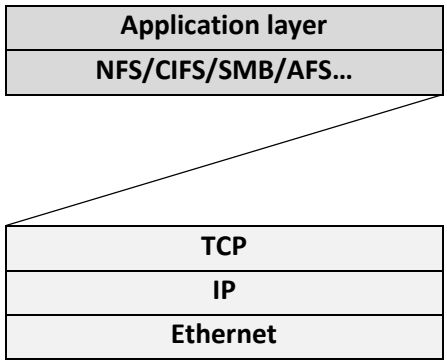
Figure 17: Network topologies

#	Topology	Nodes	Edges	Degree	Diameter	Bisection
1	**Complete graph**	N	$E(n) = \frac{n \cdot (n-1)}{2}$	$\Delta(n) = n - 1$	$d(n) = 1$	$bw_e(n) = \sqrt{\frac{n}{2}}$
2	**2D Torus (k),** k nodes line	$N(k) = k^2$	$E(k) = 2k^2$	$\Delta(k) = 4$	$d(k) = k$	$bw_e(k) = 2k$
3	**Binary fat-tree (h)**	$N(h) = 2^{h+1} - 1$	$E(h) = 2^{h+1} - 2$	$\Delta(h) = 2 + 1$	$d(n) = 2h, log_2(n)$	$bw_e(n) = 1$
4	**Hypertree (k, h),** k > 2	$N(k,h) = \frac{k^{h+1} - 2^{h+1}}{k-2}$	$E(k,h) = \frac{2k^{h+1} - k2^{h+1}}{k-2}$	$\Delta(k,h) = k + 1$	$d(k,h) = 2h$	$bw_e(4,h) = 2^{h+1}$
5	**Hypercube (k),** K > 0	$N(k) = 2^k$	$E(k) = k2^{k-1}$	$\Delta(k) = 2^k$	$d(k) = k$	$bw_e(k) = 2^{k-1}$
6	**CCC(k),** k > 3	$N(k) = k \cdot 2^k$	$E(k) = k2^{k-1} + k2^k$	$\Delta(k) = 3$	$d(k) = 2k + \lceil \frac{k}{2} \rceil - 2$	$bw_e(k) = 2^{k-1}$

Table 6: Network topologies scaling properties



(a). Protocol stack for block level "networked" storage systems



(b). Protocol stack for file level "networked" storage systems

Table 7: Protocol stacks for storages

Analysis:

- The complete-graph has the perfect **diameter** (value of 1), hypercube, cube-connected-cycles, binary-fat-trees and hypertrees has a logarithmic diameter (it is worth to mention that hypertrees has the advantage of using constant speed for links at all levels, while fat-trees must provide *fatter* links toward the tree root).
- The **node degree** is constant for torus, binary-fat-trees and hypertrees. Nevertheless, hypertrees have the ability to scale also horizontally changing the k-ary dimension. The node degree of the hypercube grows faster by logarithmic function, regarding the number of nodes in the topology.
- In terms of **fault-tolerance**, except for fat-trees, all other topologies offer alternative routing paths. Hypertrees preserve diameter in case of faulty nodes, a very important advantage. Cube style topologies have a better fault-tolerance, because they have many alternative routes between any two nodes.

De Brujin, a directed graph with m^n nodes labeled by *n* tuples over *m* alphabet, with $N(m,n) = m^n, E(m) = m^2, \Delta(\mathrm{m}) = 2\mathrm{m}\,, d(n) = n$ has also been taken into consideration. It has large number of nodes, few connections per node and preserves short distance.

When designing a topology the graph is chosen depending on traffic requirements: east-west/south-north traffic profile, low latency or high throughput and of course the balance between costs and performance.

2.7.2 Protocol stack

In terms of protocol stack used, distributed storage systems are divided into systems that require lossless mediums and systems that can use best effort mediums. Usually, storage

devices are controlled using SCSI (Small Computer System Interface) standards (i.e. commands, protocols and interfaces). Distributed storages built at block device level, caries SCSI commands through interconnection medium – SAN based systems. Since SCSI does not have mechanisms for contention or retransmission, lossless transmission environments are more appropriate. Regarding OSI model, the reliability problem can be solved at network layer 2 (data link) or layer 4 (transport). Ethernet is the most common data link protocol used, but is a best effort by design, while for example Fibre channel is a lossless medium. Basically, fibre channel protocol (FCP) runs over fibre channel infrastructure, but there is a solution where FCP frames are embedded into Ethernet frames: Fibre Channel over Ethernet (Table 7 – a).

At layer 4, reliability can be handled by TCP, but with a noticeable performance penalty degree (Table 7 – b):

- iFCP – protocol that runs over IP and provide functionalities similar with FCP
- iSCSI – protocol that runs over IP and provides functionalities similar with SCSI
- FCIP – FCP packets are encapsulated into IP

Ethernet has been enriched with a group of protocols named DCB (Data Center Bridging) by IEEE group [42] for traffic control to avoid packet dropping and therefore Ethernet has become a lossless environment and an alternative to Fibre Channel (Table 7 – a). Also, there is a trend to merge general purposes networks (LANs) with storage networks – network convergence – where Ethernet become the solution (section 5).

Distributed storages built at file level are less demanding and usually runs over TCP/IP/Ethernet stack (classic LAN) using protocols such as: NFS, CIFS or SMB (Table 7 – b).

2.7.3 Real world considerations

The capital cost of storage system infrastructure is one of the most important criteria and therefore the tradeoff between an appealing price and a good performance must be established when choosing an interconnection model. The switches are expensive and determine the way that network scales by the number of ports per device and speed. The speed of switch ports progressed in the last decade from 1GB/s to 100GB/s and nowadays an L2/3 switch has a 10GB/s port at about 1250$, 40GB/s at about 3000$ and a 100GB/s is still very expensive, about 12 000$ – so, faster links increases expenses.

Other than capital cost, the choice depends also on the data center performance requirements, such as latency and throughput. In this case the switches ports are chosen based on two criteria: to optimize the transfer speed of each node and to support aggregate speed of a number of nodes in the data center.

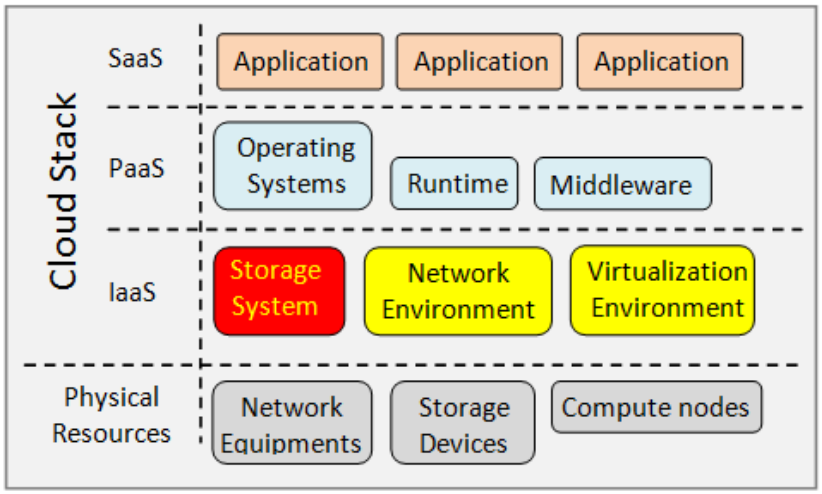


Figure 18: Cloud components

In SAN environments, for a data segment of 512KB transferred at 25MBps, a 1Gbps port supports up to 2-3 RAID devices and a 10GB/s port supports up to 25 RAID devices [30].

A fat-tree topology is expensive because of the high bandwidth needed while moving up in the tree. Nevertheless, there are methods to solve this issue, one of them being link aggregation (LAG). A butterfly topology requires fast links because the topology must accommodate the entire traffic and it does not have the fault tolerance property like the other topologies.

Mashes and torii are not very expensive, but they needs extra connections to storage devices and usually are fit for storage systems built from blocks (e.g. as the ones proposed by IBM and HP). Hypercubes are a special case of torii, but the bandwidth scales better [30].

There are other graphs that can be considered to build a topologies designed for data centers, such as De Brujin which requires a low number of switches but the capital costs put pressure on the NICs.

Also, there is a trend to combine different topologies to lower the costs and to increase performance in different areas. For example, in Ceph context the throughput capabilities has to be higher for RADOS, while for Monitors and Metadata servers the latency is more important.

2.8 Interfacing with storage systems (case study: clouds)

Basically, distributed storages are used as infrastructure for larger and more complex systems such as **Clouds**. In case of clouds, the infrastructure is composed by three main components: storages, networks and virtualization environments (Figure 18).

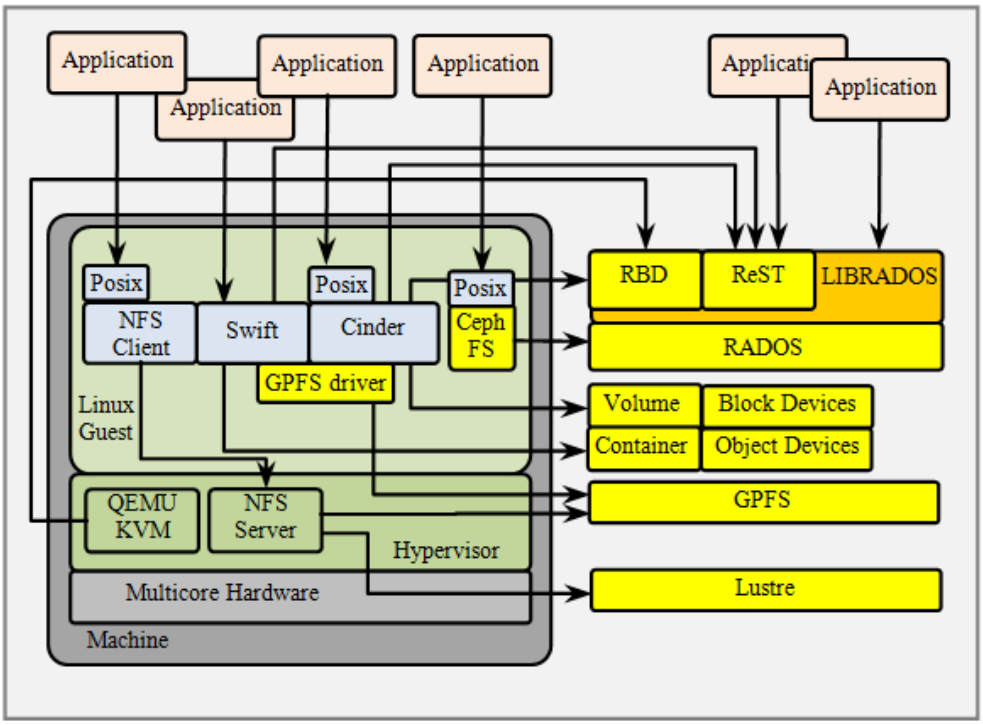


Figure 19: Open Stack storage integration

In general, storage systems have three main interfaces:

- *via hypervisor* support (exposing a block device to Linux guest);
- *via Linux guest driver* support (e.g. Ceph's Linux driver);
- *applications direct access* via HTTP, ReST or other standard or non-standard API.

Open Stack, one of the most successful cloud systems, has two main modules that provide storage to guest virtual machines and applications: **Swift** (object storage) and **Cinder** (block storage). Swift is a complicated component because it has many features including: high availability, high durability and high degree of concurrent access, while Cinder offers only block level access via iSCSI protocol to block devices.

Other than these two, **Ceph** can be easily incorporated into Open Stack due to its multiple APIs: RBD (block access), ReST, librados , Ceph-FS (using FUSE or Ceph Linux driver) and it has also support into QEMU (Quick EMUlator) and KVM (Kernel-based Virtual Machine) and offers block device level access to Linux guests (Figure 19). **GPFS** can be used through GPFS Linux driver, Lustre has a lack of support, but it can be accessed as NFS through NFS server – hypervisor support. In the future, Hadoop will be incorporated into Open Stack for its Big Data support.

CloudStack access methods are not very different from Open Stack. It has two types of storages: **primary** and **secondary**. Primary storage is used to expose a mount point to guest Linux based on hypervisor (KVM or XEN) via iSCSI, Fiber Channel or NFS: RADOS can be used as primary storage through RBD (RADOS Block Device) interface, Lustre via NFS server, etc. Secondary storage is used to store templates, OS images and snapshots. Secondary storage can use Swift (object based storage) from Open Stack with all its interfaces or NFS based file server.

In essence, most cloud systems (e.g. **Eucaliptus** or **OpenNebula**) use **SAN** or/and **NFS** technologies to access a shared storage. Where, **SAN-FS** provides block storage to VMs, while **NFSs** is used as *secondary storage* for applications databases, backups etc. It is worth to mention that there is also a trend to integrate all data/storage tiers.

3 CASE STUDY: INFRASTRUCTURE FOR ELEARNING ENVIRONMENTS

3.1 Motivation

Usually, high education is expensive and therefore eLearning platforms provide a way of affordable education with an effective cost of investments. In the last decade, cloud computing research and adoption increased greatly due to its many advantages including economic benefits, ease of management, power saving and so on. In essence, clouds provides the means to organize and deliver a wide variety of software services including **eLearning environments**. Along with cloud computing, file systems solutions were improved to meet requirements imposed by the distribution characteristics of clouds. The distribution level of file systems, data management, data seek methods, network characteristics and many other features of distributed file systems influences the performance of eLearning environments. Therefore, I propose a high level architecture of cloud platforms adequate for eLearning software by outlining several advantages that overcome issues related to distributed eLearning using Ceph as a data storage environment and several domestic network topologies **[PIST, 2014/2]**.

Why Clouds and eLearning?

eLearnig environments have several issues and some of them may be better handled by clouds systems. One of the main issues is the infrastructure. Usually, eLearning infrastructure requires huge investments, thus clouds being by definition an infrastructure provider may be used as lay foundation for eLearning applications. Clouds scales dynamically (by demand), and offer a collaborative environment as well [56] – an important feature for eLearning services. Basically, eLearning environments have huge databases of learning objects that can be stored in cloud's storage systems. Therefore, the way that data is handled in cloud influences different aspects of eLearning mechanism including searching and content delivery of learning objects. There is a wide variety of storage systems implementations for clouds, encompassing many advantages. Other than the storage system, the way that everything is linked together influences eLearning content delivering performance. There are many network topologies suitable for clouds, including fat-tree, hyper-tree, cube and hyper-cube. Researchers are still looking for better ways to link everything together (section 2.7). I took into consideration some characteristics that influence I/O performance: scalability, latency and capital costs.

3.2 Clouds storage systems and enhanced networks for eLearning environments

Clouds aims to meet several characteristics that impose a set of main requirements to storage systems, including: sharing, scalability, transparency, high availability, fault tolerance,

concurrent and consistent data access and security (section 2.2). The design of storage systems may influence the delivery, maintenance and management of learning objects, while network architecture influences the system I/O performance.

3.2.1 Data distribution for eLearning environments

eLearning environments usually have tremendous database of learning objects, which are organized per category and therefore a hierarchical organization won't help – an alternative could be semantic aware storages. There are researchers that support the idea that the hierarchical file systems will be replaced by semantic file systems [57], because other than categorization issues they also improve searching capabilities [58].

I propose a parallel and distributed file system architecture for a cloud storage system based on two layers: first layer stores learning objects in flat address space (for example RADOS can be used) and the second layer a semantic aware metadata for categorization and searching capabilities.

Another characteristic of eLearning systems is that the learners (users) are geographically spread, using different network paths to access the cloud's storage. To balance traffic load, the system must take into account the learners profile distribution and split the data cluster into distribution domains, where basically each domain is accessed from a specific gateway. The object distribution has to be replicated in each of these domains, thus the data access performance may be improved.

There is an alternative solution based on delegation points, which is a cacheable point of learning objects that in the end are distributed to a group of users. There are few drawbacks for this method including: security delegation, local storage for cached learning objects, lease method and so on. Users may access the learning system from their home using just a web page, so even if the internet provider would have such method in place there won't be any visible gain, the user could still see a big latency. The consequence of this proposal would be that the data has to be replicated on many storage devices – can be considered a good idea since the price per GB storage is usually low.

One more improvement could be to use the network links at optimal capacity and balance the load on the gateways if the network equipment supports Quantized Congestion Notification. There is a genuine method that balances the traffic load between network gateways, which uses a red-black tree to sort the available gateways based on congestion distance [59] or QCN-WFQR (section 6).

3.2.2 Clouds network

A cloud can span across a single or multiple data centers (public or private). A data center does not necessarily work as a cloud, thus specialized data centers have been developed. The design of a cloud's network infrastructure capitalizes on the previous experience acquired with data centers (DCs) and adapts these designs for the new requirements. A data center connects servers to each other and to outside worlds the same way a campus network provides users with the same type of access. The current DC's networks follows three layers architecture: **core** (provides the gateway to the outside world and high speed backplane connections between routers), **aggregation** (integrates many networking services such as: firewall, load balancing, tunnelling, intrusion detection/protection, security and usually this is the layer where l3 is connected to l2 networks) and **access** (usually L2 domain providing the necessary networking to the nodes such as compute, storage and so on). The main drawback is that this architecture limits the allowed topologies to those that support a multi-layered organization or to hybrid topologies.

I propose a solution mostly based on fat-tree topologies with redundancy. Basically, it provides a 1:1 subscription but is limited in height due to the necessity of providing an uplink capacity equal to the sum of all the downlinks. Due to this disadvantage, it is not used in a *pure* form; usually along the way oversubscription increases. Also, this topology is better suited when most of the traffic is north-south but it is not recommended for use in configurations where east-west traffic is predominant due to the fact that distant nodes cannot communicate directly. In an eLearning environment most of the data transfer is south-north, therefore this solution is a good choice.

In order to accommodate an eLearning environment, a network composed of interconnected, physical or virtual nodes needs to provide a rapid elasticity. The most important nodes considered for this proposal considering their elasticity characteristics are:

- **Compute**: provides processing and memory resources for the eLearning environment. Elasticity is done through the use of virtualization both for networking (by the use of virtual switches) and compute (by the use of virtual machines);
- **Storage**: provides access to data. Storage for compute needs is local or NAS and storage systems is distributed for the eLearning objects. Distributed storage systems offers intrinsic elasticity;
- **Switch**: provides L2 packet switching and needs to have support for virtualizing network resources using VLAN, QinQ, VXLAN, IBM DOVE or similar technologies. The management plane needs access through a secure channel to the switches to make dynamic changes;

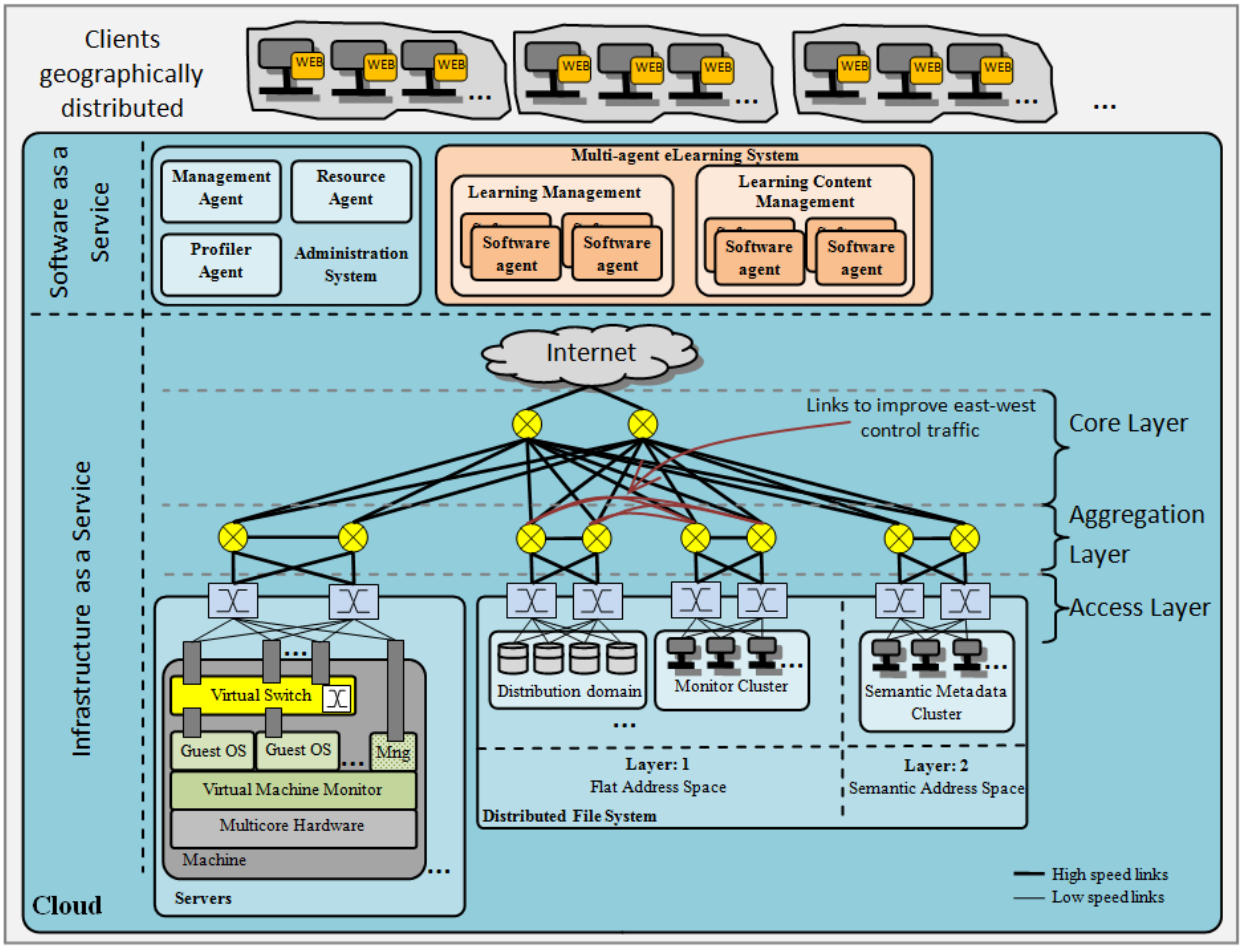


Figure 20: eLearning/ cloud – architecture proposal

- **Router**: provides L3 routing capabilities and high throughput links. Elasticity is not usually provided in current Cloud implementations, but is recommended due to the limitations imposed on the migration of the agents;
- **Intrusion Detection/Intrusion prevention and Firewall**: Uses packet inspection to detect/prevent intrusions. These nodes are parts of the cloud network and provides intrinsic protection for the environment.

3.3 Clouds and eLearning environment: proposed architecture

In the following section I propose a cloud architecture emphasizing different aspects related to storage system and network topology that may influence eLearning environments, from architectural and I/O performance point of view.

The architecture is composed of three main infrastructure components: storage system, servers and network and two main software components: administration system and multi-

agent eLearning system (Figure 20). Administration system manages server's resources (creates, migrate and destroys virtual machines and virtual switches), while eLearning multi-agent system creates, migrates and destroys agents, based on profile statistics.

3.3.1 System administration and eLearning components

The eLearning software agents are running on virtualized machines in the cloud environment. The entire system is managed by three main logic components: **resource agent** (administrates virtual resources based on profile statistics), **management agent** (administrates eLearning agents: creates, destroys and migrates agents) and **profiler agent** (gathers information about resource utilization and provides statistics).

In the absence of an eLearning system reference design, there were identified two main components: **learning management system** (delivering lectures to learners) and **learning content management system** (development and reusability of lectures). Each system is a collection of software agents that can be created, migrated or destroyed based on their status and load.

3.3.2 The storage system

In terms of storage, I propose an alternative to OGSA-GFS [60]: a two layered storage system based on Ceph components (a flat address space and a semantic aware metadata), Figure 20. Basically, Ceph is composed of two main layers: metadata cluster and RADOS. Metadata cluster manages files organization in a hierarchical form, while RADOS is a flat address space that distributes, replicates and load balances objects.

I propose a metadata cluster that implements a semantic aware namespace, instead of hierarchical namespace. There are several methods to implement such file organization: property-based, content-based or context-based [61] and several architecture directions: integrated, augmented and independent (native). I propose a native implementation with content-based semantics, because a hierarchical file organization brings no advantage and the content-based architecture categorizes files and improves searching.

Using Ceph/CRUSH, I propose a data distribution based on domains and therefore I define a placement policy and cluster map to replicate each object three times in each domain for high availability and fault tolerance.

3.3.3 The network profile

The high throughput traffic is mostly read-only and south-north from system to learners, therefore it will be small amount of replicated or rebalanced data (low east-west throughput

demands). Nevertheless, the monitors from RADOS which manages OSDs cluster by keeping storage data synchronized and collect the state of each OSD should have low latency.

A hierarchical topology facilitates south-north traffic along with a new link from monitors group to each OSDs group (Figure 20) to decrease the latency between monitors and OSDs. Also, is worth to mention that if the system grows, aggregation layers may be added, so in a hierarchical topology the east-west latency is growing.

3.3.4 Real world considerations

Storage Systems are built on elastic platforms (e.g. Data Direct Network) including storage devices and network equipment. Each platform has a set of elements with a set of features (files, blocks or objects storage devices, Infiniband connections and so on) pricing the platform.

Performance analysis of storage systems depends on a wide variety of factors, therefore I split them into several groups as follows:

- **Physical storage profile**: disks RPM, data transfer speed, disk cache size and speed, cache type, block size, connection type (e.g. SATA), RAID configuration etc.
- **Network profile**: transfer protocol performance (Fiber Channel, Inifinband or Ethernet), TCP window size, retransmission packets etc.
- **Storage system profile**: replication degree, replication method (e.g. RAID, software replication) etc.
- **Client system profile**: number of CPU cores, cores speed, page size, system load etc.

There are quite a number of storage systems alternatives, such as: Lustre, Gluster, Ceph and GPFS, GFS/Hadoop and so on. I proposed a solution based on Ceph, mainly on RADOS that I intend to use it in a semantic aware system.

Ceph performance has been analysed [11] using IOR (Interleaved Or Random used for testing performance of parallel file system using scripts that emulates behaviour of parallel applications). For OSDs, three layouts (BTRFS, EXT4 and XFS) were tested, where BTRFS had the most balanced results for I/O access. If the number of OSD replicas increases, the throughput performance has a hyperbolic profile, as expected, and also if small chunks of data are used the results were poor, as expected as well. Roughly, RADOS performance was about 70% of native hardware [11].

4 OPTIMIZATIONS OF DISTRIBUTED STORAGE SYSTEMS BASED ON HARDWARE ACCELERATORS

4.1 Motivation

Due to the advent of high speed Ethernet (40Gbps, 100Gbps or even 800Gbps), the hardware systems along with software architectures must adapt to newly increased traffic. The capacity of servers to handle such high traffic volume was exceeded. Thus, increasing port speed and CPU computation power along with leveraging different cache levels were the first technologies used to improve throughput and latency at server side. But, in the recent years the servers failed to meet even higher traffic demands due to many factors, such as networks convergence or high volume of real time traffic (e.g. voice or video) [31]. One improvement would be to increase the degree of parallelism (i.e. increase the number of nodes in clusters and split tasks at finer granularity), but this will lead to an increased complexity, power consumption and maintenance costs. Other improvement would be to increase performance of each node by offloading the CPU cores and moving specific tasks to dedicated hardware engines. Hardware engines have double impact: first, being dedicated modules perform their role faster than software which runs on general purpose cores and second, reduce significantly CPU utilization, thus increase the system performance and free CPU processing time.

Intel investigations revealed several I/O bottlenecks at receiver side divided into three overhead categories: system overhead, TCP/IP processing and memory access [31]. To meet these overheads, Intel developed *Accelerated High-Speed Networking* technology (I/OAT) – a set of features to reduce the receiver side packet processing overhead (e.g. split headers, DMA copy engine and multi-queue usage to receive frames). Intel's solution showed improvements for about 38% in CPU utilization, the number of transactions processed increased by 14% and throughput by 12%, [32].

I followed a similar idea to increase the performance of each node from a cluster by offloading the general purpose cores of packet processing at receiver side **[PIST, 2013]**. Offloading comes with flows classifications and multi-queues distributions and by using Accelerated Receive Flow Steering [33] is leveraged the parallelism of multicore systems. In other words, the hardware engines are instructed to classify and distribute flows considering the SMP characteristic of multicore systems and increase performance of parallel I/O applications running on the same machine. Also, by adding queues weights adjustments I proposed a method of reducing the latency of sensitive flows, thus making the system much responsive.

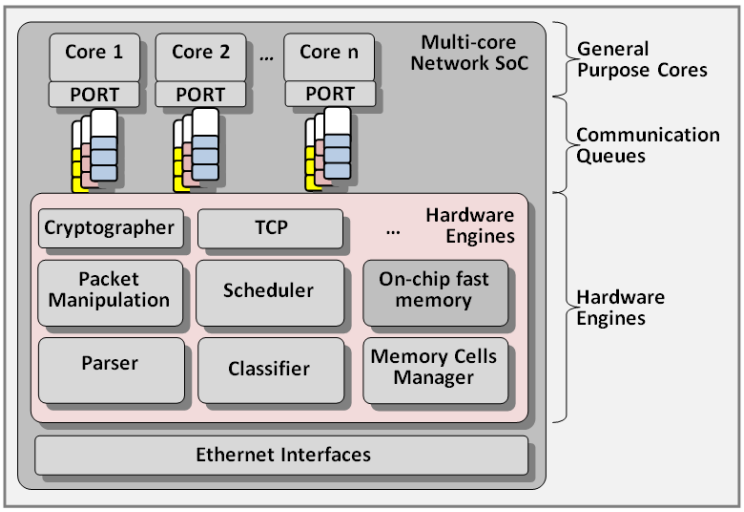


Figure 21: Block diagram of a generic packet processing SoC

4.2 Integrated packet processing engines

In general, any System on Chip (SoC) integrates a set of dedicated hardware modules along with general purpose cores into a single chip. In this thesis, I refer to a packet processing **SoC** as a chip that integrates multiple hardware engines dedicated to network specific tasks, usually handled by general purpose cores. And I refer to term **offload** (or *hardware offload*) to the migration of particular tasks from general purpose cores to dedicated engines freeing processing time.

Figure 21 presents a high-level block diagram of a generic packet processing SoC, emphasizing several important hardware engines used further:

1. **Parser**: dedicated engine for parsing frames, identifying protocols, checking packet integrity and identifying malformed frames;
2. **Classifier**: dedicated engine for classifying and distributing frames to specified flows;
3. **Packet Manipulation**: dedicated engine for frames manipulation (e.g. mangling frames headers to meet different protocols requirements);
4. **Scheduler**: dedicated engine that implements scheduling algorithms to meet different frames handling policies;
5. **Memory Cells Manager**: dedicated engine that handles memory allocations required by hardware modules;
6. **Cryptographer**: dedicated engine that implements different cryptographic algorithms (e.g. AES, CRC and so on);
7. **TCP**: dedicated engine for offloading TCP tasks (e.g. segmentation).

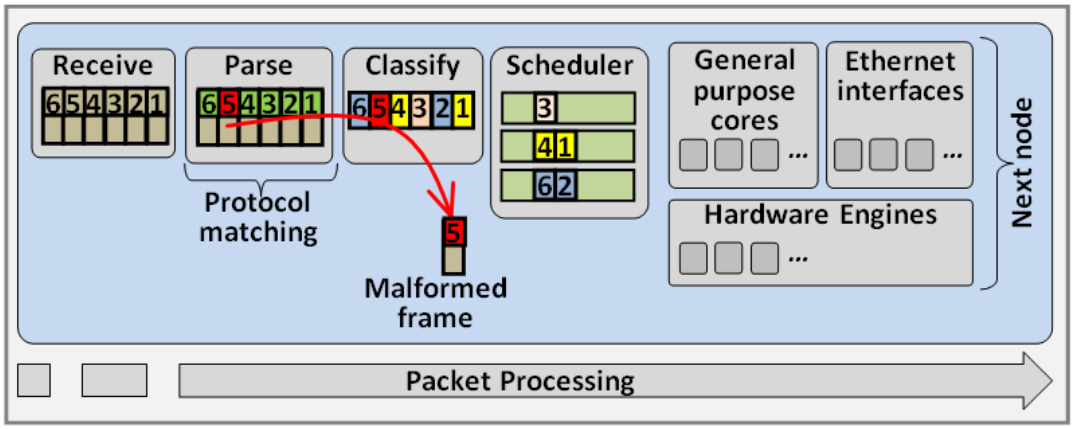


Figure 22: Typical stages for receiving packets

A packet processing SoC may have many programmable flows, therefore each transmitted or received frame is distributed to a specific flow, where each flow can be seen as a path through a graph with hardware engines as nodes and queues as edges. Also, the hardware engines are complicated modules architected to handle almost line rate traffic of small frames (e.g. 64b/frame), thus most of them have multiple processing units to handle tasks in parallel.

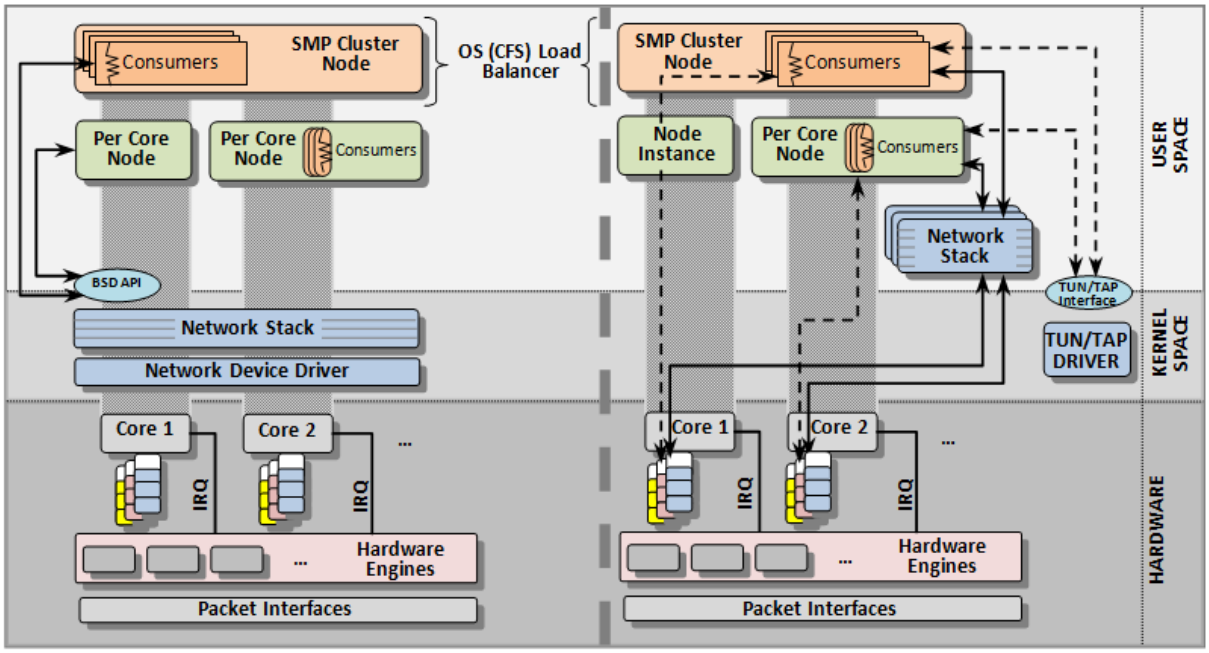
4.3 Hardware packet processing

Investigations of packet processing flows revealed that the highest consumption of resources is on receiver side rather than transmitter (where software implementations seem to be enough [32]).

Based on SoC block diagram presented earlier (Figure 21), I consider the following main stages[2] for each received frame (Figure 22):

1. **Stage 1**: frame headers and/or preset sections from payload are passed to packet processing engines;
2. **Stage 2**: the frame headers and/or preset sections from payload are parsed and matched against preconfigured protocols establishing packet integrity and identifying malformed frames;
3. **Stage 3**: the frame is classified and dispatched to preprogrammed flows based on parse results (i.e. push frame to a specific queue or group of queues);

[2]Designing a generic packet processor that could handle any number of flows with any number of engines per flow in any order without performance penalty is practical impossible, thus usually boundaries are imposed by the hardware to preserve performance and meet specific design requirements.



(a). Kernel space implementation (b). User space implementation

Figure 23: Hardware accelerated models

4. **Stage 4**: based on preprogrammed scheduling algorithms (e.g. weighted queues, Decreasing Time Algorithm, Round Robin, Shorted Job First or others) the frame is scheduled for next consumer (graph node: hardware engine or communication port);
5. **Stage 5**: the frame is pushed to next node in the processing graph.

4.4 Hardware accelerated cluster nodes

Most clusters use Linux as operating system, which by default does the packet processing in software (usually by TCP/IP stack). By using dedicated hardware, packet processing tasks are moved to hardware engines and the OS environment must be adapted to this change. Basically, the methods of offload implementations influences the system performance, such as throughput and latency.

OS is the usual solution for coexistence of different applications on the same hardware system. If the applications are operating systems, then virtualization (i.e. hypervisors such as KVM or XEN) is the solution or if groups of applications (such as independent network stacks) needs a specific isolation degree, then Linux containers (LXC) is the solution.

Given the above motivations, I propose two less restrictive models as alternatives for coexistence of applications with high and low traffic demands, using hardware acceleration and

parallel processing of flows: **Per Core Cluster Node** and **SMP Cluster Node** and two implementation directions: in **kernel** and **user space** (Figure 23).

Per Core Cluster Node model is characterized by the fact that each application instance is bound to a core, thus the scalability is limited to the number of CPU cores. The modularity is achieved by division of tasks at a smaller granularity by using dedicated threads (i.e. consumers) without being scheduled on a different core. Each consumer is bound to a specific hardware queue or group of queues that provides only specific frames to the consumer thread. The hardware engines must be instructed to classify and distribute flows to match the consumers.

SMP Cluster Node model is similar with former model with the difference that the workload is distributed over a set of cores using capabilities of hardware engines for flows classifications. One solution would be to let hardware engines to distribute workload by pushing frames on cores in "round-robin" fashion (using hash based functions), or to leave the OS scheduler to flatten the workload by scheduling consumers among the core set and instruct hardware engines to consider where the specific consumers are scheduled and push frames accordingly.

By configuring the hardware engines in concordance with software architecture and scheduler affinity, multiple instances of these models belonging to different applications in a converged system may coexist with an increased performance. It is obvious that SMP Cluster Node is befitting for high traffic volume applications with multiple consumers and also is a generic form that can be configured to behave as a Per Core Cluster Node as well if the core affinity is set to only one core.

To leverage the parallelism of multicore systems, I propose the usage of **Accelerated Receive Flow Steering (ARFS)** [33]: the frames distributions have to consider where the specific consumers were scheduled to run. This requirement imposes two things: first, each possible core where a consumer could be scheduled must have a specific queue or group of queues where the hardware engine is instructed to push the frames and second, the hardware engines must be able to steer and push the frames accordingly. In another words, the hardware engines collaborate with OS scheduler for workload balance while achieving a better performance degree.

Also, I propose two different implementations of these models: one uses the drivers in **kernel-space** (more efficient) and the other one uses drivers in **user-space** (small performance penalty, while assuring rapid development and flexible license policy). Figure 23 – (a) present the high level architecture of kernel-space implementations for both models, while Figure 23 – (b) depicts the user-space implementations.

An issue for the user-space model is the access method to a TCP/IP stack. By default Linux has the network stack implemented in kernel, thus in case of user-space implementation the applications have to deal with raw frames and lack of connection-oriented protocols. There are few solutions for adding TCP/IP support: one would be to use network tunneling/tap interfaces (thus each received frame by the user-space driver is inserted into the Linux TCP/IP stack through these interfaces and then the frame's payload is received back via sockets – the endpoint of an inter-process communication flow across a computer network), or another solution would be to use an off-the-shelf user-space TCP/IP stack (but, this method is not encouraged by the Open Community, nevertheless in some cases the user-space stack can provide better performance than the native stack [34]).

Another issue for the user-space model is the method of signaling the arrivals of frames. Linux kernel doesn't have a standard framework to signal interrupts to user-space, therefore polling[3] is one solution that doesn't violates Linux directives. Basically, the consumer puts itself to sleep if the queue is empty, or when it is scheduled to run it checks the queue and handles the frames within the time quantum set by the scheduler.

To decrease the latency of sensitive traffic, I propose a solution based on queue weighting. Basically, the queues associated with most sensitive flows have higher weights and are scheduled first, thus the sensitive flows are handled first.

How flows sensitiveness is established depends on applications purposes, thus for exemplification I propose a solution for Ceph by classifying the traffic of Monitors and OSDs in three different classes, each associated with a different group of queues with specific weights.

4.5 Case study: Hardware accelerated Ceph with QorIQ™

As case study, I used QorIQ™ P2041 integrated packet processor to decrease latency of sensitive flows and increase performance of Ceph's nodes: OSDs, monitors and metadata servers. The next subsections comprise a high-level view of QorIQ™, particularities of accelerated Ceph nodes and micro-benchmark results.

[3]POSIX OS signals are the faster inter-process communication that could be used to alert user-space processes, instead of polling.

4.5.1 QorIQ™ packet processors

QorIQ™ combines multi-general purpose cores (of power architecture[4]) with various hardware engines providing a very flexible infrastructure (**Data Path Acceleration Architecture – DPAA**) for processing high volume of traffic at high speeds. Among multiple hardware engines within DPAA, I focused on following three components: Queue Manager (**QMan** [35], pp. 6-1/299), Buffer Manager (**BMan** [35], pp. 7-1/531) and Frame Manager (**FMan** [35], pp. 8-1/594). QMan is the means by which data is passing between hardware modules, BMan role is to reduce overhead of software for memory management and FMan has three main functions (**PCD**): **P**arse fames to check packet integrity and identify protocols within frame headers, **C**lassify frames by means of generated keys (KeyGen [35], pp. 8-395/1557) based on parsing results and **D**istribute frames to queues following different distribution profiles.

Apart from standard protocols (i.e. hard-wired parser capabilities), FMan can be configured to parse and detect up to ***three*** *proprietary protocol headers or application defined fields* by means of **Software Parser** ([35], pp. 8-391/982) feature. It uses NetPDL (an XML-based language for describing packet headers [37]) to define non-standard fields configured by Frame Manager Configuration Tool. Nevertheless, the Software Parser has boundaries ([36], pp. 24) that reduces significantly its usability: each user defined fields has to be embedded between standard protocols (basically, it means that nested user defined fields are not supported), and each user defined field must follow a different standard protocol (meaning that multiple user defined protocols on top of the same standard protocol are not allowed). FMan processing flow of received frames starts with parser (hard-wired and software) which identifies protocols found in packet headers, followed by a 56bytes key generated based on parser results and a classification plan, and furthermore based on this key a specific queue is selected (i.e. distribution).

Despite of Software Parser's reduced functionality, the idea of an hardware engine capable to recognize user defined fields based on which flows classifications may be conducted is genuine and can be used to improve performance of network based systems.

[4]**Power Architecture** is a type of microprocessor architecture with RISC instruction sets (https://www.power.org/).

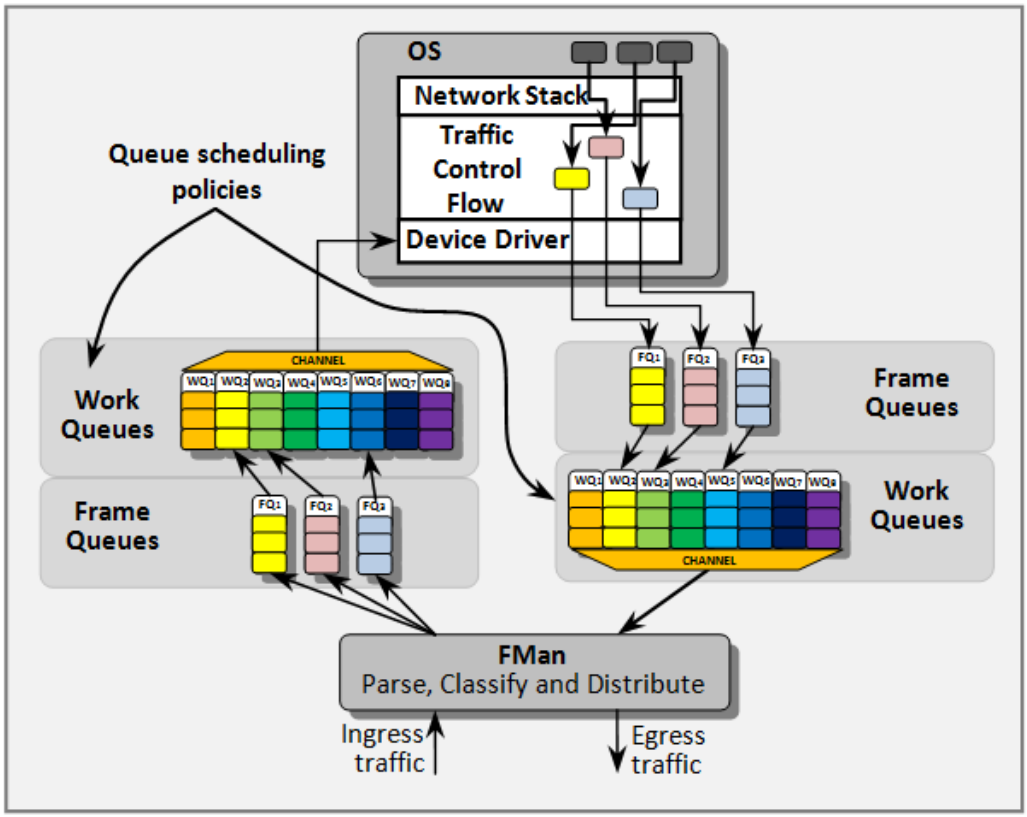


Figure 24: Ingress/Egress flow chart

In DPAA context, the unit of transmitting data is a *frame* (detailed by a frame descriptor), stored in *frame queues* (i.e. FIFO structure of frames). Frame queues granularity and creation depends on software architecture and applications purposes. Furthermore, frame queues are stored in *work queues* (and QMan applies different scheduling policies: strict priority based or round robin) grouped in *channels* of 8 prioritized items (Figure 24). Also, each channel identifies one hardware entity (e.g. CPU core, hardware engine or Ethernet port).

Besides hardware capabilities, Linux has its own traffic control called queueing discipline (qdiscs). Filters and traffic policers can be attached on ingress traffic by using Linux traffic control, while egress traffic is directed into FIFOs on which different scheduling mechanism can be applied (but, in this design I have not used any Linux queuing discipline).

4.5.2 Accelerated Ceph's nodes

Combining hardware engines capabilities with software architecture, the performance of each node in a cluster can be increased, therefore for the same performance is required a lower number of nodes which leads to smaller capital costs. Also, with the advent of converged

systems, each cluster node (following models presented in Section 4.4) may perform different roles in the same time.

One key for increasing performance of each node is the granularity of traffic division of each Ceph service in classes handled by specific consumers – following models presented in section 4.4. Basically, high traffic volume needs to be divided among multiple consumers, queues and related set of cores.

In RADOS context, OSDs nodes have the highest traffic volume (I/O mostly), while MONs need low latency traffic, since they handle OSDs failures. I propose **A-OSD** (as Accelerated OSD), **A-MDS** (as Accelerated MDS) and **A-MON** (as Accelerated Monitor). For A-OSD I propose a **SMP Cluster Node** model to be able to handle high I/O traffic, and same for A-MDS to be able to handle journals and metadata (stored by RADOS) and queries from clients. A-MON doesn't need high processing power, neither ports that supports high traffic volume, so in the context of converged systems, monitors may share the node with different other applications, therefore I propose a **Per Core Cluster Node** model.

Accelerated nodes can be implemented using QorIQ™ packet processors with FMan PCD capabilities and QMan scheduling features. Instructing FMan to classify and distribute frames according to each class particularities (e.g. see section 4.5.3 for A-RADOS classification), a performance increase is achieved. I propose for A-OSD and A-MSD a uniform distribution on queues, distributed further uniform on cores (in case where ARFS is not used), this way clients are distributed fair across the system rather than facilitating traffic of a single client. In case of ARFS usage, the queues are uniform distributed on the cores, but when a particular frame is pushed to a core's queue, the hardware engine consider where the consumer was scheduled to run. For system traffic (i.e. system consistency messages, journaling and metadata) classification I propose to use user defined flows recognition facilitated by Software Parser. A-MON has only user defined flows and therefore the only solution is the usage of Software Parser capabilities.

4.5.3 Accelerated RADOS based on queue prioritization

RADOS is composed by two distinct clusters: one of monitors and one of OSDs, where monitors are responsible for managing a large number of OSDs. Two important keys of data clusters (including RADOS) are data reliability and availability ([22], pp. 119). Usually, to make consistent data available to all parties, these must be handled in timely fashion.

For the above reasons, I propose an accelerated RADOS (**A-RADOS**) implementation based on packet processing engines. By classification and prioritization of RADOS messages I

aim to decrease latency of sensitive operations (listed in Table 4) and improve overall cluster responsiveness, thus improve data availability.

In **monitor** context, I consider that the most important message types are the ones that assure a consistent view of the system and are sensitives for cluster stability, followed by map distribution to monitors, responsible in its turn to distribute it further in the entire cluster. Therefore, I propose the following monitor traffic classes:

- **Traffic Class I**: Elections and leases messages;
- **Traffic Class II**: cluster-map updates distribution to Monitors;
- **Traffic Class III**: cluster-map distribution to all other system parties.

In **OSDs** cluster context, I consider that the messages required for system consistency are the most essential, followed by I/O traffic (i.e. read/writes and replication strategies) and storage recovery is the least sensitive operation, so I propose the following classification:

- **Traffic Class I**: heartbeats and cluster-map update propagation;
- **Traffic Class II***:* data I/O traffic (e.g. reads/writes) and data replication;
- **Traffic Class III***:* OSDs recovery.

Prioritization of traffic classes is achieved by configuring weights to flow queues, where each flow queue (or a group of flow queues – depending on hardware architecture) deals with a specific class. Thus, considering the above classifications, the first traffic classes have the highest weight and their flow queues will be scheduled first, while the last traffic classes have the lowest weight and their queues will be scheduled last, and therefore the latency of sensitive traffic is decreased while achieving a better system responsiveness.

Using a QorIQ packet processor, the prioritization of three different traffic classes is achieved by splitting work queues in three groups and configuring weights to reflect the traffic prioritizations listed above.

4.5.4 Micro-benchmark results

I consider that the number of clients served per seconds by a MDS (Ceph's application chosen as an example) reflects its performance. For testing purposes, I measured how much time a particular MDS request takes (in core ticks) and simulate using Per Cluster Node model, 4 different MDS nodes that handles requests in parallel. Using this setup, I did three different tests along with kernel adjustments: without hardware acceleration (i.e. without frame classification and distribution) and using Receive Flow Steering (software implementation only) with and without core affinity.

Model	Requests per second					CPU util.
	node 1	node 2	node 3	node 4	Avg.	
no acc.	6867	6877	6860	6882	6871	25%
RFS	7109	7117	7117	7119	7115	25%
RFS affinity	7672	7692	7717	7725	7701	27%

Table 8. Accelerated MDS micro-benchmark

Model	Requests per second			
	Class I	Class II	Class III	Total
no prioritization	2851	2902	2959	8712
queue prioritization	6556	1118	348	8022

Table 9. Accelerated RADOS micro-benchmark I

Model	Requests per second			
	Class I	Class II	Class III	Total
no prioritization	0	0	6957	6957
queue prioritization	0	0	6858	6858

Table 10. Accelerated RADOS micro-benchmark II

As hardware platform I used a QorIQ P2041 machine with 4 general purpose cores and hardware acceleration for flow steering, flow distribution and queue management (DPAA). The platform supports network connections of 1Gbps and 10Gbps, but for compatibility purposes with lab systems I've used only 1Gbps connections. The topology was very simple, composed by one PC with 1Gbps Ethernet card as client machine and P2041 as A-MDS.

For each test, I counted the number of handled requests per second (i.e. transactions). From the tests results listed in Table 8, the improvement in terms of transactions, by the test with RFS support and no affinity, was about 3.5% and comparable CPU utilization and with affinity was about 12%.

For A-RADOS I did two kinds of tests: by running concurrent requests of all three class types and by running only class III. Former tests type emphasizes that the most priority class is handled before the other classes if hardware queues prioritization support is used, while the latter shows that if only class III traffic is running the number of handled requests are comparable (Table 9 and Table 10).

5 NETWORKING CHARACTERISTICS IN CONVERGED INFRASTRUCUTRE

The main reasons of **Converged Networks** (also known as **Unified Networks Infrastructure**) are economic in nature, but also technological [38] [39] [40]: simplicity, cost saving, elasticity, requirements imposed by virtualization, better usage of servers resources, simplified management and so on.

A defining characteristic of I/O protocols for storage devices (such as SCSI) is that they do not handle lost data in timely fashion and therefore a lossless communication environment is befit. Usually SANs use Fibre Channel (used by approximately 80% of data center storage market [40]) which by design is a low latency and lossless environmental high speed (section 5.1). While Ethernet by design is a best effort communication environment and along with IP protocol it provides an end-to-end network for reliable transport protocols, such as TCP.

A converged support for LANs and SANs imposes a set of Ethernet enhancements (i.e. Data Center Bridging group of protocols) to enable a lossless medium. Also, Ethernet became a viable solution due to its advantages against FC networks, such as supported high speeds (up to 10Gbps, 100Gbps, 400Gbps or even new Intel's 800Gbps, while FC supports up to 2, 4, 8, 16 or 32Gbps just arriving) or lower capital-costs.

Nevertheless, there are several options that can be used with vanilla Ethernet too (e.g. iSCSI, iFCP or FCIP), but requires reliable protocols underneath to solve contention and congestion issues (e.g. TCP). Basically, these approaches are at a lower costs, but bring a significant performance penalty due to extra encapsulation. Usually, these are used for small and medium business along with low cost Ethernet environments [41].

Fibre Channel over Ethernet (FCoE) protocol enables the transmission of FC frames over Ethernet and it relies exclusively on DCB [42] extensions, which today comprise the following protocols:

- **Priority Flow Control** (PFC);
- **Enhanced Transmission Selection** (ETS);
- **Data Center Bridging Exchange** (DCBx);
- **Quantized Congestion Notification** (QCN).

However, these extensions can be used for any loss-sensitive application that does not have contention or congestion control mechanism and requires only L2 protocols. There are even more ingenious ideas, such us improving *TCP incast* communication problem (i.e. multiple servers simultaneously transmit TCP data to a single aggregator: TCP performance is degraded as consequence of retransmission timeouts as result of packet loss due to overwhelmed queues at network bridges level) [43].

In the next subsections I provide a brief description of these extensions with more focus on Congestion Notification (QCN) and few improvements proposed.

5.1 Fibre Channel

Fibre Channel is a set of protocols developed by American National Standards Institute (ANSI) [55]. Basically, it aims to support fast transfer of high volume of data over an interconnection topology called *Fibre Channel Switched Fabric* (which includes characteristics such as: many-to-many connectivity, device lookup, security, redundancy or zooning). In Fibre Channel context, at the basis of links lays two different ideas of transporting data to and from peripherals: channels and networks. Basically, channels are high speed with low overhead transport medium, while networks are slower with high overhead, but very flexible.

In terms of architecture, Fibre Channel it is split into 5 layers, as follows:

- FC – 4 Protocol-mapping layer: SCSI, IP or FICON, etc.
- FC – 3 Common services layer: encryption, RAID redundancy algorithms, etc.;
- FC – 2 Network layer: main framing protocols;
- FC – 1 Data link layer: byte encode/decode;
- FC – 0 Physical link: phys, lnks, etc.;

To enable different types of traffic, Fibre channel support three main service classes, each with different characteristics, as follows:

- Class 1: Dedicated bandwidth, order delivery, acknowledged end-to-end
- Class 2: Shared bandwidth, acknowledged end-to-end
- Class 3: Shared bandwidth, lossless transmission through a buffer credit

Usually, class 3 is used in SAN based storage systems, because it offers a lossless transfer medium and this way it eliminates the time penalty of SCSI recovery. Basically, recovery sequences introduce large delays in case of commands loss due to congestion points.

Lossless is provided by a buffer credit mechanism, where the receiver indicated the number of frames that can be transmitted without been forced to drop any of them – buffer to buffer flow control. Based on a similar idea, an enhancement for Ethernet was developed: Quantized Congestion Notification, detailed later in section 5.2.4.

QCN together with several other protocols were designed to enable a lossless Ethernet (a viable alternative to Fibre Channel), section 5.2.

5.2 Data Center Bridging

5.2.1 Priority Flow Control

Ethernet flow control (IEEE 802.3x PAUSE frame [44]) was first introduced in 1997, in which the receiver sends PAUSE frames to inform the sender not to send anymore packets during pause time. But, this implies that multiple flows with different QoS requirements cannot coexist on the same network segment. To enable this scenario, distinct pause frames should control distinct *communication channels* independently within the same network segment (IEEE 802.1Qbb/PFC [45]), therefore were created 8 distinct traffic classes that defines a quality of service prioritization scheme with 8 different priorities managed by separate queues (IEEE 802.1p/802.1Q).

5.2.2 Enhanced Transmission Selection

PFC guarantees a lossless environment, but each traffic flow has different bandwidth requirement based on Service Level Agreement (SLA) or architectural restrictions (e.g. Operating System implementation), therefore 802.1Qaz ETS [46] enables flows classification per traffic classes (e.g. LAN traffic, I/O SAN traffic, Network management traffic such as LLDP, cluster management traffic such as heartbeats, multicast and so on). The main idea is that each flow mapped to a class shares the same assigned bandwidth and the same class characteristics (e.g. loss).

ETS and PFC are built to coexist and the 8 priorities (defined by PFC) can be mapped to different traffic classes, where each class has assigned a bandwidth percentage at 1% granularity.

5.2.3 Data Center Bridging eXchange

In order to achieve interoperability between multiple network domains with different capabilities, peers exchange information about supported features and configurations (PFC, ETS and application priority configuration TLVs), then selects and agrees to a specific configuration [47]. Basically, it uses Link Layer Discovery Protocol [50] to exchange attributes embedded in Organizationally Specific TLVs (OUI), between partners.

5.2.4 Quantized Congestion Notification

Quantized Congestion Notification [48] ([49], pp. 1071) enables peer-to-peer congestion management by dynamically adjusting throughput due to changing bottlenecks in absence management of an upper layer protocol, such as TCP window sizing (i.e. dynamically determine how many frames to send at once without acknowledgements).

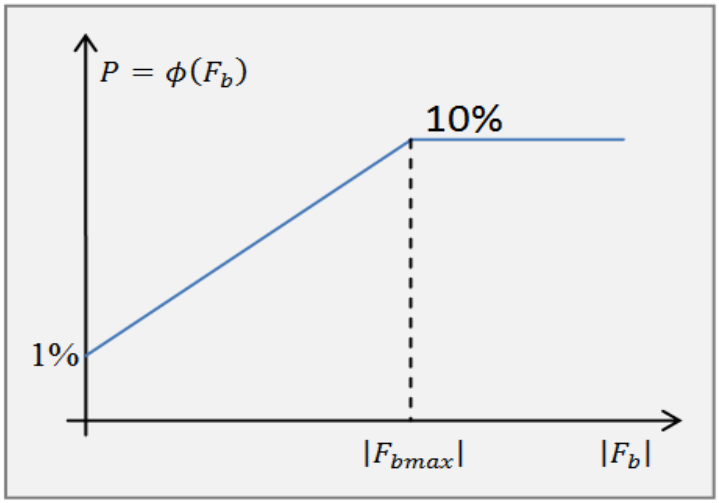


Figure 25: QCN sampling probability

It is composed by three main components: congestion, reaction and reflection points. Congestion Points (CPs) samples incoming frames and notifies sources (i.e. Reaction Points) of the sampled frames by sending Congestion Notify Messages (CNMs) to adjust their rates when the queue utilization is increasing too fast or is not decreasing fast enough against a preset equilibrium value (Figure 26).

Congestion Points computes feedback messages by combining the first derivative of queue utilization (i.e. rate excess) with the instantaneous queue utilization against a considered equilibrium (i.e. queue size excess):

$$\boldsymbol{F_b = -\left(Q_{offset} + \omega \cdot Q_\delta\right)}\text{, where}$$

$$Q_{offset} = Q - Q_{equilibrium}, \qquad (1)$$

$Q_\delta = Q - Q_{old}$ and ω is rate weight (taken to be 2 for baseline simulations).

At each sampling event, the sampling probability is updated as a linear function of $|F_b|$, with minimum probability of 1% and maximum of 10% (e.g. 100 frames/sampling reflects 1% probability), Figure 25.

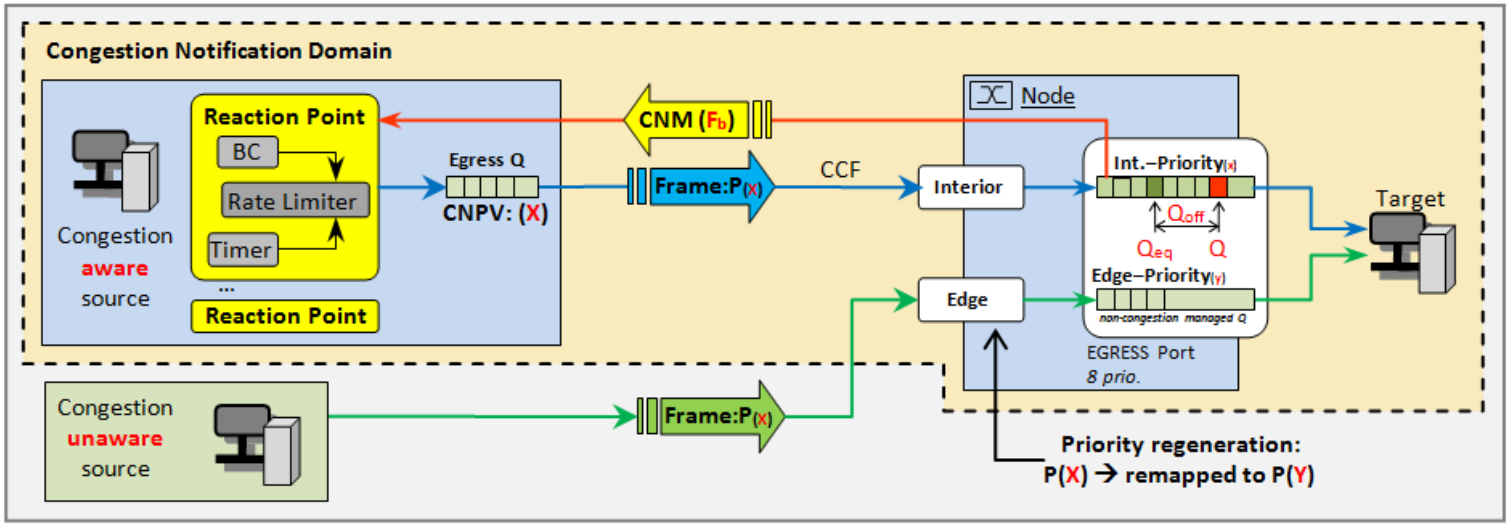


Figure 26: Quantized Congestion Notification (QCN) – high level view

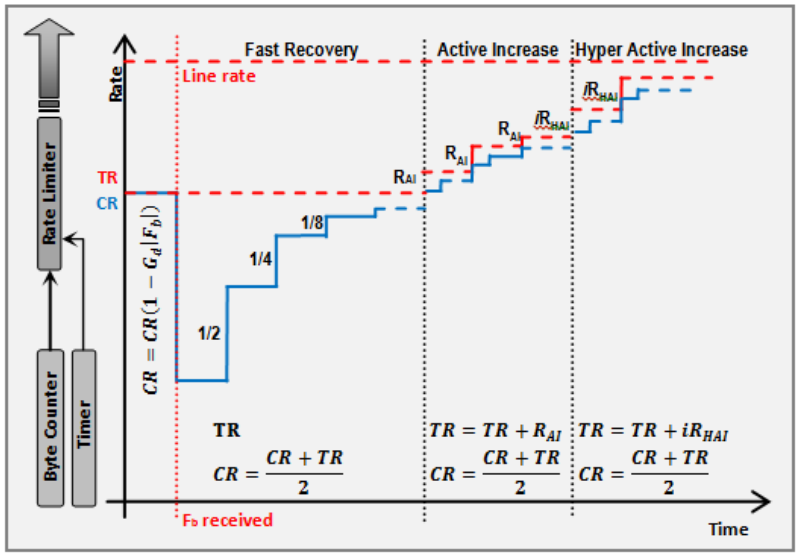


Figure 27: Reaction point's rate limiter

Reaction point's *rate limiters* (RLs) adjust *current rates* (CRs) to reach at *targeted rates* (TRs) using a *byte counter* in phases (Figure 27), as follows:

- First, feedback is received, RL sets TR as CR and CR is decreased with not more than half, as shown below:

$$\boldsymbol{TR = CR}$$

$$\boldsymbol{CR = CR \cdot (1 - G_d \cdot |F_b|)}, \text{ where}$$

G_d is chosen so that $G_d \cdot |F_{bmax}| = \frac{1}{2}$ and Fb has 6bit quantization, therefore (2)

$$F_{b-quantized} = \frac{F_b}{F_{bmax}} \cdot 64, \text{ so } G_d = 0.0078125$$

- Then, it enters in rate increase composed by three phases: Fast Recovery (FR), Active Increase (AI) and Hyper-Active Increase (HAI). In FR phase the reaction point tries to recover the lost bandwidth, while in the next phases it checks for extra bandwidth:

$$\boldsymbol{TR = TR + R}$$

$$\boldsymbol{CR = \frac{1}{2}(CR + TR)}, \text{ where}$$

R is 0 in *fast recovery*, 5Mbps for *active increase* and *i*50Mbps for and *hyper active increase* (where *i* is the i^{th} smaller cycle counted by BC and timer in AI) for 10Gbps line rate. (3)

The rate increases phases have 5 cycles each. At the end of each group of 5 cycles, if no F_b has been received, RL enters in next phase (where in HAI phase, the RL throughput can reach at line rate). In cases when CRs are very small and the BCs measured in time are very large, the phases are controlled by timers. The timer is similar with the BC and counts 5 cycles of Tms (e.g. of 10ms for a 10Gbps line rate) in FR and $\frac{T}{2}$ms in AI.

Regarding both byte counter and timer, RL is in FR phase if both BC and Timer are, and CR is updated when at least one completes a cycle. The RL is in AI phase if at least one completes a group of 5 cycles and eventually the RL is in HAI if both BC and timer are (Figure 27).

From network architecture standpoint, all congestion aware points are grouped in Congestion Notification Domains (CNDs) and flows that are congestion controlled are named Congestion Controlled Flows (CCFs), Figure 26.

5.3 QCN weaknesses and improvements

Due to existence of different types of devices within a network, the coexistence of congestion aware and unaware segments is a challenge. According to standard, this situation is solved using priority regeneration tables, therefore when frames are received from congestion unaware segments with a priority that is congestion controlled, it is remapped to different priority left unused, so this implies that at least one priority to be kept for this scenario (i.e. alternate non-CNPV priority ([49], pp. 1071).

The weakness is revealed when automatic configuration is enabled and alternate priorities are automatically chosen, and according to standard, priority chosen is the lowest unused in CNDs (Figure 28). This may affect QoS policies used for other traffic types and lead to unwanted traffic injections for that QoS domain from edge ports.

One solution would be to check each bridge within topology and verify all QoS requirements and priorities. Impractical, because a topology may have a high number of network devices and it won't be automatic anymore. The proposed solution contains a hybrid defense mode choice that allows both manual and automatic settings to use the same alternate priority value (Figure 28), therefore the administrator is able to choose the alternate priority used by automatic configuration and avoid QoS disruptions **[Patent #8891376]**.

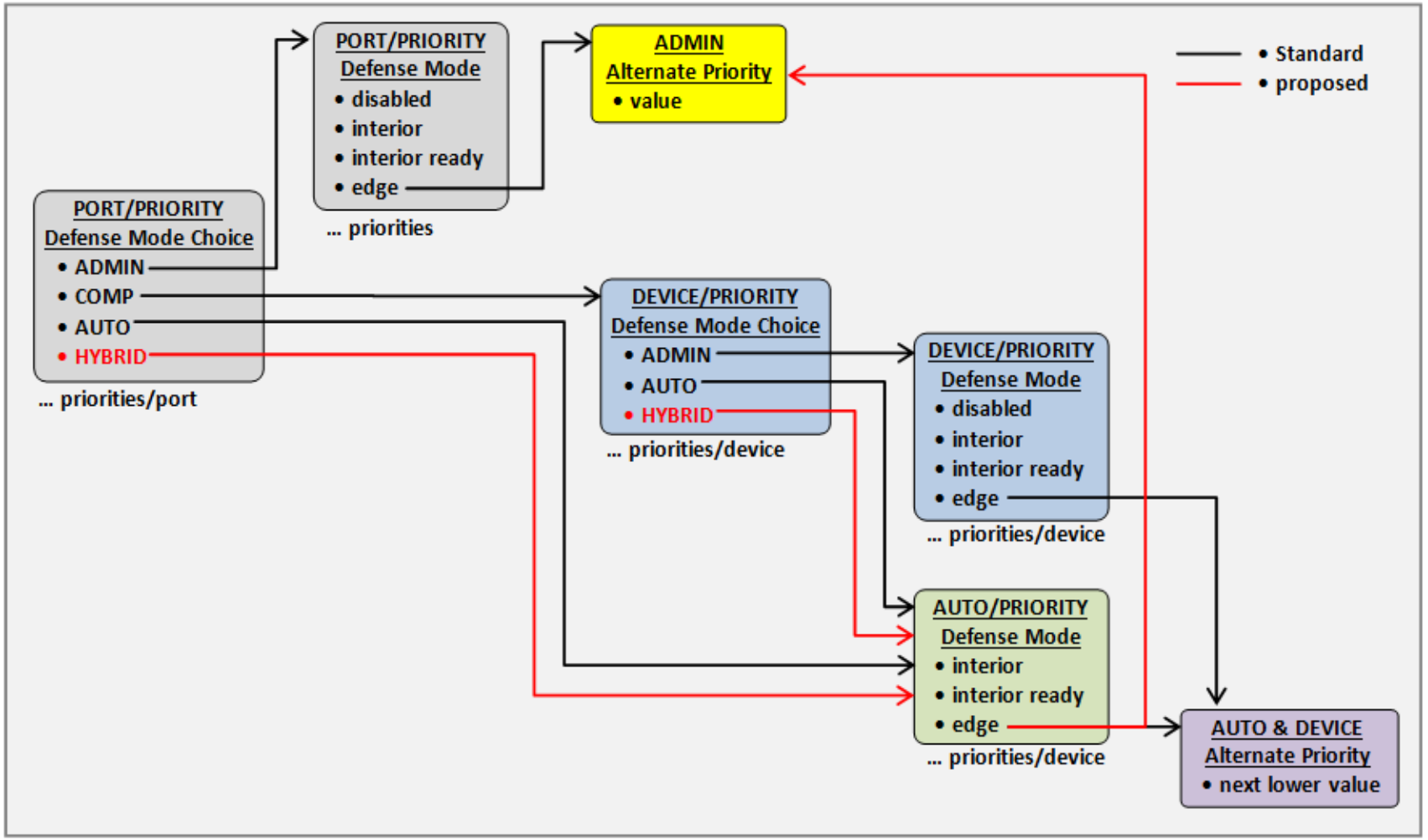


Figure 28: Quantized Congestion Notification – Congestion Notification Domain defense

Another weakness of QCN is the rate unfairness of multiple flows sharing the same bottleneck link. This comes from the fact that when a congestion point computes the feedbacks it doesn't take into consideration the rate of the reaction points. This issue has been solved by two proposed algorithms: Approximate Fairness with QCN (AF-QCN [51]) and Fair Quantized Congestion Notification (FQCN [52]).

5.3.1 The Fair QCN Solution

The main idea of FQCN is that every flow has a share rate with respect to the congested queue and the feedback computed per flow which is proportional with the exceeded rate.

Related to each congestion point (i.e. egress queue), at each sampling event t the feedback is computed as in shown is Eq. (1). At this moment t the total bytes received for each flow f_i is denoted as $B_i(t)$. Also, each flow f_i has a preconfigured weight w_i , where if all flows are fair with respect to each other, the weight is 1.

Furthermore, the fair share for each flow f_i is computed proportional with weights and received bytes, as shown below:

$$M_i(t) = \frac{w_i}{\sum w_i} \cdot \sum B_i(t) \quad (4)$$

Then the culprits are identified by comparing the shares $M_i(t)$ with the actual rates $B_i(t)$ and if the share is exceeded then the flow is added to the culprits list, denoted S^H.

Further, the share is fine grained by considering only culprits share, as shown below:

$$M_i^H(t) = \frac{w_i}{\sum_{S^H} w_i} \cdot \sum_{S^H} B_i(t) \quad (5)$$

So, the fine grained shares $M_i^H(t)$ are compared with the actual rates $B_i(t)$ and if the share is exceeded then the flow is added to the fine grained culprits list, denoted as S^R.

The actual feedback sent to each culprit is computed with respect to their shares, as shown below:

$$\Psi_{F_b}(i,t) = \frac{\frac{B_i(t)}{w_i}}{\sum_{S^R} \frac{B_i(t)}{w_i}} \cdot \Psi_{F_b}(t) \quad (6)$$

where $\Psi_{F_b}(t)$ is the feedback quantized at 64 bit.

Summarizing, the FQCN algorithm aims to things: first it identifies the overrated flows and second the feedbacks are individual to each source using per flow monitoring.

In my opinion, there are few implementation drawbacks: the byte counter per flow must be supported by hardware, because the software implementations based on interrupts or polling may have problems at high rates and the flow tables can be very large, in which case the management is hampered.

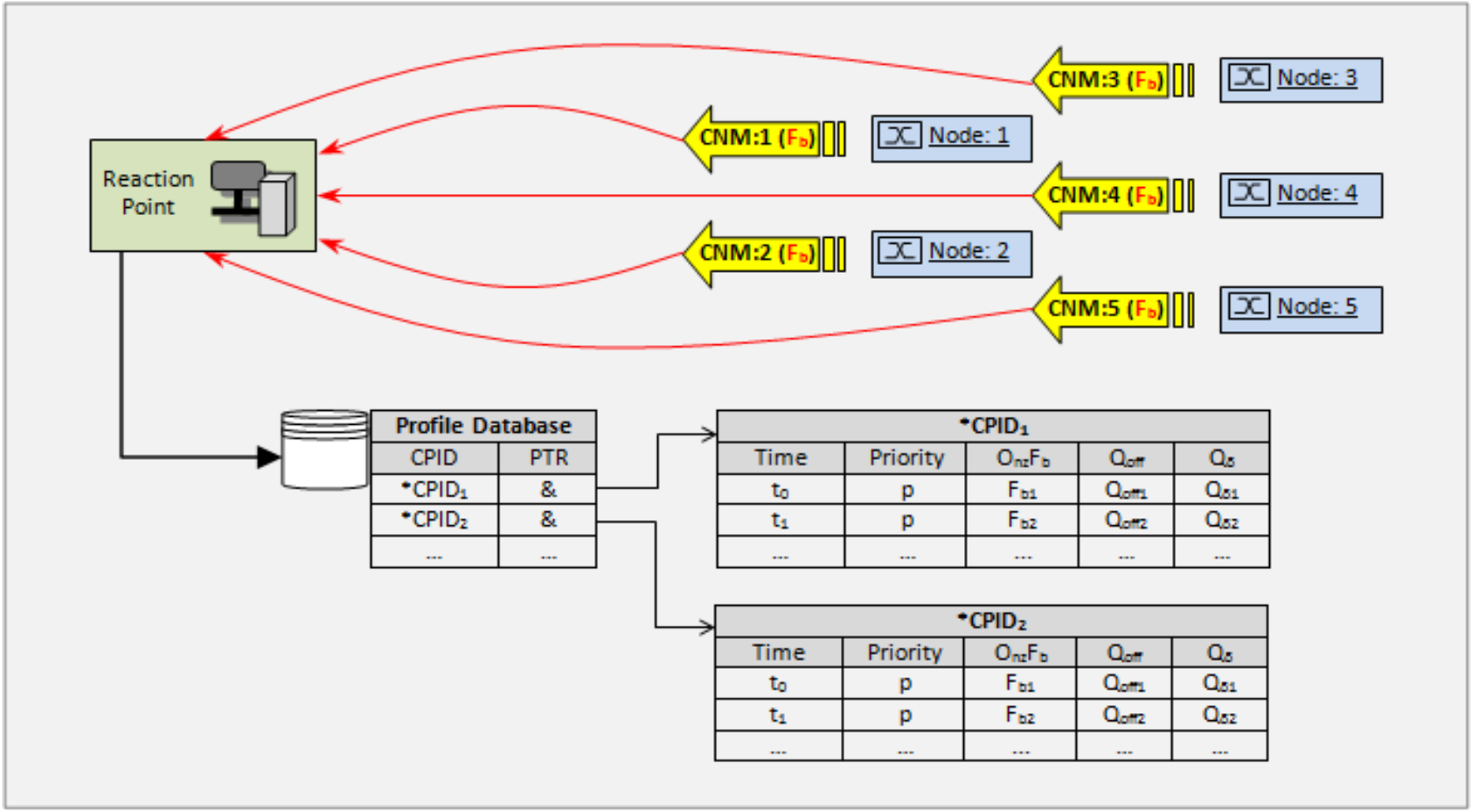


Figure 29: Quantized Congestion Notification – Congestion profiling

6 DYNAMIC LOAD BALANCING ALGORITHM BASED ON QCN NETWORKS

6.1 Motivation

It is almost impossible to predict the load profile of a network at a given time or to build an optimal topology or a configuration to ensure a uniform distribution of congestion domains, therefore decision-making based on profiling may lead to better performances.

I propose **Quantized Congestion Notification – Weighted Flow Queue Ranking** (QCN-WFQR), an algorithm based on Quantized Congestion Notification protocol **[PIST, 2015]**. The main idea of the algorithm is to compute a series of congestion indicatives used by different entities (e.g. clients, Network Profilers or SDN controllers) to balance the traffic and use better the network bandwidth. The algorithm is generic, so that it can be used in old and well known computer networks and in the novel software defined networks paradigm.

6.2 Alternative solutions

In congestion aware networks (i.e. QCN based), each reaction point receives feedback messages from multiple congestion points and adjust rate accordingly (Figure 29). To achieve a better congestion profile in the network, one solution would be that each congestion source (i.e. reaction point) to create a data base with congestion indicatives per each congestion point from which it receives feedbacks, thus each reaction point in the system is associated with indicatives received from each congestion point (CPID: congestion point ID embedded in each congestion notification message): t (time of occurrence), p (priority of sampled frame), $Q_{nz}F_b$ (quantized value of feedback), Q_{off} (queue size excess) and Q_{δ} (rate excess) **[App. #20150023172]**, Figure 29. Furthermore, reaction points may use these databases for performance analysis, based on which system administrators can change topology configuration and improve the system performance. Or reaction points may use them to adapt and influence different aspects of the network devices and obtain a better congestion profile in the system.

I chose to follow a different path, where instead of databases created by each reaction point, each congestion point creates profiles for each reaction point it servers. The main idea of the algorithm is to compute different congestion indicatives, local (by each congestion point) and system wide. Based on these indicatives the congestion profilers or SDN controllers are instructed to dynamically move flows between congestion points for a better congestion profile at system level. Also, targets can be instructed to choose a device that has the lowest contribution to system congestion.

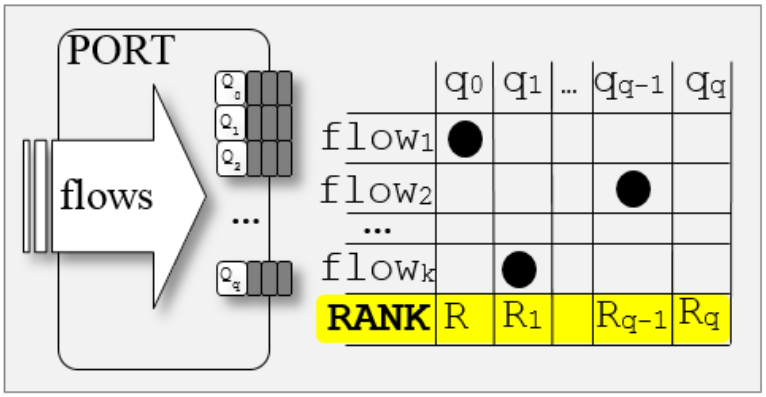


Figure 30: Port's queue ranking

6.3 QCN Weighted Flow Queue Ranking

In general, in distributed systems, the knowledge database that reflects the system and network status is huge and constantly changing (e.g. new flows are added or old ones are removed). This database can be built based on all incomplete and limited view of the status from all nodes within the system. Regarding QCN-WFQR, the focus is only on congestion status of the computer network. The main idea is that each node keeps a local information database that reflects the congestion status of a portion of the network from its point of view. Further, all local information databases are gathered by a central entity and compute a database related to the entire system. Depending on purposes of the applications, system wide or local congestion information may be used to achieve a better usage of the network and improve the system performance.

The local congestion indicatives are **flow shares**, **queues weights** and **queue ranks**, while system congestion indicatives are **flow weights** and **reaction point (or device) weights**. It is worth to mention that the system congestion indicatives are computed by a profiler (logically centralized or distributed) or by SDN controllers (depending on the network type).

Furthermore, I propose a method of QCN-WFQR usage in distributed and parallel file systems, where by using system wide congestion indicatives, a particular data clone is chosen to achieve a better balanced traffic load within the network.

Also, I propose a method of using QCN-WFQR for distributing traffic workload between multiple servers hosting the same application and migrating already established flows to alternate, less congested paths, thus reducing the need to slow down the traffic at the source.

Reaction Points	Flows	Node/queue flow shares			Congestion System Indicatives	
		Node$_1$/Queue$_i$	...	***Node$_k$/Queue$_j$***	**Flow weights**	**RP Weights**
RP$_1$	**flow$_1$**	$Share_{flow_{1/1}}$	...	$Share_{flow_{1/k}}$	$Weight_{flow_1}$	$Weight_{RP_1}$
RP$_2$	**flow$_2$**	$Share_{flow_{2/1}}$	...	$Share_{flow_{2/k}}$	$Weight_{flow_2}$	$Weight_{RP_2}$
...						
RP$_r$	**flow$_n$**	$Share_{flow_{n/1}}$	...	$Share_{flow_{n/k}}$	$Weight_{flow_n}$	$Weight_{RP_r}$

Table 11: Congestion System Indicatives

6.3.1 Defining the topology

A topology is defined as sets of network nodes, ports, queues and reaction points, as follows:

- $\mathcal{N} = \{N_1, N_2 \dots N_n\}$ set of network nodes,
- $\mathcal{P} = \{P_1, P_2 \dots P_p\}$ set of ports,
- $\mathcal{Q} = \{q_1, q_2 \dots q_l\}$ set of queues,
- $\mathcal{R} = \{R_1, R_2 \dots R_r\}$ set of reaction points and
- $\forall N_k$ has a sub-set $\mathcal{P}^{N_k} = \{P_i | P_i \in \mathcal{P}, P_i \; related \; to \; N_k\}$ of ports
- $\forall P_k$ has a sub-set $Q^{P_k} = \{q_i | q_i \in \mathcal{Q}, q_i \; related \; to \; P_k\}$ of queues

At moment *t* the network profile is defined as sets of flows and feedbacks, as follows:

- $\mathcal{F} = \{f_1, f_2 \dots f_m\}$ set of flows, where each queue q_k controls a sub-set of flows:
- $\mathcal{F}^{q_k} = \{f_i | f_i \in \mathcal{F}, f_i \; sampled \; by \; q_k, q_k \in \mathcal{Q}\}$ and each flow f_k is controlled by a sub-set of queues:
- $\mathcal{Q}^{f_k} = \{q_i | f_k \in \mathcal{F}, f_k \; sampled \; by \; q_i, q_i \in \mathcal{Q}, \}$ and each reaction point R_k adjusts rates of a sub-set of flows:
- $\mathcal{F}^{R_k} = \{f_i | f_i \in \mathcal{F}, f_i \; adjusted \; by \; R_i, R_i \in \mathcal{R}\}$, where f_k rate is adjusted at moment *t based* on received feedback $Fb_t^{q_i}(f_k)$ from queue q_i which sampled flow f_k at moment *t.*

6.3.2 QCN-WFQR indicatives

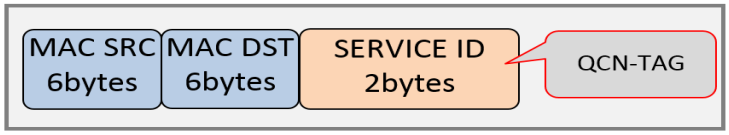
The QCN Weighted Flow Queue Ranking (QCN-WFQR) algorithm provides several congestion indicatives (per node – local) and system related, based on which decisions can be

taken and achieve a better balanced traffic load in a congestion aware network while having an overall increased performance in the system.

6.3.2.1 Flows

A computing node (e.g. server) in distributed systems offers services by means of flows. A service is identified by a service id which can be encapsulated into congestion notification tags ([49], pp.1096). As standard states, because QCN tags are optional, a service can be a portion of the frame (e.g. UDP port, IP address and so on). Also, each application must define its services and register them.

In general (to meet SDN requirements), a flow can be identified by the source address, destination address and service id, as shown below:



6.3.2.2 Ranking

A **queue rank** (7) within a port is defined as the number of flows within that queue (assuming that each flow has exactly one reaction point and exactly one destination, ***no multicast***).

To compute the ranks, each bridge (network node) keeps a table (Figure 30) per each of its ports with an aging function, with all the flows that are forwarded through each queue within the port. The algorithm is generic and therefore each port could have any number of queues (depending on hardware implementation), however the IEEE 802.1p standard enables only 8 queues.

A queue q_k rank at moment t is computed as the number of flows within the queue, as follows:

$$R_t^{q_k} = COUNT_t\langle f_i\rangle, f_i \in Q^{f_i} \quad (7)$$

6.3.2.3 Flow shares

A **flow share** (10, 16) is defined as a congestion measure from each topology's node point of view for a specific flow related to a port's queue (i.e. congestion point).

6.3.2.3.1 Flow shares based on QCN

The flow share has to contain information to be able to compare flows that shares the same congestion points and also flows with different routes and different congestion points. Therefore, the share is a combination of three indicatives: feedback average within a time frame, the frequency of feedbacks transmitted to a flow and queue rank within the time frame. The feedback is a measure of congestion related to the queue regardless the source rate, so I assume that higher rates implies higher probabilities to receive feedbacks for a flow. The rank of a queue reflects the number of flows affected in case of congestion (i.e. the main idea is to affect the smallest number of flows in case of congestion).

I eliminated feedback tables per flow and the associated aging routine by approximating the feedback average using an exponential moving average function, as shown below:

$$EMA_t = \alpha_t Y_t + (1 - \alpha_t) \cdot EMA_{t-1}$$
$$\alpha_t = 1 - e^{-\frac{\Delta t}{T}} \quad (8)$$

To reflect the frequency of transmitted feedbacks for a flow, when a feedback is sent to a specific reaction point for a specific flow, all the other flows within the queue recalculates their feedbacks averages by considering a null feedback sample, therefore flows with smaller rates are more likely to have a larger number of null samples than flows with higher rates, as shown below:

$$\Psi_t^{q_k}\left(Fb_t^{q_k}, f_i\right) = \begin{cases} \alpha_t \cdot \Psi_t^{q_k}\left(Fb_t^{q_k}\right) + (1 - \alpha_t) \cdot \Psi_{t-1}^{q_k}\left(Fb_{t-1}^{q_k}, f_i\right), q_k\ sample f_{i,} \\ (1 - \alpha_t) \cdot \Psi_{t-1}^{q_k}\left(Fb_{t-1}^{q_k}, f_i\right), otherwise \\ \boldsymbol{MIN}_{f_j \in \mathcal{F}^{q_k}} \Psi_t^{q_k}\left(Fb_t^{q_k}, f_j\right), initial\ value \end{cases} \quad (9)$$

Where $\Psi_t^{q_k}\left(Fb_t^{q_k}\right)$ is quantized feedback at moment *t* relative to queue *k*.

And $\Psi_t^{q_k}\left(Fb_t^{q_k}, f_i\right)$ is quantized feedback for f_i at moment *t* relative to queue k.

Also, the initial value has a big influence to the evolution of the EMA. Therefore instead of starting from zero, I consider the initial value as the minimum value of the existing flows sampled by q_k (this decision is based on simulations).

So, flow share for flow f_i computed by queue q_k at moment *t* is:

$$\boldsymbol{Share_t^{q_k}(f_i) = \Psi_t^{q_k}\left(Fb_t^{q_k}, f_i\right) \times R_t^{q_k}} \quad (10)$$

6.3.2.3.2 Flow shares based on FQCN (FQCN-WFQR)

If FQCN algorithm is used, briefly presented in section 5.3.1, the feedback is computed by considering the bytes received in a specific time frame *T*. Most network devices have the ability to provide average throughput per queue basis, but it is needed per flow basis, which it may be considered a drawback since it is very hard to implement this feature at hardware level. Considering the above motivation, there is a possibility to approximate throughput per flow by using the same method of exponential moving average, as shown in Eq. (11).

So, throughput at moment *t* for flow f_i related to queue *k* is:

$$Rate_t^{q_k}(f_i) = \alpha_t \cdot Bytes(frame \in f_i) + (1 - \alpha_t) \cdot Rate_{t-1}^{q_k}(f_i)\ ,$$
$$\text{Where } \alpha_t = 1 - e^{-\frac{\Delta t}{T}} \quad (11)$$

Furthermore, to compute the bytes received in time frame *T* for flow f_i related to queue *k* is easy:

$$B_t^{q_k}(f_i) = T \cdot Rate_t^{q_k}(f_i) \quad (12)$$

Based on the Eq. (12), the FQCN shares and FQCN fine grained shares are computed as follows:

$$M_t^{q_k}(f_i) = \frac{w_i^{q_k}}{\sum w_i^{q_k}} \cdot \sum B_t^{q_k}(f_i) \quad (13)$$

$$MH_t^{q_k}(f_i) = \frac{w_i^{q_k}}{\sum_{S^H} w_i^{q_k}} \cdot \sum B_t^{q_k}(f_i) \quad (14)$$

And the FQCN feedback for each culprit is computed as shown below:

$$\Psi_t^{q_k}\left(Fb_t^{q_k}, f_i\right) = \frac{\frac{B_t^{q_k}(f_i)}{w_i^{q_k}}}{\sum_{S^R} \frac{B_t^{q_k}(f_i)}{w_i^{q_k}}} \cdot \Psi_t^{q_k}\left(Fb_t^{q_k}\right) \quad (15)$$

Where $\Psi_t^{q_k}\left(Fb_t^{q_k}\right)$ is the feedback quantized at 64 bit and $\Psi_t^{q_k}\left(Fb_t^{q_k}, f_i\right)$ is quantized feedback for flow f_i

And S^R is the fine grained culprit list, explained in section 5.3.1.

Each event time, the QCN-WFQR flow share is updated similar with Eq. (10), as shown below:

$$\boldsymbol{Share_t^{q_k}(f_i) = \Psi_t^{q_k}\left(Fb_t^{q_k}, f_i\right) \times R_t^{q_k}} \quad (16)$$

The difference between **QCN-WFQR** and **FQCN-WFQR** is that FQCN consider the flow rates when computes feedback, therefore the flow share contains enough information to be able to compare flows that shares the same congestion points and also flows with different routes and different congestion points.

6.3.2.4 Weighting

A **flow weight** (17) is defined as a congestion measure from *system point of view* for a specific flow, while a **reaction point weight** (18) is defined as a congestion measure from *system point of view* for a specific reaction point.

Therefore, a **flow weight** at moment *t* for flow f_k is computed as the sum of flow shares received from all nodes (i.e. Q^{f_k}):

$$\boldsymbol{Weight_t(f_k) = \sum_{q_i \in Q^{f_k}} Share_t^{q_i}(f_k)} \quad (17)$$

And a **reaction point weight** at moment t for R_k is computed as the sum of all flow weights for flows whose rates are adjusted by it (i.e. $\mathcal{F}^{R_k}$):

$$\boldsymbol{Weight_t(R_k)} = \sum_{\boldsymbol{f_i \in \mathcal{F}^{R_k}}} \boldsymbol{Weight_t(f_i)} \tag{18}$$

A **queue weight** (19) at moment t is defined as a local congestion measure related to queue q_k and computed as sum of all feedback averages of flows controlled by the queue (i.e. $\mathcal{F}^{q_k}$) and its rank ($R_t^{q_k}$), as follows:

$$\boldsymbol{Weight_t(q_k)} = \left[\sum_{\boldsymbol{f_i \in \mathcal{F}^{q_k}}} \boldsymbol{\Psi_t^{q_k}\left(Fb_t^{q_k}, f_i\right)}\right] \times \boldsymbol{R_t^{q_k}} \tag{19}$$

This weight enables the comparison of queues based on which a specific flow may be routed or migrated achieving a better balanced traffic within the network.

There is a special situation in weights calculus, when no congestion occurred in the considered time frame and feedback averages are substantially large. In this case the congestion indicatives do not reflect the actual network congestion profile and decisions may lead to multiple congestion points. To avoid these circumstances, the feedback averages are reset if the last event occurred outside the time frame.

The congestion indicatives are heavily influenced by the starting rate of a reaction point. If the throughput of a reaction point is reset to line rate, then when it starts will congest a node and it may cause a substantially change of the congestion profile for a short period of time. To avoid this situation, the reaction point throughput must be set at minimum rate (e.g. 10Mbps when line rate is 1Gbps) rather than line rate and the throughput will increase gradually but fast without causing a burst in the network congestion profile.

6.3.3 QCN-WFQR algorithm based on standard QCN

Local congestion indicatives calculus: flow shares, queue ranks and queue weights

```
procedure Q-DO-ENQUEUE(q, p)                    ° add received packet p to sampled queue q
    Fb ← QCN-FEEDBACK(q, p)                     ° compute feedback Fb for received packet p
    if Fb < 0 then                              ° if computed Fb is negative then do QCN-WFQR algorithm
        call QCN-WFQR-UPDATE-FLOW-TABLE(q, p)   ° update flow table is packet p belongs to new flow
        call QCN-WFQR-UPDATE-QUEUE-RANK(q)      ° update queue rank (size of flow table)
        call QCN-WFQR-UPDATE-INDICATIVES(q, p, Fb)  ° update QCN-WFQR congestion indicatives
    end if
    add (q, p)                                  ° add packet p to queue q
end procedure

procedure QCN-WFQR-UPDATE-FLOW-TABLE(q, p)      ° update flow table regarding packet p (if new flow)
    if flow(p) ∉ flow_table(q) then             ° if packet p belongs to a new flow, then add flow to table
        add (flow(p), table(q))
    end if
    call QCN-WFQR-AGE-FLOW_TABLE(q)             ° remove from table older flows slipped from time frame
end procedure

procedure QCN-WFQR-UPDATE-INDICATIVES(q, p, Fb) ° local congestion indicatives: flow shares, q rank and weight
    c_time ← current time                       ° current time (for EMA calculus)
    min_ema ←  QCN-WFQR-GET-MIN-EMA(q)          ° minimum EMA  from all flows within flow table
    q_weight ← 0
    for "each entry e" ∈ flow_table(q) do       ° for each flow from table related to queue q
        if e.ema_old = 0 then                   ° initialize OLD EMA with minimum if first
            e.ema_old ← min_ema
        end if
        e.ema ← EMA (p, Fb, c_time, e);         ° new EMA for flow related to entry e
        e.flow_share ← e.ema * rank(q)          ° compute flow share for entry e
        q_weight ← q_weight + e.flow_share      ° update queue weight
    end for
end procedure
```

System related congestion indicatives calculus: flows weights and reaction points weights

```
procedure QCN-WFQR-QUERY-INDICATIVES(cpid)      ° query each CPID for changes in flow table
    for "each flow f" ∈ CPID do                 ° for each flow received
        call QCN-WFQR-UPDATE-SYS-TABLE(flow)    ° if indicatives changed update sys table
    end for
end procedure

procedure QCN-WFQR-UPDATE-SYS-TABLE(flow)       ° update sys table (Table 11)
    call QCN-WFQR-UPDATE-SHARE(flow)            ° update flow's share
    call QCN-WFQR-UPDATE-SHARE-WEIGHT(flow)     ° update flow's weight
    call QCN-WFQR-UPDATE-RP-WEIGHT(flow)        ° update reaction point weight
end procedure
```

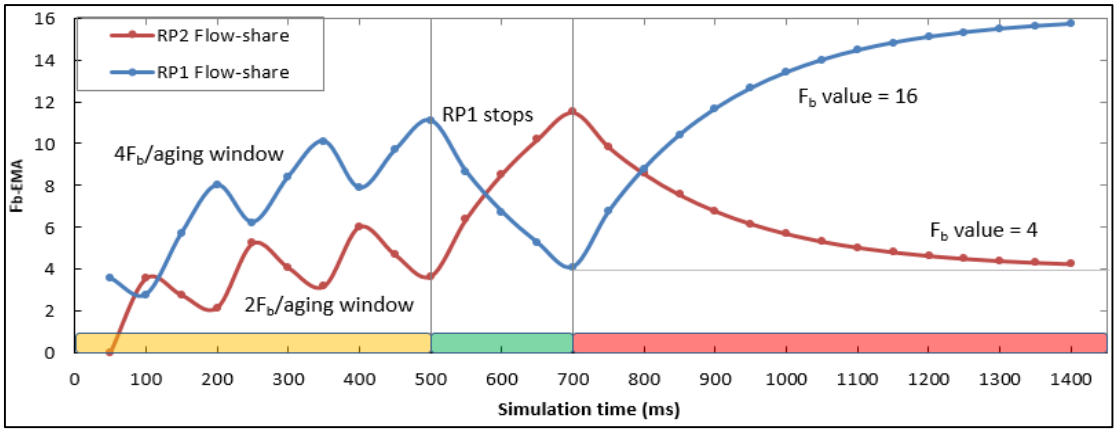


Figure 31: Flow share synthetic tests

6.3.4 Algorithm analysis

6.3.4.1 Flows shares analysis

The main idea of flow share is that it has to reflect the reaction point rate and in the same time to reflect the feedback values. I assumed that the reaction point rate is reflected by the frequency with which the feedback is sent to a specific flow, while the distance until congestion is outlined by the feedback value, so I distinguish two different situations considering two flows (f_1 and f_2), as follows:

1. $$\left.\begin{matrix} Rate(f_1) > Rate(f_2) \\ EMA_{F_b}(f_1) = EMA_{F_b}(f_2) \end{matrix}\right\} \xrightarrow{implies} Share(f_1) > Share(f_2)$$

2. $$\left.\begin{matrix} Rate(f_1) = Rate(f_2) \\ EMA_{F_b}(f_1) > EMA_{F_b}(f_2) \end{matrix}\right\} \xrightarrow{implies} Share(f_1) > Share(f_2)$$

Basically, both flow rates and feedback values are reflected in the flow share, therefore former situation enables comparison of flows that shares the same queue (same congestion conditions) but with different rates, while the later situation enables comparison of flows having same rate but on different routes and different queues with different congestion conditions (i.e. different feedback values).

For synthetic simulation I chose an aging window of 250ms and constant sampling rate of 50ms. In the first 500ms of simulation time, the first situation is emphasized by generating

feedbacks of the same value, but one flow receives twice as many feedbacks than the other so, the rate difference is reflected by the flow share as is shown in Figure 31.

Then, for the next 200ms first flow stops and its flow share is aging, while the flow share for the second flow grows. Furthermore, starting from 700ms second situation is demonstrated by sending a feedback value of 16 and 4 respectively at every 50ms. In Figure 31 it can be seen that the flow share of first flow is growing, while flow share for second flow is decreasing thus the feedback values are reflected. In conclusion the flow share responds for both rate and feedback variation.

6.3.4.2 Congestion indicatives analysis

At a particular moment a flow cannot have more than one congestion point, because a congestion point implies the smallest throughput within a flow path. But, if a different point is congested due to profile traffic changes, then the throughput must decrease even more, therefore the old congestion point will be freed (the distance until congestion grows).

In terms of flow comparison, I distinguish three different situations as follows:

1. **Solitary congestion islands**: the compared flows shares the same queue (congest the same point);
2. **Disconnected congestion islands:** the compared flows congest different queues (different congestion points);
3. **Hybrid congestion islands:** the compared flows are in solitary congestion island, but occasionally different flows may cause other congestion points and move flows into disconnect congestion islands or vice-versa (within measuring time frame T).

In case of first situation, the flow with the lowest throughput will have the lowest flow share and the flow weight will have only one component. In case of second situation, the compared flows will have different shares according to all three parameters (rate, queue ranks and feedback values) related to their congestion points. The last situation is a composition of the first two and the congestion indicatives are sums.

6.3.4.3 QCN-WFQR simulator

For simulations was used an open-source simulator: NS-3 (a discrete-event network simulator) [53]. The QCN-WFQR implementation has the following main objects: reaction point, congestion point and profiler or controller in case of SDN (Figure 32).

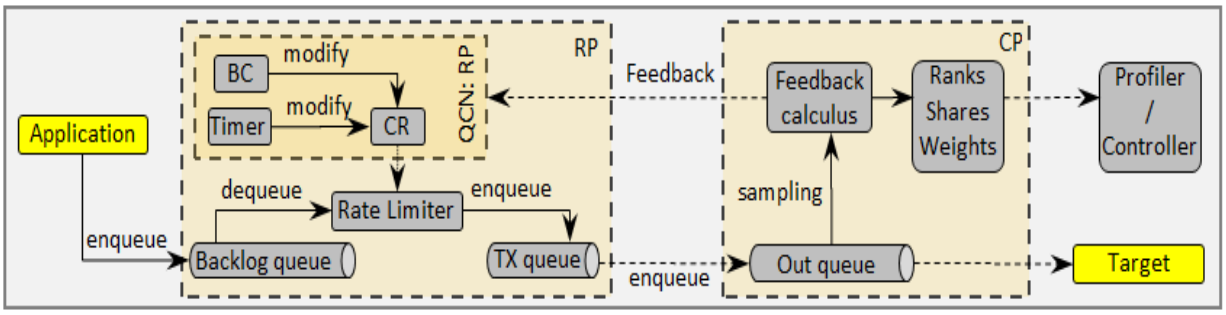


Figure 32: NS3-QCN-WFQR high-level view

Congestion Node		Reaction Point	
Queue Len.	150KB	**Byte Counter threshold**	150KB
Queue Eq.	33KB	**Timer threshold**	10ms
ω	2	**R-AI**	0.5Mbps
Share age window	2000ms	**R-HAI**	5Mbps
Table age window	250ms	**MIN Rate**	1Mbps
RTT around 50us, Link rate at 1Gbps			

Table 12: QCN-WFQR parameters

The simulation sequence starts at the application side, where the frames are enqueued in a huge *backlog queue* (thus, the starvation is avoided). The rate limiter relying on current rates moves the frames from the backlog to the TX queue and further the TX queue is dequeued based on link rate. The CR is modified by the BC and Timer according to received feedbacks and QCN parameters. Furthermore, CP samples the outgoing queue and computes feedbacks, based on which local congestion indicatives are computed. The profiler and the SDN controller interrogates the topology nodes and compute different system congestion indicatives.

Support for a centralized profiler/controller that simulates a “statistics service” was added to NS-3. The feedback messages paths were left unmodified to avoid overloading the profiler/controller (i.e. it is not aware of the feedbacks and decisions to limit the rate are taken by the switches themselves). In SDN context this may be considered a small deviation from the standard definition of a centralized control plane but, given the high rate of feedbacks generated by a congested port, it is an acceptable trade off.

6.3.4.4 QCN-WFQR simulations analysis

6.3.4.4.1 Flows contributions to system congestion profile

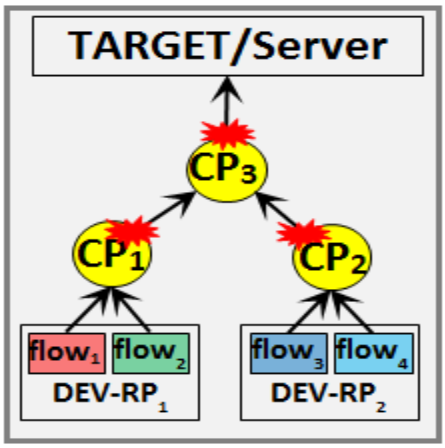
This first test aims to emphasize the flows contributions to system congestion profile: flow shares, flow weights and reaction point weights. In another words, reaction points with low flows rates has lower weights, therefore tenants can cooperate and decide upon different services locations and achieve a fair congestion profile and a better system performance (see example from section 6.3.5.1).

For simulation, a binary tree topology was used with three congestion controlled nodes (Figure 33 – a). Each child node has a device with a reaction point and two flows each, while the parent node has a device with a server (as target). Reaction points inject traffic as UDP clients, while the target consumes it as UDP server. From QCN stand point, each congestion controlled node has a transmission queue used for feedback calculus, using parameters listed in Table 12. From QCN-WFQR stand point, flow share calculus uses a window of 2 seconds, but the flow table has an age function of 250ms, so if a flow doesn't get any feedback for 250ms its share is reset (Table 12).

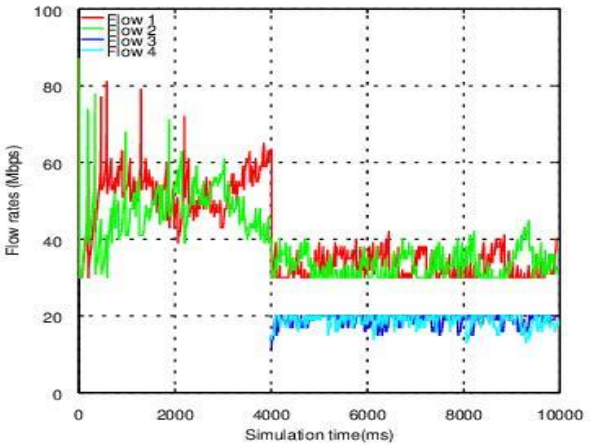
The simulation starts with RP_1 flows and five seconds later the second device begins to inject traffic. The congestion profile is heavily influenced by the fact that flows 1 and 2 have higher throughput capabilities, therefore they have a higher probability to receive feedbacks (rates are displayed in Figure 33 – b).

In the first five seconds, flows from RP_1 transmit at a much higher rates, while CP_3's queue usage is kept around equilibrium (Figure 33 – c). After that, flows from the second device start injecting traffic and CP_3's queue is pushing rate limiters throughput lower (and flow 1 and 2 rate decreases significantly). The other congestion points never get congested, because CP_3 push the throughput low enough and their queues stays below equilibrium the entire simulation time. And since only CP_3 gets congested, flows weights are the same as flows shares reported by CP_3 (Figure 33 – d, e). Also, as it can be seen from Figure 33 – (b) and (e), flow weights follows relatively same profile as their rates.

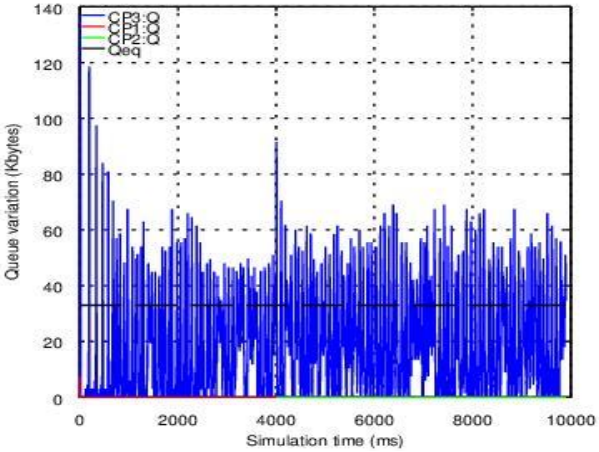
From device weights stand point, RP_1 flows ($flow_{1\&2}$) has higher throughput capabilities, then it has a higher probability to receive feedbacks, so RP_1 weight is higher than RP_2 weight (Figure 33 – f).



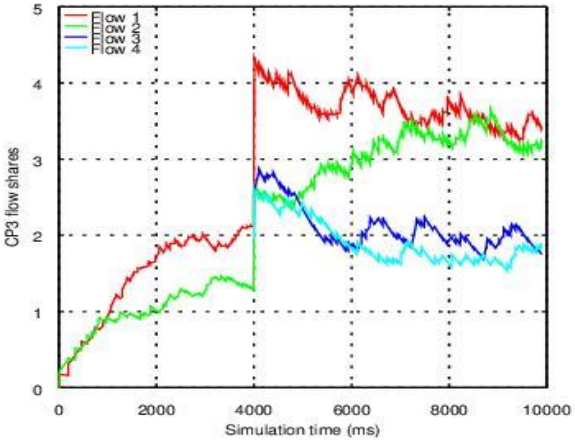
(a) Simulation topology



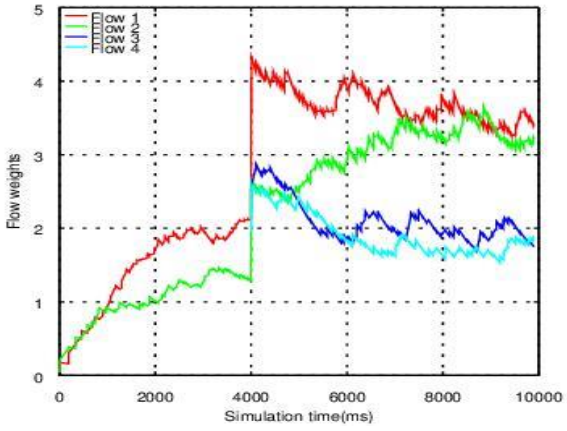
(b) Flows rates variation



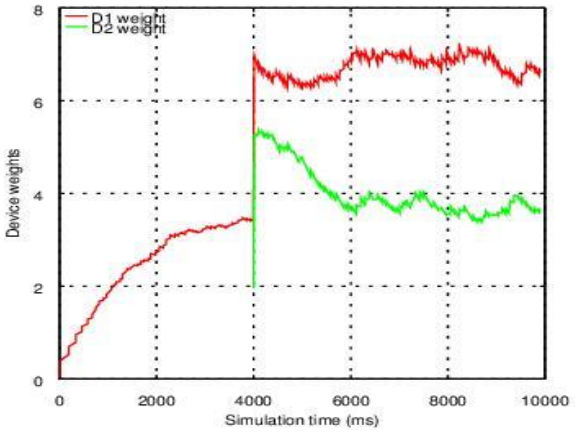
(c) Congestion points queue variation



(d) CP_3 local flow shares variation

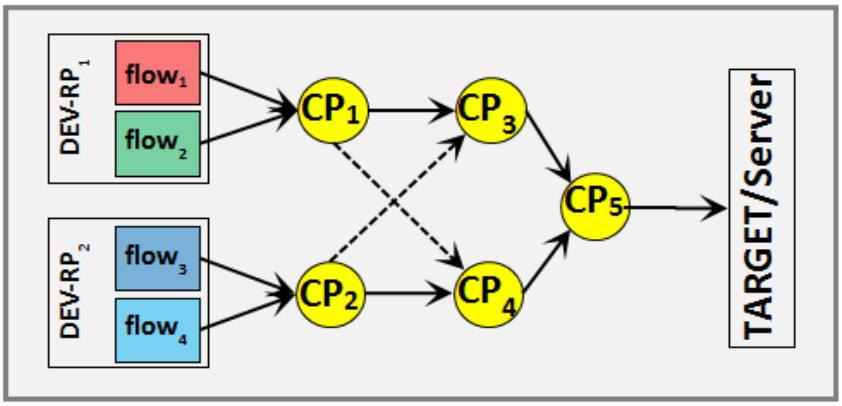


(e) Flow weights variation

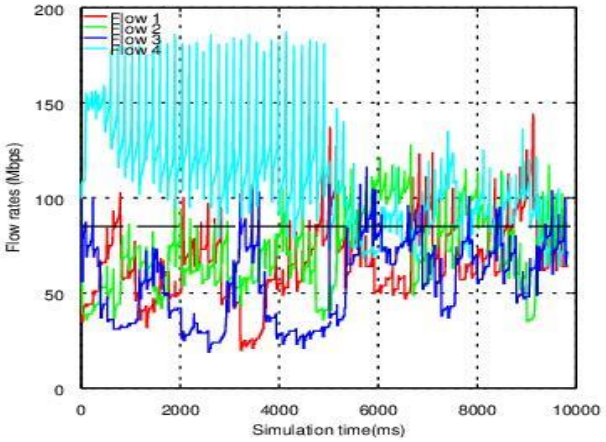


(f) Reaction points (devices) weights variation

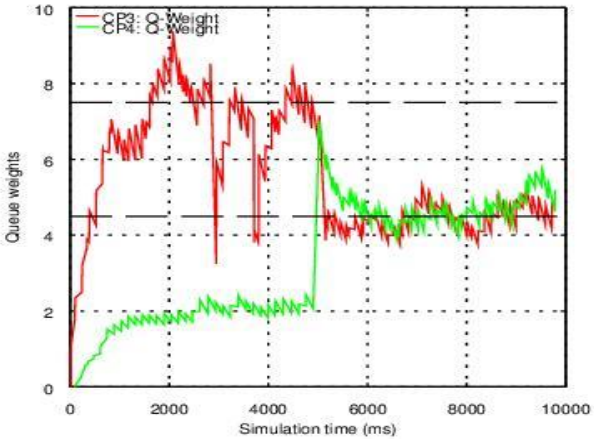
Figure 33: QCN-WFQR (NS3): simulation results for variation of flow related indicatives



(a) Simulation topology



(b) Flow rates variations



(c) CP3, CP4 queues weight variation

Figure 34: QCN-WFQR (NS3): simulation results for variation of queue related indicatives

6.3.4.4.2 Queues weights variations

The second simulation aims to emphasize that flows rates variations reflects queues weights variations, thus migration decisions may be taken to balance traffic and achieve an uniformed congestion profile among system congestion points (see example from section 6.3.5.2).

For simulation, a tree based topology was used with alternative paths, so each of the four device's flows has two alternative paths. In the first five seconds of simulation time, the congestion point CP_3 forwards and samples three flows (flow 1, 2 and 3), while congestion point CP_4 forwards and samples one flow (flow 4). It can be seen from the Figure 34 – (c) that the CP_3's weight queue is significantly higher compared to CP_4's weight queue, which controls only one flow.

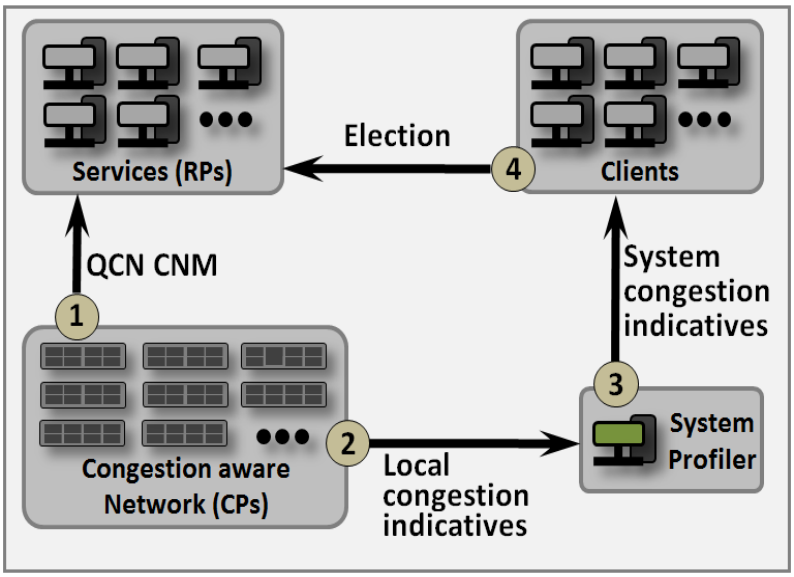


Figure 35: Elections in distributed systems with homogenous nodes

Furthermore, flow 1 is moved from CP_3 to CP_4 by the profiler/controller, which chooses an alternative path based on queues weights. The Figure 34 – (c) shows that the weight queues for the two congestion point converge, also the rates for all four flows converge as well (depicted in Figure 34 – b) achieving fair rates and a better balanced network load.

6.3.5 QCN-WFQR applications

6.3.5.1 Elections in distributed systems with homogenous nodes

In a distributed system (or cloud) with multiple classes of services, where each class has many homogenous nodes, I propose a method of chosing a particular service based in QCN-WFQR algorithm. The method aims to maximize bandwidth usage, while minimize congestion in the network and improve overall system response time (Figure 35).

If the number of clients varies and their needs are unknown, then the network congestion pattern is basically nondeterministic. When a particular service is required, I propose that the choice of a node to be based on congestion indicatives provided by QCN-WFQR system profiler.

For exemplification, in distributed and parallel file systems, a file is divided in data chunks (or objects) replicated on multiple storage devices, so a data segment can be transferred from any replica. Most distributed file systems uses a static distribution (e.g. CRUSH – Controlled Replica Under Scalable Hashing), but with advent of converged networks a static

distribution may not be the best choice. Therefore, using QCN-WFQR the replica may be chosen dynamically and improve system responsiveness by trying to avoid network bottlenecks. In this case is worth to mention that the reaction points are the storage devices.

The heart of this method is that the local congestion indicatives communicated to system profiler and the congestion indicatives communicated to the client must reflect the current system congestion status. So, if the information is communicated too often, then the system profiler may be overwhelmed or if it is too seldom, then the information will not reflect the current congestion status of the system. This means that the algorithm is very sensitive to wild congestion profile variations, so I propose to use larger data chunks.

Besides the communication of congestion indicatives, the performance and implementation of the system profiler (i.e. distributed implementation but logically centralized or completely centralized) greatly influence the system behavior. Plus, the profiler must have a very low latency connection with each congestion point and each client, so that the congestion indicatives to be exchanged in timely fashion.

I consider the following 'election' main steps (Figure 35):

- **Step-1**: QCN standard flow (feedbacks from congestion aware network – CPs towards services nodes – RPs). CPs computes local congestion indicatives (i.e. flows shares, queues ranks and queue weights).
- **Step-2**: The system profiler registers received local congestion indicatives and furthermore it computes the system congestion indicatives.
- **Step-3**: The client asks a subset of congestion indicatives from system profiler.
- **Step-4**: The client choose a service location based on received congestion indicatives (basically, the entity with the smallest indicative).

6.3.5.2 Dynamically balanced flows in computer networks

The algorithm performance is directly influenced by the latency with which the Network Profiler or SDN controller gets the relevant congestion indicatives, decides and move flows while achieving a better balanced traffic within the system topology.

The profiler/controller design follows either a centralized or distributed model [54]. Each method has its own pros and cons. For example, a centralized design has scaling limitation, bigger bottleneck probability, single point of failure, but it has a simpler complexity (basically it follows a multi-threaded design over SMP systems) and it has strong semantic consistency, while a decentralized design scales up easy and meet performance requirements, handle better data plane resilience and scalability, fault tolerant, but it has a weak consistency semantics (and it is worth to mention that a strong consistency implies complicated

implementation of synchronization algorithms) and it has much more complex implementation [54].

How congestion indicatives are communicated is strategic to relevance of the decisions based on QCN-WFQR algorithm. Basically, it has to either transmit information at short intervals overloading the controller or to filter and condense the information at the source before transmitting it to the controller.

7 CONCLUSIONS AND FUTURE WORK

This thesis begins with a general analysis of the following distributed storage systems: AFS, GFS, GPFS, Lustre and Ceph, studding different aspects such as taxonomies, architecture details and network characteristics. The analysis is followed by a proposal for eLearning infrastructures, as a case study. Afterwards, several optimizations to distributed storage systems are proposed and a novel algorithm for dynamic load balancing in congested aware networks: **QCN-WFQR**.

7.1 The original contributions of the thesis

Contributions to the analysis of distributed storage systems (chapter 2)

I proposed a simple and comprehensive taxonomy for storage systems considering four important characteristics: locality, sharing, distribution level and semantics (section 2.1). For characterization of distributed storage systems I proposed a relevant set of requirements based upon existing documentation in the field (section 2.2). Also, I highlighted the scale-out methods at different layers of file systems, thus building different types of storage systems, such as SAN or NAS (section 2.3). And to emphasize the similarities between distributed and centralized storage systems I proposed a generalized layout (section 2.5.3).

The analysis of distributed storage systems and related contributions were published in **[PIST, 2014/1]**.

Contributions to the infrastructure for eLearning environments (chapter 3)

In this chapter I proposed a parallel and distributed file system adapted for eLearning environments, in turn hosted by cloud systems. The proposed storage system can be considered an alternative solution to OGSA-GFS. It has two main components: a flat address space based on RADOS (Ceph's object store) and a module for managing the semantic aware metadata. I proposed a native implementation of the semantic aware metadata with content-based semantics to improve the searches of data in the system. In terms of network, I proposed a hierarchical topology that facilitates south-north throughput with links added to decrease the latency between RADOS's nodes.

These contributions were published in **[PIST, 2014/2]**.

Optimizations based on packet processing engines (chapter 4)

The main contribution in this chapter is the proposal of several solutions to optimize traffic intensive clusters using integrated systems of multi-general purpose cores with packet

processing engines (section 4.2 and 4.3). While Ceph addresses different optimizations at system level, my contributions focus on improving each node performance which in the end reflects the overall system performance (section 4.5).

First, the performance of each node is improved by classifying and distributing requests among cores, facilitating the parallelization of traffic consumers. Furthermore, the OS scheduler decisions are leveraged by using Accelerated Receive Flow Steering mechanisms (section 4.4). Therefore, I proposed two models and two different implementations for each model: one in kernel targeting efficiency and one in user-space facilitating a faster development and flexible license policy. As case study, I created a model using QorIQ™ platforms (section 4.5.1) for Ceph's nodes. Each model for each node type is designed based on related flow types and traffic volume (section 4.5.2).

Second, the latency of sensitive operations is decreased and the overall cluster responsiveness and availability is improved by adjusting the weights of hardware queues. Accelerated RADOS (A-RADOS) divides the traffic of monitors and OSDs in three different classes with different priorities based on importance related to Ceph (section 4.5.3).

Performance measurements that showed improved number of transactions per second and the results of the RADOS's messages processed based on priorities were presented in section 4.5.4. Besides the presented improvements there are inherent advantages of using multi-core integrated systems such as low power consumption and appealing price per performance.

The proposed optimizations based on packet processing engines and the micro-benchmark results were published in **[PIST, 2013]**.

Solutions for issues occurred in converged infrastructure for data centers (chapter 5)

A group of protocols were added to Ethernet in context of converged infrastructure for data centers (i.e. collapsing tiers and unification of general purpose networks with networks for distributed storage systems): PFC, ETS, DCBx and QCN (section 5.2). The main focus was on QCN, which has several weaknesses, such as unwanted traffic injections for QoS domains from edge ports and rate unfairness of flows that shares the same bottleneck (section 5.2).

I was part of a team that proposed a patented solution for automatic configurations of QCN parameters through LLDP in a hybrid network, where segments with and without QCN coexists considering different QoS requirements. The solution implies a new hybrid defense mode choice that allows both manual and automatic settings to use the same alternate priority value, thus the injection of unwanted traffic for a specific QoS domain is avoided (section 5.3).

The patent is public and can be inspected at **[Patent #8891376]**.

Proposed algorithm for dynamic load balancing: QCN-WFQR (chapter 6)

QCN protocol alleviate congestion in the network, but it does not address the congestion problem as a whole, where a uniform distribution of congestion domains will lead to a better utilization of bandwidth. Therefore, I proposed a partial solution that tries to solve this issue. The proposal was based on the choice of the gateway with the biggest distance until congestion and later on, it led to a more general solution that was patented, where the congestion sources (i.e. reaction points) are building databases for each congested point. This database can be used for performance analysis or can be used to influence different aspects of the network devices (section 6.2).

I have followed a different path and I proposed a **novel** and more **complete** algorithm (**Quantized Congestion Notification – Weighted Flow Queue Ranking)** that tries to solve this problem. The algorithm **QCN-WFQR** computes multiple congestion indicatives that are measures of the network load generated by flows in different points in the network, based on which different cooperative or automatic decisions may be taken to balance the workload and achieve a less congested network (section 6.3.2).

The algorithm is capable to compute the local contribution of each flow in different network points ($\boldsymbol{Share_t^{q_k}(f_i)}$) and also the system wide contribution of each flow ($\boldsymbol{Weight_t(f_k)}$). Besides flow contribution, the system is also capable to compute the contribution of each congestion source ($\boldsymbol{Weight_t(R_k)}$) to the network load. Based on these congestion indicatives any component within the system can determine the best appropriate *service* (congestion source/reaction point) location in order to cooperatively balance the workload in the network (sections 6.3.1, 6.3.2 and 6.3.3). Furthermore, I present the analysis of *flows shares* and the system wide congestion geometry (i.e. congestion islands) related to each flow (sections 6.3.4.1 and 6.3.4.2). It is worth to mention, that the QCN-WFQR algorithm makes use of the exponential moving average to store flows shares in order to minimize the memory usage on network nodes (section 6.3.2.1).

Along with the congestion indicatives listed above, the algorithm is capable to compute the importance of each queue ($\boldsymbol{R_t^{q_k}}$) related to the congestion in the system in terms of the number of handled flows and the congestion degree of each queue ($\boldsymbol{Weight_t(q_k)}$) related to all handled flows (the definition of a topology is detailed in sections 6.3.1 and 6.3.2). Using these congestion indicatives, a network can be reprogrammed to optimally balance the workload between different congestion points.

For simulations purpose I implemented QCN-WFQR in NS3 (an open source discrete-event network simulator), where besides algorithm modules, I implemented also the following QCN modules: rate limiter, reaction point and congestion point (section 6.3.4.3). I did two

different simulations aiming flows contributions to the congestion profile and variations of queues weights using tree based topologies (section 6.3.4.4). The former simulation revealed that the traffic of congestion sources is reflected in the congestion indicatives of flows and reaction points (section 6.3.4.4.1), while the later simulation revealed the reflection in the congestion indicatives of queues (section 6.3.4.4.2).

I proposed two applications of QCN-WFQR: elections of replicas in distributed storage systems based on congestion indicatives in order to achieve a better balanced congestion profile while achieving a much responsive system and a dynamically load balancer, more appropriate for SDN networks.

The patented solution can be inspected at **[App. #20150023172]**, while the QCN-WFQR algorithm with the simulations results were published in **[PIST, 2015]**.

7.2 Future work

Until now, nodes were optimized only at ingress side, therefore now I am searching for optimization methods at egress side as well. Besides classifications and distributions of flows to achieve better performances, the usage of other hardware engines is taken into consideration, such as cryptographer or TCP engines. Also, because only micro-benchmarks were done, all optimizations will be included into a wider system and tests will be conducted to see how the performance improvements at each node are reflected in a wider system with a larger number of nodes.

The QCN-WFQR algorithm has a sound approach with promising impact in the networking field, especially in Software Defined Networking. Therefore now the research focuses on implementation of different algorithms, such as Dijkstra's algorithm, using different congestion indicatives as path costs proposed in the present thesis. Since the distributed systems has several characteristics related to scaling methods, investigations are directed towards implementation of logically decentralized controllers with QCN-WFQR support and the relevance of decisions based on latency and architectures of network topologies.

The QCN-WFQR algorithm can be used to select locations of different services in more complex systems (such as clouds) and improve responsiveness, stability and I/O performance of the systems. Therefore, analyzing the relevance of QCN-WFQR is required for different traffic profiles (such as bursts and streaming) to determine the services for which this algorithm is adequate.

8 AUTHOR'S PUBLICATIONS

8.1 Patents

[App. #20150023172]. **Sorin A. Pistirica,** Dan A. Calavrezo, Casimer M. DeCusatis, Keshav G. Kamble, "**Congestion Profiling of Computer Network Devices**", Patent Pending,
USPTO: http://patents.justia.com/patent/20150023172, Jul 16 – 2013

[Patent #8891376]. **Sorin A. Pistirica**, Dan A. Calavrezo, Keshav G. Kamble, Mihail-Liviu Manolachi, "**Quantized Congestion Notification—defense mode choice extension for the alternate priority of congestion points**",
USPTO: http://patents.justia.com/patent/8891376, Oct 07 – 2013

8.2 Articles

[PIST, 2013] **Pistirica Sorin Andrei**, Caraman Mihai Claudiu, Moldoveanu Florica, Moldoveanu Alin, Asavei Victor, "**Hardware acceleration in CEPH Distributed File System**", ISPDC: IEEE 12th International Symposium on Parallel and Distributed Computing, Bucharest, June 2013, pp. 209-215, **IEEE Indexed**

[PIST, 2014/1] **Pistirica Sorin Andrei,** Victor Asavei, Horia Geanta , Florica Moldoveanu , Alin Moldoveanu , Catalin Negru , Mariana Mocanu, "**Evolution Towards Distributed Storage in a Nutshell**", HPCC: The 16th IEEE International Conference on High Performance Computing and Communications, August 2014, Paris, pp. 1267-1274, **IEEE Indexed**

[PIST, 2014/2] **Pistirica Sorin Andrei**, Asavei Victor, Egner Alexandru, Poncea Ovidiu Mihai, "**Impact of Distributed File Systems and Computer Network Technologies in eLearning environments**", eLSE: Proceedings of the 10th International Scientific Conference "eLearning and Software for Education", Bucharest, April 2014, Volume 1, pp. 85-92, **ISI Indexed**

[PIST, 2015] **Pistirica Sorin Andrei**, Poncea Ovidiu, Caraman Mihai, "**QCN based dynamically load balancing: QCN Weighted Flow Queue Ranking**", CSCS: The 20th International Conference on Control Systems and Computer Science, Bucharest, May 2015, Volume 1, pp. 197-205, **ISI Indexed**

[ASAV, 2014] Asavei Victor, Moldoveanu Alin, Moldoveanu Florica, **Pistirica Sorin Andrei**, "**Lightweight 3D MMO Framework with High GPU Offloading**", ICSTCC: 18th

International Conference On System Theory, Control and Computing, October 2014, Sinaia, pp. 708-714, **ISI Indexed**

[SIMI, 2015] Simion Andrei, Asavei Victor, **Pistirica Sorin Andrei**, Poncea Ovidiu, "**Practical GPU and voxel-based indirect illumination for real time computer games**", CSCS: The 20th International Conference on Control Systems and Computer Science, May 2015, Bucharest, Volume 1, pp. 379-384, **ISI Indexed**

[GRAD, 2015] Alexandru Grădinaru, Alin Moldoveanu, Victor Asavei, **Sorin Andrei Pistirica**, "**Case Study - OpenSimulator for 3D MMO Education**", eLSE: Proceedings of the 11th International Scientific Conference "eLearning and Software for Education", Bucharest, April 2015, **ISI indexed**

[ASAV, 2015] Victor Asavei, Alexandru Gradinaru, Alin Moldoveanu, Sorin Andrei Pistirica, Ovidiu Poncea, Alexandru Butean, "**Massively Multiplayer Online virtual spaces- classification, technologies and trends**", U.P.B. Sci. Bull., 2015, (in press, accepted for publication)

9 BIBLIOGRAPHY

[1] B-tree File system, https://btrfs.wiki.kernel.org/index.php/Btrfs_design

[2] Extended File System, http://e2fsprogs.sourceforge.net/ext2intro.html

[3] EMC Storage Pool Deep Dive: Design Considerations & Caveats, http://vjswami.com/2011/03/05/emc-storage-pool-deep-dive-design-considerations-caveats/, 2011

[4] Linux Volume Management, http://tldp.org/HOWTO/LVM-HOWTO/

[5] G.C.Foxy, K.A.Hawick, A.B.White, "Characteristics of HPC Scientific and Engineering Applications", January 1996

[6] Chang-Soo Kim, Gyoung-Bae Kim, Bum-Joo Shin, "Volume Management in SAN Environment", Parallel and Distributed Systems ICPADS, 2001, pp. 500-505

[7] Andrew S. Tanenbaum, "Distributed Operating Systems", August 25th 1994 by Prentice Hall

[8] Christian Bandulet, "The Evolution of File Systems", http://www.snia-europe.org/objects_store/Christian_Bandulet_SNIATutorial%20Basics_EvolutionFileSystems.pdf, Storage Networking Industry Association, 2012

[9] Michael Factor, Kalman Meth, Dalit Naor, Julian Satran, Ohad Rodeh, "Object Storage: The Future Building Block for Storage Systems", Local to Global Data Interoperability - Challenges and Technologies. IEEE Computer Society. pp. 119–123, 2005

[10] Information Technology - SCSI Object Based Storage Device Commands (OSD), Revision 3, 1 October 2000

[11] Information Technology - SCSI Object Based Storage Device Commands -2 (OSD-2), Revision 4, 24 July 2008

[12] J. Satran A., Teperman, "Object Store Based SAN File Systems", International Symposium of Santa Caterina on Challenges in the Internet and Interdisciplinary Research, 2004

[13] OpenAFS, Open source implementation of the Andrew Distributed File System, http://www.openafs.org/

[14] ARLA, Free AFS implementation from KTH, http://www.stacken.kth.se/project/arla/html/arla.html

[15] Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung, The Google File System , 19th ACM Symposium on Operating Systems principles, pp. 29-43, December 2003

[16] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System", 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, pp. 1-10

[17] Frank Schmuck and Roger Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters", Proceedings of the Conference on File and Storage Technologies, pp. 231–244, January 2002

[18] Peter J. Braam et al, The Lustre Storage Architecture, available at www.lustre.org, 2004.

[19] Sage A. Weil, Scott A. Brandt, Ethan L. Miller and Darrell D. E. Long, —Ceph: A Scalable, High-performance Distributed File System , 7th Conference on Operating Systems Design and Implementation, November 2006

[20] Sage A. Weil, Andrew W. Leung, Scott A. Brandt and Carlos Maltzahn, —RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters, Petascale Data Storage Workshop, November 2007

[21] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Carlos Maltzahn, "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data", Proceedings of the 2006 ACM/IEEE conference on Supercomputing, 2006

[22] Sage A. Weil, Reliable, Scalable and High-Performance Distributed Storage, PhD thesis, 2007

[23] Honicky, R.J. ,Miller, E.L., "RUSH: Balanced, Decentralized Distribution for Replicated Data in Scalable Storage Clusters", Proceedings of the 20th IEEE - 11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003, pages 146–156

[24] R. J. Honicky, Ethan L. Miller, "Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution", Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004
[25] Andrew W. Leung, Ethan L. Miller, Stephanie Jones, "Scalable Security for Petascale Parallel File Systems", Proceedings of the 2007 ACM/IEEE conference on Supercomputing, 2007
[26] Yan Li, Nakul Sanjay Dhotre, Yasuhiro Ohara, Thomas M. Kroeger, Ethan L. Miller, Darrell D. E. Long, "Horus: Fine-Grained Encryption-Based Security for Large-Scale Storage", Proceedings of the sixth workshop on Parallel Data Storage, 2013, pp. 19-24
[27] PanFS, Object RAID, https://www.panasas.com/products/panfs/object-raid
[28] Goodman, J.R.,Sequin," Hypertree: A Multiprocessor Interconnection Topology", IEEE Transactions on Computers, 1981, pp. 923-933
[29] Direct interconnection networks, http://pages.cs.wisc.edu/~tvrdik/5/html/Section5.html
[30] Andy D. Hospodor, Ethan L. Miller, "Interconnection Architectures for Petabyte-Scale High-Performance Storage Systems", 12th NASA Goddard Conference on Mass Storage Systems and Technologies, April 2004
[31] White Paper, Accelerating High-Speed Networking with Intel® I/O Acceleration Technology
[32] Karthikeyan Vaidyanathan, Dhabaleswar K. Panda, "Benefits of I/O Acceleration Technology (I/OAT) in Clusters", International Symposium on Performance Analysis of Systems & Software, pp. 220-229, 2007
[33] Scaling in the Linux Networking Stack, https://github.com/torvalds/linux/blob/master/Documentation/networking/scaling.txt, December 2011
[34] Luigi Rizzo, "netmap: a novel framework for fast packet I/O", Proceedings of the 2012 USENIX Annual Technical Conference, pp. 101-112, June 2012
[35] Freescale Semiconductor, Inc., —QorIQ Data Path Acceleration Architecture (DPAA) Reference Manual , 2011
[36] Freescale Semiconductor, Inc., —Frame Manager Configuration Tool Example Configuration and Policy, 2013
[37] Fulvio Risso, and Mario Baldi, NetPDL: An Extensible XML-based Language for Packet Header Description, Computer Networks (COMPUT NETW), vol. 50, no. 5, pp. 688-706, 2006
[38] Sangam Racherla, Silvio Erdenberger, Harish Rajagopal, Kai Ruth, "IBM Red Book, Storage and Network Convergence Using FCoE and iSCSI", International Technical Support Organization: Storage and Network Convergence Using FCoE and iSCSI, January 2014
[39] White Paper, Emulex, Top-5 Reasons for Deploying Network Convergence, 2009
[40] White Paper, Forrester Consulting, "Benefits Of SAN/LAN Convergence. Evaluating Interest In And Readiness For Unified Fabric", 2009
[41] White Paper, "Fabric convergence with lossless Ethernet and Fibre Channel over Ethernet", 2008
[42] White Paper, Ethernet Alliance, Data Center Bridging, 2010
[43] Devkota P., Reddy A.L.N., "Performance of Quantized Congestion Notification in TCP Incast Scenarios of Data Centers", Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2010
[44] IEEE Standard 802.3x PAUSE, 1997
[45] 802.1Qbb PFC – Priority-based Flow Control, 802.1Qbb Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment 17, 2010
[46] 802.1Qaz ETS – Enhanced Transmissions Protocol, 802.1az Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment 18,2011
[47] Manoj Wadekar (Qlogic), et al, DCB Capability Exchange Protocol Base Specification Rev 1.01
[48] 802.1Qau QCN – Quantized Congestion Notification, , 802.1Qau Media Access Control (MAC)

Bridges and Virtual Bridged Local Area Networks Amendment 13,2010
[49] IEEE Standard for Local and metropolitan area networks—Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks, 31 August 2011
[50] 802.1AB Station and Media Access Control Connectivity Discovery, IEEE Standard for Local and metropolitan area networks, 2009
[51] Abdul Kabbani, Mohammad Alizadeh, Masato Yasuda, Rong Panz and Balaji Prabhakar, AF-QCN: Approximate Fairness with Quantized Congestion Notification for Multi-tenanted Data Centers, Proceedings of the 18th IEEE Symposium on High Performance Interconnects, pp. 58-65, 2010
[52] Yan Zhang, Nirwan Ansari, Fair Quantized Congestion Notification in Data Center Networks, IEEE Transactions on Communications, pp. 4690 - 4699, 28 November 2013
[53] Discrete Network Simulator, http://www.nsnam.org/
[54] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky and Steve Uhlig, Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE Vol. 103, No. 1, January 2015
[55] Fibre Channel Backbone – 5 (FC-BB-5), INCITS working draft proposed American National Standard for Information Technology, Rev. 2.0, June 4, 2009
[56] Shudayfat Eman Ahmad, Moldoveanu Alin, Moldoveanu Florica, Gradinaru Alexandru. Virtual Reality-based Biology Learning Module, 9th International Conference eLearning and Software for Education. Carol I Natl Defence Univ Publishing House, vol. 2, ISSN 2066-026X, pp. 621 --626, April 2013
[57] Venu Vasudevan, Paul Pazandak, Semantic File System Survey, http://www.objs.com/survey/OFSExt.htm, 1996
[58] Margo Seltzer, Nicholas Murphy, “Hierarchical File Systems are Dead”, USENIX HOTOS, May 2009
[59] Ip.com, IPCOM000230977D, A Method and System for Storage Server Selection based on a Network Congestion Status, September 20, 2013
[60] SungHo Chin, JongHyuk Lee, HwaMin Lee, DaeWon Lee, HeonChang Yu, Pillwoo Lee, “OGSA-GFS : A OGSA based Grid File System”, Proceedings of the First International Conference on Semantics, Knowledge, and Grid (SKG 2005), 2006
[61] OGSA-DAI (Open Grid Services Architecture-Data Access and IntegrationHung Ba Ngo, Christian Bac, Frederique Silber-Chaussumier, Thang Quyet Le, “Towards Ontology-based Semantic File Systems”, 2007 IEEE International Conference on Research, Innovation and Vision for the Future, pp. 8-13

10 ACRONYMS AND DEFINITIONS

DAS	**Direct Attached Storage:** refers to a storage system directly attached to the machine accessing the data.
SAN	**Storage Area Network**: refers to a dedicated network that provides block level access to storage systems.
NAS	**Network Attached Storage**: refers to a storage system that provides file level access through a computer network.
SCSI	**Small Computer System Interface**: a set of interface standards for communication with peripheral hardware, such as: printers, storage devices and so on.
SAS	**Serial Attached SCSI:** a new serial protocol that moves data to and from storage devices.
sATA	**Serial ATA**: a computer bus interface that connects host bus adapters to mass storage devices.
Cloud Computing	Complex system for hosting service based applications in a networked system invisible for clients.
HPC	**High Performance Computing**: high speed computing system (nanoseconds based calculus).
Data block	Smallest data structure of an uninterpreted sequence of bytes or bits, having a well-defined length, managed by a storage device unit.
Data file	Self-contained information organized in a sequence of bytes and metadata managed by the overlying Operating System.
Data object	A data structure composed by a sequence of bytes or bits of flexible size and a variable amount of metadata that describes it.
inode	**Index node:** metadata information about a stored element in file systems implemented in *nix operating systems.
FC	**Fibre Channel**: high speed network technology used to connect storage devices

FCP	**Fibre Channel Protocol**: transport protocol predominantly used for SCSI commands over FC networks.
RAID	**RAID–Redundant Array of Independent Disks:** a system of multiple storage devices combined into a single logical unit for redundancy and performance improvement.
LUN	**Logical Unit Number:** a logical number used to identify the device addressed through SCSI protocol.
LBA	**Logical Block Addressing:** a method of locating data blocks, typically a linear scheme, on computer storage devices.
POSIX	**Portable Operating System Interface:** a set of standard operating system interfaces based on UNIX OS.
OSD	**Object Storage Device:** device that implements the standard (i.e. an extension to SCSI) in which data is organized and accessed as objects.
COW	**Copy on write:** optimization strategy that provides pointers to resources to callers, until the callers change or copy it.
LAN	**Local Area Network**: computer network that interconnects nodes in a limited area.
IP	**Internet Protocol**: communication protocol where a node is identified by an IP address.
TCP	**Transmission Control Protocol**: reliable transport protocol.
FCoE	**Fibre Channel over Ethernet**: network technology that encapsulated Fibre Channel frames over Ethernet medium.
FCIP	**Fibre Channel over IP:** protocol that encapsulates FC frames over IP protocol.
iFCP	**Internet Fibre Channel Protocol:** protocol that runs over IP and provide functionalities similar with FCP
iSCSI	**Internet Small Computer System Interface:** – protocol that runs over IP and provides functionalities similar with SCSI
SoC	**System on Chip:** a hardware system that integrates multiple computer components into a single chip.

DCB	**Data Center Bridging:** a set of Ethernet enhancements for data center infrastructure.
PFC	**Priority Flow Control (IEEE 802.1Qbb)**
ETS	**Enhanced Transmission Selection (IEEE 802.1Qaz)**
QCN	**Quantized Congestion Notification (IEEE 802.1Qau)**
DCBx	**Data Center Bridging eXchange**
LLDP	**Link Layer Discovery Protocol:** link layer protocol used for advertising nodes capabilities within a computer network.
TLV	**Type, Length, Value:** protocol for advertising optional information between nodes within a computer network.
CCF	**Congested Controlled Flow**
CND	**Congestion Notification Domain**
SLA	**Service Level Agreement**