2013 IEEE 12th International Symposium on Parallel and Distributed Computing

Hardware Acceleration in Ceph Distributed File System

Pistirica Sorin Andrei, Moldoveanu Florica,
Moldoveanu Alin, Asavei Victor
Faculty of Automatic Control and Computer Science
University Politehnica of Bucharest
Bucharest, Romania
{andrei.pistirica, florica.moldoveanu,
alin.moldoveanu, victor.asavei}@cs.pub.ro

Caraman Mihai Claudiu
Faculty of Electrical Engineering and Computer Science
Transylvania University of Brasov
Brasov, Romania
mcaraman@unitbv.ro

Abstract— **Cloud computing becomes more and more popular providing the means to organize and deliver almost any kind of data and software services. Since it is inherently scalable to support rapid economic growth and productivity, its distributed data storage layer needs to be capable to address these requirements. A distributed file system contains a large number of cluster nodes and a large number of clients interacting with it. In this article, we propose two acceleration mechanisms based on multi-core network SoC to maximize each cluster node performance. First, the requests to the cluster node are balanced evenly on the platform cores, and second the requests are classified to decrease latency of sensitive operations and improve the overall cluster responsiveness and availability. For implementation, tests and measurements we have used a novel open source distributed file system: Ceph. Among many advantages over the competitors of Ceph we mention preservation of POSIX API, completely decoupled data and metadata and usage of object storage devices instead of block devices.**

Keywords—multicore, hardware accelerators, clustered storage, distributed file system, Ceph

I. INTRODUCTION

Cloud computing applications are becoming nowadays more like common distributed systems. They provide access, even for small businesses, to technology that previously was out of their reach, thus providing the possibility to compete with larger businesses. Cloud computing provides infrastructure as a service for software development and operation: loan software instead of buying it, scale applications to business needs, cut back on system hardware costs and maintenance, use the applications that are automatically kept up to date, and so on.

The number of worldwide internet users grew more than five times in the last ten years and this has also led to an increase of the requirements imposed upon the internet services they use. For example the social network today has billions of users and they view or upload millions of photos every second. To support these increasing requirements, both processing power and data storage have been developed.

The growth of cloud computing and number of internet users poses several important challenges for the distributed file systems: data transfer bottlenecks, performance, scalability, availability, reliability etc. One of the novel distributed file system which tries to meet all the requirements is Ceph [4]. Other than being an open source system in the Linux kernel, it has several advantages including preservation of the POSIX API. In order to integrate easily with customer tools, it provides virtually unlimited storage space, it balances system load between cluster nodes to improve availability and performance and it provides data reliability. Ceph's architecture inherit basic concepts from Google File System, a representative distributed file system used as a foundation for Cloud Computing [1].

The trend in processor development is to emphasize multi-core CPUs as the answer to scaling system performance. Also, software has to leverage the multi-core parallel processing capabilities.

One of the challenges in distributed systems, mainly in distributed file systems, is the enhancement of the network hardware performance to make optimal use of the available network bandwidth. The cluster nodes, other than dealing with storage, are also network based applications. With the development trend of multi-core processors, researchers combine multi-core parallel processing with the hardware multi-function network capabilities. The latest hardware network devices perform multiple functions such as packet manipulation, classification, distribution, security and much more. One of the main challenges is the use of parallel processing and dedicated network hardware processors in an integrated way.

In this paper we propose several optimizations: a solution for parallelizing and evenly distributing the requests on a multi-core SoC platform for network communications to improve cluster node performance and also we also propose a method to decrease the latency of sensitive operations and improve the overall cluster responsiveness and availability based on hardware queue management scheduling and congestion mechanisms.

For implementation purposes we have chosen a QorIQ^{TM} multi-core communication SoC [5], and Ceph distributed file system due to its advantages over competitors.

The article binds the proposed solutions to Ceph, therefore first it overviews this distributed file system in section II. In section III it presents the state-of-the-art and the justification of the proposals. In sections IV, V and VI it describes the solution's architecture, advantages and the tests and measurements to emphasize the obtained performances. Lastly, section VII concludes this work.

II. Ceph Overview

From a system architecture point of view, Ceph has two main layers (see figure 1): a layer for managing namespaces (metadata servers) and one for distributing data objects (monitors and object storage devices) called RADOS [3]. For simplicity, we will call a metadata server, monitor or object storage device simply *cluster node*.

A. Metadata Servers

Ceph has a file system metadata distributed across a cluster called *Metadata Cluster*. Its main purpose is to maintain a consistent file access. Due to its distributed characteristic it also dynamically balance the load based on file popularity.

Being distributed it has few major advantages: the metadata reliability is achieved by storing its data in RADOS (spread across OSDs as any regular file) and it is tolerant to any number of arbitrary node crashes, but do not tolerate to byzantine,

B. RADOS

RADOS is a distributed object store mechanism that provides reliability in an autonomous manner [3]. Designed primarily to lay the foundation for Ceph distributed file system, it can be used in other infrastructures like block devices for Virtual Machines in Infrastructure as a Service clouds.

RADOS manages a cluster of intelligent *Object Storage Devices* (OSDs) and transparently provides replication, failure detection and failure recovery to assure a high level of data availability. A small number of *Monitors* provides a simple OSDs *Cluster Map* which describes the state of the OSDs in the cluster. Objects are identified by a unique *Object Identifier* and they have different requirements, thus they have different replication degree controlled by *Placement Groups* (PGs). Ceph's clients, MDSs or internal OSDs make use of CRASH [3], a deterministic algorithm, to find which PG provides the *Object Identifier*, a replication parameter and the *Cluster Map*.

Clients, with direct access to OSDs, require just the *Object Identifier* and the *Cluster Map* to identify the target OSDs for the store *Object*. Operations on OSDs are asynchronous by their nature.

Monitors are relatively small in number and serve all the entities involved in the system. A *Leader* is elected based on quorum. The *Leader* leases *Monitors* to distribute the *Cluster Map*, usually as deltas from different epochs. *Monitors* aggregate heartbeats from OSDs to detect OSDs availability. If a change is detected a new epoch is generated and propagated throughout *Monitors*.

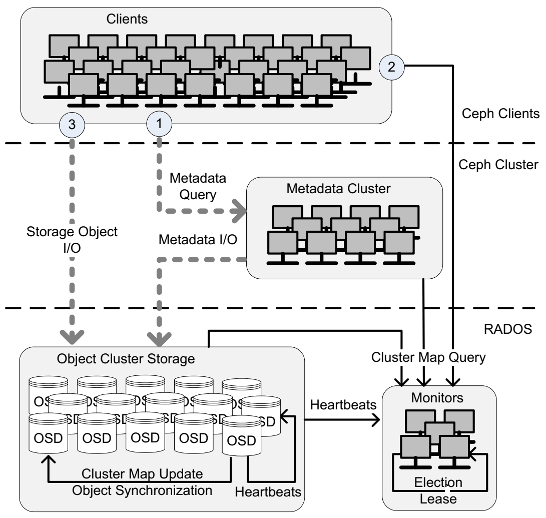

Fig. 1. Ceph architecture.

OSDs self-manage consistent data replication, recovery and migration, by relying on a high bandwidth and low latency of the dedicated network.

III. Related Optimizations

In the next subsections we will describe the optimizations, mainly at system architecture level and our proposed solutions – cluster node optimizations.

Ceph addresses different optimizations for load balancing at system level [2], while we address load balancing at cluster node level. We maximize the cluster node performance by using multi-core network SoC platforms that leverages parallel processing along with hardware accelerators.

In the last 10 years, Ethernet data bandwidth capacity was increased from 10 megabits per second to 100 gigabits per second [10]. The high data rates of the network support a very large number of parallel requests from cluster entities and clients. Thus, a high number of requests are supported by network, but might not be supported by the computation power of the cluster node. By using the capabilities of the network multi-core SoC platforms, several jobs are offloaded from the host cores by its hardware accelerators and with load balancing techniques improve the performance of multi-processor systems.

A. Metadata Servers

In order to improve I/O profile, Ceph stores dentries (i.e. file/directory names) along with inodes (index node, information about a file system object except data content and its name). The advantage is that the dentries and inodes for the entire directory are fetched together into the *Metadata Server* (MDS) cache in a single read (when a lookup or read dir operation is executed). Nevertheless, there are two disadvantages, on the one hand it affects the POSIX semantics (hard link support missing), and by

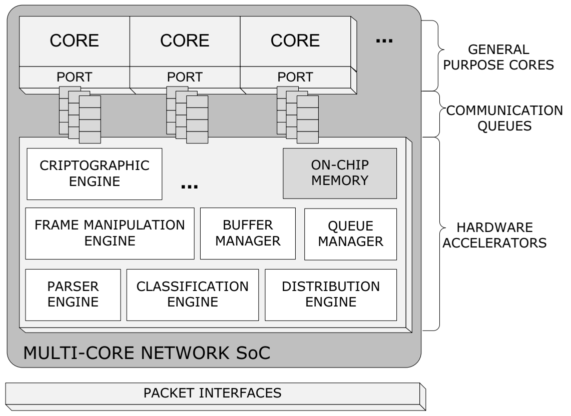

Fig. 2. Multi-core network SoC generic architecture

the other hand if the directory grows very large the model does not scale well.

Ceph addresses these disadvantages by using an *anchor table* to store *remote dentries* (thus, making available hard links), by partitioning the directory hierarchy into directory fragments it tries to solve the load imbalance of Metadata Servers.

Another situation addressed is of a large number of clients that access the same file/directory by sending the current replication information (which is cached), so the clients will know which MDS to contact in the future for popular and unpopular items.

B. RADOS

There are also several optimizations to leverage performance that are taken into consideration by the current RADOS implementation. They are described in the following subsection.

Usually, data safety comes with performance degradation. By design RADOS detach them, allowing excellent performance without affecting the consistency and safety.

The OSD replication algorithm has several important characteristics that influence its performance, like optimization of the message exchange and usage of parallel updates with more replicas. Both the latency and the memory utilization on the tail OSD are lowered.

The *write* operations from cluster nodes were also designed for performance by decoupling notifications for memory data and serialized data.

One of the most important characteristic of CRASH algorithm is that the most PGs remain unchanged when some OSDs change states. These avoid massive data reshuffling and lower network activity as well.

IV. MULTI-CORE NETWORK SOC ARCHITECTURE

A multi-core network SoC has multiple hardware accelerators, each being specialized for a particular job and configured to work together for different common goals. A SoC generic architecture is presented in Fig. 2.

The hardware accelerators must have such flexibility to be configured to match different requirements. Basically, the computation logic of the hardware accelerators forms a graph for each type of frame. This means each received frame will cover a certain graph where each node is a hardware accelerator and the graph's arcs are the *hardware queues*. Each frame is part of a certain category called *flow.* Each flow has a source and a destination (e.g. the source could be a core and the destination a packet interface or vice versa).

A *multi-core network SoC* has different hardware accelerators. Our generic architecture will emphasize the most common ones. The *Parser Engine* parses the frames and it recognizes different standard protocols from frame (it could be even configured to recognize preconfigured proprietary protocols) and generates an output structure called *parse results*. A certain profile (e.g. mask) is applied to *parse results,* based on which the frame is classified by the *Classification Engine* module to match a specific flow. Based on a classification profile, the *Distribution Engine* module decides the next node (e.g. the next hardware accelerator) in the graph. The next node could be any other hardware accelerator, a core or a packet interface. The links between graph nodes are achieved by *hardware queues* (or simply *queues*), which are managed by a different accelerator named *Queue Manager*. The *Queue Manager* has also the ability to schedule the hardware queues based on priorities assigned to each one to match certain application requirements.

The accelerators could change the frame in the process according to a preset profile; therefore certain parts of the frame must be accessed very fast. To achieve this requirement all the modules must work with a very fast memory (on-chip memory). Usually only the frame headers (ISO-OSI 2, 3 and 4 protocols headers) has to be stored, parsed and manipulated.

The frame's payload is stored in an external memory partitioned in *cells.* They are managed by the *Buffer Manager* that acts as a hardware allocator where the cells are the smallest allocation unit. Any hardware accelerator or host application can ask for *cells* (allocation stage), and when the job is done the *cells* are freed (free stage). The cells are partitioned in *buckets*. A *bucket* can be shared between cores or can be configured as private. The shared *buckets* are usually used by the shared queues (shared between hardware system cores), while the private *buckets* are used by the private queues. Sharing is used to leverage forwarding mechanism, otherwise the frame has to be copied from input to output bucket, affecting the system performance.

Cryptographic Engine implements different types of encryption (e.g. AES – Advanced Encryption Standard, SHA1-Cryptographic Hash Function and so on) and it is used, for example, for *Internet Protocol Security* (IPSec) or the computation of the frame's FCS (Frame Check Sequence).

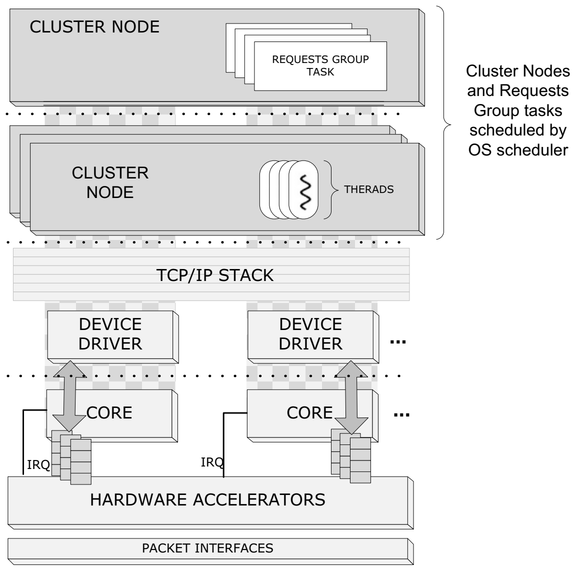

Fig. 3. Kernel-space design

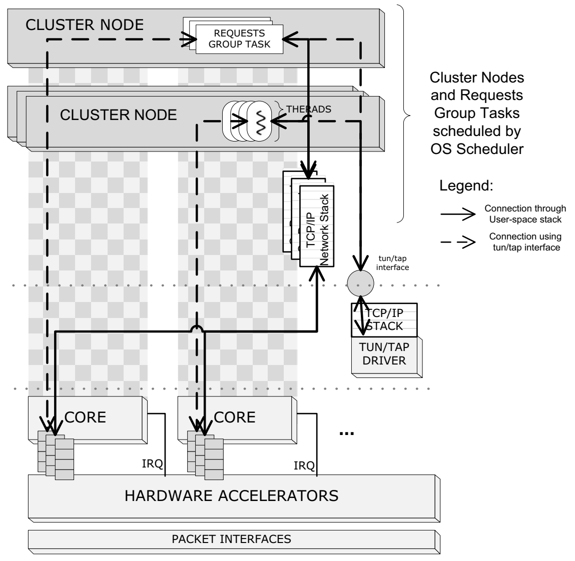

Fig. 4. User-space design

As we said before, the queues can be shared between all cores or can be private to a single core. The difference between them is that from the shared queue any core can consume frames, while from private queue only the designated core. The private queues are useful, for example, when different tasks (affine to a core) process only a certain type of frames.

A. *Real World Chip*

For our implementation we have chosen QorIQ^{TM}, power architecture based multi-core network SoC. Its hardware accelerators are very flexible and are suitable for our purpose.

The platform comes into several flavors, each with a different number of cores and accelerators. Its network accelerators contain different hardware modules: the *Queue Manager* (manage queues which link together all hardware accelerators, cores and packet interfaces), the *Buffer Manager*, the *Frame Manager* (parses, classifies and distributes frames), *Security Engine* (implements different types of encryption algorithms) and *Pattern Matching Engine* (capable of scanning data from streams for patterns that match a specification in a given database).

V. HARDWARE ACCELERATED CEPH

We propose two models (*Per Core Cluster Node* and *SMP Cluster Node*) for hardware accelerated processing of parallel requests and two implementations in both kernel-space and user-space.

A. *Cluster Node Models*

The first model is based on multiple cluster nodes instances running in parallel on the same hardware system and the classification engine directs their requests to the core where the designated instance is scheduled. We name this model *Per Core Cluster Node*. The distribution is made per cluster node instance, and the workload of cores is balanced by the OS scheduler. This is a coarse distribution, where one hardware system acts like several cluster nodes.

The second model is based on single instances of a cluster node with multiple threads and the requests are classified by type in a group and handled by a specific thread. We name this model *SMP Cluster Node*. The number of groups defines the degree of parallelism. Each group is in fact a thread, specialized to deal with a certain group of requests (e.g. one group dealing with metadata operations and other specialized in replicating the metadata fragments). The *Classification Engine* is the hardware module which identifies the request group type.

The models are chosen based on the system configuration: number of cores, number of network connections and network speed. *Per Core Cluster Node* is more suitable on a system with large number of high-speed network connection and large number of cores; while in *SMP Cluster Node* the groups must be chosen in a way that it facilitate the best system workload balance to maximize its performance.

B. *Models Implementations*

We also propose two implementations for these two 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 3 depicts the high-level architecture of the kernel-space implementation, while the figure 4 depicts the user-space implementation.

The mechanism starts with *hardware accelerator* which first parses the ingress frames to identify their types, then classifies and distributes them to the right hardware queues (where each queue is associated with a particular group of flows (requests) or a cluster node instance), from which the destination entity consumes them, leveraging the parallelism of the multi-core system. This method is also known as **Accelerated Receive Flow Steering** [8].

Furthermore we explain the data flow from bottom up. First, the hardware accelerators are configured to parse and recognize the request type. Then the request (i.e. frame) is classified by the *Classification Engine* to match a specific flow (to a request group or cluster node) and enqueue to the related hardware queue. The request identifier (i.e. frame pattern) is not a protocol standard pattern, thus the accelerators must support the ability to match *proprietary* patterns. The QorIQ$^{\text{TM}}$'s *Frame Manager* has this ability by using a function named *Soft Parser* [5]. Last, after the frame was enqueued, the Queue Manager signals the proper core and the frame is dequeued and handled by the designated entity (depending of the used model). To enhance system performance and balance the workload on the cores on a SMP Linux system, we are using different network scaling methods [8].

Each core accesses the hardware queues through a *port* and consumes frames. Therefore, the port registers must be mapped in the corresponding address space (kernel-space or user-space) in order for the application to access the queues and process requests.

The data path for the kernel-space model follows the scaling method from the Linux Networking Stack [8]. When a request is received, the hardware accelerators steers and enqueue it to a proper queue, then issues an interrupt and the network device further passes it to network stack. To leverage the Linux SMP characteristics, the RFS (Accelerated Receive Flow Steering) mechanism overcome scheduled application on a certain CPU to perform frame processing. In the end the application receives the request and processes it. Basically, the workload balance is done by the OS scheduler and Accelerated RFS enhances the performance for multi-processor systems.

One issue for the user-space model is access to a TCP/IP stack. The network stack is in kernel-space, thus the user-space driver deals with raw frames. In order to have TCP/IP support (i.e. connection oriented) there are multiple solutions. One solution is to use tun/tap interfaces (network tunneling/network tap). When a frame is received by the user-space driver, the frame is inserted into the TCP/IP stack through tun/tap interfaces and then the frame's payload is received back using sockets (i.e. endpoint of an inter-process communication flow across a computer network) [7]. Other solution is to use a TCP/IP stack in user-space. In some cases the user-space stack can provide better performance than the native kernel space stack [6].

In the user-space model, polling is required. When a request is received, steered, distributed and enqueued, the thread designated to process that type of request will poll the queue and handle it, when it is scheduled to run. If there is nothing to process, the thread will put itself to sleep.

These models can be implemented for all types of cluster nodes. For testing purposes, we have used kernel implementation of the *Per Core Cluster Node* of the MDS, further called A-MDS (Accelerated Metadata Server). Because in Ceph architecture, metadata is completely decoupled from the distributed object storage, the workload of the metadata server nodes consists of small data structures and its performance affects the entire system. Therefore, the speed with which the MDS deals with clients requests is sensitive. By using the proposed models, we leverage the hardware accelerators and improve performance for multi-processor systems.

C. Accelerated RADOS

We propose also an Accelerated RADOS (A-RADOS) implementation based on hardware queue scheduling (based on weighted prioritization) and congestion notification.

The Queue Manager module from SoC allows us to set distinct priorities per *flow queues* (i.e. hardware queues). By associating *flow queues* with cluster operation types, we manage to decrease the latency of sensitive operations and improve the overall cluster responsiveness and availability.

Since the Queue Manager supports queue prioritization as well (see section III), we have also classified and prioritized operations and messages that take place on A-RADOS cluster nodes.

We group Monitor's traffic in three classes:

- *Traffic Class I*: Election, lease, and heartbeat messages
- *Traffic Class II*: Map Cluster update distribution to Monitors
- *Traffic Class III*: Map Cluster distribution to Clients, MDS and OSDs

Top priority will be assigned to the class I messages that are related to the Leader, Monitors and OSDs availability (election, lease, and heartbeat messages which are sensitive for cluster stability).

The class II traffic deals with *Cluster Map* updates distribution to *Monitors,* responsible in its turn to distribute it further in the whole cluster.

The class III traffic has the lowest priority (less sensitive), groups the *Cluster Map* distribution to *Clients*, MDS and OSDs.

We also group ODS's traffic in three classes:

- *Traffic Class I*: OSDs heartbeats, Cluster Map epoch update propagation
- *Traffic Class II:* Client I/O traffic (e.g. reads/writes), OSDs replication
- *Traffic Class III:* OSDs recovery and migration

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.1 A-MDS micro-benchmark

Top priority will be assigned to the class consisting in OSDs heartbeats and *Cluster Map* epoch update propagation messages required for consistency (i.e. *Traffic Class I*).

To the second class (i.e. *Traffic Class II*), which groups operations like read/writes and associated synchronous replication operations (i.e. *Traffic Class II*) will be assigned a lower priority.

The last class of operations (i.e. *Traffic Class III*) that has the lowest priority deals with the storage recovery and migration of data replication.

VI. Tests and Measurements

For micro-benchmarking we are using a QorIQ^{TM} platform (i.e. P2041 flavor) with 4 cores and hardware accelerators for flow steering, distribution and queue management. The platform's network connection speed is 1Gbps, nevertheless it supports up to 10Gbps. The P2041 is configured as: metadata server and object storage device. Basically, we measure the performance gained with hardware accelerators enhancements.

First, we've measured the ability of A-MDS to handle I/O requests by using Receive Flow Steering method combined with P2041 hardware accelerators support. For measurements we've used *Per Core Cluster Node* model detailed in section V. The topology used was simple, composed by one PC with 1Gbps Ethernet card and P2041 as Accelerated MDS (**A-MDS**).

For meaningful measurements, we tested the Accelerated cluster node against non-accelerated model in the same conditions. We have chosen the kernel space model, since the non-accelerated solution uses native network stack to be able to compare the two micro-benchmark results.

We measured on P2041 the number of handled requests per second with and without acceleration using 4 instances each time. From the tests results listed in table 1, the improvement achieved by RFS was about 3.5%. If each cluster node instance is affine to a single core the improvement was about 12%.

For A-RADOS we did two kinds of tests: by running concurrent requests of all three class types and by running only class III type. Former tests type emphasize that the most priority class is handled before the other classes if hardware queues prioritization support is used, while the former shows that if only class III traffic is running the number of handled requests are comparable. The A-RADOS micro-benchmarks are listed in table 2 and 3.

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

Table.2 A-RADOS micro-benchmark 1

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

Table.3 A-RADOS micro-benchmark 2

VII. Conclusions

We have presented in this paper a hardware accelerated solution based on multi-core network SoC platforms that improve the performance of the novel Ceph distributed file system. While the original Ceph system addresses different optimizations at system level, our contribution focus on improving the cluster node performance.

First, we have improved the performance of the cluster nodes by parallelizing and evenly distributing the requests using Accelerated Receive Packet Steering method. We achieved this by leveraging the hardware accelerators capabilities along with multi-core parallel processing. We proposed two design implementations one in the Linux kernel and the other in user space.

Second, we proposed an Accelerated RADOS (A-RADOS) implementation that decreases latency of sensitive operations and improves the overall cluster responsiveness and availability. We leverage hardware queue management scheduling and congestion mechanisms by defining three classes of message priorities for Monitors and Object Store Devices.

We presented performance measurements that show handled operations requests improvements on MDS and RADOS message processing. Beside the presented improvements there are inherent advantages of using multi-core communication SoC such as low power consumption and appealing price per performance.

VIII. Future Work

For future work we plan to add a model for Egress Traffic Shaping along with Data Center Bridging (DCB). The DCB refers to a set of enhancements to Ethernet local area networks for use in data center environments like Quantized Congestion Notification [9]. We also plan to use the *Cryptographic Engine* to implement a security mechanism in A-RADOS.

IX. Acknowledgments

We would like to thank Freescale for providing an evaluation board to do micro-benchmarks for our solutions.

References

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

[2] 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

[3] 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

[4] Sage A.Weil, "Ceph: Reliable, Scalable, and High-performance Distributed File System", Ph.D. thesis, University of California, Santa Cruz, December 2007

[5] Freescale Semiconductor, Inc., "QorIQ Data Path Acceleration Architecture (DPAA) Reference Manual", 2011

[6] Luigi Rizzo, "netmap: a novel framework for fast packet I/O", Proceedings of the 2012 USENIX Annual Technical Conference, pp. 101-112, June 2012

[7] Steve D. Pate, "UNIX® Filesystems: Evolution, Design, and Implementation", Wiley Publishing Inc., 2003

[8] Scaling in the Linux Networking Stack, https://github.com/torvalds/linux/blob/master/Documentation/networking/scaling.txt, December 2011

[9] IEEE Standard for Local and metropolitan area networks— Virtual Bridged Local Area Networks, Amendment 13: Congestion Notification, March 2, 2009

[10] NTT Communication, The Evolution of Ethernet, white paper, June 2010

[11] LAN Ethernet Maximum Rates, Generation, Capturing & Monitoring, http://wiki.networksecuritytoolkit.org/nstwiki/index.php/LAN_Ethernet_Maximum_Rates,_Generation,_Capturing_%26_Monitoring