LightStor

A scalable, virtual storage network, optimized for SSDs

G S Madhusudan
Matias Bjørling, Jesper Madsen, Philippe Bonnet (IT University of Copenhagen)

Legal Stuff

- The contents of this document is licensed under
  - Creative Commons Attribution-ShareAlike International (CC BY-SA) V 4.0
  - For simple English explanation see “http://creativecommons.org/licenses/by-sa/4.0/“
  - For the actual license terms see “ http://creativecommons.org/licenses/by-sa/4.0/legalcode”
- Standard itself when created will be licensed under an open standards license, I am still trying to identify the right license.
- I undertake that the chosen license will have the following characteristics
  - the proposed standard will be license, royalty and patent free (I cannot of course override existing legal rights of patent or copyright holders).
  - the proposed standard will have license terms that will ensure that the adopter's of the standard cannot assert any patent or other IP rights that can prevent any other user from using or implementing the LightStor standard
  - No coercive techniques like forcible purchase/use of reference implementations will be used to permit use of standards. You will be allowed to use the standard freely in any manner you see for for any business model provided you do not assert any patent/IP rights, use any coercive technique to prevent any one else from implementing this standard.
  - Derivative works will have to provide the same rights as this license

Scope

- Define a storage fabric standard scalable in capacity and bandwidth
- The standard's key features are
  - Clear separation of logical storage layers from physical storage layers.
  - Virtual channels with flow control and QoS support
  - An extendible, declarative **Storage Configuration Language** to specify fabric virtual topology, storage behaviour like fail-over/RAID/replication and QoS parameters to define SLAs.
  - A clear separation of control and data planes at the architecture and fabric level with appropriate virtual channel support
- Define the storage components attached to the fabric and specify their normative functionality
- Specify portions of the T10 stack that apply to this proposal and the proposed extension

Storage Configuration Language

- The Storage Configuration language is crucial to the working of a LightStor instance since it would be practically impossible to configure and maintain a dynamic fabric of this complexity through any other means
- Ideally an existing control language should be used, options include
  - Languages from the networking world like frenetic, procera, netcore
  - Or languages like Quasr used for controlling and configuring arbitrary resource networks https://www.usenix.org/legacy/publications/library/proceedings/tcl96/grady.html
- The language will consist of declarative components and agent like logic to be run on various LightStor components
  - XML based languages do not allow this to be done in an intuitive fashion
- Since the storage requirements for different environments can vary dramatically, LightStor will specify templates for different scenarios - a reference SCL library
  - A social network may require a scale-out fabric with lesser emphasis on transactional semantics
  - This will allow systems integrators to quickly provision standardized configurations with minimal effort

Software Defined Storage

- LightStor is not intended to be an apocryphal Software defined Storage proposal since the phrase has been usurped by the Marketing world and so by definition its use will serve more to confuse rather than clarify
  - But the pervasive nature of the “Software defined Storage” concept will most likely force implementers of this standard to call LightStor as an SDS system in spite of the author's protests !
- The standard has specific goals and a well defined topology and component definition. These goals and definitions do coincide with a “canonical” SDS system

Proposal Overview

- A storage fabric standard based on an optical variant of RapidIO 3.0, optimized for large packet sizes (4k)
- Aimed at systems using SSDs/NVRAM as primary storage devices and HDDs as secondary media
- Define new Storage API optimized for SSDs and future NVRAMs
- Decentralized scheme with distributed metadata – non-centralized management is a key feature of this standard ensuring no single point of failure
- Storage components with significant compute ability to allow running user code in virtualized containers
- Amalgam of existing standards to the extent possible

Rationale – Convergence Trends

- Convergence of networking, storage and computing requires a new unified interconnect standard based on a easily routable network fabric with low-latency switching
  - Should allow unlimited scaling and multi-terabit interconnect speeds
  - The fabric should also allow unlimited redundant paths for 100% fabric availability

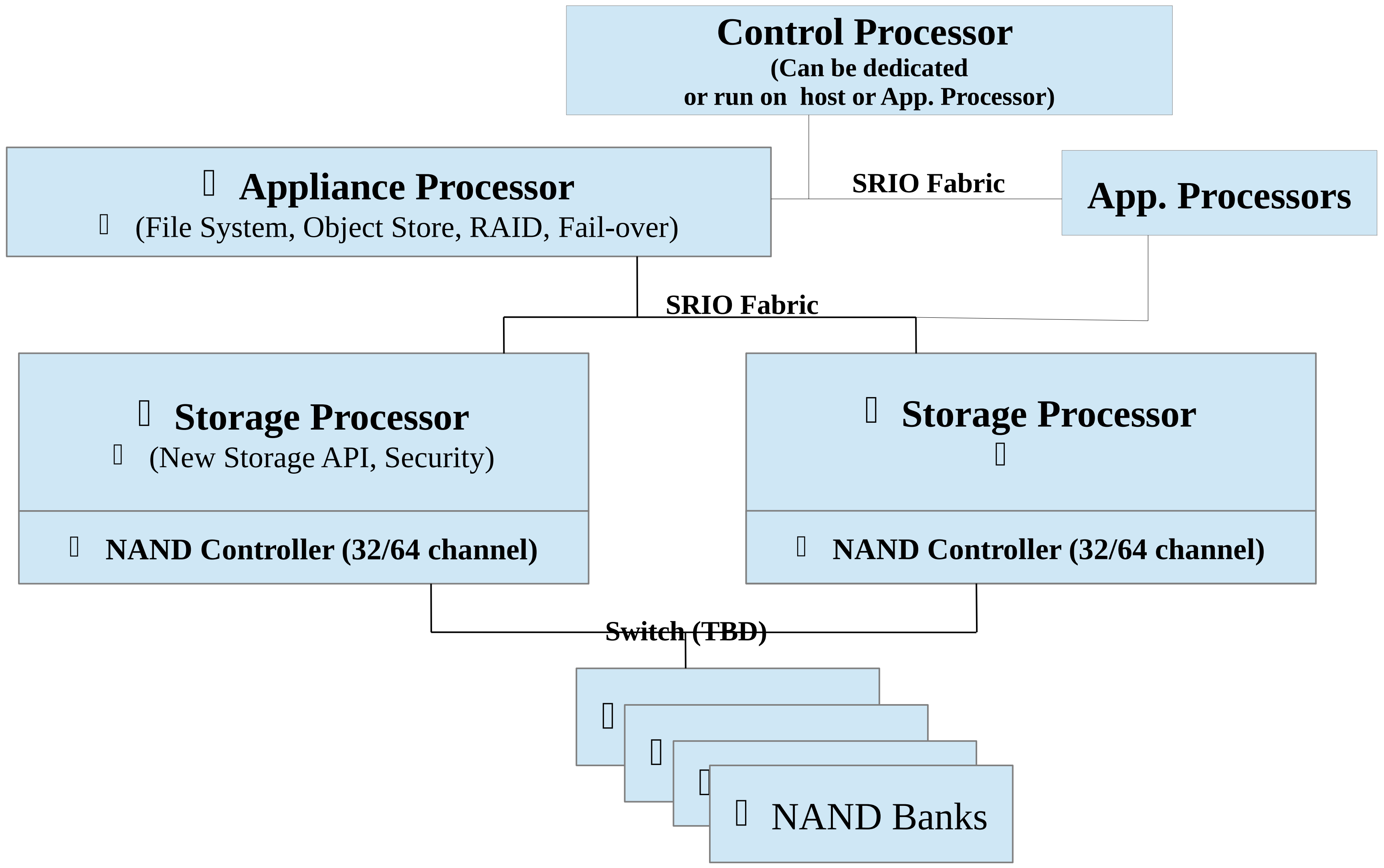
Rationale – Limitations of Storage HW standards

- Existing standards have a variety of problems
  - Ethernet variants have too high an overhead/latency
  - Infiniband/FC et al are too specialized and too expensive
  - NVM Express or any other PCIe based NVM standard addresses SSDs but have no fabric component

Rationale – Storage APIs

- Existing Storage APIs are primarily block storage APIs which focus mainly on the LBA to PBA mapping. SSDS are fundamentally different from HDDs and require a different API to utilize their capabilities
  - Specifically Flash systems have
    - Different reliability/durability characteristics
    - Latencies are not symmetrical
    - Direct erase is not feasible and hence erase needs to be optimized at the application level

Proposed Normative Architecture


Control Processor
(Can be dedicated
or run on host or App. Processor)
Appliance Processor
(File System, Object Store, RAID, Fail-over)
SRIO Fabric
App. Processors
SRIO Fabric
Storage Processor
(New Storage API, Security)
NAND Controller (32/64 channel)
Storage Processor
NAND Controller (32/64 channel)
Switch (TBD)
NAND Banks

Specification Hierarchy

Host Processor (can host Control Functionality)

Appliance Processor (optional)

- **Applications, File Systems**
- **+ optionally Control Processor Functions**

Storage Processor

- **Storage and Security API**

Serial RapidIO 3.0 with Extensions

- **4k PDU**
- **Virtualization**

802.3bm (DWDM Fiber)

et

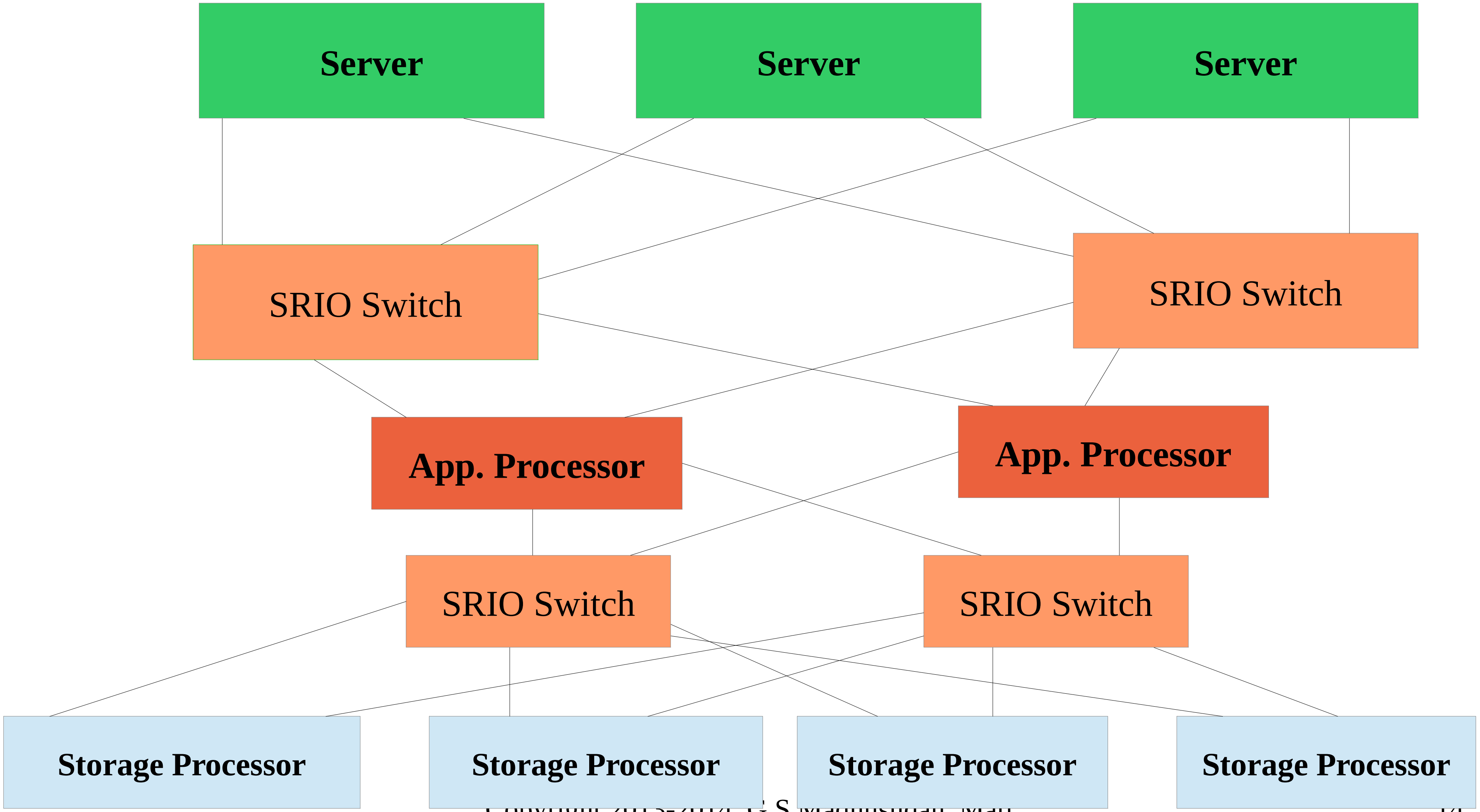
Standard Components

- Virtualized Serial RapidIO Switch fabric
- Control Plane
  - Control processor
- Data Plane
  - Storage Processor
  - Appliance processor
  - Standard NAND modules
    - LightStor will leverage upcoming standard SSD mechanical standards
    - A NAND bank switch is being proposed but feasibility has to be examined.

Fabric Topology

- No specific topology is envisaged since it is a switched fabric and SRIO allows flexible routing
- Node IDs and node count limitations will be determined by RapidIO
- Performance and redundancy will dictate the topology required for specific deployments
- Switching latency is expected to be 100ns or lower per hop, with end to end application latency in the order of 1-2 uS.

Example Dual Redundant Topology



Serial RapidIO 3.0

- The upcoming Serial RapidIO standard is proposed to be used as the interconnect
- It is proposed to use the 10/25G SERDES version of this standard. A LightStor port will consist of 4, 8 or 16 lanes of 10/25 G each transported over optical links.
- Extensions to the SRIO standard if necessary
- Use of optical interconnect instead of copper
- 10 lanes of 10G or 4 lanes of 25G muxed onto the 802.3bm standard's 100G optical link
- Extending max packet size of SRIO to 4k
  - TBD based on performance of 256 byte packets

Control Plane

Control Processor

- The Control Plane of LightStor is a collection of Control Processors
  - The control processing function can be hosted on a dedicated machine or hosted on a host or an appliance processor. The preferred configuration is a dedicated machine
  - In keeping with LightStor's philosophy of redundancy, multiple Control Processors can be configured with replicated metadata.
  - Control Commands will run on dedicated virtual channels with guaranteed QoS
- LightStor will specify a standard set of APIs (T10 + significant extensions), vendors can extend this API
- APIs and behaviour will be specified for
  - Redundancy/Fail-over – Snapshots, Replication, Copy
  - Storage pool/Volumes, Enclosure, Migration, Provisioning, Service class/QoS
  - Security   – Key management, Capabilities (experimental), will leverage Storage Processor's encryption capabilities
  - Control Processor app security – all SCL apps will be digitally signed and distributed to the data plane components using a Public/Private key mechanism
    - The standard will specify security characteristics of the OS and the HW components (Symmetric keys – 256 bit AES, Public keys 2048 bit RSA or Elliptic Curve)
    - HW support for secure boot and verification of signed applications will be required

Control Processor Applications

- The control processor applications are written in the Storage Configuration language
- Applications will have two components
  - a configuration component that specifies features like topology, failover scheme, RAID levels etc.
  - a dynamic component (basically a collection of agent code) that will be invoked by various components of the system based on event triggers
    - Data plane components will have a control plane agent running in a protected VM to execute
- Control processor applications typically run in a decoupled mode
  - the data plane components are initialized with the policies and agent code specified in the SCL application
  - the control plane's proxy in the data plane component is responsible for enforcing these policies and exceptions
  - But provision is made for raising exception to a control processor. This is not expected to be the normal behaviour

Data Plane

Storage Processor – Virtualized Addressing

- The Storage Processor will provide an SSD optimized API.
- Specific optimizations for file systems, caching systems, key-value based systems and database systems
- Virtual Storage API with unified global address,
  - Unified address range across multiple storage processors to be achieved by using distributed metadata. (host + SP or host only - Final arch. TBD)
    - Allow client applications access to a single level store and removes the need to deal with various page mapping issues
    - Not using host is less optimal than running the FTL completely on the host but allows multiple hosts to share a global address space.
      - This makes shared disk architectures trivial to implement
      - Node based access control and distributed lock manager support is a natural extension to this

Storage API – Distributed FTL

- Distributed FTL (host + storage processor)
  - TRIM like functionality will now be more optimal since decisions like garbage collection can be taken at the right layer in the storage processing stack
  - Will allow mitigation of bulk erasure latencies
  - Application controlled wear levelling
  - Log structured allocation strategy
  - Allows coupling of caching and FTL strategy for efficient operations

Storage API – Native data store support

- Atomic storage API – allows delegation of transactional writes to the storage layer instead of doing it sub-optimally at application layers
  - Atomic batching of multiple I/OS
  - FTL layers optimized for atomic writes
    - Initially log structured FTL is envisaged but other schemes can be added
- Key-Value API – allows optimized implementation of Hadoop like systems that rely on key-value stores

Storage API - Other

- The proposed API will also support the following functionality
  - User defined commands (limits and context of UDCs is a major area of concern.)
  - Performance – Write Caching, NVRAM/DRAM support, QoS/priority
  - Failover – RAID, separate RAID scheme for block level metadata
  - Management – SSD configuration, cache, environment/enclosure, redundancy
  - Security – encryption (AES), Authentication, user defined encryption

Storage API – hosting OS code

- OS storage layer can be partly hosted on Storage processor
  - Micro kernel like distributed server strategy
- Hybrid memory cube can be used to share DRAM with host

Appliance Processor

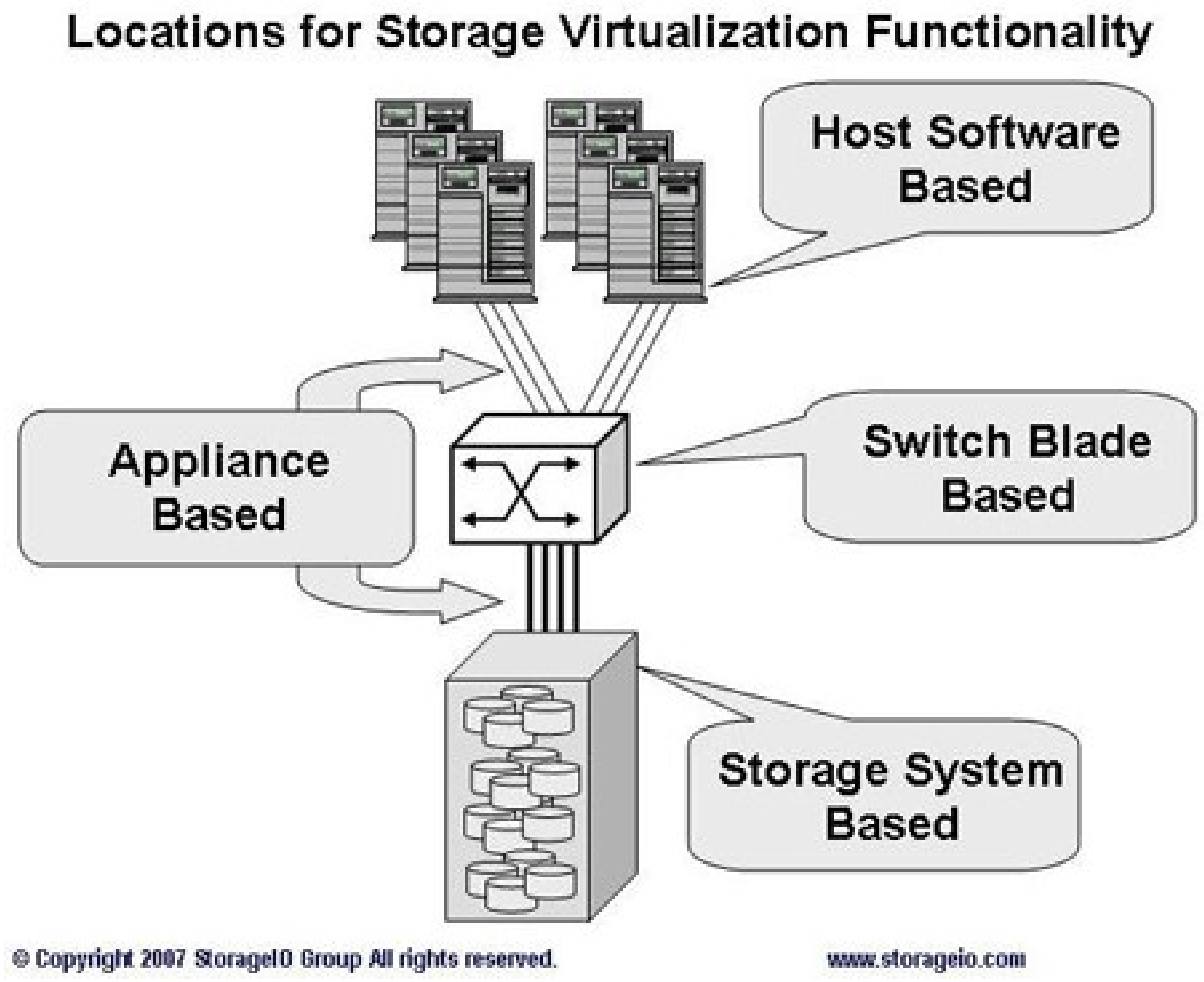
- The Appliance processor hosts standard storage appliance functionality and exposes this through a standardized interface.
  - It is envisaged that storage functionality like RDBMS storage layers, File systems, backup/replication can be delegated to the appliance processor
  - Since the API is standardized, users can mix and match appliance processors from multiple vendors
- Standard T10 File (CIFS, NFS) and Object Store level commands
- LightStor Extensions
  - User defined commands
  - Performance – Easy Clustering semantics for scale out, Mirroring
  - Redundancy/Fail-over – Snapshots, Replication, Copy
  - Management (client side) – Cluster/Redundancy, Storage pool/Volumes, Enclosure, Migration, Provisioning, Service class/QoS
  - Security (client side) – Key management, Capabilities (experimental), will leverage Storage Processor's encryption capabilities
  - Storage Optimization - Compression, De-duplication (TBD – partly to be done on Storage processor)

Virtualization

Virtualization

- LightStor attempts to provide standardized storage virtualization
  - Virtualized fabric provided by enhancements proposed to SRIO (changes are not anticipated to be major since message passing is easily virtualizable)
  - Storage Virtualization provided by the node servers
    - If mapped to currently available solutions, LightStor can be thought of as providing storage server virtualization with network virtualization provided by the SRIO fabric
- Virtualization management is an open issue
  - T10 does not talk too much about it
- Existing host side virtualization can continue to be used but can now leverage LightStor's virtualization features to reduce CPU load
  - Integration with host side VM is to be sorted out

Virtualization


Locations for Storage Virtualization Functionality
Host Software Based
Appliance Based
Switch Blade Based
Storage System Based
© Copyright 2007 StorageIO Group All rights reserved.
www.storageio.com

Staging and Implementation

Staging

- LightStor represents a comprehensive effort at consolidating various storage components into a comprehensive standard. Such a standard will necessarily have to be rolled out in stages
  - Phase 1 – Block level
  - Phase 2 – Node level
  - Phase 3 – Support for SSD + HDD tiering
- The interconnect part of the standard will also depend on SRIO 3.0 roll out. The phases here would probably be
  - Phase 1 – 40 G optical links (4 x 10G)
  - Phase 2 - 100G (10x10G or 4 x 25G)
  - Phase 3 – 400G (16 x 25G, 8 x 50G)

Next Steps

- Identify a technical standards committee sponsor to create a study group
- Study group will identify and include other interested stakeholders interested in this activity
- Study group expected to last 3-6 months to create a Project Authorization Request (PAR) submitted for approval to standards board for forming the working group
- The standards development process from creation of working group to completing draft is expected to be about 18-24 months

Next Steps

- Broad areas for Study Group
  - Identifying T10 commands subset
  - Defining new commands
  - RapidIO extensions
  - Define semantics user defined command mechanism
- Entities working with me in this effort – IIT-Madras, Freescale, IDT, Redhat, EnterpriseDB, RapidIO Trade Association
  - Dell Networking Group, IBM are aware of the effort and are considering joining the effort.

Reference Implementation

- It is also imperative that a complete reference implementation be developed to enable implementers to validate their systems
- The reference system will focus on features to demonstrate the various aspects of the LightStor standard.

Contact Information

G S Madhusudan

Principal Research Scientist

Department of Computer Science and Engineering

IIT-Madras

Chennai, India