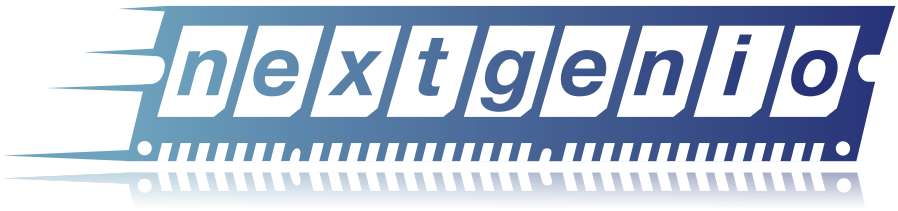

D2.4 The NEXTGenIO Architecture

WP2 "Architecture"

Project Acronym	NEXTGenIO
Project Name	Next Generation I/O for Exascale
Project Number	671591
Instrument	Research and Innovation Action
Thematic Priority	FETHPC-1-2014 HPC Core Technologies, Programming Environments and Algorithms for Extreme Parallelism and Extreme Data Applications
Due date	30/09/2017
Submission date	29/09/2017
Project start date	1st October 2015
Project duration	36 months
Deliverable lead organisation	UEDIN
Status	Final
Version	1.0
Author(s)	Adrian Jackson, Iakovos Panourgias (UEDIN), Bernhard Homölle (FUJITSU), Alberto Miranda, Ramon Nou, Javier Conejero (BSC), Marcelo Cintra (INTEL), Simon Smart, Antonino Bonanni (ECMWF), Holger Brunst, Christian Herold, Muhammad Sarim Zafar (TUD)
Reviewer(s)	Hans-Christian Hoppe (INTEL), Michèle Weiland (UEDIN), Bernhard Homölle (FUJITSU)

Dissemination level	
PU	*PU - Public*

Version History

Version	Date	Comments, Changes, Status	Authors & contributors
0.1	14/08/17	Initial draft created	Adrian Jackson (UEDIN)
0.2	01/09/17	Review draft finished	Adrian Jackson (UEDIN)
0.3	04/09/17	Fixed layout and incorporated Christian's changes	Adrian Jackson (UEDIN), Christian Herold (TUD)
0.4	12/09/17	Update of systemware architectures, correcting some mistakes and re-ordering sections.	Adrian Jackson (UEDIN)
1.0	22/09/17	Incorporating review comments	Adrian Jackson, Michèle Weiland (UEDIN), Bernhard Homölle (FUJITSU), Hans-Christian Hoppe, Marcelo Cintra (INTEL)

Table of Contents

1 Executive Summary 7
2 Introduction 8
2.1 Glossary 8
3 Non-volatile Memory 12
3.1 SCM modes of operation 13
3.2 Non-volatile memory software ecosystem 14
3.3 Alternatives to SCM 14
4 Requirements 16
5 Hardware Architecture 17
5.1 Hardware building blocks 17
5.1.1 Main server node 17
5.1.2 Ethernet switch 19
5.1.3 Network switch 19
5.1.4 Login nodes 19
5.1.5 Boot nodes 19
5.1.6 Gateway nodes 20
5.2 Prototype considerations 20
5.2.1 RDMA options for the prototype 20
5.2.2 Applicability of the architecture without access to SCM 21
5.3 System configuration 21
5.4 ExaFLOP vision 23
5.4.1 Scalability to an ExaFLOP and beyond 24
5.4.2 Scalability into 100 PFLOP/s 24
6 Systemware Architecture 25
6.1 Systemware Overview 25
6.2 Login Nodes 26
6.2.1 Communication Libraries 27
6.2.2 Compilers 27
6.2.3 I/O Libraries 28
6.2.4 Java 28
6.2.5 Job Launcher 28
6.2.6 Job Scheduler 28
6.2.7 Local Filesystem 29
6.2.8 Meta Scheduler 29
6.2.9 Object Store 29

6.2.10 Operating System 29
6.2.11 Performance Monitoring 29
6.2.12 Performance Visualisation and Debugging Tools 30
6.2.13 Python 30
6.2.14 Scheduling Tools 30
6.2.15 Scientific Libraries 30
6.2.16 Task-based programming model 31
6.3 Gateway Nodes 31
6.3.1 Operating System 31
6.4 Boot Nodes 31
6.4.1 Operating System 32
6.4.2 Management/Monitoring Software 32
6.4.3 PXE-Boot Server 32
6.4.4 DHCP Server 33
6.4.5 DNS Server 33
6.4.6 Network Management 33
6.4.7 Local Filesystem 33
6.5 Complete Compute Nodes 33
6.5.1 Automatic Check-pointer 34
6.5.2 Communication Libraries 34
6.5.3 Data Movers 35
6.5.4 Data Scheduler 35
6.5.5 I/O Libraries 35
6.5.6 Java 35
6.5.7 Job Scheduler 35
6.5.8 Management/Monitoring Software 35
6.5.9 Multi-node SCM Filesystem 36
6.5.10 Object Store 36
6.5.11 Operating System 36
6.5.12 Performance Monitoring 36
6.5.13 Persistent Memory Access Library 37
6.5.14 Python 37
6.5.15 SCM Driver 37
6.5.16 SCM Local Filesystem 37
6.5.17 Task-based Runtime Environment 37
6.5.18 Volatile Memory Access Library 38

7 Usage Patterns 39
7.1 General Purpose HPC system using multi-node SCM filesystem 39
7.1.1 Description 39
7.1.2 Components 39
7.1.3 Data lifecycle 40
7.2 General Purpose HPC system improving throughput 40
7.2.1 Description 40
7.2.2 Components 40
7.2.3 Data lifecycle 40
7.3 Resilience in specialist HPC system 41
7.3.1 Description 41
7.3.2 Components 41
7.3.3 Data lifecycle 41
7.4 Large Memory 42
7.4.1 Description 42
7.4.2 Components 42
7.4.3 Data lifecycle 42
7.5 Workflows 42
7.5.1 Description 42
7.5.2 Components 43
7.5.3 Data lifecycle 43
7.6 Data Analytics 43
7.6.1 Description 43
7.6.2 Components 44
7.6.3 Data lifecycle 44
7.7 Live process pausing and migration 44
7.7.1 Description 44
7.7.2 Components 45
7.7.3 Data lifecycle 45
7.8 Object Store 45
7.8.1 Description 45
7.8.2 Components 45
7.8.3 Data lifecycle 46
7.9 Weather forecasting component overview 46
7.9.1 Description 46
7.9.2 Components 46

7.9.3 Data lifecycle 47
7.10 Distributed workflow management 47
7.10.1 Description 47
7.10.2 Components 47
7.10.3 Data lifecycle 47
8 Conclusions 49
9 References 50
10 Appendix 51

Table of Figures

Figure 1: One-level memory (1LM). 13
Figure 2: Two-level memory (2LM). 14
Figure 3: pmem.io software architecture (source: RedHat). 14
Figure 4: Main node DRAM / NVM (SCM) configuration. 18
Figure 5: Unified NEXTGenIO server node (source: FUJITSU). 18
Figure 6: Server node block diagram. 19
Figure 7: Local and RDMA SCM latency with hardware RDMA. 20
Figure 8: Local and remote SCMs with RDMA emulation. 21
Figure 9: The NEXTGenIO prototype rack. 22
Figure 10: Prototype rack details (NOTE: capacities/sizes of memory and other hardware component specifications are based on current expectations, rather than exact specifications of the prototype). 22
Figure 11: Scaling the NEXTGenIO architecture beyond one rack. 23
Figure 12: Intra-rack vs. inter-rack connectivity. 24
Figure 13: Systemware architecture. 26
Figure 14: Login node systemware components. 27
Figure 15: Boot node systemware components 32
Figure 16: Complete compute node systemware components 34
Figure 17: General Purpose HPC system using multi-node SCM filesystem. 51
Figure 18: Resilience in specialist HPC system. 52
Figure 19: Large memory. 53
Figure 20: Workflows. 54

Table of Tables

Table 1: NEXTGenIO scalability (3D torus projections). 23

1 Executive Summary

NEXTGenIO is developing a prototype high performance computing (HPC) and high performance data analytics (HPDA) system that integrates a byte-addressable storage class memory (SCM) into a standard compute cluster to provide greatly increased I/O performance for computational simulation and data analytics tasks.

To enable us to develop a prototype that can be used by a wide range of computational simulation application, and data analytic tasks, we have undertaken a requirements-driven design process to create hardware and software architectures for the system. These architectures both outline the components and integration of the prototype system, and define our vision of what is required to integrate and exploit SCM to enable a generation of Exascale systems with sufficient I/O performance to ensure a wide range of workloads can be supported.

The hardware architecture, which is designed to scale up to an ExaFLOP system, uses high performance processors coupled with SCM in NVRAM (non-volatile random access memory) form, traditional DRAM memory, and an Omni-Path high performance network, to provide a set of complete compute nodes that can undertake both high performance computational and high performance data analysis.

The systemware (system software), which supports the hardware in the system, will enable parallel I/O using the SCM technology, provide a multi-node filesystem for users to exploit, enable use of object storage techniques, and provide automatic check-pointing if desired by a user. These features, along with other systemware components, will enable the system to support traditional parallel applications with high efficiency, and newer computing modes such as high performance data analytics.

The architectures defined in this document are also demonstrated with descriptions of relevant use cases that illustrate which systemware components will be used to undertake various actions on the hardware by user applications.

2 Introduction

The NEXTGenIO project is developing a prototype HPC system that utilises byte-addressable SCM hardware to provide greatly improved I/O performance for HPC and HPDA applications. A key part of developing the prototype is the design of the different components of the system.

This document outlines the architectures we have designed for the NEXTGenIO prototype and future Exascale HPC/HPDA systems, building on a detailed requirements capture undertaken in the project.

Our aim in this process has been to design a prototype system that can be used to evaluate and exploit SCM for large scale computations and data analysis, and to design a hardware and software architecture that could be exploited to integrate SCM with Exascale sized HPC and HPDA systems.

NEXTGenIO is building a hardware prototype using Intel's 3D XPoint™ memory, and we have designed a software architecture that is applicable to other instances of SCM once they become available. The functionality outlined is sufficiently generic that it can exploit a range of different hardware to provide the persistent and high performance storage functionality that 3D XPoint™ offers.

The remainder of this document outlines the key features of SCM that we are exploiting in NEXTGenIO, the general requirements that we have identified for our architectures, and the hardware and software architectures that we have designed. It concludes with a discussion of common usage patterns we envisage will be used to exploit the performance benefits SCM can bring.

2.1 Glossary

1LM	1 level memory. This represents the mode where DRAM and SCM are used as separate random-access memory systems, with their own memory address space. This means that it is possible to address all the DRAM and SCM from an application, resulting in an available memory space of the total DRAM plus the total SCM installed in a node. 1LM allows persistent memory instructions to be issued.
2LM	2 level memory. This represents the mode where the DRAM is used as a transparent cache for the SCM in the system. Applications do not see the DRAM memory address space, only the SCM memory address space. This means the total memory available to applications is the size of the total SCM memory in the node. 2LM cannot issue persistent memory instructions; it can only be used in load/store (volatile) mode.
1GE	1 Gigabit Ethernet
10GE	10 Gigabit Ethernet
100G	100 Gigabit (dual simplex)
3D XPoint™	Intel's emerging SCM technology that will be available in NVMe SSD and NVDIMM forms
AEP	Apache Pass, code name for Intel NVDIMMs based on 3D XPoint™ memory
API	Application programming interface
CPU	Compute processing unit
CCN	Complete compute node
DDR4	Double data rate DRAM access protocol (version 4)

DHCP	Dynamic Host Configuration Protocol
DIMM	Dual inline memory, the mainstream pluggable form factor for DRAM
DMA	Direct memory access (using a hardware state machine to copy data)
DNS	Domain Name System, a standard Internet address resolution system
DP	Dual processor
DPA	DIMM physical address
DP Xeon	DP Intel® Xeon® platform
DRAM	Dynamic random access memory
DMTF	Distributed Management Task Force, an industry standards body for computer system management topics
ECWMF	European Centre for Medium-Range Weather Forecasts
ExaFLOP	10^{18} FLOP/s
FLOP	Floating point operation (in our context 64bit/dual precision)
Gb, GB	Gigabyte (1024^3 bytes, JEDEC standard)
GIOPS	10^9 IOPS
GPU	Graphical processing unit
HDD	Hard disk drive. Primarily used to refer to spinning disks
HPC	High Performance Computing
IB	InfiniBand
IB-EDR	IB “eight data rate” (100Gb/s per link)
IFT card	Internal faceplate transition card (interfaces internal iOPA cables coming from CPUs to server faceplate)
IFS	Integrated Forecasting System, a simulation framework from ECMWF
InfiniBand	Industry standard HPC communication network
I/O	Input / output
iOPA	Integrated OPA (with HFI on the CPU package, as opposed to having it on a PCIe card)
IOPS	I/O operations per second
IPMI	Intelligent Platform Management Interface
JEDEC	JEDEC Solid State Technology Association, formerly known as the Joint Electron Device Engineering Council; governs DRAM specifications
Lustre	Parallel distributed file system; Open source
LNET, LNET	Lustre network protocol
MIOPS	10^6 IOPS
MPI	Message Passing Interface, standard for inter-process communication heavily used in parallel computing
MW	Mega Watt (10^6 Watts)

NAND flash	Mainstream NVM used in SSDs today
NUMA	Non-uniform memory access
NVM	Non-volatile memory (generic category, used in SSDs and SCM)
NVMe	NVM Express, a transport interface for PCIe attached SSDs
NVMF	NVM over fabrics, the remote version of NVMe
NVML	NVM library at [5]
NVRAM	Non-volatile RAM (DIMM based). Implemented by NVDIMMs, being one kind of NVM, that supports both non-volatile RAM access and persistent storage
NVDIMM	Non-volatile memory in DIMM form fact that supports both non-volatile RAM access and persistent storage. Intel NVDIMM, also known as AEP.
OEM	Original equipment manufacturer
Omni-Path	High-performance interconnect fabric developed by Intel for use in HPC systems
OPA	Omni-Path
O/S	Operating system
PB	Petabyte (1024^5 bytes, JEDEC standard)
PCI-Express	Peripheral Component Interconnect Express. Communication bus between processors and expansion cards or peripheral devices.
PCIe	PCI-Express
Persistent	Data that is retained beyond power failure
PFLOP	Peta-FLOP, 10^{15} FLOP/s
PMEM, pmem	Persistent memory, another name for SCM
POSIX	Portable Operating System Interface
POSIX file system	A file system that adheres to the POSIX standard, specifically for File and Directory operations.
PXE	Pre-boot Execution Environment, run before O/S is booted
QPI	Quick Path Interconnect
Quick Path Interconnect	Intel term for the communication link between Intel® Xeon® CPUs on a DP system
RAM	Random access memory
ramdisk	DRAM based block storage emulation
RDMA	Remote DMA, DMA over a network
SCM	Byte-Addressable Storage class memory, referring to memory technologies that can bridge the huge performance gap between DRAM and HDDs while providing large data capacities and persistence of data
SNIA	Storage Networking Industry Association

SNMP	Simple Network Management Protocol
SSD	Solid state disk drive
SHDD	Solid state HDD, using a NVM buffer inside a HDD
TB	Terabyte (1024^4 bytes, JEDEC standard)
TFTP	Trivial File Transfer Protocol
TOP500	TOP500 supercomputer list, http://www.top500.org
U	Rack height unit (1.75”)
Xeon®	Intel’s brand name for x86 server processors
Xeon Phi™	Intel’s manycore x86 line of CPUs for HPC systems

3 Non-volatile Memory

The aim of the NEXTGenIO project is to exploit a new generation of memory technologies that we see bringing great performance and functionality benefits for future computer systems. Non-volatile memory, memory that retains any data stored within it even when it does not have power, has been exploited by consumer electronics and computer systems for many years. The flash memory cards used in cameras and mobile phones are an example of such hardware, used for data storage. More recently, flash memory has been used for high performance I/O in the form of Solid State Disk (SSD) drives, providing higher bandwidth and lower latency than traditional Hard Disk Drives (HDD), but typically with lower capacities.

Whilst flash memory can provide fast I/O performance for computer systems, there are some draw backs. It has limited endurance when compare to HDD technology, limiting the number of modifications of memory cells and thus the effective lifetime of the flash storage. It is often also more expensive than other storage technologies. However, SSD storage, and enterprise level SSD drives are heavily used for I/O intensive functionality in large scale computer systems.

Byte-addressable storage class memory takes a new generation of non-volatile memory that is directly accessible via CPU load/store operations, has higher durability than standard flash memory, and higher read and write performance, and packages it in the same form factor (i.e. the same connector and size) as the main memory used in computers (DRAM). This allows the memory to be installed and used alongside DRAM based main memory, controlled by the same memory controller. This means that applications running on the system can access it directly as if it was main memory, including true random data access at Byte or cache line granularity. This is very different than block based flash storage, which requires intensive I/O operations for such data access.

The performance of byte-addressable SCM is projected to be lower than main memory (with a latency of ~5-10x of DDR4 memory when connected to the same memory channels), but much faster than SSDs or HDDs. It is also projected to be of much larger capacity than DRAM, around 2-5x denser (i.e. 2-5x more capacity in the same form factor).

This new class of memory offers very large memory capacity for servers, long term very high performance persistent storage within the NVRAM memory space of the servers, and the ability to undertake I/O (reading and writing data) in a new way. Direct access from applications to individual bytes of data in the SCM is very different from the block-oriented way I/O is currently implemented.

SCM has the potential to enable synchronous, byte level I/O, moving away from the asynchronous block-based file I/O applications currently rely on. In current asynchronous I/O, the driver software issues an I/O command by handing over an I/O request into a queue to an active controller. Processing of that command may happen after some delay if there are other commands already being processed.

Next, the I/O command is actively started (as with storage read/write or networking transmit), or it is recognised as a receiving descriptor for an asynchronous I/O command coming from an external source (e.g. a networking receive). Once the I/O operation is finished by the controller, the driver is notified (usually by an interrupt) to complete the operation in software.

With SCM providing much lower latencies than external disk drives, this traditional I/O model, using interrupts, is inefficient because of the overhead of context switches between user and kernel mode (which can typically take thousands of CPU instructions).

Furthermore, with SCM it becomes possible to implement remote access to data stored in the memory using RDMA technology over a suitable interconnect. Using high performance networks can enable access to data stored in SCM in remote nodes faster than accessing local high performance SSDs via traditional I/O interfaces and stacks inside a node.

Therefore, it is possible to use SCM to greatly improve I/O performance within a server, increase the memory capacity of a server, or provide a remote data store with high performance access for a group

of servers to share. Such storage hardware can also be scaled up by adding more SCM memory in a server, or adding more nodes to the remote data store, allowing the I/O performance of a system to scale as required.

However, if SCM is provisioned in the servers in a compute system, there must be software support for managing data within the SCM. This includes moving data as required for the jobs running on the system, and providing the functionality to let applications run on any server and still utilise the SCM for fast I/O and storage (i.e. applications should be able to access SCM in remote nodes if the system is configured with SCM only in a subset of all nodes).

As SCM is persistent, it also has potential to be used for resiliency, providing backup for data from active applications, or providing long term storage for databases or data stores required by a range of applications. With support from the systemware, servers could be enabled to handle power loss without experiencing data loss, efficiently and transparently recovering from power failure and resuming applications from their latest running state, and maintaining data with little overhead in terms of performance.

3.1 SCM modes of operation

Ongoing developments in non-volatile / storage class memory have impacted the discussion of memory hierarchies. A common model that has been proposed includes the ability to configure main memory and SCM in two different modes: 1-level memory and 2-level memory [2].

1-level memory, or 1LM, has main memory (DRAM) and NVRAM as two separate memory spaces, both accessible by applications. This is very similar to the Flat Mode [3] configuration of the high bandwidth, on-package, MCDRAM in current Intel® Xeon Phi™ processors (code name "Knights Landing" or KNL). The DRAM will be handled via standard memory API's such as malloc and represent the O/S visible main memory size. The NVRAM will be managed by standard file system API such as mmap and present the non-volatile part of the system memory. Both allow direct CPU load/store operation. In order to take advantage of SCM in 1LM mode, the systemware or applications have to be adapted to use these two distinct address spaces.

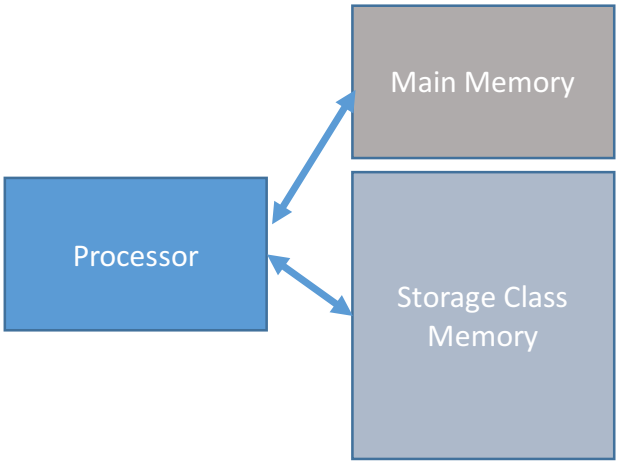

Figure 1: One-level memory (1LM).

2-level memory, or 2LM, configures DRAM as a cache in front of the NVRAM. Applications only see the memory space of the SCM, data being used is stored in DRAM, and moved to SCM when no longer immediately required by the memory controller (as in standard CPU caches). This is very similar to the Cache Mode[3] configuration of MCDRAM on KNL processors.

This mode of operation does not require applications to be altered to exploit the capacity of SCM, and aims to give memory access performance at main memory speeds whilst providing the large memory

space of SCM. However, how well the main memory cache performs will depend on the specific memory requirements and access pattern of a given application. Furthermore, persistence of the NVRAM contents cannot be longer guaranteed, due to the volatile DRAM cache in front of the NVRAM, so the non-volatile characteristics of SCM are not exploited.

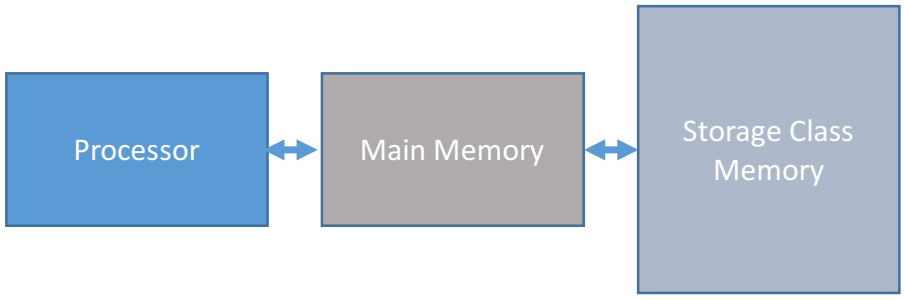

Figure 2: Two-level memory (2LM).

3.2 Non-volatile memory software ecosystem

In recent years the Storage Networking Industry Association (SNIA) has been working on a software architecture for SCM with persistent load/store access. These operating system independent concepts have now materialised into the Linux pmem.io [5] library.

This approach re-uses the naming scheme of files as traditional persistent entities and map the SCM regions into the address space of a process. Once the mapping has been done, the file descriptor is no longer needed and can be closed.

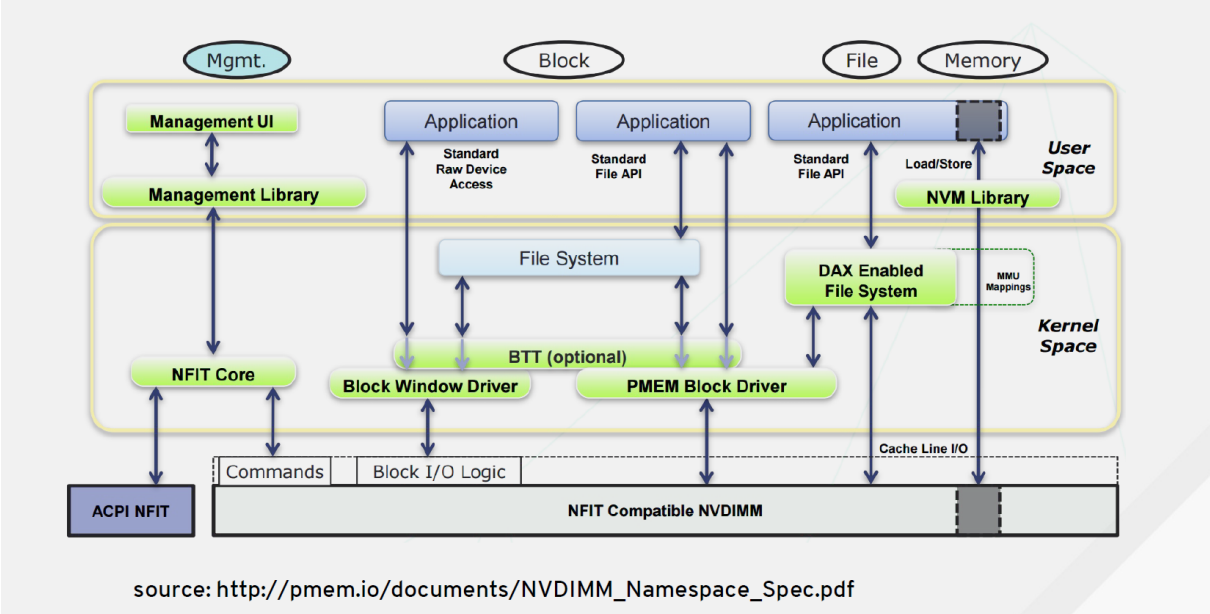

Figure 3: pmem.io software architecture (source: RedHat).

3.3 Alternatives to SCM

There are existing technological solutions that are offering similar functionality to SCM and that can also be exploited for high performance I/O. One example is NVMe devices: SSDs that are attached to the PCIe bus and support the NVM Express interface. Indeed, Intel already has a line of an NVMe device on the market that use on 3D XPoint™ memory technology, called Intel® Optane™ [6]. Other vendors have a large range of NVMe devices on the market, most of them based on different variations of Flash technology.

NVMe devices have the potential to provide byte-level storage access, using the pmem.io libraries. A file can be opened and presented as a memory space for an application, and then can be used directly as memory by that application.

NVMe devices offer the potential to remove the overhead of file access (i.e. data access through file reads and writes) when performing I/O and enable the development of applications that exploit SCM functionality. However, given that NVMe devices are connected via the PCIe bus, and have a disk controller on the device through which access is managed, NVMe devices do not provide the same level of performance that SCM offers. Indeed, as these devices still use block-based data access, fine grained memory operations can require whole blocks of data to be loaded or stored to the device, rather than individual bytes.

Memory mapped files are another example of SCM like functionality using existing technology. Memory mapped files allow I/O storage to be accessed as if it was memory, through the virtual memory manager. However, the data still resides on a traditional storage device and it can therefore only be accessed at the performance rate that this storage device allows.

Memory mapped files allow for some of the block storage costs to be avoided (i.e. those costs caused by the asynchronous data access and interrupts), but are limited to the size of DRAM available to an application, have coarser grained persistency (requiring full blocks to be flushed to disk to ensure persistency), with much higher persistency costs than byte-addressable SCM.

4 Requirements

To design our hardware and systemware architectures that will enable us to construct a prototype system exploiting SCM for HPC and HPDA applications, we identified and examined the usage scenarios for such a system and used them to generate a set of requirement.

Whilst detailed reporting of those requirements is not the aim of this document, we will outline the general requirements we considered key when designing the system:

1. The system should be as close as possible to a standard, production, large scale compute system.
2. Applications must be able to run on the system without changes, other than possibly requiring re-compilation, and be able to use the SCM to provide performance benefits.
3. It must be possible to modify applications to directly use the SCM and exploit the full performance the hardware provides.
4. The hardware and systemware must be able to support a heterogeneous SCM environment. We consider that systems where some nodes have SCM and some do not may be desirable for a range of users or system providers; we therefore want our hardware and systemware architectures to be able to support such configurations.
5. The system must be able to support multiple users and multiple applications running at the same time.
6. The system must be flexible enough to allow any application to run on any compute node.
7. It should be possible for multiple applications to share a single compute node, or for user jobs to specify that exclusive node access is required.
8. The system must support the sharing of data between applications or user jobs. In particular we recognise that *workflows* of applications are an important use case for SCM in such systems.
9. The system should support I/O from applications using non-filesystem based approaches. One candidate could be an object store.
10. The system should provide data access to SCM between compute nodes to allow applications to access remote data and to allow the system to manage data in SCM.
11. The architectures should enable the design of an ExaFLOP range system with an I/O performance sufficient to ensure I/O is not a performance limiter when scaling applications on such a system.
12. The system should allow different storage targets to be configured for a given application. This requires software being configured for the set of compute nodes allocated to a given user job.
13. The system must support profiling and debugging tools to enable users to understand the performance and behaviour of their programs.

5 Hardware Architecture

Developing a prototype platform to test and evaluate the performance and functionality benefits of SCM is one of the key objectives of the NEXTGenIO project. The SCM technology we are using, 3D XPoint™ memory, is a new type of memory that requires specific hardware support. 3D XPoint™ memory cannot simply be installed in an existing system.

However, one of the major benefits of SCM (NVRAM) is that it comes in the same hardware form factor as standard memory (DRAM, DDR4 for 3D XPoint™). This means we can design and build a system that can take both standard memory and SCM in the same node and provide both to users.

SCM does required specific processor support, primarily because memory controllers in modern computing systems are directly integrated into the processor itself. This means, if we want to use a new memory technology we need processors with support for that new memory in their memory controllers. For instance, for 3D XPoint™, Intel has announced a new generation of Intel® Xeon® processors with the code name "Cascade Lake" for 2018 release that will include such support.

Furthermore, as we are combining two types of memory in the servers (or nodes) in the system, we need to ensure that there is capacity to install sufficient amounts of standard memory and SCM. As we are looking to provide a prototype system where large scale computational simulation and data analytics tasks can be undertaken, we need to provide sufficient hardware capacity to install sufficiently large amounts of standard DRAM memory and SCM at the same time. In other words, we need a system that has space for a large number of memory modules, or DIMMs, to be installed.

The NEXTGenIO prototype system is designed to provides for very high SCM capacities and I/O bandwidth, combining is 3D XPoint™ technology with Intel's new Omni-Path interconnect to provide extremely high I/O performance for applications.

A single NEXTGenIO rack holds up to 32 Xeon® servers, Complete Compute Nodes (CCNs), connected by the Omni-Path network, with two login nodes to support local compilation, visualisation, and rack management. Furthermore, two boot servers are provided to store and serve the operating system and software required by the CCNs, and two gateway servers to enable access to an external filesystem for long term storage, and for performance comparison with the I/O provided by the SCM.

The architecture is designed with scalability in mind, facilitating the connection of NEXTGenIO racks into larger systems, reaching out well into the ExaFLOP range. We enable software to access local SCM almost as fast as DRAM, and to access and any other CCN's SCM in the cluster faster than accessing a fast storage device (i.e. a fast SSD) inside the node itself.

This SCM focussed system architecture allows fundamentally new designs for scalable HPC software and provides the functionality required to make the data centre power consumption become the final scaling limiter, not the system architecture.

Using future processors and network generations, we expect an ExaFLOP configuration based on the NEXTGenIO architecture to become practical within five years.

5.1 Hardware building blocks

5.1.1 Main server node

Since NEXTGenIO is centred on using SCM as a new, ultra-fast storage layer between memory (DRAM) and disk, we have to understand the goals and constraints on how to configure DRAM and NVM in the servers being used.

As shown in Figure 4, the NEXTGenIO server node is a dual processor node and supports SCM. As we are looking for high performance nodes, we need to have all 12 memory channels (as provided by Intel's current "Skylake" generation of Intel® Xeon® processors and announced for "Cascade Lake") populated with some DRAM, to enable utilising the total available DRAM bandwidth in the node.

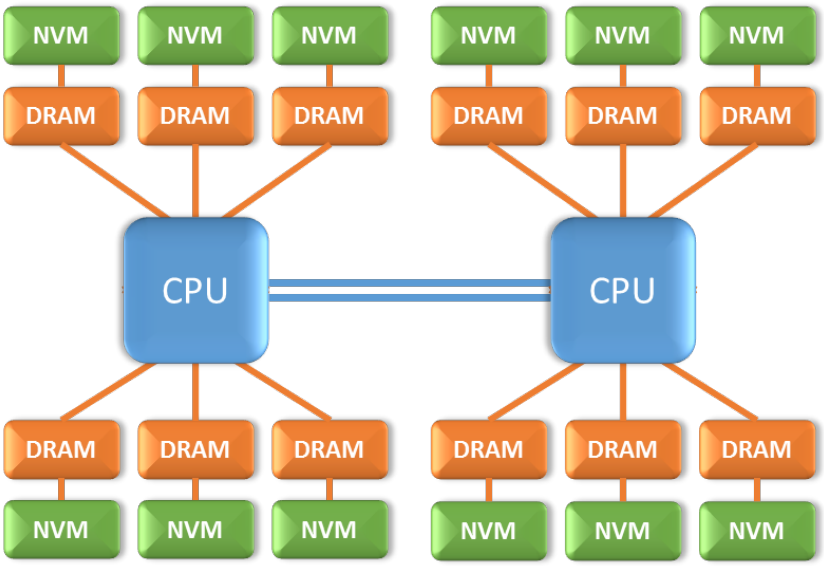

Figure 4: Main node DRAM / NVM (SCM) configuration.

Since we are also interested in maximum SCM bandwidth and capacity, we also need to populate all 12 memory channels in the node with SCM. Together, that makes 12 DRAM-DIMMs plus 12 NVDIMMs, pairing up in each channel. Of course, devices on the same channel share the available DDR4 channel's bandwidth.

The server hardware is implemented in a 1U form factor (Figure 5), which is a good compromise between configurability and density.

Figure 5: Unified NEXTGenIO server node (source: FUJITSU).

Figure 6 shows the NEXTGenIO server node implementation block diagram. While most details are not relevant here, some are important. In total, there are three PCIe x16 slots available for add-in cards in the system. We will develop support for processors with integrated network connection i.e. one channel coming off each CPU; this currently requires a PCIe slot to carry an "IFT Card", holding the modules to receive external cables (shown on the upper right in the picture). Consequently, that leaves two PCIe x16 slots for other I/O cards.

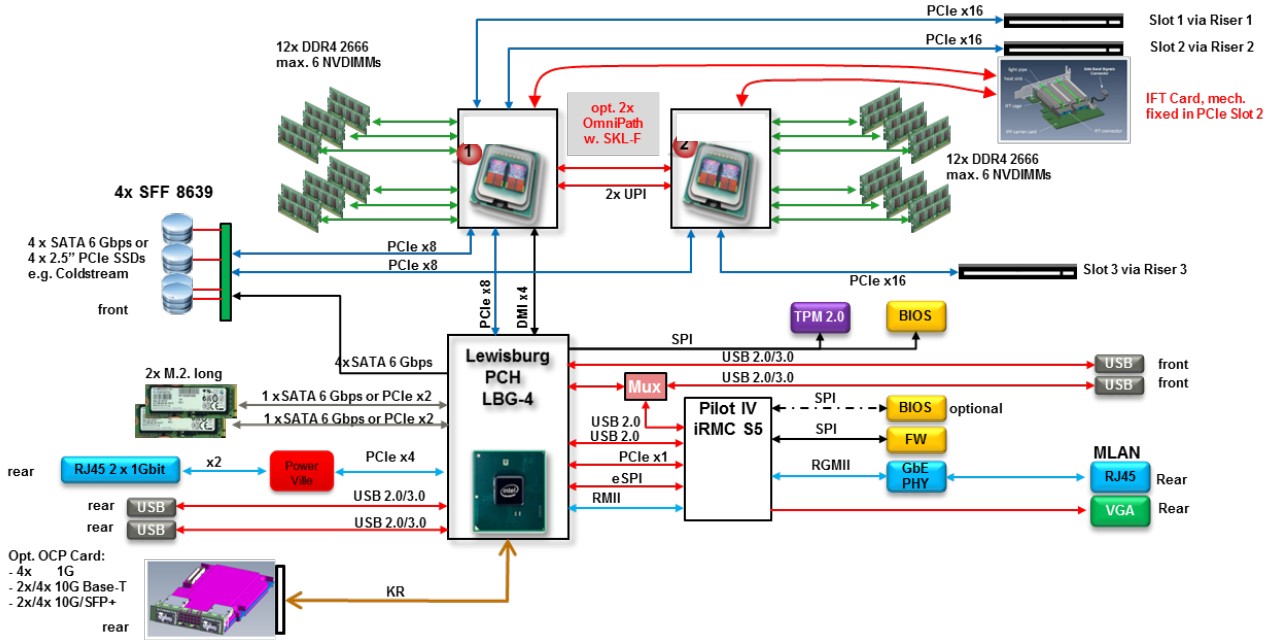

Figure 6: Server node block diagram.

5.1.2 Ethernet switch

For system management and general-purpose inter-node communication, two Ethernet networks are included: 1GE for the management LAN and slow data connections, and 10GE to connect to the data centre LAN as well as fast connections within the rack.

5.1.3 Network switch

The hardware architecture supports the integration of most common high performance networks to provide the fast, high bandwidth, interconnect between the nodes in the system. This includes Infiniband based networks, Omni-Path Architecture and fast Ethernet and Omni-Path options, connected via PCI Express or (for Omni-Path) integrated into a multi-chip CPU package.

One feature that is necessary for a system to support all the project's requirements is remote data access over the network. RDMA offers the potential for high performance access to remote SCM, which is important to enable some of the use cases we are considering.

5.1.4 Login nodes

As with any HPC cluster, we need to support login nodes to access the system, perform compilations and remote visualisation. Login nodes have different requirements to the compute nodes, and as such will be provisioned with high memory and processing capabilities, along with local storage to enable fast compilation of applications.

This local storage can be provided by SCM in the nodes or local SSDs, depending on the cost and availability of these components. SCM could provide the benefit of a large memory space for the login nodes (i.e. if it supported 2LM functionality).

5.1.5 Boot nodes

To enable booting of the compute nodes via the network, for easy administration of the operating system, and to install software on the compute nodes, we provide a number of boot nodes. These have 1GE network connections to the compute servers, connected through 10GE switches (to reduce the risk of becoming saturated during system initialisation).

Boot nodes typically have low requirements on compute performance, i.e. even a mainstream single CPU is expected to be sufficient. The storage capacity requirements are also moderate. However, to avoid bottlenecks on boot storms, SSDs or SHDDs (HDDs with an internal NAND-Flash buffer) seem to be appropriate.

5.1.6 Gateway nodes

If an Omni-Path network is used as the main interconnect within the system, and we need to connect the system to an external Lustre filesystem based on Infiniband, we will need some gateway servers that bridge between the two interconnects. Intel have techniques for building such gateways using standard Intel® Xeon® servers and Linux operating system.

5.2 Prototype considerations

The choice of network for the prototype is between Infiniband and Omni-Path based solutions. Whilst Infiniband offers hardware RDMA, Omni-Path provides the potential for a network integrated on to the processor (integrated OPA or iOPA). For the NEXTGenIO prototype we decided to evaluate the integrated network functionality of Omni-Path.

5.2.1 RDMA options for the prototype

There are a number of ways we can provide remote data access in the prototype system. If we utilised an Infiniband based network we could enable RDMA (supported by the network adapter hardware), providing very low latency (1 microsecond or less) and high bandwidth access, with small I/O operations. This hardware RDMA also does not use much of the compute resource in a node as the network adapter has its own processing hardware to support the RDMA options.

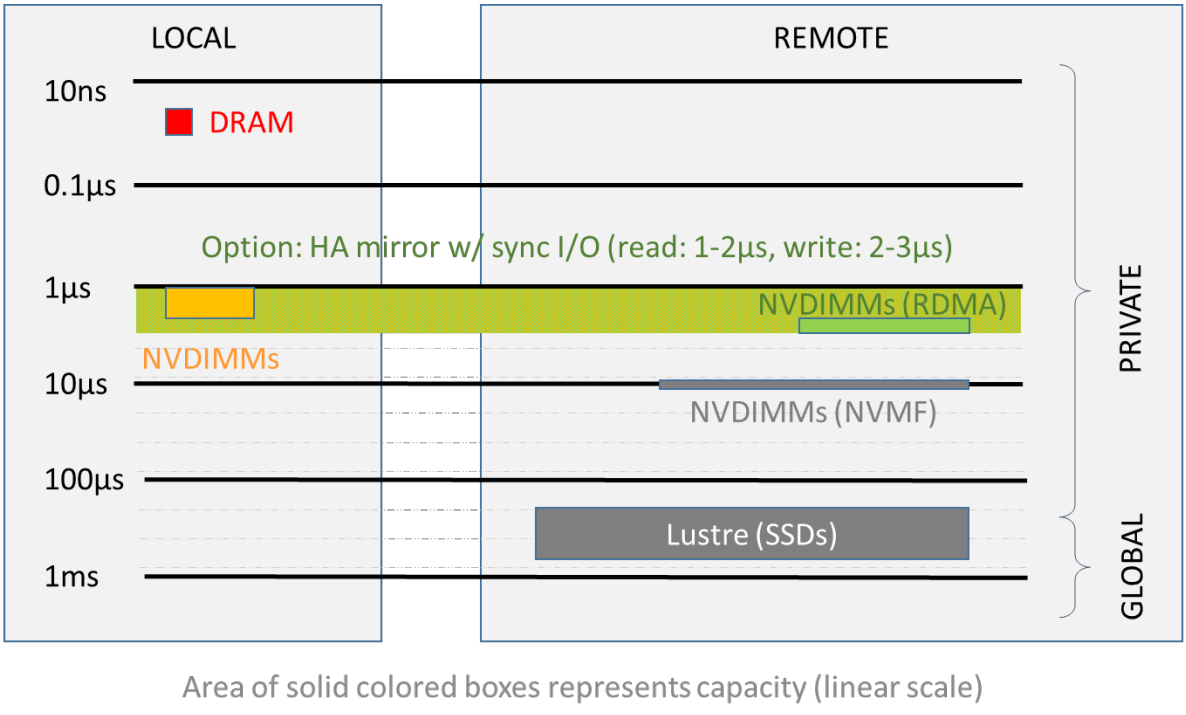

Figure 7: Local and RDMA SCM latency with hardware RDMA.

However, the current generation of Omni-Path network adapters does not support hardware accelerated RDMA (future generations of Omni-Path are scheduled to have such support). In this scenario, parts of the RDMA operations need to be executed by the host CPU. Assuming an optimised RDMA emulation target using polling, we expect a remote SCM block read/write latency of around 4

to 5 microseconds (Figure 8). Even though this is significantly higher than with hardware accelerated RDMA, it will still be in the range for effective use of synchronous I/O.

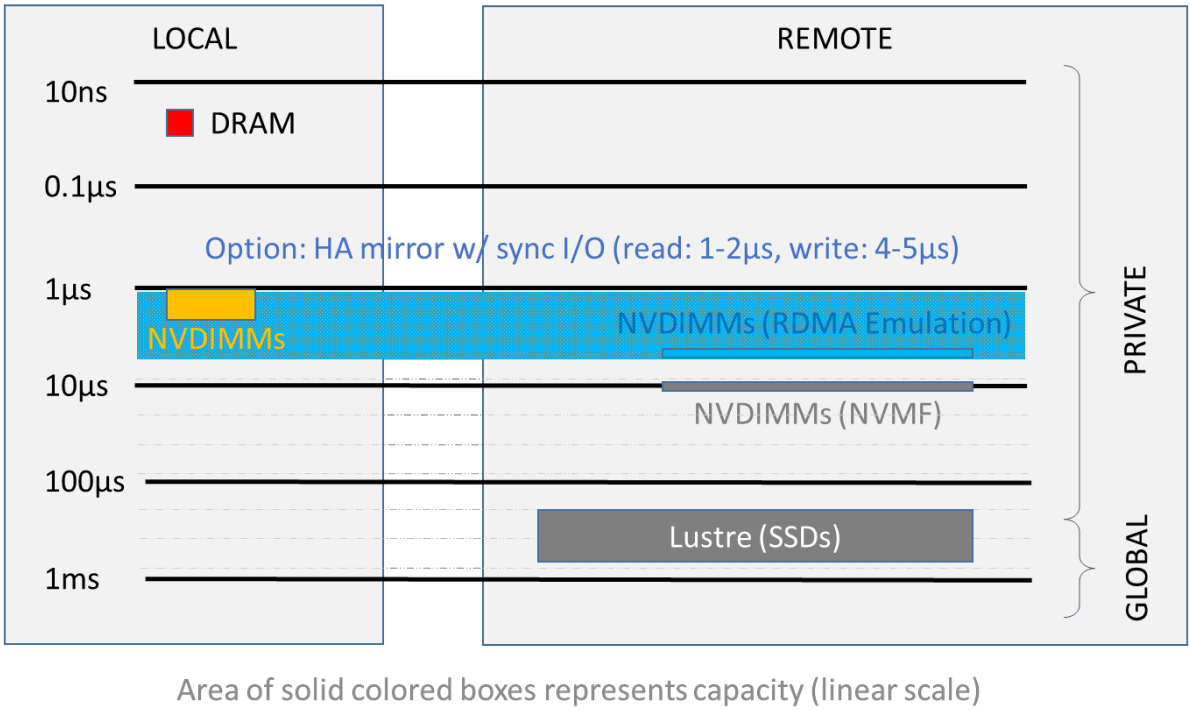

Figure 8: Local and remote SCMs with RDMA emulation.

5.2.2 Applicability of the architecture without access to SCM

Whilst this hardware architecture is designed around SCM, it is more generally applicable. For instance, local (in-node) SCM could be replaced by local non-volatile disk drives, using NVM-Express (NVMe) devices. These provide byte level access to persistent memory from applications, and are compatible with the pmem.io library functionality. It would also be possible to replace in-node SCM with remote NVMe devices, using NVM over Fabrics (NVMF).

NVMe latency and bandwidth will not be of the same level of performance compared to the SCM we will use in our prototype architecture (i.e. 3D Xpoint™ memory), with best sustained NVMe latencies for current technology being around 100-300 µs, and bandwidth of around 2.5-3.5 GB/s (for a x4 PCIe connection), compared to 300-500ns and 8-10 GB/s for the SCM (per DIMM). The NVMF latency will be higher than with remote SCM accessed by RDMA. We expect around 10-20 µs (read/write) with 3D XPoint™-SSDs (Optane) and around 10 µs (read/write) with a SCM based NVMF target.

5.3 System configuration

The NEXTGenIO prototype rack provides approximately 100 TB NVRAM capacity based on Intel's 3D XPoint™ technology [1]. It is accessible from any (and to any) CCN in less than 5 microseconds, with very high I/O bandwidth (almost 400 MIOPs).

The rack holds up to 32 Intel® Xeon® servers, also known as complete compute nodes (CCN), connected by Intel® Omni-Path Architecture fabric. Two login nodes support local compilation, visualisation, and also rack management. Two boot servers enable network boot, and two gateways to an external file system. The node configurations details and network connectivity are shown in Figure 10.

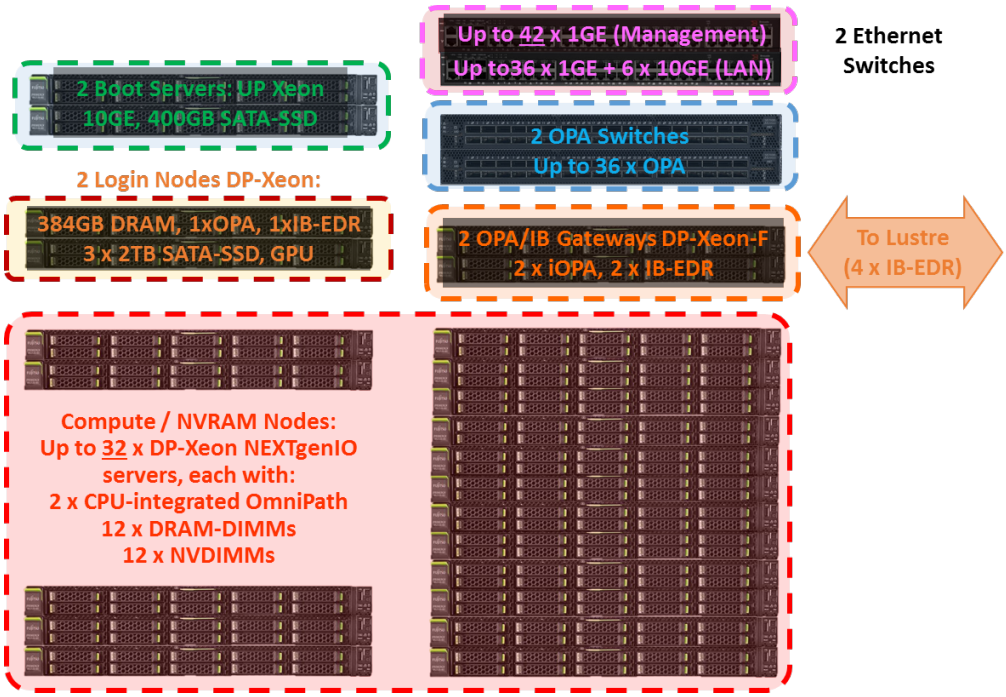

Figure 9: The NEXTGenIO prototype rack.

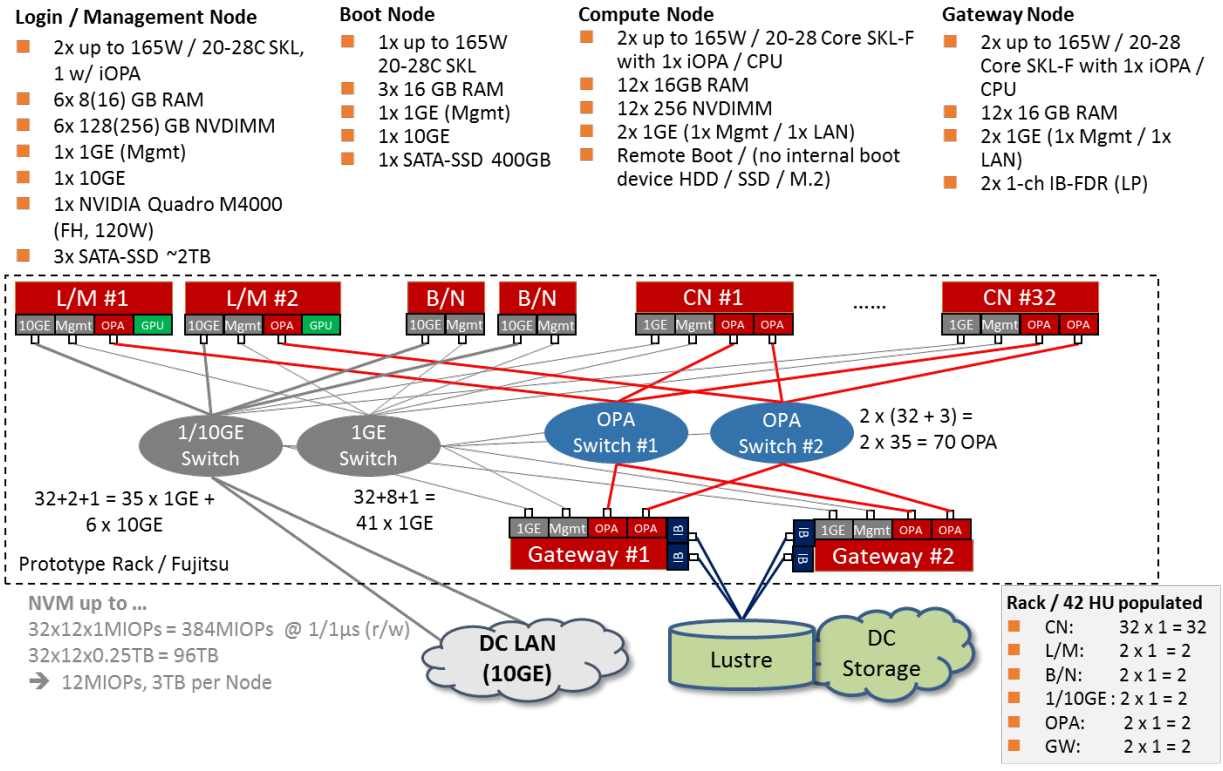

Figure 10: Prototype rack details (NOTE: capacities/sizes of memory and other hardware component specifications are based on current expectations, rather than exact specifications of the prototype).

Assuming 2 to 4 TFLOPS per node, the NEXTGenIO rack with 32 nodes achieves 64-128 TFLOPS and approximately 384 MIOPS, i.e. around 1.5 TB/s local block storage. For the purpose of the prototype this scale is sufficient, but one key aspect of this work is to show how this could scale into the ExaFLOP range.

5.4 ExaFLOP vision

Since the architecture supports nodes using the two integrated network ports of the CPUs, we have two PCIe x16 slots available per node. Using two single-channel network cards, we can extend the single rack system to many racks, but still preserve the same hardware/software architecture. It would just add a second level of "slower" connectivity, i.e. between racks. "Slower" in this context means higher latency and lower bandwidth per node, but still keeping the concept of synchronous I/O, at least up to a range of 10 to 20 microseconds latency.

With a 5-hop network we can link 864 racks with 50 to 100 PFLOPs, 83 PB NVM, and 332 GIOPs:

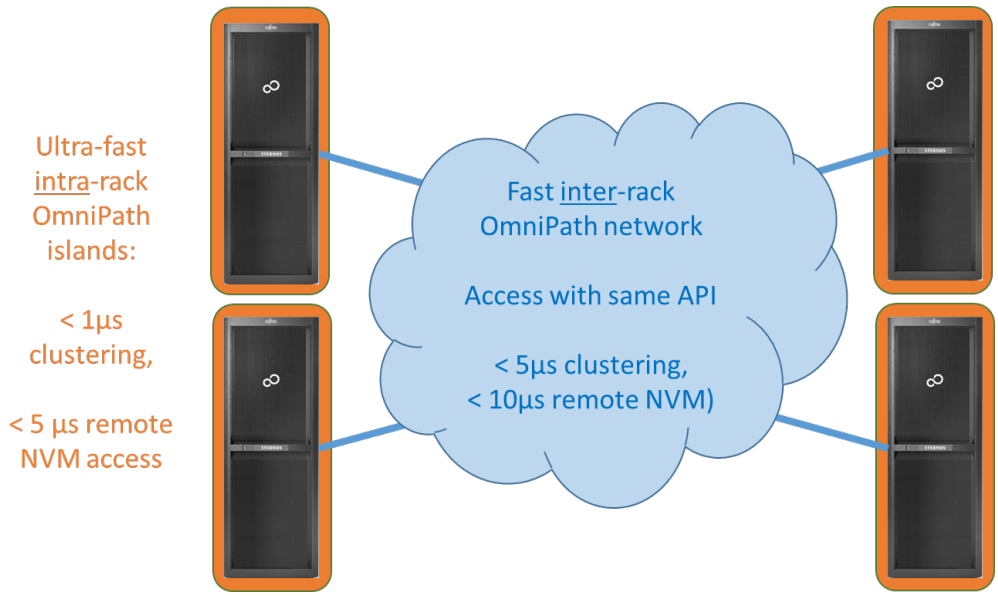

Figure 11: Scaling the NEXTGenIO architecture beyond one rack.

With cubical 3D torus configurations, we can project scaling to the ExaFLOP range – still using the same NEXTGenIO hardware/software architecture (Table 1).

Total # Nodes (Intel DP Skylake)	PFLOP (Min Estimate)	PFLOP (Max Estimate)	Total SCM Capacity (PB)	Total SCM I/O B/W (TB/s)	Total Power (MW)
768	1.5	3	2	36	0.4
3,072	6	12	9	144	1.5
24,576	48	96	72	1,152	12
82,944	162	324	243	3,888	41
196,608	384	768	576	9,216	98
384,000	750	1,500	1,125	18,000	192

Table 1: NEXTGenIO scalability (3D torus projections).

At 750 to 1,500 PFLOPs, the total SCM capacity would be 1.1 Exabyte. The total I/O bandwidth would be 18 PB/s, which corresponds to the speed of approximately 6 million current day PCIe-based SSDs.

The FLOP to I/O bandwidth ratio would be 1:50, i.e. at a similar level to the FLOP to memory bandwidth ratio that HPC systems aspire too.

The power needed for a 1 ExaFLOP system based on current Intel processors would be impractical – almost 200MW. However, based on future CPUs and interconnects, an ExaFLOP configuration within an affordable 20-30MW should become feasible early in the next decade.

5.4.1 Scalability to an ExaFLOP and beyond

Technically, the OPA link layer can address more than 10 million nodes[4]. Such cluster size would be far beyond any practical power limit (5 GW). Early next decade, however, a dual processor (DP) node may have 50 TFLOPs or more in the same power envelope. Then, 50,000 nodes would achieve 2.5 ExaFLOPs in a much more practical power budget of 25MW.

5.4.2 Scalability into 100 PFLOP/s

According to Intel, a 5-hop OPA cluster can link up to 27,648 nodes. With our NEXTGenIO rack and a single OPA inter-rack link per node, this would lead to the following configuration (see also Figure 12):

- 864 racks with 27,648 NEXTGenIO nodes
- 50 to 100 PFLOPs
- 83 PB of SCM with 332 GIOPs @ 4kB = 1.36 PB/s local I/O
- 14 MW power (assuming 500W/node including infrastructure)

The inter-rack bandwidth would be dependent on the network topology. Not knowing the configuration details, we guess the bandwidth per node will be lower than the 2 x 100G available in our prototype rack. The latency, however, would be similar. Assuming 1 microsecond or less round-trip per hop, the MPI latency should stay below 5 microseconds, and remote SCM via RDMA emulation below 10 microseconds. This would enable synchronous remote I/O SCM storage access, but at somewhat lower efficiency than within the racks.

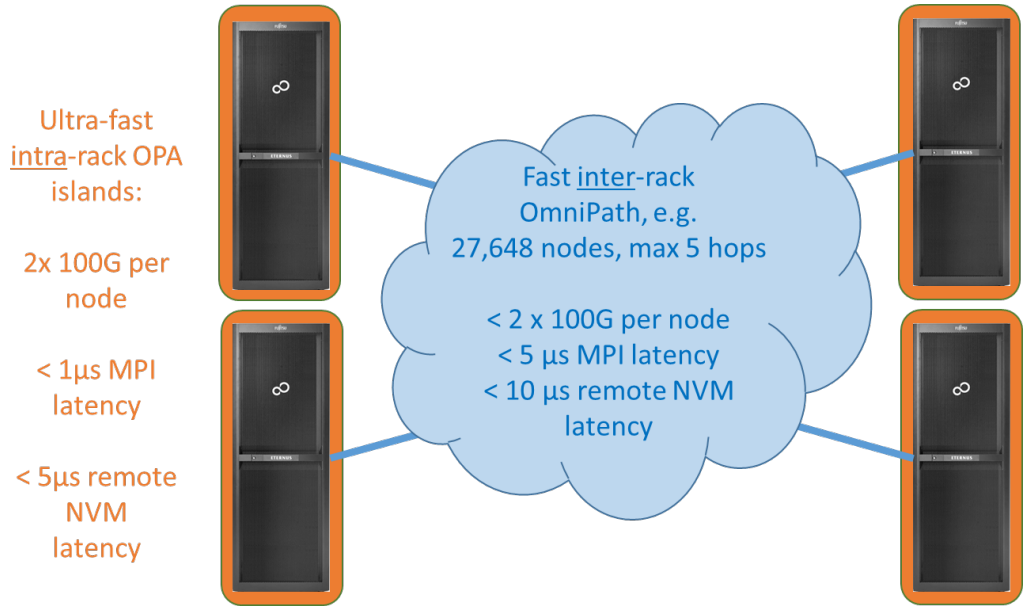

Figure 12: Intra-rack vs. inter-rack connectivity.

Since the bandwidth per node may be limited by this inter-rack bandwidth restrictions, the software should be aware of this extra network level. The scheduler and applications should therefore ensure that inter-rack accesses are less frequent than intra-rack accesses. This is very similar to today's NUMA memory access optimisation within compute nodes.

6 Systemware Architecture

The systemware for the NEXTGenIO architecture includes all the software that runs on the system (excluding user applications). As described in Section 5, there are four main components in the prototype:

- Login nodes
- Boot/Management nodes
- Gateway nodes
- Complete compute nodes

For large scale versions of the system we are designing, we would expect there to be a fifth type of node, Job Launcher/Scheduler nodes, which would run the systemware components that schedule and run user jobs, and collect performance data from the system as a whole. For full-scale production systems, it would not be desirable to run the scheduling and monitoring systems from the login nodes as user behaviour could compromise the scheduling components and interrupt the running of jobs on the system, and the small number of login nodes might be overwhelmed by the load created by monitoring, orchestrating and scheduling a very large system. Furthermore, separating out the login and job scheduling/launching nodes allows the enforcement of different levels of security on those components, thus facilitating a secure system overall. However, given the scale of our prototype, this functionality has been assigned to the login nodes instead of requiring a separate set of nodes.

There are also other hardware components that are important to the prototype system we are developing, but are not part of the core systemware architecture, specifically:

- Network switches, these will require management functionality in our prototype system, but they are not part of what we consider the core NEXTGenIO systemware as they are simply a component of the network they belong to. We will use standard components for the selected network fabric.

We have added these additional components to the systemware diagrams below, but they are not described them in the following sections; we view them as standalone components that are not key to the architectures we are designing in this project (i.e. they could be replaced with other components and still provide the same functionality to our system).

6.1 Systemware Overview

The systemware provides the software functionality necessary to fulfil the requirements we have identified. It provides a number of different sets of functionality, related to different ways to exploit SCM.

Briefly, from a user's perspective the system enables the following scenarios:

1. Users can request systemware components to load/store data in SCM prior to a job starting, or after a job has completed. This can be thought of as similar to current burst buffer technology. This can address the requirement for users to be able to exploit SCM without changing their applications.
2. Users can directly exploit SCM by modifying their applications to implement direct memory access and management. This offers users the ability to exploit the best performance SCM can provide, but requires application developers to undertake the task of programming for SCM themselves, and ensure they are using it in an efficient manner.
3. Provide a filesystem built on the SCM in the CCNs. This allows users to exploit SCM for I/O operations without having to fundamentally change how I/O is implemented in their applications. However, it does not enable the benefit of moving away from filesystem operations that SCM can provide.

4. Provide an object store that exploits the SCM to enable users to explore different mechanisms for storing and accessing data from their applications.
5. Enable the sharing of data between applications through SCM. For example, this may be sharing data between different components of the same computational workflow, or be the sharing of a common dataset between a group of users.
6. Ensure data access is restricted to those authorised to access that data and enable deletion or encryption of data to make sure those access restrictions are maintained
7. Provision of profiling and debugging tools to allow application developers to understand the performance characteristics of SCM, investigate how their applications are utilising it, and identify any bugs that occur during development.
8. Efficient check-pointing for applications, if requested by users.
9. Provide both 1LM and 2LM modes if they are supported by the SCM hardware.
10. Enable or disable systemware components as required for a given user application to reduce the performance impact of the systemware to a running application, if this is not using those systemware components.

We have not designed the systemware to support the following functionality:

1. Automatic copying or mirroring of application data between nodes without a job explicitly requesting it.

The systemware architecture we have defined is outlined in the following diagram. We will describe it in more detail in the following sections. Each distinct type of node in the system has its own part in the systemware architecture.

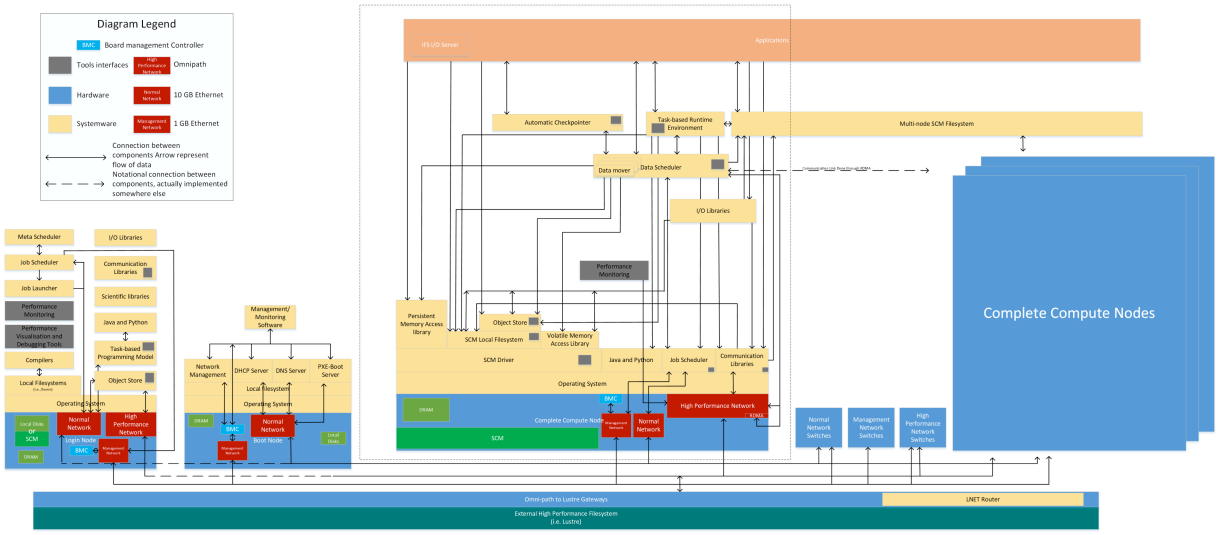

Figure 13: Systemware architecture.

6.2 Login Nodes

The primary purpose of the login nodes is to provide resources for users to perform data movement, source code compilation, job submission, and job management. Users will be able to log into these nodes, and will have access to a local filesystem on the login nodes as well as to a global parallel filesystem, which is also accessible from the CCNs.

The systemware components for the login nodes are shown in Figure 14, which is a subsection of the overall systemware architecture shown in Figure 13.

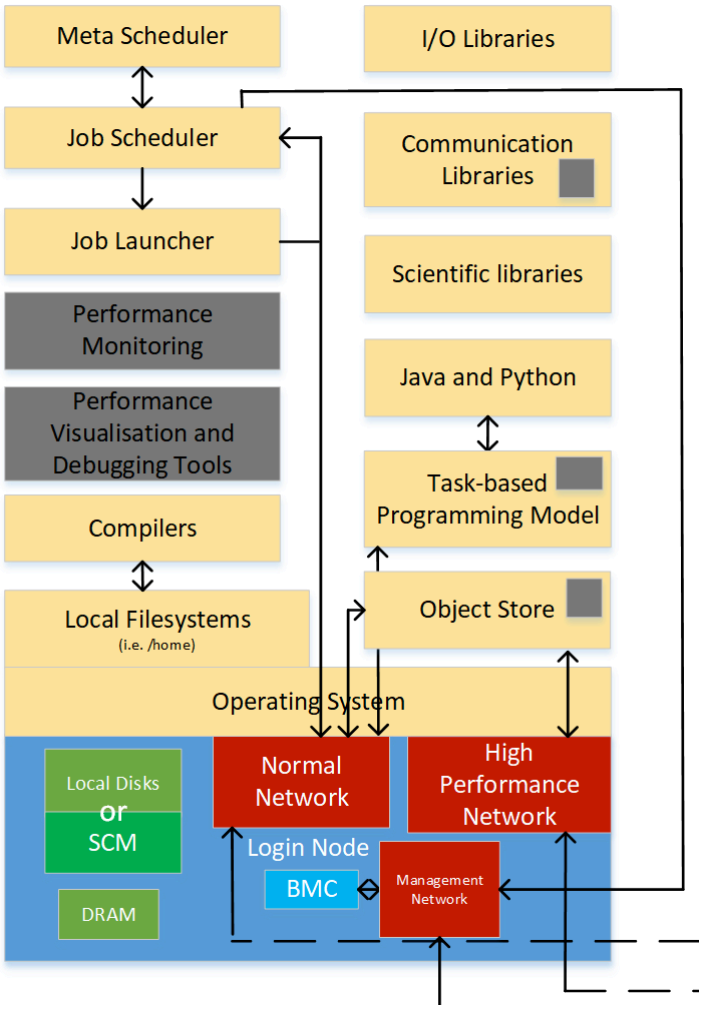

Figure 14: Login node systemware components.

6.2.1 Communication Libraries

Compatible MPI libraries/functionality is essential to support the HPC applications that will be using the system. As many parallel applications use MPI-I/O functionality the MPI library should support at least the MPI-2.0 APIs and functionality, and MPI-I/O should be optimised to work with SCM.

6.2.2 Compilers

Applications need to be compiled to use particular hardware platforms. Compute nodes for HPC systems generally are not setup for direct user access or for compilation, so login nodes are required on a HPC system to let users build/compile their applications, and launch their jobs through the batch system.

Compilation can require reading and writing of large numbers of small data files, which can be slow on parallel filesystems. SCM could speed up such compilation by providing a faster filesystem for such compilation.

HPC applications are generally C, C++, or Fortran programs. As such it is essential to have support for those three languages in the compilers on the system. There may be varying levels of standards usage and compliance, particularly in the C++ and Fortran codes (i.e. C++11 or FORTRAN2015 features could be used by some codes), so it is necessary to understand the language support for C++ and Fortran that the available compilers provide.

HPC applications generally use some parallel functionality, primarily MPI and OpenMP (although other parallel functionality is used, such as OpenACC, OpenCL, CAF or UPC). Therefore the compiler needs to also support linking to communication libraries for MPI (and other distributed memory parallelisation approaches) based programs, and to node-local parallelisation libraries/functionality such as OpenMP.

6.2.3 I/O Libraries

Many applications use I/O libraries, such as HDF5 or NetCDF, to simplify I/O for particular application areas, to provide metadata for datasets, or to integrate data with other applications/tools.

I/O libraries may be built on top of other I/O libraries (i.e. HDF5 may build on MPI-I/O for parallel functionality, and NetCDF on top of HDF5).

6.2.4 Java

Java is a general-purpose, concurrent, class-based and object-oriented computer programming language which has gained an important popularity, and consequently it is being widely adopted within the scientific community. The latest Java version is 8.

Some new task based programming environments, such as COMPSs, are implemented in Java, consequently Java support must be provided within the login nodes in order to be able to execute it enable such environments within the login nodes (e.g. submit a PyCOMPSs application to the queuing system and set the environment within the assigned CCN).

6.2.5 Job Launcher

For performance and security reasons the CCNs cannot be accessed directly by users. Jobs are submitted using a job scheduler which takes input from the user (via a job scheduling/batch script) which specifies the application to be run, the resources to be used, and other data that is required for successful running of the job. The batch system then schedules and runs the job on the CCNs when resources are available and following defined scheduling policies.

Often the application is actually run by a job launcher, a program specified by the user in the batch script which starts the parallel application on the CCNs that the batch system has allocated to the job. The job launcher generally is part of parallel libraries on the system, specifically the MPI libraries and associated ecosystem.

6.2.6 Job Scheduler

A job scheduler is responsible for allocating nodes to run parallel jobs. In order to provide this basic functionality, a job scheduler must:

- Provide authentication and authorisation for users and groups
- Provide accounting and historical record keeping
- Implement several scheduling algorithms
- Allow interactions by providing a set of helper application (and a programmatic API)
- Provide a backup job scheduler for a fault-tolerant configuration
- Allocate and run jobs in a timely manner
- Manage and track data associated with user jobs

The job scheduler must be aware of SCM resources in the system, allow users to specify SCM requirements and data movement requirements, understand the configuration of CCNs and allow users to specify configuration requirements of CCNs for their jobs.

The job scheduler will communicate with, and control, job scheduling and data scheduling components on the CCNs, to enable the system to run in as efficient a manner as possible and ensure user data is secure and safe.

There will be multiple scheduling components in the job scheduler, including components that can schedule jobs based on data location or energy characteristics (the data aware job scheduler and the energy aware job scheduler).

6.2.7 Local Filesystem

For performance reasons, users of HPC systems should have access to both a high performance parallel filesystem (/work), such as Lustre, as well as a standard backed-up NFS filesystem (/home).

CCN may have access only to the high performance filesystem, while login nodes have access to both so that the user can securely store data on a backed up system while not putting unnecessary strain on the high performance filesystem.

Compilation can create a large number of small files, especially when compiling large programs, and this can have a significant impact on the performance of parallel filesystems. Having a more local filesystem on the login nodes, especially one that can be backed up if necessary, is essential to enable efficient compilation and high performance applications on the CCN.

6.2.8 Meta Scheduler

The meta scheduler coordinates the scheduling decisions between the Data Aware Job Scheduler and the Energy Aware Job Scheduler. For example, instead of moving data it can schedule jobs where the data is located, or decide to stage-in and stage-out data during the execution of the job if there is spare network capacity. This functionality will use the job scheduler, and through it the data schedulers on CCNs, providing dynamic, adaptive, scheduling to maximise the efficient use of the machine.

6.2.9 Object Store

Any object store we have running in the system may require tools available on the login nodes for users to access, query, update, and remove data from the object store. Therefore, we should have the ability to run object store access software on the login nodes.

Furthermore, some object stores have management and control functionality that can be used to administer and monitor the object store. The login node would be the appropriate location for that functionality as well.

6.2.10 Operating System

The operating system on the Login nodes requires the functionality to provide external access to the system; undertake compilations and visualisation/graphical operations. Furthermore, the meta and job schedulers will run on the login nodes.

The login nodes may have their own local boot media as well as the potential to PXE-boot from the boot node. Beside standard functions the O/S must support the CPUs that support SCM, a GPU as well as the high performance network interface.

6.2.11 Performance Monitoring

Studying system and application performance requires a performance-monitoring component. These performance monitoring interfaces will be responsible for performance data collection during the runtime of an HPC application. This covers both data about the application, and data on the actions of the systemware.

In many cases the performance monitoring infrastructure is integrated into, or attached to, the applications to be monitored. For certain performance aspects, e.g. the scheduling of jobs, it is necessary to provide a self-contained monitoring component that runs independently as part of the systemware.

6.2.12 Performance Visualisation and Debugging Tools

Performance profiling is statistical analysis to study the overall performance of HPC applications. Performance tracing allows the recording of individual events from an application. The resulting trace file enables in-depth analysis of the application's runtime behaviour and is typically visualised using a time series. In contrast, debugging tools provide access to an application while it is running to address correctness issues as they happen.

All three forms of performance tracking require a sophisticated graphical interface and various systemware components to provide their service. In this project the tools Allinea Map, Vampir/Score-P, and Allinea DDT will be deployed. Together, they provide comprehensive in-depth analysis aiding to resolve errors, bottlenecks and perform optimisations.

Vampir/Score-P is a framework for performance analysis, which enables developers to quickly study the application behaviour at a fine grain level and perform in-depth root-cause analysis. This level of detail is achieved by means of recording performance events and storing them to the trace file. An important feature of Vampir is the intuitive and graphical representation of detailed performance events on a time axis and aggregated profiles.

These tools differ from the monitoring component of Section 6.2.11 in that they are used by application developers during code development and optimisation, rather than the data collection components of performance monitoring/debugging.

6.2.13 Python

Python is a free, interpreted, object-oriented, high-level programming language with dynamic semantics. It is characterised by enabling the rapid application development, as well as being used as scripting or glue language to connect existing components together. It is becoming widely adopted within the scientific community due to its quick learning and development process, as well as for the existing scientific libraries, such as NumPy and SciPy.

In order to run Python applications, a Python interpreter is needed within the login nodes. Thus, in order to provide support for scientific applications, some libraries may be of interest (e.g. NumPy or SciPy).

Given that the applications that will be executed in NEXTGenIO that take advantage of PyCOMPSs are Python applications, we need Python support within the login nodes.

6.2.14 Scheduling Tools

Scheduling tools are used to interact with the job scheduler (either by invoking a command/application or programmatically using a well-defined API). Interactions with a job scheduler include:

- Submitting a batch script or single application for a parallel run
- Transmit a file to nodes allocated for a particular job
- Query the scheduler about the current state of jobs/nodes
- Cancel a running or pending job
- Get/update the state of jobs/nodes
- Display accounting/historical data about nodes/jobs/users

6.2.15 Scientific Libraries

Many of the operations carried out in scientific applications can be composed from standard components. These have been developed and optimised extensively by domain specialists as well as hardware and compiler vendors. High-performance computational libraries are often optimised for the particular systems they are to be used on.

6.2.16 Task-based programming model

New programming models for exploitation of large scale HPC systems aim to tackle programmability and performance for very large numbers of program processes or threads. One such approach is to use a task-based programming model, where developers specify the tasks required for their application and the runtime environment of the programming model distributes and runs these tasks in as efficient way as possible.

One example of such a model is PyCOMPSs, which is the Python binding of COMP Superscalar (COMPSs), a task-based programming model that aims to ease the development of applications for distributed infrastructures.

It offers a simple programming model based on sequential development: a PyCOMPSs application is a plain sequential Python script with annotations that define tasks and their data dependencies. In the model, the user is mainly responsible for (i) identifying the functions to be executed as asynchronous parallel tasks and (ii) annotating them with a standard Python decorator.

The login nodes must have all the components of the task-based programming model required for development of applications and submissions/management of jobs based on this technology.

6.3 Gateway Nodes

The primary purpose of the gateway nodes is to provide access to external storage systems. The nodes provide high bandwidth gateway functionality from IB to OPA and vice versa.

6.3.1 Operating System

The operating system on the Gateway nodes requires the functionality to provide external access to storage systems. For the prototype system, the bridging software running on the Gateway nodes will be the LNET Router software from Intel. Beside standard functions this component must support that LNET router software, the new processors to be used in the prototype, and the network interfaces required to bridge between the main compute network and the external storage system network.

6.4 Boot Nodes

The boot nodes provide management and software provisioning functionality for the system. They will provide the CCNs with software images or downloads to enable CCNs to operate without local storage.

They will only be accessible by system administrators. The boot node systemware components are outlined in Figure 15.

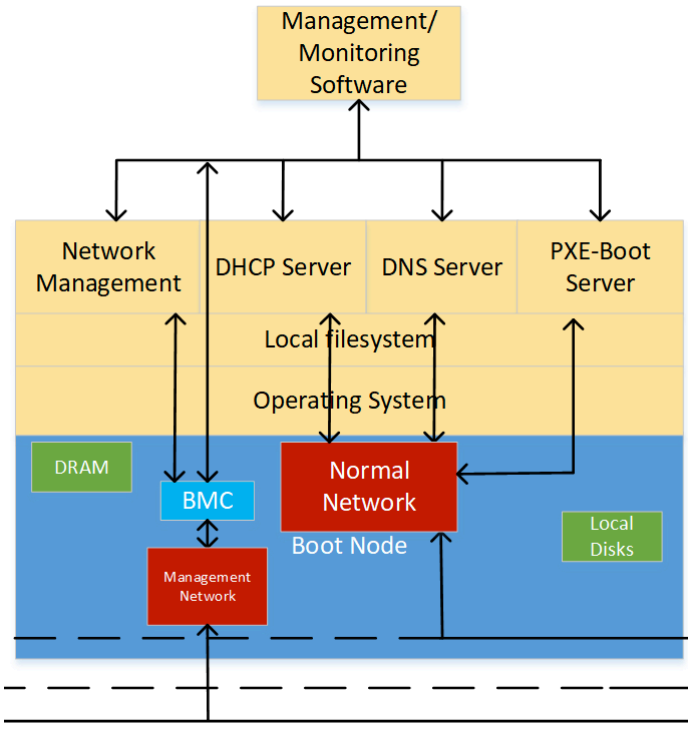

Figure 15: Boot node systemware components

6.4.1 Operating System

The operating system on the Boot nodes requires the functionality to host the Pre-boot Execution Environment (PXE) to provide boot service via Trivial File Transfer Protocol (TFTP) to other nodes.

The boot nodes will also offer Domain Name System (DNS) and Dynamic Host Configuration Protocol (DHCP) services within the prototype domain. The boot node will host management and monitoring software and is the central hub of the NEXTGenIO prototype management system.

This includes CCN, Ethernet network and Omni-Path Fabric management. The boot nodes will be used for management access to all other nodes, the Ethernet switches and the Omni-Path switches. Therefore the operating system will need to support the management, monitoring, DNS, DHCP, and PXE-Boot software that will be run on it.

6.4.2 Management/Monitoring Software

The CCN and login nodes within the system require some software to manage their operation, start-up and shutdown. This includes monitoring the health of individual nodes, provisioning software images to boot compute nodes from, and managing the lifecycle of the whole system. Several different interfaces to the board management controllers (BMC) on each node are available.

The traditional Intelligent Platform Management Interface (IPMI) over LAN, Simple Network Management Protocol (SNMP) or the new RESTful Redfish (DMTF) can be used to retrieve and set management data and functions via remote out-of-band (Management LAN) interface.

6.4.3 PXE-Boot Server

The boot server provides functionality for CCNs, or login node, to boot via the network rather than having to have an operating system installed locally. This, in turn, means that the CCNs can operate without local storage/disks attached.

6.4.4 DHCP Server

Dynamic Host Configuration Protocol (DHCP) is a client/server protocol that automatically provides a host with its IP address and other related configuration information such as the subnet mask and default gateway.

This component provides IP addresses on the normal network for the CCN, login, and boot nodes in the system.

6.4.5 DNS Server

The DNS server is any computer that provides a name service for a set of computers on a network. A DNS server run special-purpose networking software with a public IP address, and contains a database of network names and addresses for other hosts.

6.4.6 Network Management

Software that configures and monitors the switches and networks used in the system is essential to ensure the correct running of the prototype. This will involve managing all of the networks we will deploy in the system through the management network.

6.4.7 Local Filesystem

The services on the boot node, such as the PXE-Boot service, need a place to store data (such as the boot images).

6.5 Complete Compute Nodes

The CCNs provide the computational resource for the system. They will not be directly accessible by users; user jobs will be run through scheduling components installed on the nodes.

There will be a large number of CCNs in the system and they will all have the same software configuration(s). It is expected that users will be able to reconfigure CCNs for different modes of hardware operation (rebooting between different SCM configuration modes), and potentially to enable or disable different systemware components (such as the object store or SCM filesystem if those not in use).

Therefore, whilst the defined systemware architecture for the CCNs will be uniform, different components may be active or enabled on different CCNs at any given time, depending on the jobs that are being run on the system.

The NEXTGenIO architectures are also designed to allow some level of heterogeneity in the configuration of CCNs, with the potential to either have the same amount of SCM across all nodes, or to have systems where some nodes have no SCM. This is to allow systems to be deployed where SCM is provided in a subset of the CCNs, with the benefits of SCM available across the system using the functionality of the systemware NEXTGenIO is designing and implementing.

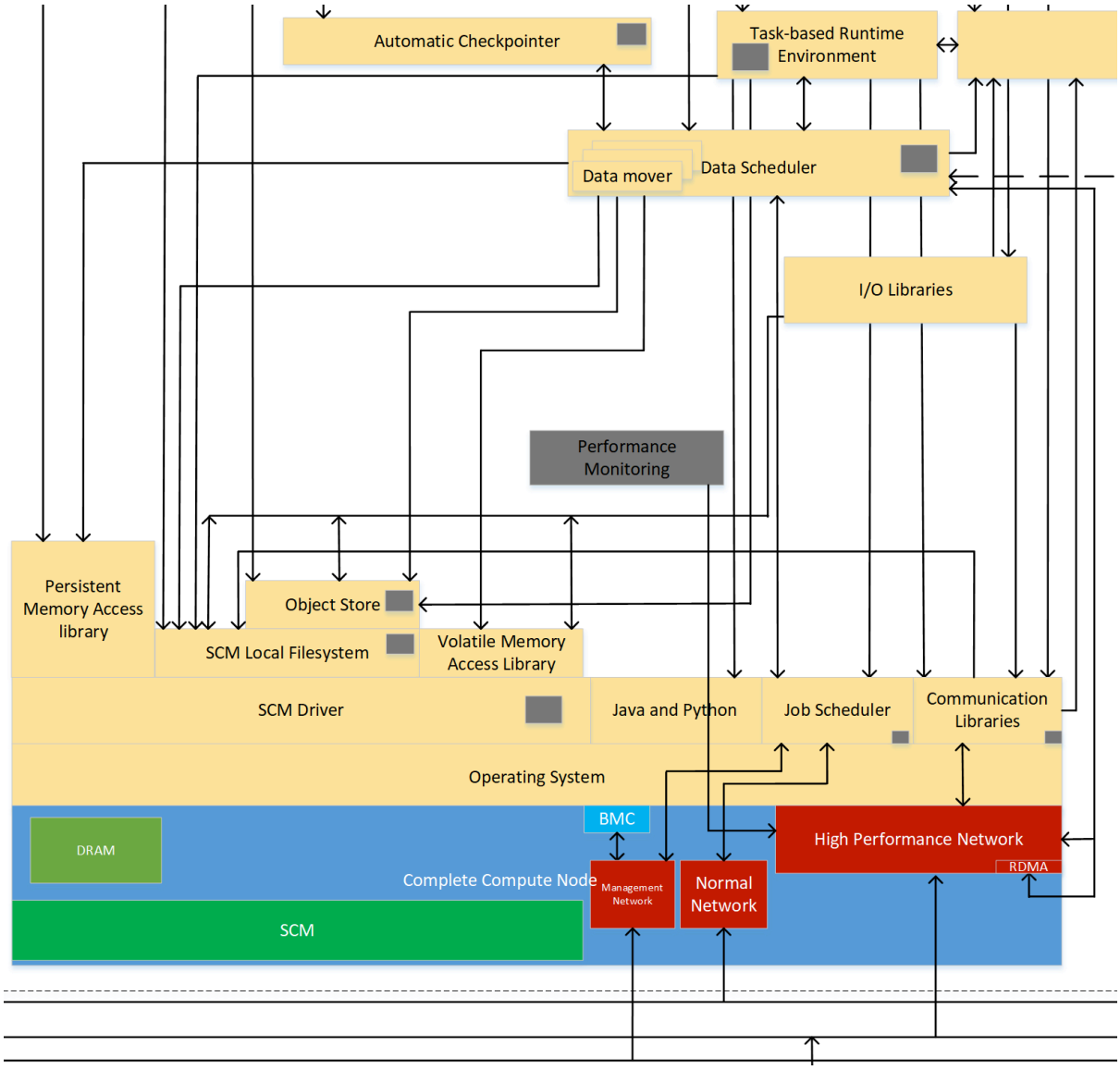

Figure 16: Complete compute node systemware components

6.5.1 Automatic Check-pointer

Applications that run for long periods of time (over a few hours) generally have to check-point (write some data to a persistent storage system, like a filesystem, that can be used to restart the application if necessary) to cope with hardware failures and job queue limits.

Currently applications perform their own check-pointing, however with SCM on node it could be possible for the automatic check-pointing system component to automatically check-point whole application instances into SCM periodically to remove the need for applications to check-point themselves.

6.5.2 Communication Libraries

Communication libraries must be available on the CCNs for use by HPC and HPDA applications if they have been compiled dynamically (i.e. the operating system looks for the library object files when the application runs rather than including them at compile time).

6.5.3 Data Movers

Data movers are used by the data scheduler to undertake specific data operations. This could be copying data between different filesystems, from local to remote CCNs, and between different data storage targets (for instance between an object store and a filesystem).

There will be different data movers for different operations (i.e. one for moving files around, or another for moving data to a remote CCN).

6.5.4 Data Scheduler

The data scheduler can migrate data between different hardware levels, within a single node, independently from an application, and between different nodes in the system, specifically to and from SCM on other nodes over the high performance network (probably using RDMA). This functionality can be requested by other software components (including applications).

The data scheduler also keeps track of data and what hardware that data is located in. On the CCN the data scheduler component provides the local functionality to move data between storage levels (e.g. filesystem, SCM, DRAM) as instructed by the higher level scheduling component (the job scheduler on the CCN in coordination with the master job scheduler).

The data scheduler is primarily associated with moving data for a given job or application, this may be staging data in prior to a job starting or staging data out after a job has finished.

The data scheduler is also responsible for handling remote data accesses.

6.5.5 I/O Libraries

Many applications use I/O libraries, such as HDF5 or NetCDF, to simplify I/O for particular application areas, to provide metadata for datasets, or to integrate data with other applications/tools.

I/O libraries may be built on top of other I/O libraries (i.e. HDF5 may build on MPI-I/O for parallel functionality, and NetCDF on top of HDF5).

6.5.6 Java

See subsection 6.2.4.

6.5.7 Job Scheduler

A job scheduler agent running on the CCNs is responsible for running an application which is part of a parallel job. In order to provide this basic functionality, a job scheduler agent must:

- Be able to update the current configuration environment of a node;
- Execute a parallel application;
- Monitor all tasks running on a node;
- Kill/cancel tasks running on a node;
- Communicate with the master job scheduler (in order to provide updates);
- Accept commands from the master job scheduler;
- Be able to act if communication with the master job scheduler is lost.
- Be able to communicate with the data scheduler to understand the current state of data in the system
- Pass commands to the data scheduler

6.5.8 Management/Monitoring Software

The compute and login nodes within the system require some software to manage their operation, start-up and shutdown. This includes monitoring the health of individual nodes, provisioning software images to boot CCNs from, and managing the lifecycle of the whole system.

This component runs on each of the CCN and is used by the management/monitoring software components on the boot nodes to undertake management or monitoring operations on the CCNs.

6.5.9 Multi-node SCM Filesystem

The multi-node SCM Filesystem provides a lightweight POSIX filesystem that has two main goals: first, to hide the complexity of the different levels of local storage present in a computing node (i.e. DRAM, SCM and high performance parallel filesystem) by aggregating them into a virtual storage device and, secondly, to allow computing nodes to export SCM regions so that they can be used transparently by other nodes, thus allowing for a larger storage capacity and the improved efficiency of collaborative, multi-node I/O operations.

The rationale behind this software component is that there are many legacy HPC applications which may be hard or costly to modify to use SCM technology, it would be desirable that they could at least get some of the benefit of the NEXTGenIO architecture without any modifications.

Additionally, the multi-node SCM Filesystem offers the following opportunities and advantages: (1) policies can be added to automatically control the level at which data is stored (DRAM, SCM or high performance parallel filesystem) and when it should be migrated; (2) the opportunity to distribute I/O between the SCM of a set of nodes and keep the results in a "cooperative burst buffer", which could be reused between jobs or written to long-term storage when the cluster load is appropriate; and (3) offer an API to schedulers and applications so that they can better communicate their needs to the filesystem.

6.5.10 Object Store

Recently some HPC applications have started using object stores to manage data instead of, or as a complement to, traditional file systems. Object stores provide applications with variable granularity data in the form of individually identified objects instead of files in a directory hierarchy. Such object stores appear to be a good match to distributed storage, especially one that can provide user-level byte-granularity access (as SCM technology offers).

Intel have been working on one such object store, DAOS. Another example is dataClay, a data store developed by BSC that allows accessing persistent objects as if they were in memory. dataClay can store functions or code-snippets, as well as data, that can be used by the PyCOMPSs task-based programming system.

6.5.11 Operating System

The operating system on the CCNs must be configured to allow high performance computations and to support new non-volatile main memory functionality. It must allow the implementation of new storage paradigms such as direct load/store access of persistent main memory, without page caches, but also distributed shared storage based on SCM via a high-speed low latency high bandwidth interconnect.

To avoid traditional local storage devices such as HDD or SSD the O/S must support a memory disk to boot straight from the boot loader on the boot node. As well as the standard functions the O/S must support the SCM hardware, processors that can support SCM, and the network interface used for the main compute network.

6.5.12 Performance Monitoring

This performance monitoring component is responsible for collecting performance information from an application execution. In the first case, performance monitoring component attaches itself to the execution of the application. Different states of the program under observation are inspected by periodically interrupting the running program. Examples would be `setitimer()` or overflow triggers of hardware counters that are triggered periodically to observe the application execution.

In order to inspect the execution state of an application, call-path and information from the hardware performance counters are utilised. Call paths provide information regarding the functions and regions within the application whereas performance counters provide information regarding the hardware related activities. The performance monitoring component directly attaches itself with the interfaces which can provide information regarding the application execution. These interfaces include PAPI, hwloc, netloc, procfs and NVML.

In some cases, this component will query hardware resources on the node. For this purpose, systemware monitoring interfaces, such as interfaces to the scheduler, are used. The scheduling API provides a monitoring mechanism to interpret the overall resource consumption of the application on a node.

Moreover, the scheduler will also be responsible for managing the optimal utilisation of SCM resources, requiring it to maintain up to date information on the status of SCM. Consequently, the information collected by the scheduler can be passed to the performance monitoring component.

6.5.13 Persistent Memory Access Library

One way to provide simpler and reusable SCM access and management methods is to use a library suite, which can then be used by both user applications and systemware. Among the core expected functionality, such libraries should provide mechanisms for applications or systemware to create and manage persistent memory regions and their keys, including across power cycles.

The library should also provide mechanisms to facilitate storage of persistent data, such as mechanisms for reliable transactional updates. Optionally, the libraries could provide methods for manipulation of common higher-level data structures in persistent memory, such as lists and trees and even simple key-value stores.

6.5.14 Python

See subsection 6.2.13.

6.5.15 SCM Driver

Just like with any hardware device, the SCM must be exposed to the rest of the system and the O/S through a specific device driver. We expect this to be provided by the SCM vendor (in this case Intel). We do not expect to need to perform changes to the device driver.

6.5.16 SCM Local Filesystem

One of the most straightforward methods to exploit SCM for persistent storage in each compute node is to use a local filesystem. While any filesystem can be used with the SCM, in order to exploit the byte-level load/store access to SCM it is necessary that the local filesystem supports the DAX extensions. These extensions allow users to bypass all the traditional block oriented structures in the I/O stack, such as the page cache.

6.5.17 Task-based Runtime Environment

Task-based programming models generally require a runtime component that executes and controls tasks associated with a task-based program on compute nodes.

The PyCOMPSs runtime manages the user application exploiting the inherent concurrency of the script, automatically detecting and enforcing the data dependencies between tasks and spawning these tasks to the available resources, which in this scenario is considered to be able to handle persistent objects across the infrastructure.

6.5.18 Volatile Memory Access Library

While the system can transparently provide large volatile memory to applications on top of combined DRAM and SCM (in 2LM mode), some applications may wish to manage their own volatile data and DRAM.

One example would be applications explicitly storing frequently accessed data in DRAM and less frequently accessed data in NVRAM, but not requiring persistency of the data in NVRAM. Another example would be applications implementing a software cache using NVRAM of data mirrored in external storage devices. In such cases, a mechanism is required to allow applications to specify where (either DRAM or NVRAM) memory should be allocated.

7 Usage Patterns

To allow a completer understanding of how a system developed from our architectures can be used, we outline some of the possible usage scenarios we have developed during the requirements gathering and architecture design processes in the project.

The following sections provide details on the functionality that the hardware and systemware can offer applications. We build on use cases, and associated usage scenarios, we have identified to outline the systemware and hardware components used by a given application, and the lifecycle of the data in those components.

For each usage scenario, we describe which systemware components are active, how data flows through these components and the user application, where data resides, and the state of the system and data after the application has finished. Sequence diagrams that illustrate the data lifecycle can be found in the Appendix (see page 51).

7.1 General Purpose HPC system using multi-node SCM filesystem

7.1.1 Description

Many HPC systems host a wide range of jobs, from small single node jobs to full system jobs. For the UK national HPC service that UEDIN operate we see a breakdown of jobs that is roughly as follows:

- Breakdown by compute time
  - 15% < 96 cores
  - 80% < 96-12000 cores
  - 5% > 12000 cores
- Breakdown by number of jobs submitted
  - 70% < 96 cores
  - 29% < 96-12000 cores
  - 1% > 12000 cores

As there is a wide range of jobs on the system, we see a wide range of I/O patterns, from applications that do very little I/O to those that are dominated by I/O. We estimate the I/O cost is less than 10% for the majority of applications. Most applications will perform check-pointing to cope with the maximum runtime limit on the system.

Such a system generally uses a parallel filesystem and there are no disks in the compute nodes. All I/O is done to the parallel filesystem over the high performance communication network fabric used for inter-process communications.

As this is a general purpose usage scenario there is potential for applications running on a system like this to use all the different components available in the NEXTGenIO systemware and hardware architecture. However, for this example we will restrict ourselves to applications that have not been modified, are simply re-compiled, and write their data to files in a standard filesystem. Traditionally such applications would use the parallel filesystem attached to a HPC system to perform their I/O, would be run using a batch system, and would operate within the resource limits placed on them by that batch system.

We assume that applications use the multi-node SCM filesystem for I/O whilst the application is running, and the external high performance filesystem for data storage after an application has finished.

For this usage scenario we are using SCM in 1LM node.

7.1.2 Components

- Job Scheduler

- Multi-node SCM filesystem
- I/O Libraries
- Communication Libraries

7.1.3 Data lifecycle

1. Scheduling requests submitted.
   a. At this point the executable to be run is either stored on the external high performance filesystem or on the local filesystem on the login nodes, and the data for the application is stored on the external high performance filesystem.
2. Job scheduler allocates resources for an application.
   a. The job scheduler ensures that nodes are in 1LM mode and that the multi-node SCM filesystem component is operational on the nodes being used by this application.
3. Once the nodes that the job has been allocated to are available the data scheduler copies input data from the external high performance filesystem to the multi-node SCM filesystem.
   a. This step is optional; an application could read it's input data directly from the external high performance filesystem, but using the multi-node SCM filesystem will deliver better performance.
4. Job Launcher starts the user application on the CCN.
5. Application reads data from the multi-node SCM filesystem.
6. Application writes data to the multi-node SCM filesystem.
7. Application finishes.
8. Data Scheduler is triggered and moves data from multi-node SCM filesystem to the external high performance filesystem.

7.2 General Purpose HPC system improving throughput

7.2.1 Description

This is a general purpose usage scenario, meaning there is potential for applications to use all the different components available in the NEXTGenIO systemware and hardware architecture. However, for this example we are considering how to use the NEXTGenIO systemware to optimise the overall utilisation or energy efficiency of the system.

We are assuming that CCNs are in varied states, with various data loaded on to SCM on different CCNs. The schedulers are used by the meta scheduler to ensure that efficient use is made of the whole machine or particular resources that are of interest (i.e. total energy consumed, power limits, or time for throughput of jobs).

7.2.2 Components

- Data Aware Job Scheduler
- Energy Aware Job Scheduler
- Meta Scheduler
- Multi-node SCM filesystem
- I/O Libraries
- Communication Libraries

7.2.3 Data lifecycle

1. Scheduling requests submitted.
   a. At this point the executable to be run is either stored on the external high performance filesystem or on the local filesystem on the login nodes, and the data for the application may be on the external filesystem or on SCMs somewhere within the system.

2. Meta scheduler is informed of the job scheduling requests and undertakes an analysis of what resources would be most efficient to use for this job taking into account the constraints that the meta scheduler assigned.
3. Meta scheduler generates job and data scheduler instructions.
4. Job scheduler allocates resources for an application.
5. Once the nodes that the job has been allocated to are available the data scheduler copies input data from the external high performance filesystem to the multi-node SCM filesystem.
    a. This step is optional, the data may already be on-node or may still be on the external filesystem.
6. Job Launcher starts the user application on the CCN.
7. Application reads data from the multi-node SCM filesystem.
8. Application writes data to the multi-node SCM filesystem.
9. Application finishes.
10. Data Scheduler may be triggered to move data from SCM storage to the external high performance filesystem.

7.3 Resilience in specialist HPC system

7.3.1 Description

Another category of HPC system are those that are used for a single academic or industrial discipline. This type of system can still be very large, but will typically run a smaller number of longer jobs, and each job will be larger than is usually seen on shared HPC system. Check-pointing is critical for the safe operation of jobs to ensure that any hardware failure does not result in losing a large job.

Applications currently manually check-point their running state and datasets to enable restart on failure of jobs or jobs reaching batch system time limits. SCM can allow for O/S-level check-pointing, rather than application level check-pointing, with the O/S triggering a dump of an application to SCM memory for future use independently of the application itself. It can also allow for applications to check-point to SCM manually.

For this usage scenario, we are using the SCM for check-pointing of running applications to ensure that applications can be recovered after hardware failures. In this context, we are using the SCM in 1LM mode, although 2LM mode with an App-Direct region is also possible.

7.3.2 Components

- Job Scheduler
- Automatic Check-pointer
- Communication Libraries
- I/O Libraries

7.3.3 Data lifecycle

1. Job scheduling request submitted.
    a. Scheduling request includes automatic check-pointing requirements.
2. Job scheduler allocates resources for an application.
3. Once the nodes the job has been allocated to are available the job launcher starts the user application.
4. The user application notifies the automatic check-pointer that check-pointing is required, and at what frequency.
    a. Some applications may notify the automatic check-pointer when check-pointing can occur.
    b. Others may let the check-pointing occur as decided by the check-pointer.
5. Two scenarios are now possible.

a. Application completes successfully.
    i. Scheduler notifies the automatic check-pointer that check-pointers are no longer required.
    ii. Check-pointer removes check-points.
b. Application failed for some reason.
    i. Re-run job submitted with a check-point ID provided to resume the run.
    ii. Application runs to completion as above.

If check-point data was written to the multi-node SCM filesystem instead of straight to SCM then it is possible that jobs could be restarted without failed nodes provided the multi-node SCM filesystem has provided data backup/replication/migration functionality to ensure the check-point data is available on multiple nodes, not just the node it was written on.

7.4 Large Memory

7.4.1 Description

Applications areas such as bio-informatics often rely on large scale shared memory systems to enable very large data sets to be loaded and processed. For these applications, reading and writing data from/to storage systems takes a long time, so reducing this cost is important.

For this example, we are considering using the SCM as a very large memory space to perform in-memory processing of full datasets, reading the data into memory once and then writing the data back to the high performance filesystem at the end of the computation. In this context, we are using the SCM in 2LM mode and a single job will only use one node in the system.

7.4.2 Components

- Job Scheduler
- I/O Libraries

7.4.3 Data lifecycle

1. Job scheduling request submitted.
    a. Scheduling request includes information that 2LM mode is required.
2. Job scheduler allocates a node for the application.
3. Once the node that the job has been allocated to is available the job scheduler ensures it is in 2LM mode.
    a. Either this is simply selecting a node already in that mode, or;
    b. The scheduler initiates node reboot as required to get the node into 2LM mode.
4. The job scheduler starts the user application on the node.
5. The user application loads data from the external high performance filesystem into memory.
6. The memory controller hardware loads data into SCM and uses the DRAM as cache for that SCM.
7. Application processes data.
8. Application writes data back to the external high performance filesystem.
9. Application finishes and job scheduler releases the node used.

7.5 Workflows

7.5.1 Description

Jobs from an increasing number of scientific areas are often characterised by a workflow/job chaining approach, where one application may read data, then produce output/write data, which is in turn used by another application. Reading and writing data to be passed between applications can takes a long time, therefore reducing this cost is important.

In this context we are using SCM in 1LM mode.

7.5.2 Components

- Job Scheduler
- I/O Libraries
- Data Scheduler
- Data Movers

7.5.3 Data lifecycle

1. Job scheduling request submitted.
    a. Scheduling request includes information that 1LM mode is required.
2. Job scheduler allocates nodes for the application.
    a. This may include restarting nodes to ensure they are in 1LM mode.
3. The job scheduler starts the first user application in the workflow on the nodes.
4. The user application loads data from the external high performance filesystem into memory.
    a. This may be SCM or DRAM.
5. Application processes data.
6. Application writes data to SCM.
7. Application finishes and informs Data Scheduler of data to be retained.
8. Data Scheduler takes control of data.
    a. This ensures data is not lost once the owning application finishes.
    b. The Data Scheduler or Scheduling Daemons will generate a key for future applications to use, to ensure the security of that data, and will return that key to the finishing application or user script.
9. Next application in the user workflow submitted to the job scheduler with notification that it requires previously written data.
    a. The application or script will use the previously generated key to take control of the data stored in SCM.
10. Job scheduler allocates nodes for the application.
    a. It is likely these will be the same nodes as those used by the previous application, but may not be. If they are different nodes, the job and data schedulers are responsible to ensure workflow data is available when the next workflow application starts.
11. The job scheduler starts the next user application in the workflow on the nodes allocated to that application.
12. Application processes data.
13. Application writes data to SCM.
14. Application finishes and informs Data Scheduler of data to be retained.
15. Data Scheduler takes control of data.
    a. This ensures data is not lost once the owning application finishes.
    b. The Data Scheduler or Scheduling Daemons will generate a key for future applications to use to, ensure the security of that data, and will return that key to the finishing application or user script.
16. Steps 9 – 13 repeated as required.
17. Final application in the workflow finishes.
    a. Data Scheduler uses Data Movers to write data to be stored back to the external high performance filesystem.
    b. Nodes released by job scheduler.

7.6 Data Analytics

7.6.1 Description

Data analytics usage cases generally have very large data sets that are analysed often, and the analysis performs small amounts of work per data point. There are therefore large numbers of data reads, and

small amounts of compute or data writes. Often the main overhead comes from loading the data prior to analysis, or from converting the data into a format that can be queried, rather than from the analysis or querying itself.

Once data has been loaded it is often queried or analysed many times. Persistence of large datasets in SCM is beneficial as it enables data to be available after reboots and hardware faults. Persistence of the analysis data (i.e. individual analysis runs) is not important. If an individual analysis run is lost that can be re-run.

7.6.2 Components

- Job Scheduler

7.6.3 Data lifecycle

1. Request submitted to job scheduler for data set to be loaded into SCM on CCNs from the high performance filesystem.
2. The job scheduler instructs specific CCNs data schedulers to load the specified data in to the appropriate place in the system.
    a. This could be local SCM on individual nodes.
    b. This could be the multi-node SCM filesystem.
    c. This could the object store.
3. The data scheduler loads the data and stores a reference to what data has been stored and on which CCN.
4. A user submits a request to the job scheduler to run an application.
5. Job scheduler allocates resources for an application.
    a. The scheduling takes into account the data required by the job when allocating resources by liaising with the data scheduler.
    b. If the data is not present on any resources, or the requested resources do not match where the data is available then the job is rejected.
6. Once the nodes that the job has been allocated are available the job launcher starts the user application.
7. The application uses the data available on the nodes.
8. The application writes any data it produces back to the high performance filesystem.
9. The application finishes.
10. Compute resources are freed up for use.
11. Steps 4-10 are repeated as required.
    a. Multiple jobs can run using the pre-loaded data at the same time as required.
12. Once the data is no longer required on the system the job scheduler is notified that the data can be removed.
13. The job scheduler notifies the data schedulers that they can remove the data.
14. The data scheduler removes the data.

7.7 Live process pausing and migration

7.7.1 Description

Batch systems for HPC are designed to allow a range of job lengths and sizes to be run. However, if there are urgent or large scale jobs waiting in the batch queue, this involves allocating resources for these future jobs and blocking resources until these jobs are ready to run (this is often called backfilling/queue draining).

SCM gives the possibility to pause live jobs by check-pointer their states from DRAM to SCM to free up resources to run urgent jobs quickly. This cuts down the backfilling/draining time for large scale or

urgent jobs. It is also possible for applications to be migrated between nodes through RDMA access to SCM.

To achieve this functionality we will use the automatic check-pointer and job scheduler work together to migrate a running application into a stored state in SCM, run a new application, and then restore the application from the check-point once the resources are once again free.

This relies on having a portion of the SCM available, regardless of whether 1LM or 2LM mode is being used, to ensure that there is sufficient space for a running application to be check-pointed.

7.7.2 Components

- Job Scheduler
- Automatic Check-pointer
- Job Launcher

7.7.3 Data lifecycle

1. Application is running on a set of CCN.
2. A high priority job is submitted to the job scheduler.
3. Job scheduler allocates a set of nodes to the high priority job.
4. Job scheduler notifies the automatic check-pointer on each of the allocated nodes to check-point the jobs.
5. Check-pointer takes a check-point and stores it in SCM.
6. Check-pointer notifies the job scheduler that the check-point has been taken and where it is.
7. Job scheduler kills the running application(s).
8. Job launcher starts the high priority application.
9. High priority application runs to completion.
10. Job scheduler reallocates the nodes to the previous jobs and resubmits their job scripts with restore from automatic check-pointing enabled.
11. Application(s) restarts.
12. Job scheduler notifies automatic check-pointer that check point data can be removed.
13. Application(s) runs to completion.
14. Nodes are released for future use.

7.8 Object Store

7.8.1 Description

Object stores are a data storage approach that manages data as objects, compared to traditional storage architectures like file systems, where data is managed as a file hierarchy, and block storage where data is dealt with in chunks, called blocks. As object stores provide storage that can mirror standard data access in most applications, they offer the potential for fine grain, and low overhead, data storage and access.

Matching the low overheads of object stores, compared to file access, with the high performance of SCM, when compared to traditional I/O devices, offers extremely high I/O performance and new ways of operating for HPC and HPDA applications.

This scenario involves an application that has been modified to use an object store, running on a system with SCM, with at least part of the SCM available for direct access from the object store.

7.8.2 Components

- Object store
- Job Scheduler

7.8.3 Data lifecycle

1. Object store service is initiated through the job scheduler.
2. Job is submitted requesting nodes with object store enabled.
3. Job scheduler checks which nodes have object store available.
4. Job scheduler allocates required nodes.
    a. Job submission fails if node request cannot be matched.
5. Job scheduler runs job on required node.
6. Job runs and uses object store.
7. Job finishes.
8. Steps 3-7 repeated as required.
9. Object store service is closed through job scheduler.
10. Data scheduler(s) can persist object store to external high performance filesystem if required.
    a. If not persisted then object store is lost at the end of a job.

7.9 Weather forecasting component overview

7.9.1 Description

The operational workflow at ECMWF is complex, and involves many (approx. 10,000) tasks tied together by an external control system, with substantial inter-stage data flows.

Within NEXTGenIO, the focus is on adapting the flow of data between the various components, to avoid data transfers to-and-from the high-performance filesystem within the time-critical pathway. This is likely to involve some form of object store, on nodes containing SCM, which can retain the relevant data in memory throughout the critical time periods. How this functionality fits into the overall workflow is yet to be explored, and in particular it is not clear the extent to which this data storage should be collocated with other operational components.

The overall weather forecasting workflow, although it is purely an application, expects to operate with substantial reserved resources, and very short queue times, and thus to have a very high level of control of how the overall workflow operates on the machine. As a result, some of the components appear to have systemware scope, despite being part of the application not the systemware.

There are two pseudo-systemware components that are part of the ECMWF operational workflow:

- **Workflow Controller**: ECMWF uses a custom workflow controller that runs continuously on a node outside the HPC system. It interrogates and submits jobs to the scheduler, and receives status updates directly from the HPC jobs over the IP network.
- **Hot Object Store:** A domain specific multi-node object store for meteorological data that can be addressed using the same requests as the existing indexed data stores at ECMWF. This may, or may not be collocated with other application components, or operate as its own task on the HPC prototype. It makes use of the Persistent Memory Access Library to make use of the large memory capacity available on SCM. It communicates with other HPC tasks over the high-performance network.

Within this workflow, SCM will be explicitly managed in 1LM mode.

7.9.2 Components

- Job Scheduler
- Persistent Memory Access Library
- Communication Libraries (MPI)
- High-performance filesystem
- I/O libraries (GRIB)

7.9.3 Data lifecycle

Note the description above of the non-systemware components called the “workflow controller” and “hot object store”.

1. Data arrives from a wide variety of external sources on an ongoing basis and pre-processed before being stored on the high-performance parallel filesystem
2. During the time-critical window, the Workflow Controller monitors the status of the operational tasks, ensuring that the correct tasks are submitted at the correct time
3. At the start of the time-critical window, the pre-processed data is read from the high-performance parallel filesystem into the 52 primary computational tasks
4. At intervals output fields stored the Hot Object Store, storing the data in SCM using the Persistent Memory Access Library. In a non-time-critical manner, this data *may* also be duplicated onto the high performance filesystem as this time
5. Post-processing processes are triggered at the appropriate moment by the workflow controller. They retrieve the relevant data from the Hot Object Store over the high-performance network
6. The output of post-processing is stored on the high-performance filesystem for later retrieval.
7. Data archiving tasks, outside the time-critical pathway, retrieve meteorological input data, and output fields from the high performance filesystem and the hot object store for external archiving.

7.10 Distributed workflow management

7.10.1 Description

PyCOMPSs is the Python binding of COMP Superscalar (COMPSs), which is a task-based programming model that aims to ease the development of applications for distributed infrastructures.

PyCOMPSs manages the user application and interacts with dataClay in order to handle persistent objects across the infrastructure. In the model, the user is mainly responsible for (i) identifying the functions to be executed as asynchronous parallel tasks, (ii) annotating them with a standard Python decorator and (iii) determine the persistent objects.

An example application for this use case can be the K-means algorithm or a life science application or similar that uses the K-means algorithm in its analysis with persistent objects.

7.10.2 Components

- Task-based programming model and environment: PyCOMPSs
- Object store: dataClay

7.10.3 Data lifecycle

1. User requests the execution of the application with a set of parameters using COMPSs launching scripts.
    a. At this point, the executable can be on the external high performance filesystem or on the local filesystem in the login nodes.
    b. If a dataset is requested, it is stored on the external high performance filesystem.
    c. The user requests to use dataClay in combination with PyCOMPSs (this step may be optional for other applications).
2. COMPSs launching scripts create a job template which is submitted to the job scheduler.
3. The Job scheduler allocates resources for PyCOMPSs.
4. Once the nodes that the job scheduler has allocated are available, COMPSs starts and configures the required processes within each node in order to setup a master-worker structure.
    a. The first node of the allocation will be considered the master and the rest of the nodes will be considered workers.
    b. If the user has requested to use dataClay, this deployment process also triggers the dataClay deployment before starting the user application execution.

5. When the entire deployment has finished, PyCOMPSs starts the user application execution on the assigned CCN.
6. Dataset source:
    a. Option 1: the application creates a dataset and makes it persistent within dataClay;
    b. Option 2: the dataset is already within dataClay.
7. The application starts computing in order to achieve the result.
    a. The tasks are deployed across the CCN.
    b. Each task can interact with dataClay if has to read or write data.
8. The application finishes.
9. Data scheduler(s) can persist object store to external high performance filesystem if required.
    a. If not persisted then object store is lost at the end of a job.

8 Conclusions

We have outlined the hardware and systemware architectures we have designed to exploit Storage Class Memory for HPC and HPDA applications. We have demonstrated that such an architecture has the potential to scale up to Exascale-class systems in the near future, with the potential for extremely high I/O performance and memory capacity.

Efficiently utilising SCM for computational simulation and data analytic applications will either require application modifications, to enable applications to directly target and manage SCM, or intelligent systemware to support applications using SCM for I/O, memory, and check-pointing.

We have described a range of systemware components that the NEXTGenIO project is developing and deploying to evaluate SCM for large scale applications. We will be evaluating this systemware, along with direct access applications, and the raw performance of SCM, using a prototype HPC system that NEXTGenIO is developing.

9 References

[1] "3D XPoint™ Technology Revolutionizes Storage Memory" http://www.intel.com/content/www/us/en/architecture-and-technology/3D-XPoint™-technology-animation.html

[2] https://www.google.com/patents/US20150178204

[3] https://colfaxresearch.com/knl-mcdram/

[4] Intel Omni-Path Whitepaper from Hot Interconnects 2015: https://plan.seek.intel.com/LandingPage-IntelFabricsWebinarSeries-Omni-PathWhitePaper-3850

[5] pmem.io: http://pmem.io/

[6] https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-ssd-dc-p4800x-brief.pdf

10 Appendix

The following pages contain sequence diagrams for usage patterns described in Section 7.

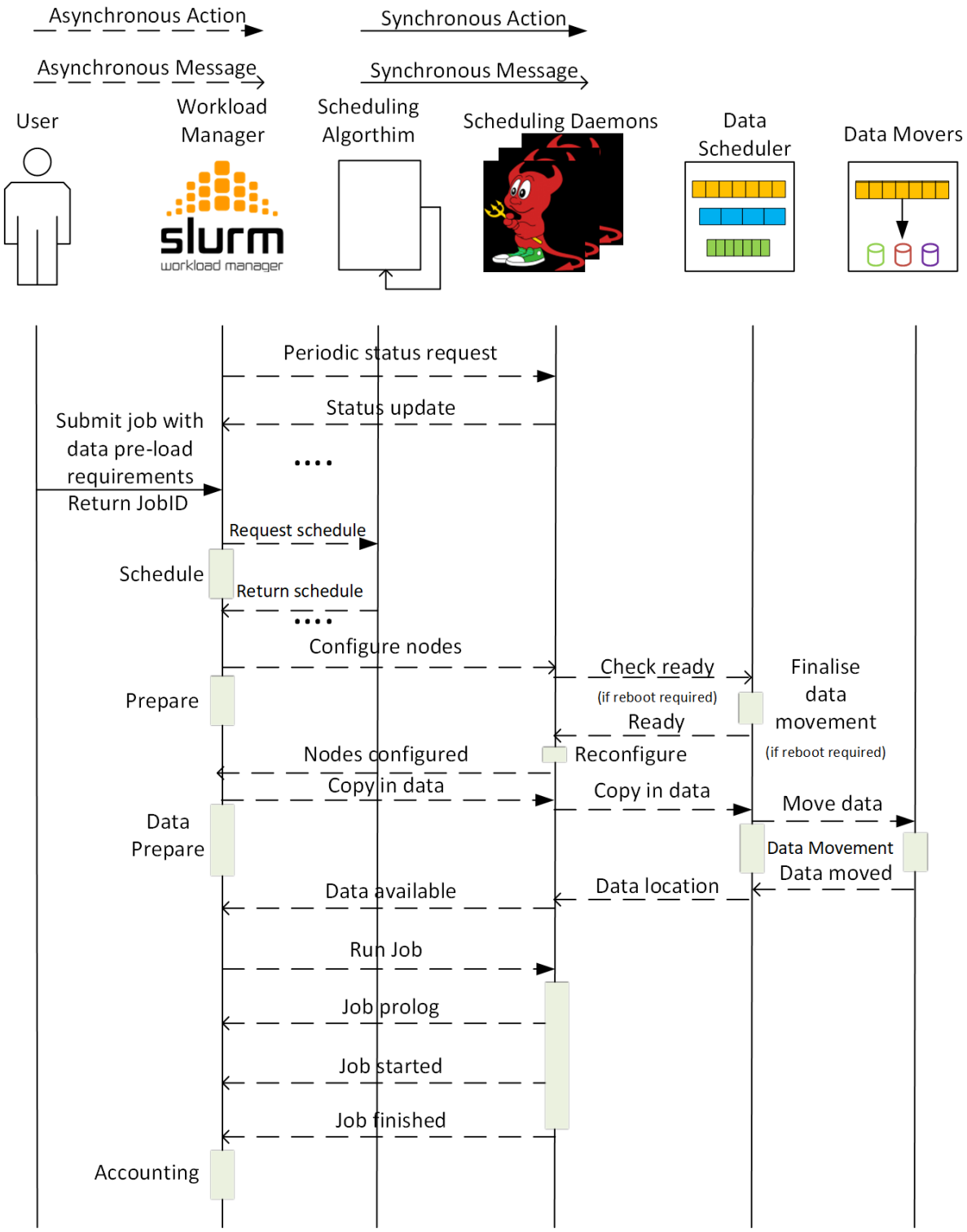

Figure 17: General Purpose HPC system using multi-node SCM filesystem.

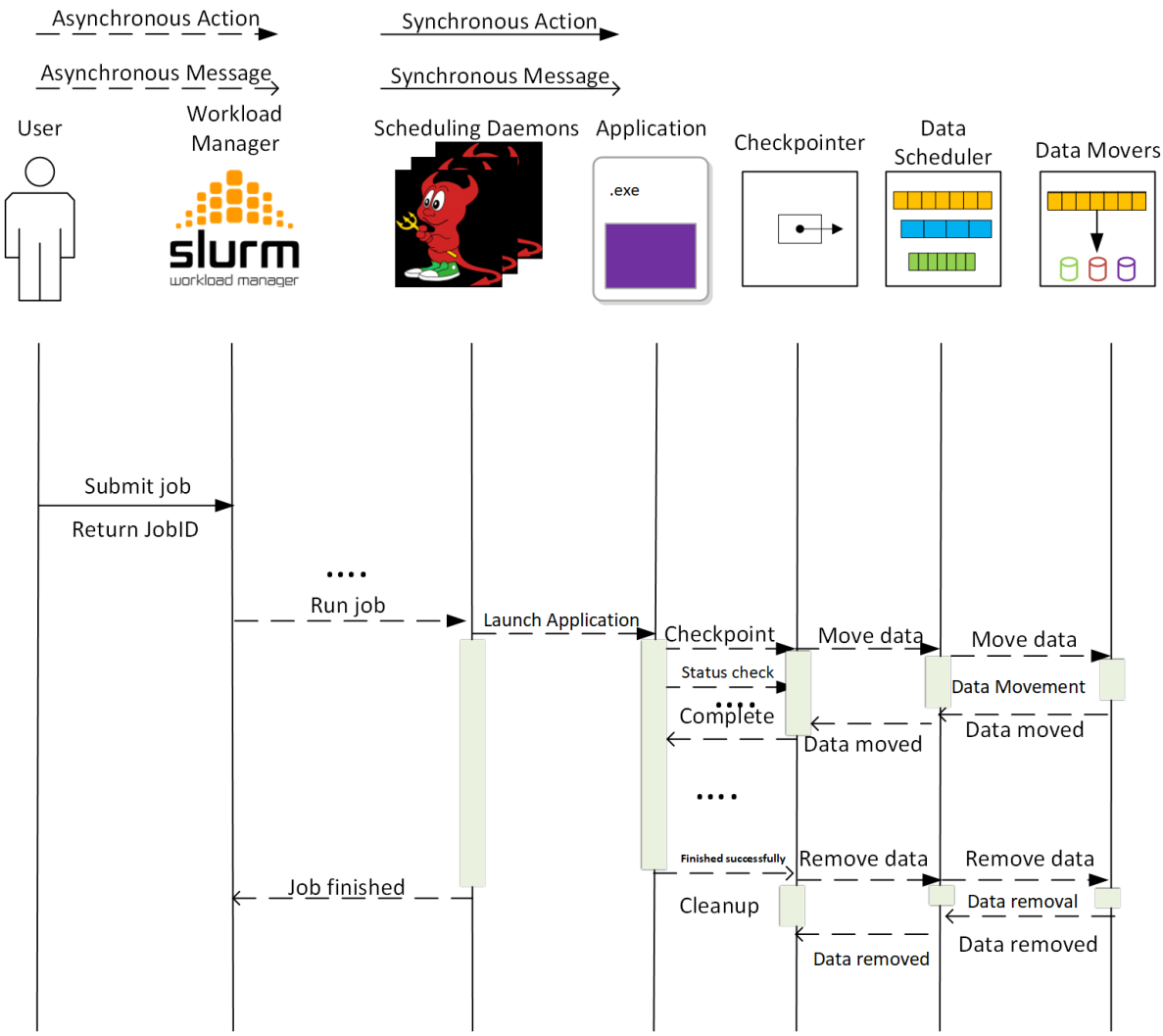

Figure 18: Resilience in specialist HPC system.

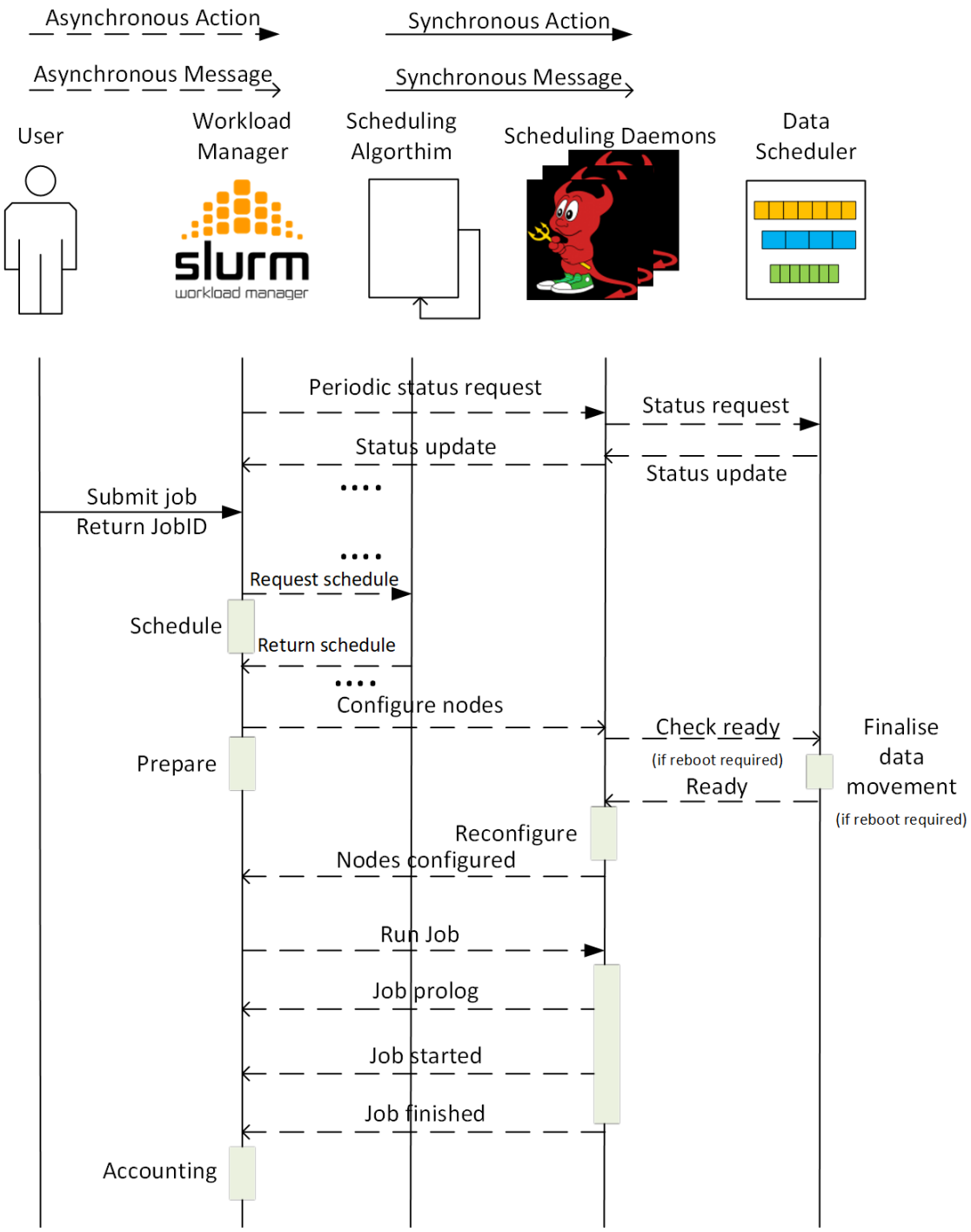

Figure 19: Large memory.

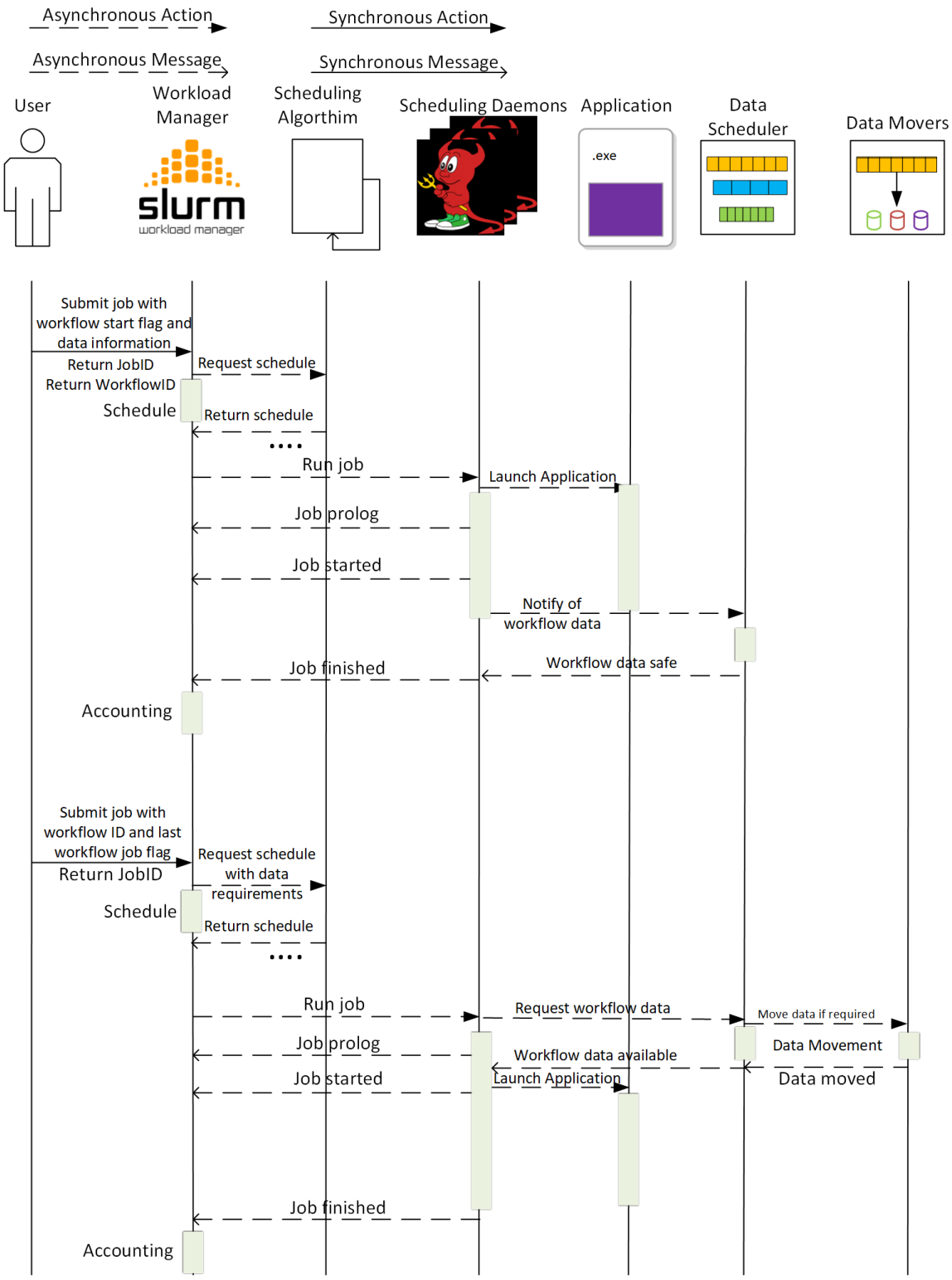

Figure 20: Workflows.

