nextgenio

SYSTEMWARE ARCHITECTURE & USAGE SCENARIOS

NEXTGenIO Architecture White Paper

Authors: Adrian Jackson, Iakovos Panourgias (EPCC), Bernhard Homölle (SVA), Alberto Miranda, Ramon Nou, Javier Conejero (BSC), Marcelo Cintra (Intel), Simon Smart, Antonino Bonanni (ECMWF), Holger Brunst, Christian Herold, Sarim Zafar (TU Dresden)

11th October 2018

www.nextgenio.eu

Next Generation I/O for the Exascale
This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 671951

Contents

Table of Contents

Foreword		3
About NEXTGenIO		3
1	Executive Summary	4
2	Introduction	5
	2.1 Glossary	5
3	Non-volatile Memory	7
	3.1 B-APM modes of operation	8
	3.2 Non-volatile memory software ecosystem	9
	3.3 Alternatives to B-APM	10
4	Requirements	11
5	Systemware Architecture	12
	5.1 Systemware Overview	12
	5.2 Complete Compute Nodes	14
6	Conclusions	18
7	References	19

Table of Figures

Figure 1: One-level memory (1LM). 8

Figure 2: Two-level memory (2LM). 9

Figure 3: pmem.io software architecture (source: RedHat). 9

Figure 4: Systemware architecture. 13

Figure 5: Complete compute node systemware components 14

Foreword

BY **DR MICHÈLE WEILAND, NEXTGENIO PROJECT MANAGER**

NEXTGenIO is working on improved I/O performance for HPC and data analytics workloads. The project is building a prototype hardware system with byte-addressable persistent memory on the compute nodes, as well as developing the systemware stack that will enable the transparent use of this memory.

Another key outcome from the project's research is the publication of a series of White Papers, covering topics ranging from the architecture requirements and design, to the systemware layer and applications. Each White Paper addresses one of the core challenges or developments that were addressed as part of NEXTGenIO.

The White Paper describes the systemware architecture that was developed based on a detailed requirements capture exercise informed by HPC and data analytics usage scenarios.

About nextgenio

Current HPC systems perform on the order of tens to hundreds of petaFLOPs. Although this already represents one million billion computations per second, more complex demands on scientific modelling and simulation mean even faster computation is necessary. The next step is Exascale computing, which is up to 1000x faster than current Petascale systems. Researchers in HPC are aiming to build an HPC system capable of Exascale computation by 2022.

One of the major roadblocks to achieving this goal is the I/O bottleneck. Current systems are capable of processing data quickly, but speeds are limited by how fast the system is able to read and write data. This represents a significant loss of time and energy in the system. Being able to widen, and ultimately eliminate, this bottleneck would significantly increase the performance and efficiency of HPC systems.

NEXTGenIO will solve the problem by bridging the gap between memory and storage using Intel's revolutionary new Optane DC Persistent Memory, which will sit between conventional memory and disk storage. NEXTGenIO will design the hardware and software to exploit the new memory technology. The goal is to build a system with 100x faster I/O than current HPC systems, a significant step towards Exascale computation.

The advances that Optane DC Persistent Memory and NEXTGenIO represent are transformational across the computing sector.

1 Executive Summary

NEXTGenIO is developing a prototype high performance computing (HPC) and high performance data analytics (HPDA) system that integrates byte-addressable persistent memory (B-APM) 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 B-APM to enable a generation of Exascale systems with sufficient I/O performance to ensure a wide range of workloads can be supported.

The systemware (system software), which supports the hardware in the system, will enable parallel I/O using the B-APM 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.

2 Introduction

The NEXTGenIO project is developing a prototype HPC system that utilises B-APM 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 white paper sets out the key features of B-APM that we are exploiting in NEXTGenIO and goes on to outline the systemware architecture 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 B-APM for large scale computations and data analysis, and to design a hardware and software architecture that could be exploited to integrate B-APM 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 B-APM 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.

2.1 Glossary of Acronyms

Acronym	Description
1LM	1 level memory. This represents the mode where DRAM and B-APM 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 B-APM from an application, resulting in an available memory space of the total DRAM plus the total B-APM 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 B-APM in the system. Applications do not see the DRAM memory address space, only the B-APM memory address space. This means the total memory available to applications is the size of the total B-APM memory in the node. 2LM cannot issue persistent memory instructions; it can only be used in load/store (volatile) mode.
3D XPoint™	Intel's emerging B-APM technology that will be available in NVMe SSD and NVDIMM forms
AEP	Apache Pass, code name for Intel NVDIMMs based on 3D XPointTM memory
API	Application programming interface
B-APM	Byte-Addressable Persistent 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
CPU	Compute processing unit
CCN	Complete compute node
DDR4	Double data rate DRAM access protocol (version 4)
DIMM	Dual inline memory, the mainstream pluggable form factor for DRAM
DMA	Direct memory access (using a hardware state machine to copy data)
DP	Dual processor
DP Xeon	DP Intel® Xeon® platform
DRAM	Dynamic random access memory
ECWMF	European Centre for Medium-Range Weather Forecasts
ExaFLOP	1018 FLOP/s
FLOP	Floating point operation (in our context 64bit/dual precision)
Gb, GB	Gigabyte (10243 bytes, JEDEC standard)
HDD	Hard disk drive. Primarily used to refer to spinning disks
HPC	High Performance Computing

IFS	Integrated Forecasting System, a simulation framework from ECMWF
I/O	Input / output
JEDEC	JEDEC Solid State Technology Association, formerly known as the Joint Electron Device Engineering Council; governs DRAM specifications
MPI	Message Passing Interface, standard for inter-process communication heavily used in parallel computing
NVM	Non-volatile memory (generic category, used in SSDs and B-APM)
NVMe	NVM Express, a transport interface for PCIe attached SSDs
NVML	NVM library at [3]
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.
O/S	Operating system
PB	Petabyte (10245 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, 1015 FLOP/s
PMEM, pmem	Persistent memory, another name for B-APM
POSIX	Portable Operating System Interface
POSIX file system	A file system that adheres to the POSIX standard, specifically for File and Directory operations.
RAM	Random access memory
ramdisk	DRAM based block storage emulation
RDMA	Remote DMA, DMA over a network
SNIA	Storage Networking Industry Association
SSD	Solid state disk drive
SHDD	Solid state HDD, using a NVM buffer inside a HDD
TB	Terabyte (10244 bytes, JEDEC standard)
TOP500	TOP500 supercomputer list, http://www.top500.org
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 persistent memory (B-APM) 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 B-APM 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 NVDIMM 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 B-APM is very different from the block-oriented way I/O is currently implemented.

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

Therefore, it is possible to use B-APM 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 B-APM 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 B-APM is provisioned in the servers in a compute system, there must be software support for managing data within the B-APM. 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 B-APM for fast I/O and storage (i.e. applications should be able to access B-APM in remote nodes if the system is configured with B-APM only in a subset of all nodes).

As B-APM 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 B-APM modes of operation

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

1-level memory, or 1LM, has main memory (DRAM) and B-APM as two separate memory spaces, both accessible by applications. This is very similar to the Flat Mode [2] 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 APIs such as malloc and represent the O/S visible main memory size. The B-APM will be accessed by non-volatile library functionality, such as the PMDK library, and present the non-volatile part of the system memory. Both allow direct CPU load/store operation. In order to take advantage of B-APM in 1LM mode, the systemware or applications have to be adapted to use these two distinct address spaces.

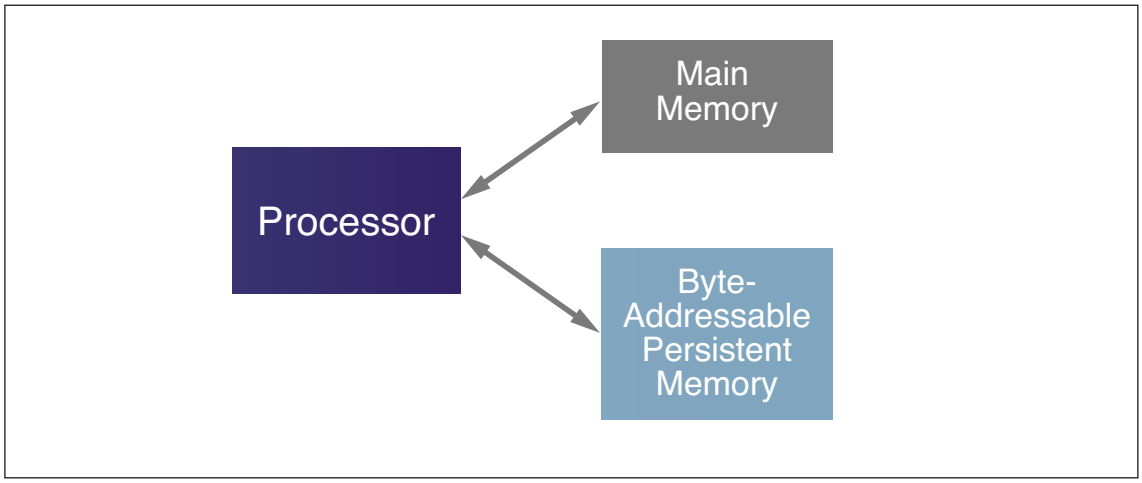

Figure 1: One-level memory (1LM).

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

This mode of operation does not require applications to be altered to exploit the capacity of B-APM, and aims to give memory access performance at main memory speeds whilst providing the large memory space of B-APM. 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 B-APM contents cannot be longer guaranteed, due to the volatile DRAM cache in front of the B-APM, so the non-volatile characteristics of B-APM are not usable.

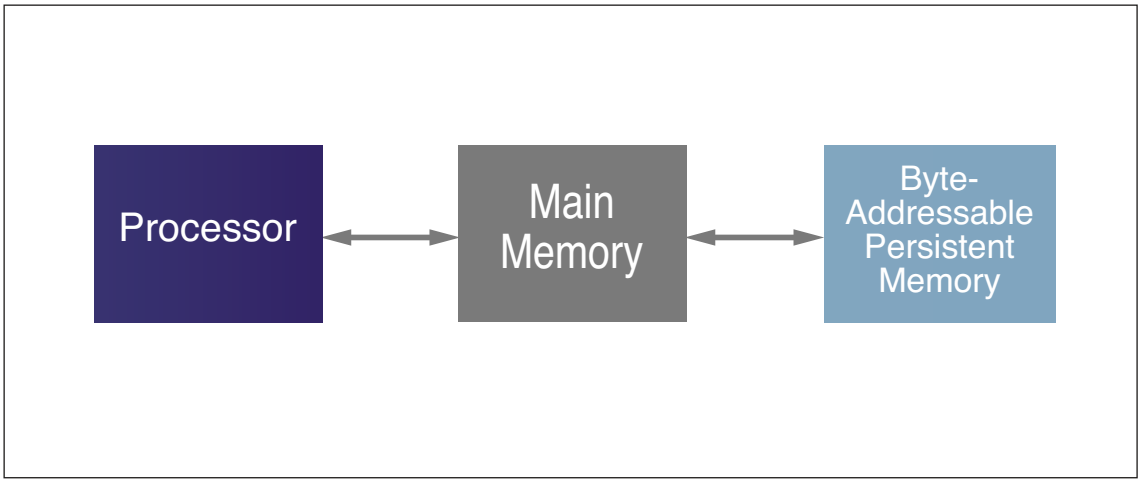

Figure 2: Two-level memory (2LM).

3.2 Non-volatile memory software ecosystem

In recent years the Storage Networking Industry Association (SNIA) has been working on a software architecture for B-APM with persistent load/store access. These operating system independent concepts have now materialised into the Linux PMDK[3] library.

This approach re-uses the naming scheme of files as traditional persistent entities and map the B-APM 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.

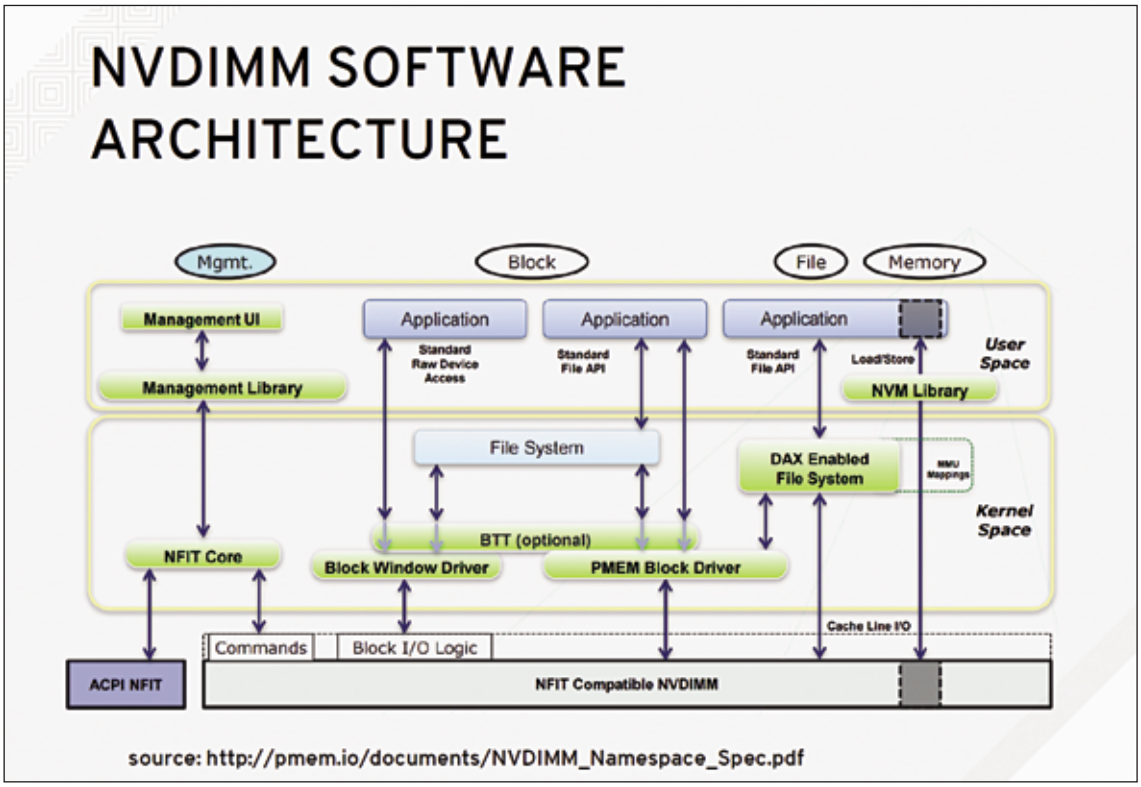

Figure 3: pmem.io software architecture (source: RedHat).

3.3 Alternatives to B-APM

There are existing technological solutions that are offering similar functionality to B-APM 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™ [4]. Other vendors have a large range of NVMe devices on the market, most of them based on different variations of Flash memory technology.

NVMe devices have the potential to provide byte-level storage access, using the PMDK 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 B-APM 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 B-APM 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 B-APM 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 B-APM.

4 Requirements

To design our hardware and systemware architectures that will enable us to construct a prototype system exploiting B-APM 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 B-APM to provide performance benefits.

3. It must be possible to modify applications to directly use the B-APM and exploit the full performance the hardware provides.
4. The hardware and systemware must be able to support a heterogeneous B-APM environment. We consider that systems where some nodes have B-APM 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 B-APM 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 B-APM between compute nodes to allow applications to access remote data and to allow the system to manage data in B-APM.

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 Systemware Architecture

The systemware for the NEXTGenIO architecture includes all the software that runs on the system (excluding user applications). 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).

5.1 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 B-APM. Briefly, from a user's perspective the system enables the following scenarios:

1. Users can request systemware components to load/store data in B-APM 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 B-APM without changing their applications.

2. Users can directly exploit B-APM by modifying their applications to implement direct memory access and management. This offers users the ability to exploit the best performance B-APM can provide, but requires application developers to undertake the task of programming for B-APM themselves, and ensure they are using it in an efficient manner.

3. Provide a filesystem built on the B-APM in the CCNs. This allows users to exploit B-APM 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 B-APM can provide.

4. Provide an object store that exploits the B-APM to enable users to explore different mechanisms for storing and accessing data from their applications.

5. Enable the sharing of data between applications through B-APM. 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 B-APM, 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 B-APM 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.

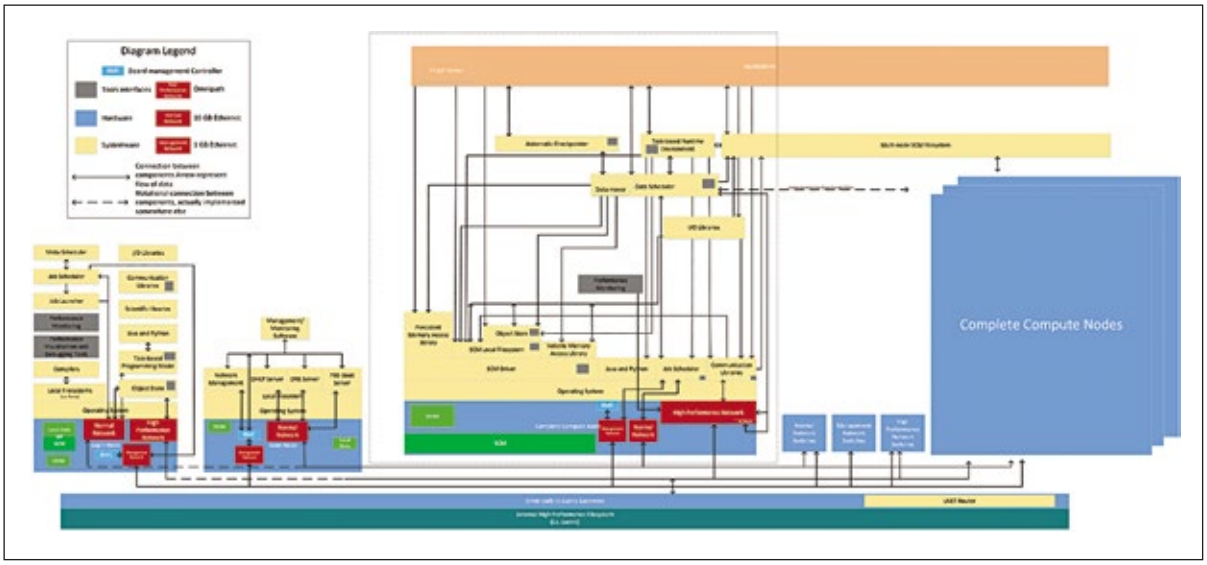

Figure 4: Systemware architecture.

5.2 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 B-APM configuration modes), and potentially to enable or disable different systemware components (such as the object store or B-APM 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 B-APM across all nodes, or to have systems where some nodes have no B-APM. This is to allow systems to be deployed where B-APM is provided in a subset of the CCNs, with the benefits of B-APM available across the system using the functionality of the systemware NEXTGenIO is designing and implementing.

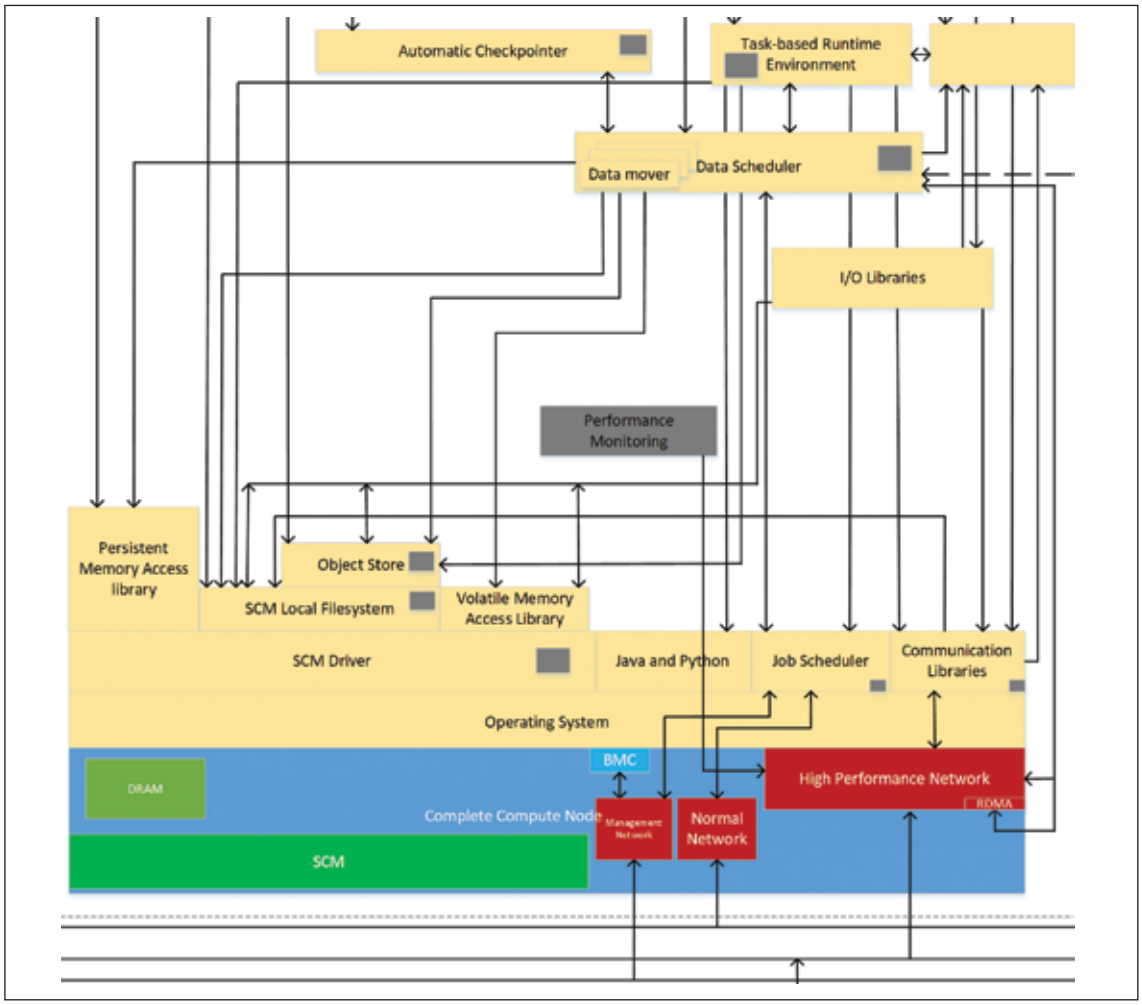

Figure 5: Complete compute node systemware components.

5.2.1 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 B-APM on node it could be possible for the automatic check-pointing system component to automatically check-point whole application instances into B-APM periodically to remove the need for applications to check-point themselves.

5.2.2 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).

5.2.3 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).

5.2.4 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 B-APM on other nodes over the high performance network (i.e. 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, B-APM, 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.

5.2.5 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).

5.2.6 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

5.2.7 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.

5.2.8 Multi-node B-APM Filesystem

The multi-node B-APM 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, B-APM and high performance parallel filesystem) by aggregating them into a virtual storage device and, secondly, to allow computing nodes to export B-APM 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 as there are many legacy HPC applications which may be hard or costly to directly modify to use B-APM 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 B-APM Filesystem offers the following opportunities and advantages: (1) policies can be added to automatically control the level at which data is stored (DRAM, B-APM or high performance parallel filesystem) and when it should be migrated; (2) the opportunity to distribute I/O between the B-APM 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.

5.2.9 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 B-APM 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.

5.2.10 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 B-APM 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 B-APM hardware, processors that can support B-APM, and the network interface used for the main compute network.

5.2.11 Performance Monitoring

This performance monitoring component is responsible for collecting performance information from an application execution. In the first case, the performance monitoring component would attach itself to the execution of the application. Different states of the program under observation would be 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 to interfaces such as 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 B-APM resources, requiring it to maintain up to date information on the status of B-APM. Consequently, the information collected by the scheduler can be passed to the performance monitoring component.

5.2.12 Persistent Memory Access Library

One way to provide simpler and reusable B-APM 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.

5.2.13 B-APM Driver

Just like with any hardware device, the B-APM 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 B-APM vendor (in this case Intel). We do not expect to need to perform changes to the device driver.

5.2.14 B-APM Local Filesystem

One of the most straightforward methods to exploit B-APM for persistent storage in each compute node is to use a local filesystem. While any filesystem can be used with the B-APM, in order to exploit the byte-level load/store access to B-APM 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.

5.2.15 Volatile Memory Access Library

While the system can transparently provide large volatile memory to applications on top of combined DRAM and B-APM (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.

6 Conclusions

We have outlined the systemware architecture we have designed to exploit Storage Class Memory for HPC and HPDA applications.

Efficiently utilising B-APM for computational simulation and data analytic applications will either require application modifications, to enable applications to directly target and manage B-APM, or intelligent systemware to support applications using B-APM 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 B-APM for large scale applications. We will be evaluating this systemware, along with direct access applications, and the raw performance of B-APM, using a prototype HPC system that NEXTGenIO is developing.

7 References

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

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

[3] Persistent Memory Development Kit: http://pmem.io/pmdk/

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