NVDIMM-Ns

Boosting Application Performance with NVDIMM-Ns

White Paper M-WP004

Corporate Headquarters: 39870 Eureka Dr., Newark, CA 94560, USA • Tel: (510) 623-1231 • Fax: (510) 623-1434 • E-mail: info@smartm.com
Customer Service: Tel: (+1) 978-303-8500 • Email: customers@smartm.com
Latin America: Tel: (+55) 11 4417-7200 • Email: sac@smartm.com
Europe: Tel: (+44) 7825-084427 • Email: euro.sales@smartm.com
Asia/Pacific: Tel: (+65) 6232-2858 • Email: infoasia@smartm.com

Table of Contents

Overview 3
NVDIMM-Ns – How they work 4
NVDIMM-Ns - A growing industry standard 4
NVDIMM-Ns – Their advantages 4
NVDIMM-Ns - Access to persistent memory modules 5
NVDIMM-Ns – Use case 6
NVDIMM-Ns – The next step 7

Overview

Traditional server and storage systems have a two-level storage model: One level accesses volatile data in memory (DRAM) with a load/store interface. The second level accesses persistent data in storage (flash) with a file system interface. The problem is that the operating system (OS) and file system (FS) code and buffering for storage lead to performance inefficiencies. Persistent memory is one solution now being used to improve system performance and reduce bottlenecks in high-availability, high-reliability server and storage applications. "Hot data" is stored within portions of high speed DRAM memory that is made persistent by the use of non-volatile DIMMs known as NVDIMM-N. Hot data is data that needs to be accessed frequently. It is typically critical information that needs to be accessed quickly and is often used by a company that requires quick decision making. Also, other latency and performance-sensitive data such as transaction logs and checkpoint images can be stored and accessed from the persistent sections of main memory to dramatically improve application performance without the risk of data loss.

NVDIMM-Ns – How they work

NVDIMM-N modules combine DRAM and flash onto a standard JEDEC registered DIMM whereby the data in the DRAMs is protected by using flash memory. In the event of a system power loss, a backup is triggered and the data is transferred to the flash memory. Only the NVDIMM-N needs to be powered until the data within the DRAM devices are written to flash memory, not to the entire system. Once the transfer is complete, the NVDIMM-N can be powered off.

When the system power is restored, the data is restored to the DRAM from the flash memory. The flash memory on the NVDIMM-N is not accessible by the system until another power outage should occur.

Application Examples

Two application examples of how system performance is dramatically improved with the use of NVDIMM-N include transaction logging and check pointing. Transaction logging records change to databases in a journal file-and-check, pointing records to the state of the database at a given moment in time. Instead of this data being stored into flash, such as an SSD, and taking hundreds of microseconds to store and access while going through layers of I/O, the data is now directly mapped into main memory and is accessed in hundreds of nanoseconds. The diagram below shows the concept of how NVDIMM-Ns leverage the speed of DRAMs and the persistence of nonvolatile memory to improve system performance.

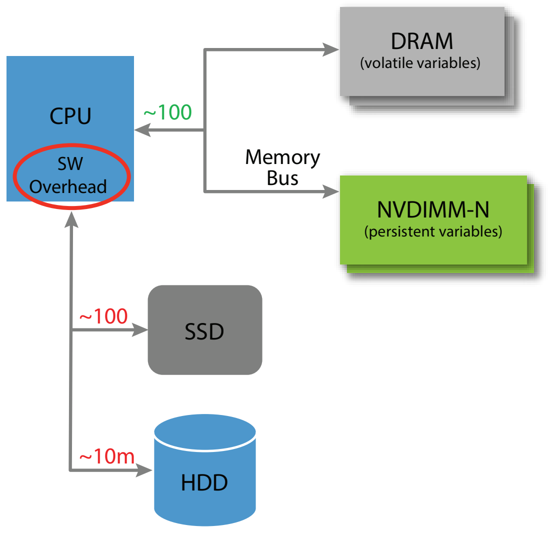

Figure 1

NVDIMM-Ns – A growing industry standard

Many DDR4 NVDIMM-N-enabled server and storage systems are available today from companies like Intel, Supermicro and others. NVDIMM-N-enabled motherboards have the ADR (Asynchronous DRAM Refresh) signal routed from an Intel processor (Grantley, Purely, or Broadwell) to the DIMM sockets. This is required to put DRAMs into self-refresh before a power loss condition occurs. Other requirements are a BIOS and MRC (Memory Reference Code) configured to recognize NVDIMM-N modules when they are plugged in. The BIOS and MRC detect, initialize, and boot the NVDIMM-N modules. They are then mapped as a persistent section of memory within main memory: Block vs. Load/store Access (Memory Mapped)

NVDIMM-Ns – Their advantages

NVDIMM-Ns can be accessed as block devices or as load/store devices. Block-level storage presents the NVDIMM-N to servers as a SSD that is installed in a DIMM socket. Load/store architecture allows NVDIMM-Ns to be accessed by load and store operations and all values for an operation need to be loaded from memory and be present in registers. Following the operation, the result needs to be stored back to memory. This operation represents a direct, byte-addressable path to main memory. The real benefits of NVDIMM-Ns are realized when using load/store access architecture. NVDIMM-Ns are memory mapped and made PM-aware and directly access DRAM without any block drivers. Software in the data path is thus eliminated. The table below illustrates performance benchmark testing results using a new SDM (Software Defined Memory) file system developed by Plexistor.

Operation per second

	NVMe (ZFS)	NVMe (XFS)	NVDIMM-N (SDM)	
Random 4KB write Single threaded FIO benchmark	21,200	11,300	492,456	x 23
Random 4KB write Multi threaded FIO benchmark	28,700	72,760	3,680,270	x 128
Random 128B write Multi threaded FIO benchmark	21,200	105,830	6,848,541	x 913
SQL Database SPEC SFS 2014 Database	-	-	-	
MongoDB NoSQL MongoDB v3.1 on WiredTiger. Load scenario	11,610	18,090	41,000	x 4

Latency in µs

	NVMe (ZFS)	NVMe (XFS)	NVDIMM-N (SDM)	
Random 4KB write Single threaded FIO benchmark	840	88	2	x 420
Random 4KB write Multi threaded FIO benchmark	623	246	4	x 156
Random 128B write Multi threaded FIO benchmark	840	168	2	x 65
SQL Database SPEC SFS 2014 Database	120	110	5	x 25
MongoDB NoSQL MongoDB v3.1 on WiredTiger. Load scenario	830	534	227	x 4

ES-2650 v3 CPU, 32GB DRAM, 32GB NVDIMM-N, NVMe

Figure 2 - Software-Defined Memory

The table on the bottom right of the previous page compares the performance between four 16GB DDR4 NVDIMMs and one 400GB NVMe PCIe SSD. Based on the dramatic increase in operations per second and the dramatic decrease in latency, there is a clear end user benefit when adopting NVDIMMs.

In addition to the file system developed by Plexistor, changes to the Linux operating system have been made to make it 'NVDIMM-N-aware'. The 4.3.3 Linux Kernel has the built-in drivers for the byte addressable access (pmem) and for the block access (blk) to the NVDIMM-N modules. The block access uses advanced mechanisms such as the Block Transfer Window (BTW) that is provided by the NFIT (NVDIMM-N Firmware Interface Table) protocol -- a new way of persistent memory representation per the ACPI 6.0 specification. In addition, the NFIT protocol provides many advanced methods of organizing the byte addressable persistent memory as well.

The BIOS must be NFIT capable to make use of those advanced features. For the legacy BIOS that provides the basic support for the NVDIMM-N modules, the 4.3.3 Linux Kernel supports the byte addressable by default, as well as a simple block mode access through its pmem driver. The built-in pmem device driver creates the device link, one device per CPU socket. That is, if the mother board has two CPU sockets that are populated, all the NVDIMM-N modules directly accessible from Socket-0 (QPI) would be grouped under /dev/pmem0 and all the NVDIMM-N modules directly accessible from Socket-1 (QPI) would be grouped under /dev/pmem1, respectively. The individual NVDIMM-N modules across the memory channels for a socket are accessed according to the memory interleave rules. This transfer might cause serious issues for the block mode access, and hence, is discouraged as it does not have sophisticated error recovery methods. NFIT BIOS is recommended for block mode access.

NVDIMM-Ns - Access to Persistent Memory Modules

BLOCK MODE

The pmem driver, by default provides the BLOCK mode access for the underlying Persistent Memory modules. For example, just like any other BLOCK device, /dev/pmem0 can be used as follows:

```
dd if=/dev/zero of=/dev/pmem0
```

The EXT4 file system in the Linux 4.2 Kernel has a new feature called the "Direct Access" support (DAX) that eliminates the file system buffering of data. For our example, the recommended usage is:

```
mkfs.ext4 /dev/pmem0
mount -o  dax /dev/pmem0 /mount_point
```

Once mounted at the "mount_point" regular file systems operations – such as copying and deleting the files -- could be done. To unmount, use:

```
umount /mount_point
```

In the event of a power loss, the underlying Persistent Memory Modules (when properly ARMed) must keep the data persistent.

MEMORY MAPPED MODE

An application program in the user space could do a memory map on the /dev/pmem0 and access them as a byte addressable entity. The underlying pmem driver provides that capability, too. This requires writing some application code in C/C++. (In our tests, the usage of O_DIRECT seemed to indicate a major improvement in IO performance.)

Sample Code: (Use this for guidance only.)

```
uint64_t length, offset = 0;
char *mmapp = NULL;

uint8_t write_operation = 0;
uint32_t result;

if ((fd = open("/dev/pmem0", O_RDWR | O_DIRECT))
< 0) {
perror("open failed.");
exit(1);
}

// Get the length of the device.
if ( ioctl( fd, BLKGETSIZE64, &length )) {
perror("get size failed.");
exit(1);
}

mmapp = mmap(
NULL,
length,
PROT_READ | PROT_WRITE,
MAP_FILE | MAP_SHARED,
fd,
offset
);

if ( mmapp == MAP_FAILED ) {
perror("memory map failed.");
exit(1);
}

if (write_operation)) {
result = do_write_test(mmapp, length);
} else {
result = do_readverify_test(mmapp, length);
}

if (munmap(mmapp, length) < 0) {
perror ("unmamp failed");
exit(1);
}

close(fd);
exit(0);
```

NVDIMM-Ns – Use Case

All the modern file systems (EXT4, ZFS, etc.,) use a transaction log based mechanism to record the changes to the file system that are not yet committed to the persistent data storage. This is to safeguard the file system against any failure from power loss or other failure conditions. In event of any failures, the file system would replay the transaction log and commit the changes to the persistent data storage to bring the file system to a consistent state.

The reason for this multi stage commit process used by the file systems is due to the latencies involved with the conventional persistent data storage media such as the hard drives, and the SAN. By recording the transactions to a separate logging device, the file system could respond to its clients much faster, and increase the overall throughput.

It is to be noted that the storage for the transaction logs must be persistent too. Otherwise, it would defeat its purpose of existence. Previously, a second hard drive, a drive smaller in capacity and with a high RPM, was used for keeping the transaction logs, as the size requirement would not demand large capacity media. Storage was typically factored by the file system bandwidth requirements. Conventional wisdom to have hard drives “short stroked” – limited to a very little capacity -- by limiting the drive head moving across a limited number of sectors on the drive platter.

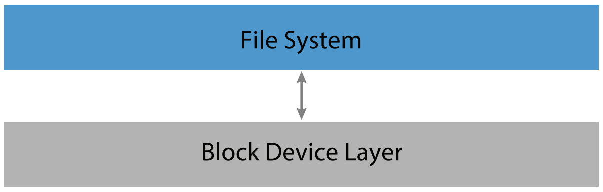

Figure 3

Later, usage of solid state storage devices for keeping the transaction logs gained prominence. This offered a variety of advantages, as they not only had latencies that were of magnitude lesser than the hard drives, but short stroking was not needed at all. Still, a solid state storage device that is constantly written for every file system transaction must adhere to a very high life cycle expectancy to meet the demands of the file system.

As seen, irrespective of a dedicated hard drive or a solid state storage device, I/O for the transaction log had to share the same backend storage sub system for the file system’s persistent data storage. (Some configurations used a separate host bus adapter (HBA) to mitigate this problem.) Typically, it was a SAS, or a fiber channel, or a SATA back end connection. This process added layers of protocol usage. Also, being used as a block storage device, the transaction logs had to be written in the block format. This called for data format conversions within the file system software.

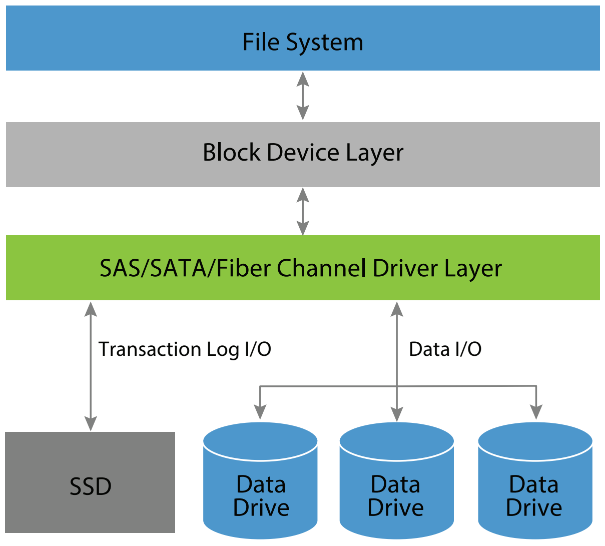

Figure 4

The arrival of NVDIMM-N removes many shortcomings for usage as a device for transaction logs, as seen below:

1. It is a device on the memory bus. It does not have to share or interfere with the file system’s I/O to its persistent data storage [on the SAS, Fiber Channel, or SATA bus].
2. The latency of the device on the memory bus is of multiple orders of magnitude less than any of the storage I/O bus.
3. The device could be accessed as a byte-addressable entity, meaning no need to organize the data in block formats.
4. The data could be stored, as is, without any overlay of the storage protocols such as SAS or fiber channel. This makes it simple to administer.
5. The access to the persistent part of the device is used only on power down or power failure conditions. This gives a near-memory kind of performance and also a much longer life expectancy for the underlying flash chips.
6. The biggest advantage is that the existing file systems could use the NVDIMM-N devices without any changes to its software modules. (Note: The NVDIMM-N devices are always usable as a conventional block torage. In such cases, the file systems would not need any changes.)

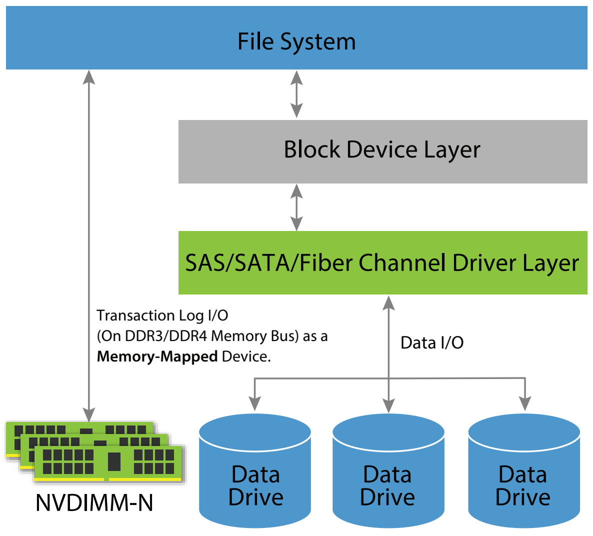

Figure 5

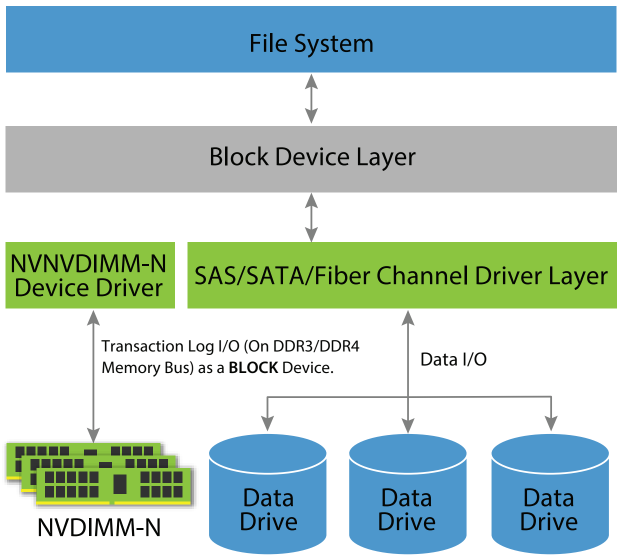
File System
Block Device Layer
NVNVDIMM-N Device Driver
SAS/SATA/Fiber Channel Driver Layer
Transaction Log I/O (On DDR3/DDR4
Memory Bus) as a **BLOCK** Device.
Data I/O
NVDIMM-N
Data Drive
Data Drive
Data Drive

Figure 6 - NVDIMM-N device usage as a Block Device by the File System

NVDIMM-Ns – The next step

In conclusion, NVDIMMs have been described as the next logical step toward the adoption of Storage Class Memory (SCM). NVDIMMs are driving the implementation of hardware, software and I/O stack changes to make standard use of the benefits of nonvolatile random access memory. While memory solutions continue to evolve as they always have, NVDIMMs are the current ideal solution to improve system performance and reduce bottlenecks in high-availability, high-reliability server and storage applications.