Using Active NVRAM for Cloud I/O

Sudarsun Kannan, Dejan Milojicic, Vanish Talwar
HP Labs
Palo Alto, CA, USA
sudarsun.kannan, dejan.milojicic, vanish.talwar@hp.com

Ada Gavrilovska, Karsten Schwan
Georgia Institute of Technology
Atlanta, GA, USA
ada, schwan@cc.gatech.edu

Hasan Abbasi
Oak Ridge National Laboratory
Oak Ridge, TN, USA
habbasi@ornl.gov

***Abstract*—A well-known problem for large scale cloud applications is how to scale their I/O performance. While next generation storage class memories like phase change memory and Memristors offer potential for high I/O bandwidths, if left unchecked, the raw volumes and rates of I/O already present in current cloud applications can quickly overwhelm future I/O infrastructures. This fact is motivating research on 'data staging' in which I/O and data movement actions are enhanced with computations that process data before or while moving it across I/O channels – in situ – to filter or reduce it, to better organize it for subsequent access (e.g., by other applications as in coupled codes), or to analyze it to quickly derive important insights about the application producing those large data volumes. This paper proposes a technique that uses and exploits 'Active NVRAM' (non volatile memory) for staging I/O. Active NVRAMs are node-local NVRAMs that are embedded with a low power system-on-chip compute element. These active compute elements can be used to operate on output data asynchronously with the tasks performed by computational node elements, to reduce data or to perform some of the data processing required for data analytics before data is moved to longer term storage. The paper describes the Active NVRAM design, sample ways in which it is used for I/O acceleration, and initial performance results evaluating the opportunities for and limitations of the Active NVRAM approach.**

***Keywords*-NVRAM, Cloud, In-situ processing**

I. Introduction

It is well known that data volumes are growing faster than Moore's law, almost 56 times over the past seven years, in fact. Further, with more services moving toward cloud-based datacenters for scaling, this data explosion trend is predicted to continue. Finally, for both enterprise/cloud infrastructures and for future high performance machines, while processing power is important for applications to scale, equally important are efficient methods to store generated data (I/O), to operate on these data, absorb and analyze it, and decimate the rest.

While much prior work has focused on improving data storage efficiency by increasing I/O bandwidths via hardware and software techniques, e.g., by using new storage class memories like NVRAM, there has been less research on efficient mechanisms for I/O data processing (post processing). By I/O data processing, we refer to data operations that can either be done online, while computations are proceedings, or offline, and such computations may be performed on nodes that are local or remote to where data is generated. In response, in high performance computing, particularly as we begin to move toward the exascale, there has been recent work that deals with output data via 'in situ' methods to analyze or visualize the enormous data volumes produced by large-scale simulations [1]. In the enterprise space, with the increased importance of online data analytics and monitoring, there are now multiple infrastructures for online data processing, including those using streaming or real-time versions of Hadoop [2, 3]. However, these infrastructures continue to rely on existing techniques for I/O scaling, such as distributed file systems like the Google File System (GFS) or its HDFS open source counterpart [4]. With such techniques, the petabytes of data consumed and produced for online data analytics are stored to shared persistent storage disks and/or moved over network for analytics processing. Unfortunately, always fetching and re-fetching data from shared disks and/or over the network to perform post processing is not efficient, resulting in substantial costs due to data movement and network traffic. In response, C. Wang et al. [5] argue that one way to attain scalability in analytics and monitoring is to analyze data locally and filter monitoring information to reduce total data volumes passed across monitoring topologies or to disk. D. Logothetis et al. [6] explore inefficiencies in moving terabytes of data for log processing to centralized locations, stressing the need for fast in-situ post processing.

Data post processing serves two purposes:

1) Post processing like data filtering or compression reduces the magnitude of data moved across I/O channels, which also facilitates data management. This can directly benefit I/O performance.
2) Post processing methods like sorting, indexing, and layout reorganization are important for data analytics, including to increase performance by improving data layout or formatting [7].

This paper specifically focuses on improving the performance of post processing for large scale applications in cloud and utility data center systems. Toward that end, we adopt the method of data staging developed for the HPC domain, but instead of moving data to other 'staging' nodes, as done in prior HPC research [1], we propose and evaluate

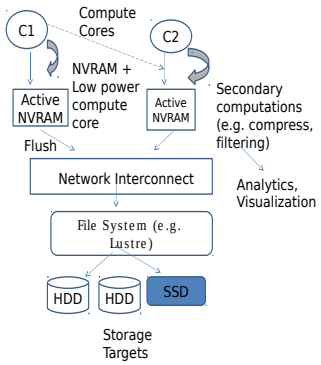

Figure 1. Architecture

a novel mechanism that uses *Active NVRAM* to improve I/O throughput as well as post processing performance. Active NVRAM is non-volatile memory coupled with a low power compute element (see Figure 1) attached to server or compute nodes. Active NVRAM [8] is based on a system-on-a-chip architecture providing high memory bandwidth to hosts, where each active NVRAM has its own runtime. NVRAM technologies such as Memristor and PCM offer 100X faster read-write performance and have endurance of approximately a million writes. They are scalable in terms of storage density, with the ability to store multiple bits per cell, and they require low energy (approx. 5-10 pJ/bit to write and no refresh required) when compared to DRAM making them a suitable replacement for disks on common compute or server nodes. In fact, our preliminary evaluations show high I/O performance gains when using active NVRAM, and we also analyze the effectiveness of the Active NVRAM approach for post processing using representative benchmarks. We argue that, by using Active NVRAM, significant data post processing operations can be accomplished efficiently.

Our technical contributions include the following:

1) *asynchronous I/O processing via NVRAM* – a framework that permits applications to use NVRAM for high performance I/O; the framework supports asynchronous I/O data post processing using active NVRAM, with minimal changes required to applications;
2) *experimental evaluation* – experimental results for a common I/O post processing application – distributed sorting and compression; and
3) *assessment of opportunities and limitations* – by explaining and evaluating the overheads associated with the Active NVRAM-based staging approach.

II. Related Work

C. Wang et al. [5] motivate the need for processing data 'closer' to computation in the cloud. D. Logothetis et al. [6] discuss the time sensitivity aspects of log data post processing in the cloud and the need for efficient in-situ processing, mirroring earlier work in the HPC space on I/O data staging [1] and in situ processing [9]. In comparison, and since in-situ processing performed on compute or server cores can add overheads to applications, we explore opportunities for reducing such overheads via Active NVRAM. Specifically, we exploit node local data storage and use NVRAM like Memristor and PCM to improve both I/O performance/throughput and post processing performance for I/O data. H. Monti et al. [10] use cloud for such data analysis. In cloud environments, with restricted memory and network bandwidths for commodity servers and with increasing I/O data volumes and rates, the time taken to move data from compute core to separate staging nodes will add overheads that may not be acceptable to applications. We will evaluate those experimentally in our future work.

III. Active NVRAM Usage

Post processing examples. While current approaches perform 'on-compute-core' processing or do so on dedicated staging nodes, we propose using Active NVRAM. To evaluate the viability of doing so, we have implemented (1) a sample data reduction technique, i.e., the compression of I/O data, and (2) a representative data reorganization method, i.e., a distributed merge sort, as driving examples. While data reduction via compression can improve I/O performance, sorting is a useful example for data reorganization, since it is commonly used in data analytics [11], for example, in mapreduce-based analytics to sort input files. Further, compression is computationally intensive, whereas sorting is communication intensive, thus representing two different ends of the spectrum of computations Active NVRAM may perform. Experimental evaluations will analyze the performance implications of using such post processing codes.

Other Active NVRAM applications. While this paper focuses on post processing with active NVRAM, there are additional opportunities for using activity in conjunction with I/O, some of which were identified in early work on 'active disks' [12]. Specifically, when offloading I/O operations to active NVRAM, one could consider the needs of data intensive applications, which one can divide into two data components: (1) their disk based chunks or files stored on some set of nodes and (2) their in-memory key value store using dedicated nodes loaded with database values. The key value *store and fetch* operations are time sensitive and do not require much processing, yet using memory is costly in terms of power. To address power and cost issues, FAWN [13] proposes a system architecture for using 'wimpy' nodes (low power cores) with Flash-based storage devices for efficient key-value store operations. In cloud-based environment, not all applications can scale with wimpy nodes making it difficult to replace all cloud servers with wimpy nodes in which case, node-level active NVRAM can be added to servers machines and used for such key value store operations, while the database and decision engines execute on powerful cores. The approach

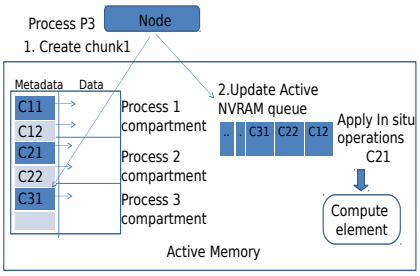

Figure 2. Design

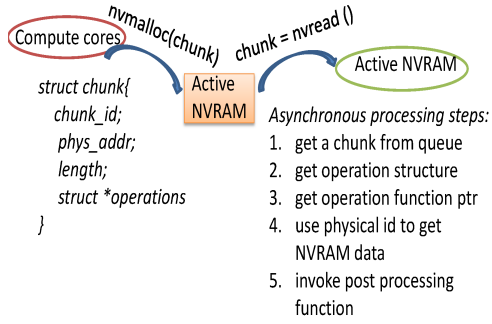

Figure 3. Asynchronous Data Processing

has potential additional power benefits, as shown in [14], We plan to investigate these applications in our future work.

Importance of Active NVRAM in the Cloud. Poor I/O bandwidth for large scale applications in virtualized cloud environments is a well know problem. Despite various I/O optimizations, a major cause of poor I/O bandwidth performance is the underlying virtualization technology. Sharing physical I/O channels, such as Gigabit Ethernet, between different virtual machines [15] can degrade bandwidth to roughly 87 MB for large scale instances in EC2, for example. Further, for large scale applications, inter VM contention for I/O bandwidth can add substantial overheads. We believe our NVRAM design can provide substantial benifits for cloud applications. This paper's measurements are done on a non-virtualized system, and our future work would evaluate the impact of using active NVRAM for large scale cloud setup by providing virtualization support for our framework.

IV. Active NVRAM Framework

The goal of the Active NVRAM framework (AMF) is to provide applications with simple memory based interfaces for accessing NVRAM. Generally, large scale applications have multiple processes or threads, each of which reads/writes I/O data in chunks to active NVRAM (see Figure 2). A chunk is a sequential stream of data described with metadata that contains information about the physical address of the chunk in active NVRAM (see Figure 3), chunk length, data type, and a set of flags describing the chunk's state. Metadata also contains information about the set of operations to be performed as a part of post processing on each chunk.

NVRAM Allocation and Data Commit. To write data to active NVRAM from a source data structure (e.g., variable, arrays, structure), applications make *nvmalloc()* calls. The call results in the allocation of memory in DRAM and also an equivalent allocation of memory in active NVRAM. All updates to the allocated memory before the commit operations are buffered in DRAM. This avoids the write latency to non volatile memory and indirectly addresses the wear leveling issues of NVRAM. Active NVRAM is persistent and so applications decide when data should be committed using *nvcommit()*. A commit guarantees an atomic failsafe update to NVRAM. Since non-volatile memories are not currently available, the AMF uses memory mapped files to emulate persistence. AMF works with any memory mapping compatible hardware.

Asynchronous Data Processing. One goal of our framework is to decouple compute cores and the active NVRAM compute element. They behave in a producer consumer model, where the compute cores run applications, write data in form of chunks to NVRAM, and continue with whatever computations they are currently performing (e.g., compute phase after I/O phase). Each process writes data to its own compartment. A *compartment* is a process independent container containing its NVRAM data (similar to process address space). It avoids inter NVRAM data conflicts. Figure 3 shows the basic chunk structure and psuedocode of how data chunks are processed by NVRAM.

In our current design all threads of an application write their metadata to a single global work queue apart from writing data to their respective process compartment (see Figure 2). The active element dequeues metadata corresponding to a chunk, and uses chunk information to find the physical address of the chunk. The active element also uses metadata to get the list of post processing operations that needs to be applied over the data chunk. The operations are specified by the application. In our current design, it is responsibility of the application to denote the operations, as different applications could have different post processing requriments.

To summarize, our Active NVRAM data structure design takes both I/O performance and post processing mechanisms into consideration. The applications requires minimal changes with few additional memory based interfaces. In our future work, we would like to enhance our design to exploit application knowledge with flexible customizations to enhance performance.

V. Experimental Evaluation

To evaluate the use of Active NVRAM for I/O staging, we use an Infiniband-based cluster at HP. The cluster nodes run non-virtualized Linux OS. Each node has quad core Intel Xeon processors with node local hard disks. To emulate active NVRAM, each node uses three cores for running the actual computation, and one core is dedicated for the active NVRAM compute element. The experimental results:

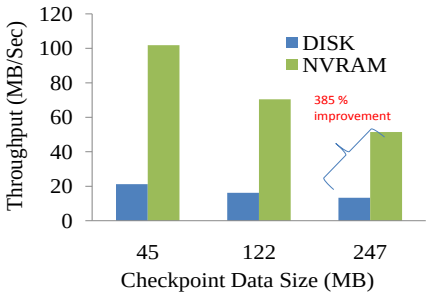

Figure 4. Throuput comparison - Disk, Ramdisk, NVRAM

1) validate the benefits of using NVRAM for I/O in terms of throughput;
2) provide a comparison of performance tradeoffs for using Active NVRAM with respect to compute core in-situ processing; and
3) analyze application overheads due to software and hardware limitations in using Active NVRAM.

A. I/O Throughput

For evaluation, we use the NAS-NPB benchmarks [16], which provides a good mix of applications with different characteristics, such as communication intensive, I/O intensive, and embarrassingly parallel applications. We primarily use the I/O intensive BTIO benchmark. Figure 4 compares the I/O throughput of applications using disk storage in NFS mode vs. Active NVRAM. We use application checkpointing as the source of I/O data.

Observations: As expected, using NFS results in poor throughput. For the largest data size in our experiment, we observe around 385% gains over disk. Further, with increasing I/O data sizes, the disk throughput further decreases because of increased NFS contention. Though we use NFS for our initial results, we believe using NVRAM can have substantial performance improvement over high performance file systems like GFS in cloud. With increase in data size, though active NVRAM performs better, the throughput slightly reduces. This is because DRAM free pages are used to emulate NVRAM by memory mapping files. With increasing data size, the memory used by the application increases and so reduction of overall free pages used to emulate active NVRAM. This results in increased page flusing which is why the effective througput decreases. We believe such overheads will be absent when using real NVRAM hardware.

B. Performance impact of using NVRAM

The next experiment evaluates the implication of using NVRAM for data post processing, using data sorting as an example. We use again the I/O intensive BTIO benchmark for experiments. Figures 5 and 6 compare application runtime for on-compute core post processing, indicated by red bars, with the active NVRAM approach, indicated by green bars, by varying the number of compute cores and the total I/O data size, respectively.

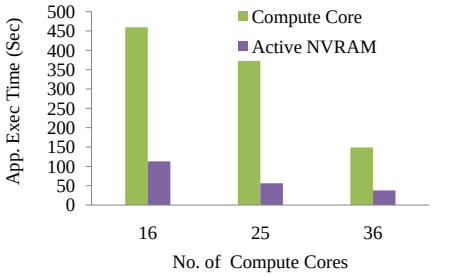

Figure 5. Sorting performance vs. compute cores.

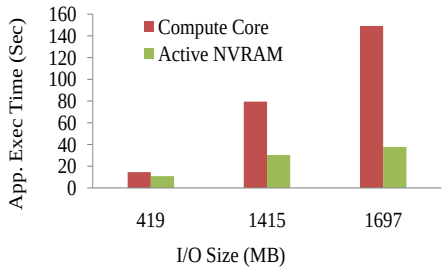

Figure 6. Sorting performance vs. data size.

Observations: We first keep the data size constant (1.6 GB) and vary the number of application cores (see Figure 5). The total number of application cores is same for the 'on-compute core' and the active NVRAM approach. When the *data size (MB) to no. of compute cores* ratio is large, Active NVRAM has the best performance compared to smaller ratios. Even for smaller data to compute core ratio, the performance improvement is higher than 100% for the active NVRAM approach. Figure 6 keeps the compute cores to constant (36) and varies the data size. The performance improvements are similar to the previous experiments. The reason for this behavior can be attributed to the communication intensive nature of sorting. For small data sizes, post processing after each checkpoint step does not add much overhead to application run time. But with increasing data size, although the 'on compute core' approach uses powerful compute cores, communication latencies have substantial impact on application runtime, whereas the asynchronous processing approach in Active NVRAM hides such overheads. Also, increasing the number of post processing threads is not an effective solution in case of the 'on-compute core' approach. With increase in threads sharing physical cores, parallel application run time gets seriously impacted.

In contrast to these results, Figures 7 and 8 provide insights on using computation intensive compression for post processing, without any communication. The algorithm is a 'lossy compression' and provides a 4:1 data reduction. The graphs indicate almost no performance difference when using the 'on-compute core' for post processing compared with active NVRAM, for different *data size to no. of compute cores* ratios.

Observations: The reason for this behavior can be attributed to the following:

1) For the on compute core approach, although compres-

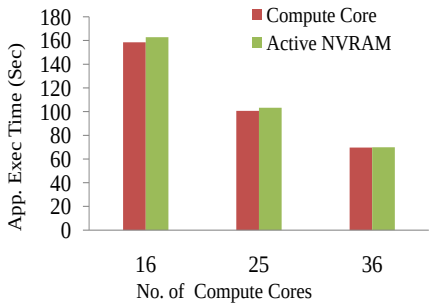

Figure 7. Compression performance vs. compute cores

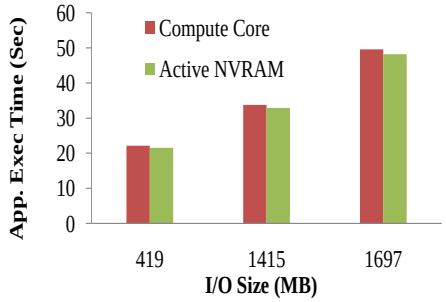

Figure 8. Compression performance vs. data size

sion is performed synchronously with computation, the ratio of compute cores to active elements is 3:1, which explains why this approach adds less than 2 percent overhead to application run time.

2) The time spent on compression is around 4 percent of the application's runtime, so there is not much significant gain in performing work asynchronously in active NVRAM. In our future work, we plan to use different compression techniques to better assess performance tradeoffs for on core vs. in Active NVRAM configurations.

C. Other Active NVRAM overheads

The compute cores and active compute element share the NVRAM device. While the compute cores write data to NVRAM, the active NVRAM reads data from NVRAM. For large chunk sizes there exists a contention reducing the effective I/O bandwidth to the application. Our initial results using the BTIO benchmark show overheads around 4-5%. This holds true for other I/O interfaces like network devices when post processing requires distributed processing (e.g., sorting). Our future work will explore methods to reduce such synchronization overheads and communication overheads via efficient NVRAM access and usage scheduling.

VI. Conclusions and Future Work

This paper analyzes the opportunities of using Active NVRAM for large scale data (I/O) intensive applications, using data post processing as a driving example. High performance application benchmarks are used to first evaluate I/O performance gains, followed by an analysis of the tradeoffs seen when doing post processing on Active NVRAM. Preliminary results show over 350% I/O throughput benefits when using NVRAM vs. disks. Further, while for sort based post processing our mechanism shows 160% improvement in performance, for compute intensive operations there is little benefit. These preliminary results are based on limited scales and are driven by benchmarks. In our next phase of research, we would like to improve the experiment scale and evaluate performance for different data post processing examples. We will also address some of the performance overheads of in-situ processing.

References

[1] H. Abbasi, et al., "Datastager: scalable data staging services for petascale applications," ser. HPDC. NY: ACM, 2009.
[2] "S4," http://s4.io/.
[3] "Flume," http://flumebase.org/.
[4] S. Jeffrey, et al., "The hadoop distributed filesystem: Balancing portability and performance," in *ISPASS*, 2010.
[5] C. Wang, et al., "A flexible architecture integrating monitoring and analytics for managing large-scale data centers," ser. ICAC, NY, 2011.
[6] D. Logothetis, et al., "In-situ mapreduce for log processing," ser. USENIX ATC'11, Berkeley, CA, 2011.
[7] J. Lofstead, et al., "Managing variability in the io performance of petascale storage systems," ser. SC, Washington, DC, 2010.
[8] P. Ranganathan, "From microprocessors to nanostores: Rethinking data-centric systems," *Computer*, 2011.
[9] F. Zheng, et al., "PreDatA - Preparatory Data Analytics on Peta-Scale Machines," in *IPDPS*, Atlanta, GA, 2010.
[10] H. Monti, et al., "CATCH: A Cloud-based Adaptive Data Transfer Service for HPC," in *IPDPS*, Anchorage, Alaska, 2011.
[11] R. Ananthanarayanan, et al., "Cloud analytics: do we really need to reinvent the storage stack?" ser. HotCloud, Berkeley, CA, 2009.
[12] E. Riedel, "Active disks: remote execution for network-attached storage," Ph.D. dissertation, Pittsburgh, PA, 1999.
[13] D. G. Andersen, et al., "Fawn: a fast array of wimpy nodes," ser. SOSP, NY, USA, 2009.
[14] H. Amur, et al., "Robust and flexible power-proportional storage," ser. SoCC, NY, 2010.
[15] S. Ostermann, et al., "A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing," in *Cloud Computing*, ser. ICST, Berlin, Heidelberg, 2010.
[16] "NAS Parallel Benchmarks," http://www.nas.nasa.gov/Resources/Software/npb.html.