Active Flash: Out-of-core Data Analytics on Flash Storage

Simona Boboila[*][‡], Youngjae Kim[†], Sudharshan S. Vazhkudai[†], Peter Desnoyers[*] and Galen M. Shipman[†]
[*]Northeastern University, [†]Oak Ridge National Laboratory
[*]{simona, pjd}@ccs.neu.edu, [†]{kimy1, vazhkudaiss, gshipman}@ornl.gov

***Abstract*—Next generation science will increasingly come to rely on the ability to perform efficient, on-the-fly analytics of data generated by high-performance computing (HPC) simulations, modeling complex physical phenomena. Scientific computing workflows are stymied by the traditional chaining of simulation and data analysis, creating multiple rounds of redundant reads and writes to the storage system, which grows in cost with the ever-increasing gap between compute and storage speeds in HPC clusters. Recent HPC acquisitions have introduced compute node-local flash storage as a means to alleviate this I/O bottleneck.**

We propose a novel approach, *Active Flash*, to expedite data analysis pipelines by migrating to the location of the data, the flash device itself. We argue that Active Flash has the potential to enable *true out-of-core* data analytics by freeing up both the compute core and the associated main memory. By performing analysis locally, dependence on limited bandwidth to a central storage system is reduced, while allowing this analysis to proceed in parallel with the main application. In addition, offloading work from the host to the more power-efficient controller reduces peak system power usage, which is already in the megawatt range and poses a major barrier to HPC system scalability.

We propose an architecture for Active Flash, explore energy and performance trade-offs in moving computation from host to storage, demonstrate the ability of appropriate embedded controllers to perform data analysis and reduction tasks at speeds sufficient for this application, and present a simulation study of Active Flash scheduling policies. These results show the viability of the Active Flash model, and its capability to potentially have a transformative impact on scientific data analysis.

I. Introduction

Scientific discovery today is becoming increasingly driven by extreme-scale computer simulations—a class of application characterized by long-running computations across a large cluster of compute nodes, generating huge amounts of data. As an example, a single 100,000-core run of the Gyrokinetic Tokamak Simulation (GTS) [49] fusion application on the Jaguar system at Oak Ridge National Laboratory (ORNL) produces roughly 50 TB of output data over a single 10 to 12-hour run. As these systems scale, however, I/O performance has failed to keep pace: the Jaguar system, currently number 3 on the Top500 list [47], incorporates over 250,000 cores and 1.2 GB of memory per core, but with a 240 GB/s parallel storage system has a peak I/O rate of less than 1 MB/s per core. With current scaling trends, as systems grow to larger numbers of more powerful cores, less and less storage bandwidth will be available for the computational output of these cores.

[‡] The author was an intern at Oak Ridge National Laboratory during the summer of 2011.

Massively parallel simulations such as GTS are only part of the scientific workflow, however. To derive knowledge from the volumes of data created in such simulations, it is necessary to analyze this data, performing varied tasks such as data reduction, feature extraction, statistical processing, and visualization. Current workflows typically involve repeated steps of reading and writing data stored on a shared data store (in this case the ORNL Spider [2] system, a center-wide Lustre [1] parallel file system), further straining the I/O capacity of the system.

As the scale of computational science continues to increase to peta-scale systems with millions of cores, I/O demands of both the front-end simulation and the resulting data analysis workflow are unlikely to be achievable by further scaling of the architectures in use today. Recent systems have incorporated node-local storage as a complement to limited-bandwidth central file systems; such systems include Tsubame2 at the Tokyo Institute of Technology [19] and Gordon [3] at the San Diego Supercomputing Center (SDSC). The Gordon system, as an example, is composed of 1024 16-core nodes, each with 64 GB DRAM and paired with a high-end 256 GB solid-state drive (SSD) capable of 200 MB/s streaming I/O.

Architectures such as this allow for *in situ* processing of simulation output, where applications schedule post-processing tasks such as feature extraction (e.g. for remote visualization) alongside simulation [30], [53]. By doing so, simulation output may be accessed locally on the nodes where it is produced, reducing the vast amounts of data generated on many-core nodes before any centralized collection steps. The advantage gained by avoiding multiple rounds of redundant I/O can only increase as storage subsystem performance continues to lag behind computation in large HPC systems.

Current approaches to in-situ data analysis in extreme-scale systems either use some fraction of the cores on the compute nodes to execute analysis routines, or utilize some partition of nodes that is part of the compute allocation of the simulation job [53]. For example, prior work at ORNL has dedicated a percentage of compute nodes to storing and analyzing output of the simulation application before storage of final results to the parallel file system [5], [27], [40].

Although such *in-core* in-situ analysis is able to avoid bottlenecks associated with central storage, it may adversely impact the main simulation. It competes with the main application not only for compute cycles, but for DRAM as well. Memory is becoming a critical resource in HPC systems, responsible for a significant portion of the cost and power budget today even as the memory-to-FLOP ratio has been steadily declining, from 0.85 for the No. 1 machine on Top500 in 1997 to 0.13 for Jaguar and 0.01 for the projected exaflop machine in 2018 [36], [47]. In addition, for a substantial class of HPC applications characterized by close, fine-grained synchronization between computation on different nodes, non-determinacy resulting from competing CPU usage can result in severe performance impairment [48], as such *jitter* causes additional waiting for "stragglers" at each communication step.

In addition to competing with the main application for time, this sort of in-situ analysis also consumes additional energy during a simulation. Power is rapidly becoming a limiting factor in the scaling of today's HPC systems—the average power consumption of the top 10 HPC systems today is 4.56 MW [47], with the No. 1 system drawing 12.66 MW. If peak power has become a constraint, then feasible solutions for addressing other system shortcomings must fit within that power constraint.

Rather than combining primary and post-processing computation on the same compute nodes, an alternate approach is what might be termed *true out-of-core*[1] data analysis, performed within the storage system rather than on the compute node. Earlier attempts to combine computation and storage have focused on adding application programmability on disk-resident CPUs [42] and parallel file system storage nodes [21]. This *Active Storage* approach has drawn renewed interest in recent years with the commercial success of Netezza [34], a specialized "data warehouse" system for analyzing data from business applications. In HPC contexts, however, an isolated analysis cluster such as Netezza poses many of the same scalability problems as centralized storage systems.

The recent availability of high-capacity solid-state storage, however, opens the possibility of a new architecture for combining storage and computation, which we term *Active Flash*. This approach takes advantage of the low latency and architectural flexibility of flash-based storage devices to enable highly distributed, scalable data analysis and post-processing in HPC environments. It does so by implementing, within the storage system, a toolbox of analysis and reduction algorithms for scientific data. This functionality is in turn made available to applications via both a tightly-coupled API as well as more loosely-coupled mechanisms based on shared access to files.

The contributions of this work include:

- A proposed architecture for Active Flash,
- Exploration of energy and performance trade-offs in moving computation from host systems to Active Flash,
- Demonstration of the ability of representative embedded controllers to perform data analysis and reduction tasks at competitive performance levels relative to modern node hardware, as evidence of the feasibility of this approach,
- A simulation study of Active Flash scheduling policies, examining the possibilities of scheduling controller-based computation both between and during host I/O.

In particular, we begin by describing SSD architecture and proposed Active Flash extensions, and then investigating different aspects of its feasibility, focusing on (a) energy savings and related performance trade-offs inherent in off-loading computation onto lower-power but lower-performance storage CPUs (Section III), (b) feasibility of realistic data analysis and reduction algorithms on such CPUs (Section IV), and (c) a simulation-based study (Section V) examining the degree to which storage and computation tasks compete for resources on the SSD. We finally survey prior work in Section VI and conclude.

II. Background

A. General SSD architecture

An SSD as shown in Figure 1 is a small general-purpose computer, based on a 32-bit CPU and typically 64-128 MB of DRAM, or more for high-end PCI-based devices. In addition it contains Host Interface Logic, implementing e.g. a SATA or PCIe target, a Flash Controller handling internal data transfers and error correction (ECC), and an array of NAND flash devices comprising the storage itself.

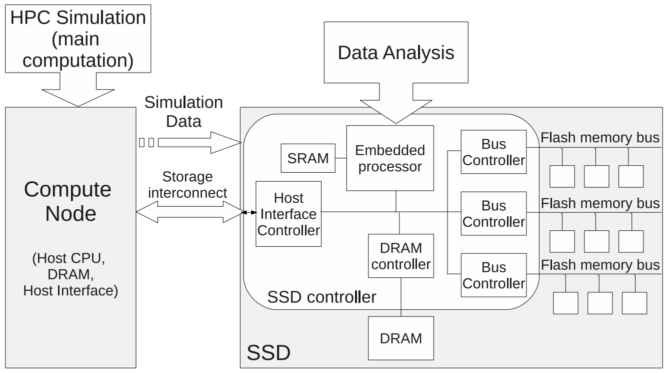


Fig. 1: Active Flash: the HPC simulation (main computation) is running on the compute node (host CPU), the simulation data is sent to the storage device (SSD) and the data analysis is carried out on the embedded processor (SSD controller). The general architecture of an SSD [23] is illustrated.

The internal architecture is designed around the operational characteristics of NAND flash. Storage on these chips is organized in *pages* which are read or written as units; these operations consist of a *command*, a *busy period*, and a *data transfer phase*. When reading a page, the busy period is fairly short (e.g. 50 μs per page, which is typically 4 KB) and is followed by the data transfer phase, which at today's flash chip speeds will typically take 40-100 μS for a 4 KB page at a flash bus speed of 40-100 MB/s. Writes are preceded by the data transfer phase, at the same speed, but require a busy period of 200-300 μs or more. Pages are organized in *erase blocks* of 64-256 pages (typically 128) which must be erased

[1]Typically the term "out-of-core" refers to the use of external storage as part of an algorithm processing data sets larger than memory. In the case described here, however, not only data but computation is shifted from memory and main CPU to the storage system.

as a unit before pages may be re-written; this operation is time-consuming (2 ms or more) but rare in comparison to writes.

Write bandwidth to a single flash chip is limited by operation latency; e.g. a 300 μs latency for writing 4 KB page results in a maximum throughput (i.e. with infinite bus speed) of less than 14 MB/s. High bandwidth is obtained by performing write operations on many chips simultaneously, across multiple chips sharing the same bus or *channel* (*multi-way* interleaving), as the bus is only needed by a particular chip during the data transfer phase, and across multiple buses (*multi-channel* interleaving).

The page write / block erase mechanism provided by NAND flash is fundamentally different from the re-writable sectors supported by hard disk drives (HDDs). It is hidden by flash management firmware termed the Flash Translation Layer (FTL), which performs out-of-place writes with re-mapping in order to present an HDD-like re-writable sector interface to the host. These tasks may be performed by a relatively low-end controller in inexpensive consumer devices (e.g. the Indilinx Barefoot incorporates an 80 MHz ARM CPU), while higher-end SSDs use speedier CPUs to reduce latency, such as the 4 780 MHz Tensilica cores in the OCZ RevoDrive X2.

B. Active Flash feasibility and architecture

These higher-end CPUs are what enable Active Flash, the architecture of which is shown in Figure 1. We assume that the HPC simulation—i.e. the main application—runs on the *compute node* or host CPU, generating large volumes of data which are sent to the storage device (SSD). By carrying out data analysis on the SSD itself, we avoid multiple rounds of redundant I/Os between the compute node and the storage device, and the overhead of these I/Os. In order to offload such processing, we take advantage of the following characteristics of today's SSDs:

- High I/O bandwidth: SSDs offer high I/O bandwidth due to interleaving techniques over multiple channels and flash chips; this bandwidth may be increased by using more channels (typically 8 on consumer devices to 16 or more on high-end ones) or flash chips with higher-speed interfaces. Typical read/write throughput values for contemporary SSDs are 150–250 MB/s, and up to 400-500 MB/s for PCIe-based devices such as the OCZ RevoDrive PCI-Express SSD [35].
- Availability of idle times in workloads: Although NAND flash management uses some CPU time on the embedded CPU, processor utilization on SSDs is highly dependent on workloads. The processor is idle between I/O accesses, and as higher and higher-speed CPUs are used to reduce per-request latency, may even be substantially idle in the middle of requests as well. These idle periods may in turn be used to run tasks that are offloaded from the host.
- High-performance embedded processors: a number of fairly high-performance CPUs have been used in SSDs, as mentioned previously; however there are also many other 'mobile-class' processors which fit the cost and power budgets of mid-to-high end SSDs. (e.g. the ARM Cortex-A9 [11] dual-core and quad-core CPUs, capable of operating at 2000 MHz) The comparatively low investment to develop a new SSD platform would make feasible an Active Flash device targeted to the HPC market.

We assume an active flash storage device based on the SSD architecture we have described, with significant computational power (although much less than that of the compute nodes) and low-latency high-bandwidth access to on-board flash. An out-of-band interface over the storage channel (e.g. using operations to a secondary LUN [9]) is provided for the host to send requests to the active flash device. These requests specify operations to be performed, but not data transfer, which is carried out by normal block write operations. The active flash commands indicate which logical block addresses (LBAs) correspond to the input and output of an operation; these would typically correspond to files within the host file system containing data in a standard self-describing scientific data format such as NetCDF [41] or HDF5 [20].

III. Performance–Energy Tradeoffs

We analyze the performance–energy tradeoffs of the Active Flash model. We generalize the study to a *hybrid* model, in which the SSD controller is used in conjunction with the host CPU to perform the data analysis. A fraction f of the data analysis is carried out on controller, and the rest on the host CPU. Moving the entire analysis on the controller is a specific case of the hybrid model, when $f = 1$. The two scenarios compared are:

- **baseline:** the entire data analysis is performed on the host CPU.
- **hybrid:** a part of the data analysis is carried out on the SSD controller; the rest, if any, is running on the host CPU.

We consider two HPC scenarios, which may occur in the baseline model and the host-side of the hybrid model:

- **alternate:** data analysis alternates with other jobs (e.g. HPC simulation). When data analysis is performed, it fully utilizes the CPU.
- **concurrent:** data analysis runs concurrently with other jobs (e.g. HPC simulation). It utilizes only a fraction of the CPU compute power.

Performance and energy consumption of the data analysis task are determined chiefly by data transfer and computation. Assuming data transfer takes place over a low-powered bus such as SATA/SAS (developed with mobile use in mind) the contribution of data transfer to energy consumption should be negligible, and will be ignored. This data transfer, however, plays a significant role in performance, giving the hybrid model a significant advantage over the baseline model, as fewer transfers occur, with no data ever being transferred from the controller back to the host CPU.

A. Performance study

Table I gives a complete list of variables used in this study. They address time, speed, energy, and CPU utilization.

Working alone (i.e. the baseline model), the host CPU takes time t_b to finish the entire computation. The controller is s

TABLE I: List of variables.

Data analysis parameters:	
baseline:	
t_b	total computation time (CPU time)
u_b	host CPU utilization
ΔE_b	host CPU energy consumption
hybrid:	
t_c	computation time on controller (CPU time)
u_c, u_h	controller and host CPU utilization
f	fraction of data analysis carried out on controller
S_e	effective slowdown (CPU time ratio)
S	visible slowdown (wall clock time ratio)
$\Delta E_c, \Delta E_h$	controller, host CPU energy consumption
ΔE	energy savings
Device parameters:	
s_h, s_c	host CPU speed, controller speed
s	s_h/s_c
H_{idle}, H_{load}	host CPU idle and load power consumption
C_{idle}, C_{load}	controller idle and load power consumption
ΔP_h	$H_{load} - H_{idle}$
ΔP_c	$C_{load} - C_{idle}$
p	$\Delta P_c / \Delta P_h$

times slower than the host CPU. Thus it finishes its share f of the data analysis in:

$$t_c = f \cdot s \cdot t_b \tag{1}$$

The effective slowdown of the computation in the hybrid model compared to the baseline model is:

$$S_e = \frac{t_c}{t_b} = f \cdot s \tag{2}$$

For *alternate* data analysis, the effective slowdown (CPU time ratio) equals the visible slowdown (wall clock time ratio).

For *concurrent* data analysis, the visible slowdown may be smaller than the effective slowdown (due to task parallelism on the host CPU), and depends on u_b, the fraction of time the data analysis job uses the CPU (i.e. the CPU utilization due to data analysis):

$$S = \frac{t_c}{t_b/u_b} = f \cdot s \cdot u_b \tag{3}$$

The fraction f of data analysis performed on the controller determines the visible slowdown. Moreover, for every u_b we can determine the fraction f to move on the controller such that the data analysis job incurs no visible slowdown cost, or can finish even faster than in the baseline approach.

A fraction f of the data analysis job running on an s times slower processor uses: $u_c = f \cdot s \cdot u_b$. We consider that the controller invests all its computation cycles in the data analysis: $u_c = 1$. Thus $f = 1/(s \cdot u_b)$. The entire work is done on the controller ($f = 1$) at $u_b = 1/s$.

To summarize, if $u_b \leq 1/s$ (e.g. due to competing load on the host CPU from the data-generating application), we can move the entire data analysis on the controller with no visible slowdown. (If $u_b < 1/s$ there is actually a speedup). If $u_b > 1/s$, we can move a fraction $f = 1/(s \cdot u_b)$ of the data analysis on the controller with no visible slowdown.

$$f = \begin{cases} 1, & \text{for } u_b \in [0, 1/s) \\ 1/(s \cdot u_b), & \text{for } u_b \in [1/s, 1] \end{cases} \tag{4}$$

In addition, a few host CPU cycles have become available for other jobs. The remaining host CPU utilization due to data analysis is:

$$u_h = (1-f) \cdot u_b = \begin{cases} 0, & \text{for } u_b \in [0, 1/s) \\ u_b - 1/s, & \text{for } u_b \in [1/s, 1] \end{cases} \tag{5}$$

B. Energy study

The energy savings in the hybrid model compared to the baseline model are:

$$\Delta E = 1 - \frac{\Delta E_h + \Delta E_c}{\Delta E_b} \tag{6}$$

The energy consumption of the host CPU in the hybrid model decreases with the fraction of work transferred to the controller: $\Delta E_h = (1-f) \cdot \Delta E_b$. Thus $\Delta E = f - \Delta E_c / \Delta E_b$.

The energy consumption over a time interval Δt at a power rate P is: $E = \Delta t \times P$. Considering a ΔP increase in power consumption between the idle and load processor states, the equation becomes:

$$\Delta E = f - \frac{t_c}{t_b} \cdot \frac{\Delta P_c}{\Delta P_h} \tag{7}$$

Finally, using the t_c formula determined in Equation (1), the energy savings are:

$$\Delta E = (1 - sp) \cdot f \tag{8}$$

C. Experimental study

In this section, we present a concrete estimation of energy savings compared to job slowdown. We conducted power usage and timing measurements on the following platforms to obtain realistic results:

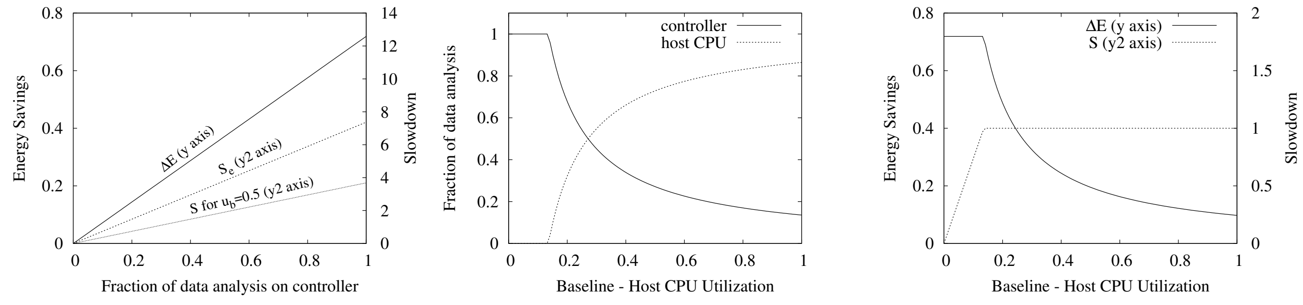
- Embedded CPU (controller): We measured an example of a high-end 32-bit controller, the 1 GHz ARM Cortex-A9 MPCore dual-core CPU running on the Pandaboard [37] development system.
- Host CPU: The Intel Core 2 Quad CPU Q6600 at 2.4 GHz.

We benchmarked the speed of the two processors for a single internal core, with Dhrystone [13]. Although Dhrystone is not necessarily an accurate predictor of application performance, results in later sections show that it gives a fairly good approximation for the ones we will look at (Section IV). We measured whole-system idle and load power consumption in each case. Table II presents the measured values and the resulting parameters s and p.

TABLE II: Measured parameters for speed and power consumption, and derived parameters s and p.

s_h (DMIPS)	s_c (DMIPS)	ΔP_h (W)	ΔP_c (W)	s	p
16215	2200	21	0.8	7.3	0.038

Figure 2 shows the performance-energy tradeoffs of the hybrid model. Figure 2a illustrates the energy savings and slowdown depending on the fraction of data analysis carried out on the controller. The x-axis shows the fraction f of data analysis performed on the controller. In the specific case of $f = 1$, i.e. the entire data analysis is performed on the controller, the energy savings reach the peak value of 0.72 at the effective slowdown cost of 7.3 (the S_e line at $f = 1$). However, if the data analysis job utilizes only half of the CPU time in the baseline model by sharing it with other concurrent



(a) Energy savings versus slowdown **(b)** Data analysis split without slowdown **(c)** Energy savings without slowdown

Fig. 2: Performance-Energy tradeoffs for the Hybrid Model. The controller is used in conjunction with the Host CPU to carry out part of the computation. No job slowdown occurs for $S = 1$.

jobs (the 'S for $u_b = 0.5$' line), its visible job slowdown in the hybrid model is proportionally less (e.g. for $f = 1$ the visible slowdown is about 3.6).

Figures 2b and 2c investigate how to split data analysis in the hybrid model to obtain energy savings without any visible job slowdown. Figure 2b shows the fraction of data analysis on each processor in this case. If the baseline host CPU utilization is smaller than 0.13 ($u_b < 0.13$ on the x-axis), the entire data analysis can be accommodate on the controller without a visible slowdown. Moreover, Figure 2c shows that in this case ($u_b < 0.13$), performing the entire data analysis on the controller gives a speedup ($S < 1$, y2-axis) and peak energy savings of 0.72. Even in the worst case (at $u_b = 1$ on the x-axis), i.e. full host CPU utilization due to data analysis (baseline model, alternate), the controller is able to free the host CPU by about 0.13 (Figure 2b), while saving about 0.1 of the baseline energy consumption (Figure 2c) without slowing down the data analysis.

D. Discussion

These results indicate that moving the entire data analysis on to the controller, or even just a fraction of it, can give significant energy savings. Moreover, the fraction of data analysis to be carried out on the controller can be tuned (based on the baseline host CPU utilization due to the data analysis job) to control the job slowdown cost. In some cases, energy savings can be obtained without slowing down the data analysis.

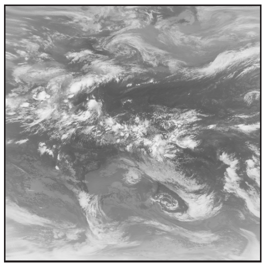
IV. Data Analysis Applications

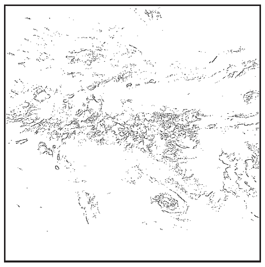
To demonstrate the feasibility of the Active Flash model in real-world cases, we examine four data analysis applications from representative domains of high performance computing. The post-processing performed in these examples is driven by the contrast between the large volumes of high-precision data which may be needed to represent the state of a simulation closely enough for it to evolve accurately over time, as compared to the lesser amount of detail which may be needed in order to examine the state of the simulated system at a single point in time. *Data reduction* encompasses a range of application-agnostic methods of lossy (e.g. decimation, precision reduction, etc.) or lossless compression; our analysis examines simple lossless data compression on scientific data in several formats. In contrast, *feature detection* refers to more application-aware computations, tailored to extract relevant data in a particular domain. We investigate two fairly general-purpose feature-detection algorithms—edge and extrema detection—on meteorological and medical data, as well as a specialized algorithm for heartbeat detection.

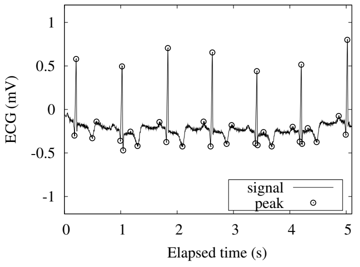
In estimating performance requirements for an Active Flash implementation, we note that HPC simulations do not necessarily output data at a constant rate. In particular, an alternate output behavior is that of checkpointing, where the computational state on all nodes is periodically written to storage, allowing recovery to the latest such checkpoint in the case of interruption due to e.g. hardware failure. Although transparent mechanisms for performing checkpointing exist [6], application-implemented checkpoints using files which may serve as the final output of the simulation are in fact common. We consider the case where an application writes a checkpoint to local SSD at regular intervals; the checkpoint is then post-processed on the Active Flash device for e.g. central collection and visualization. In this scenario we have a deadline for Active Storage computation; this processing must complete in the window before the next checkpoint is written. The size of these checkpoints is bounded by the node memory size, and their frequency is limited by the duration of the checkpoint write process, and the desire to minimize the overhead of these periodic halts on the progress of the main computation.

A. Description of applications

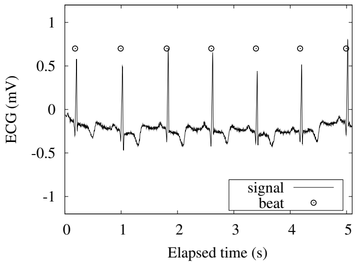
Edge detection: Edge detection is an example of feature extraction applied to image processing, in which specific portions (i.e. the edges) of a digital image are detected and isolated by identifying sudden changes in image brightness. We use SUSAN, an open source, low-level image processing application [45] examined in earlier studies of active storage [43], to detect edges in a set of weather images collected with GMS (Japan's Geostationary Meteorological Satellite system), which are publicly available on the Weather Home, Kochi University website [17]. Detecting useful patterns in meteorological data is clearly of practical value, in both forecasting and longer term climate analysis; edge detection


(a) Original image


(b) Detected edges



(c) Local extrema



(d) Detected heartbeats

Fig. 3: (a), (b) – Edge detection applied to an image rendering weather information from June 11, 2011, provided by GMS-5 (Japan Meteorological Agency) and the Kochi University Weather Home. The edges were detected with a brightness threshold of 40. (c) – Finding local maxima and minima (peaks) in a wave signal with the threshold distance set to 0.1. (d) – Detecting heartbeats in an electrocardiogram (ECG). For (c) and (d), the input data represents an ECG recording from the MIT-BIH Arrhythmia Database over a 5 seconds interval of recorded data.

has been used to identify region outliers associated with severe weather events [28], as well as to detect clouds and air flows. In Figure 3 we see a sample image, as well as the corresponding detected edges.

Finding local extrema: Finding the local maxima and minima in a noisy signal is a problem which appears in several fields, typically as one of the steps in peak detection algorithms, along with signal smoothing, baseline correction, and others. A comprehensive study of public peak detection algorithms on simulation and real data can be found in Yang et al [52].

We use the open source implementation available at [51] to detect local extrema in a wave signal, using a method [7] which looks for peaks above their surroundings on both sides by some threshold distance, and valleys below by a corresponding threshold. We apply this application to a set of ECG (electrocardiogram) signals from the MIT-BIH Arrhythmia Database [33]; example results may be seen in Figure 3c, using a threshold distance (delta) of 0.1.

Heartbeat detection: Heartbeat detection is a signal processing application with great impact in medical fields; although typically used with real patient data, the rise of computational science in medical fields [39] leads to applications of such algorithms in the HPC simulation environments targeted by Active Flash. We evaluate the performance of the SQRS heartbeat detection algorithm [15], which approximates the slope of an ECG signal by applying a convolution filter on a window of 10 values. It then compares this filtered signal against a variable threshold to decide whether a normal beat was identified; sample output is shown in Figure 3d. We evaluate an open source implementation of the SQRS algorithm from PhysioNet [18], applied to ECG signals from the MIT-BIH Arrhythmia Database.

Data compression: Data compression is used in many scientific domains to reduce the storage footprint and increase the effective network bandwidth. In a recent study, Welton *et* al. [50] point out the advantages of decoupling data compression from the HPC software to provide portable and transparent data compression services. With Active Flash, we propose to take the decoupling idea one step further, and harness the idle controller resources to carry out data compression on the SSD.

We use the LZO (Lempel-Ziv-Oberhumer) lossless compression method which favors speed over compression ratio [29]. Experiments were conducted using two common HPC data formats encountered in scientific fields: NetCDF (binary) data and text-encoded data. The data sets are extracted from freely available scientific data sources for atmospheric and geosciences research (NetCDF format) [10], and bioinformatics (text format) [16].

B. Experimental setup

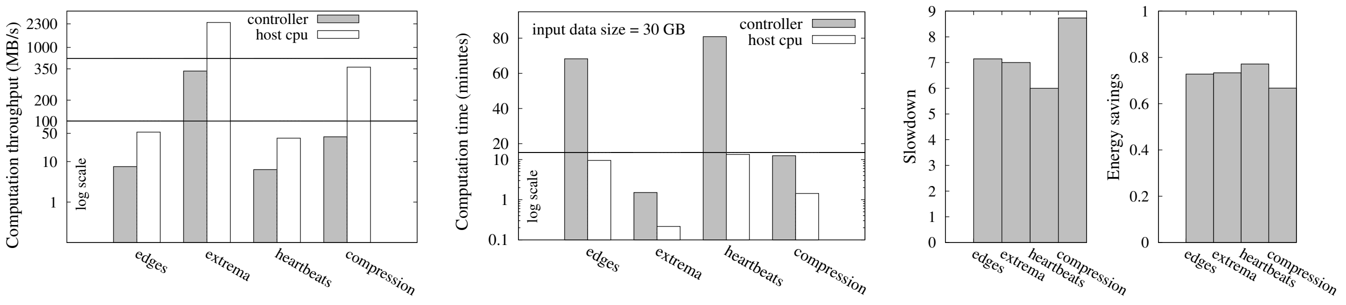
The experimental platforms used are the same as in Section III-C: the Pandaboard development system featuring a dual-core 1 GHz ARM Cortex-A9 MPCore CPU (controller), 1 GB of DDR2 SDRAM, and running Linux kernel 3.0, and a host machine featuring an Intel Core 2 Quad CPU Q6600 at 2.4 GHz, 4 GB of DDR2 SDRAM, and running Linux kernel 2.6.32. The applications chosen were all platform-independent C language programs; to reduce the effect of compiler differences GCC 4.6.1 was used on both platforms for all tests. Measurements were made of the computation phase of each program (i.e. excluding any time taken by input or output) running on a single core with no competing processes; measurements were made on both the host CPU and the controller.

C. Results

A summary of measured timings and data reduction values is given in Tabel III.

TABLE III: Data analysis applications. Measured timings and data reduction.

Application	**Computation throughput (MB/s)**		**Data reduction**
	controller	**host CPU**	**(%)**
Edge detection	7.5	53.5	97
Local extrema	339	2375	96
Heartbeat detection	6.3	38	99
Compression			
average bin&txt	41	358	49
binary (netcdf)	49.5	495	33
text	32.5	222	65



(a) Computation throughput **(b)** Computation time for 30 GB data processed **(c)** Slowdown versus energy savings

Fig. 4: (a) – Computation throughput, and (b) – computation time for 30 GB input data on the controller and on the host CPU. The bottom part of each figure uses a log-scale to observe low y-axis values in detail. (c) – Slowdown and energy savings estimated with Equation (7) using $f = 1$, and the measured times and power values.

Figure 4a gives a comparative view of computation speeds on controller and host CPU for the four data analysis applications. Edge detection and heartbeat detection are more computation intensive, compared to the other two applications. We used the detailed log-scale to display these speed values of about 7 MB/s on the controller. Data compression is next in terms of computation speed, averaging at about 41 MB/s. Unlike the previous applications, local extrema detection is less computationally intensive. Current interfaces commonly transfer data at about 200 MB/s. Thus this application is I/O bound instead of computation speed limited.

Figure 4b illustrates the time needed to process 30 GB of data (half the memory size per node of the Gordon system) at the measured speeds. Compression and local extrema detection are very fast even on the controller (under 15 minutes). Edge and heartbeat detection are slower, but still deliver acceptable timings (70-80 minutes) for a realistic scientific computation. We note that these measurements use fairly compact input data formats; it is likely that actual simulation data will be less dense due to the need for higher precision for each data point. Since the runtime of these algorithms is typically a function of the input size in data points, throughput in practice is likely to be higher.

Figure 4c shows how many times longer it takes to run these applications on controller, instead of the host CPU. On average, the slowdown is about 7.2, which confirms the benchmarking results from Section III. The same figure shows energy savings of about 0.75. The measured computation speeds for each application, and the measured power values of each test platform (Section III-C) are used in Equation (7) to derive the fraction of saved energy.

In all cases, the output can be represented much more compactly than the original data (it contains less information). The feature extraction applications delivered a substantial data reduction of over 90%, while compression averaged at about 50% for the two input data formats studied. We observe that compressing binary data is faster than compressing text, at the expense of a smaller compression ratio (the text format is more compressible than binary NetCDF). The input and output are in binary format for heartbeats and edge detection, and text format for local extrema.

D. Discussion

These results indicate that off-loading data analysis to a storage device based on a high-end controller has the potential to deliver acceptable performance in a high performance scientific computing environment. Using heartbeat detection as an example, the rate at which the ECGSYN electrocardiogram signal simulator [14] generates output data on our Intel host CPU is 3.2 MB/s of text data, equivalent to 0.26 MB/s in the binary format assumed in Figure 4. Even assuming 16 cores per host node, each producing simulation output at this rate, the total output is comfortably less than the 6.3 MB/s which could be handled on the controller. Alternately, assuming a checkpoint post-processing model, we see that the worst-case time for processing a volume of data equal to a realistic node checkpoint size is roughly an hour, making it realistic to consider flash-based processing of checkpoints in parallel with the main computation. Best suited for Active Flash are applications with minimal to no data dependencies, such as the ones illustrated here. Subsets of weather/ECG simulation data can be analyzed independently, without the need to exchange partial results among the compute and storage nodes. Also, we assume that jobs run without contention, as nodes are typically dedicated for an HPC application at a time.

V. Scheduling Data Analysis on Flash

In this section, we propose several policies to schedule both data analysis on the flash device and flash management tasks, i.e. garbage collection (GC). GC is typically triggered when the number of free blocks drops below a pre-defined threshold, suspending host I/O requests until completion; it is therefore important to schedule analysis and GC in a way that optimizes both analysis as well as overall device I/O performance.

A. Scheduling policies

The scheduling policies examined are as follows:

On-the-fly data analysis – Data written to the flash device is analyzed while it is still in the controller's DRAM, before being written to flash. The primary advantage of this approach is that it has the potential to significantly reduce the I/O traffic within the device by obviating the need to re-read (and thus re-write) data from the SSD flash. However, the success of this approach is dependent on factors such as the rate at which data is output by the main application, the computation throughput on the controller and the size of the controller DRAM. If data cannot be analyzed as fast as it is produced by the host-resident application, then the application must be throttled until the analysis can catch up.

Data analysis during idle times – In *idle-time* data analysis, controller-resident computation is scheduled only during idle times, when the main application is not performing I/O. Most HPC I/O workloads are bursty, with distinct periods of intense I/O and computation [24]; for these workloads, it is possible to accurately predict idle times [31], [32], and we exploit these idle times to schedule data analysis on the controller. This increases the I/O traffic inside the flash device, as data must be read from flash back into DRAM, and after computation written back to flash. However, our results indicate that, in many cases (i.e. computation bound data analysis), the additional background I/O does not hurt overall I/O performance.

Idle-time data analysis plus GC management – With *idle-time-GC* scheduling, optimistic garbage collection tasks as well as data analysis are controlled by the scheduler. Since GC will occur when absolutely necessary regardless of scheduling, data analysis is given priority: if an idle time is detected, but there is no data to be processed for data analysis, then, GC is scheduled to run instead. This complements the default GC policy, where GC is invoked when the amount of available space drops below a minimum threshold. Pushing GC earlier during idle times may incur additional write amplification than if GC were triggered later, because fewer pages are stale by the time GC is invoked. However, this early GC does not affect perfomance since it happens only when the device is idle.

B. *Simulator implementation and setup*

We have used the Microsoft Research SSD simulator [4], which is based on DiskSim [8] and has been used in several other studies [25], [26], [44]. We have simulated a NAND flash SSD, with the parameters described in Table IV. We have extended the event-driven SSD simulator to evaluate the three scheduling policies. In addition to the default parameters for SSD simulation, the Active Flash simulator needs additional parameters, which are shown in Table V.

TABLE IV: SSD Parameters.

Parameter	Value
Total capacity	64 GB
Flash chip elements	16
Planes per element	8
Blocks per plane	2048
Pages per block	64
Page size	4 KB
Reserved free blocks	15 %
Minimum free blocks	5 %
FTL mapping scheme	Page-level
Cleaning policy	Greedy
Page read latency	0.025 ms
Page write latency	0.2 ms
Block erase latency	1.5 ms
Chip transfer latency per byte	25 ns

TABLE V: Data analysis-related parameters in the SSD simulator.

Parameter	Value
Computation time per page of input	application-specific
Data reduction ratio	application-specific
GC-idle threshold (fraction of reserved space)	0.9

The MSR SSD implementation captures the I/O traffic parallelism over flash chip elements. However, the controller is a resource shared by all of the flash chips. While I/O requests to different flash chips may be scheduled simultaneously, computation (i.e. processing a new data unit on the controller) can only start after the previous one has ended. Our extension to the simulator accounts for this fundamental difference between handling I/O streams and computation.

We implemented the idle-time scheduling policy, wherein data analysis is triggered when the I/O queue is empty and there are no incoming requests. A typical GC policy such as that implemented in this simulator will invoke GC when the amount of free space drops below a minimum threshold (Table IV - 5% of the storage space, equivalent to 0.33 of the reserved space). In order to implement the idle-time-GC data analysis policy, we introduced an additional GC threshold, the *GC-idle* threshold, set to a high value (0.9 of the reserved space, equivalent to 13.5% of the storage space) to allow additional dirty space to be reclaimed during idle times.

While we expect a high degree of sequentiality in HPC data, we have experimented with worst-case conditions. We have simulated a synthetic write workload, consisting of small random writes, to represent the data generated by the main application on the host CPU. The request lengths are exponentially distributed, with a 4K mean value, and the inter-arrival rate (modeled by a uniform distribution) is set accordingly in each experiment to simulate different data generation rates of the scientific application running on the host CPU. The volume of write requests issued is 1 GB in every test.

C. *Results*

Figure 5 illustrates the potential bottlenecks in the Active Flash model: the computation throughput of the analysis on the controller, the flash management activity, in particular GC, and the I/O bandwidth of flash. In our results, we evaluated the scheduling policies by studying the effects of these limiting factors.

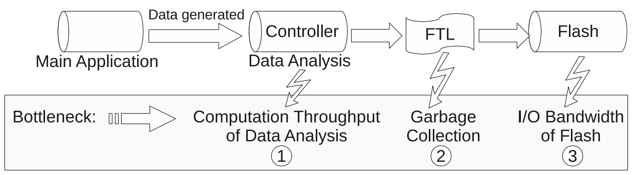


Fig. 5: Limiting factors in Active Flash.

To ensure high internal activity, the entire logical space of the SSD is initially filled with valid data. As the state of the

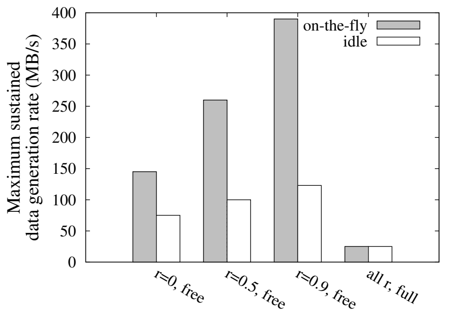


Fig. 6: I/O bound data analysis. Maximum sustained data generation rate of the scientific application for 'on-the-fly' and 'idle-time' data analysis running entirely on the controller, for cases 'free' (no GC), and 'full' (intensive GC). r = data reduction ratio (values of r experimented with: 0, 0.5, 0.9).

reserved block list plays a critical role in SSD performance, we considered the following two experimental conditions for free block list:

- **free**: All reserved blocks are initially free. A previous GC process has already reclaimed the space.
- **full**: The reserved blocks are initially filled with invalid data, resulting from previous write requests (updates). We maintained only a very small number of free blocks in order for the GC process to work. Victim blocks selected for cleaning contain mostly invalid pages, and possibly a few valid pages. Before the block is erased, the valid pages are moved in a new free block. A minimal number of free blocks (in our experiments, 5 blocks per flash chip) ensures that there is space to save the valid data during cleaning.

In the following experiments, we refer to the data generation rate of the main application running on the host CPU as *data generation rate*, and to the computation throughput of the data analysis running on the SSD controller as *computation throughput*.

I/O bound data analysis: We evaluated an I/O bound data analysis (e.g. local extrema detection–Section IV), in which the I/O bandwith of flash represents the main bottleneck (bottleneck 3 in Figure 5). With contemporary SSDs, featuring high I/O bandwidth of 150-250 MB/s, and even higher for some PCIe-based SSDs (400-500 MB/s), the case of I/O bound data analysis is expected to be less common.

In these experiments, we compare on-the-fly and idle-time data analysis scheduling policies, when all analysis was performed on the controller (case $f = 1$ in the hybrid model from Section III). A very high throughput (390 MB/s) was set for controller-based data analysis, so that the maximum write bandwith (145 MB/s) of the flash array in our simulated SSD would be the bottleneck; results are presented in Figure 6.

Case 'free' (no GC): If the entire reserved space of the emulated SSD is initially free, the SSD can accommodate 1 GB of write requests without the need to invoke GC. In this case, the controller-resident data analysis can keep up with a maximum data generation rate which highly depends on the data reduction resulting from analysis.

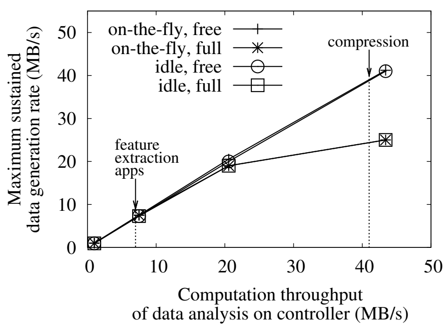


Fig. 7: Computation bound data analysis. Maximum sustained data generation rate of the scientific application depending on the computation throughput of in-storage data analysis for cases 'free' (no GC), and 'full' (intensive GC).

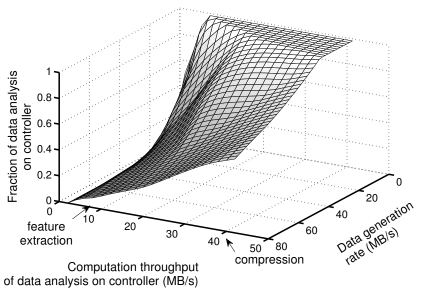
First we present the results for the *on-the-fly* policy. If no data reduction was obtained after data analysis ($r = 0$), then the same volume of data is written to the SSD, and the maximum sustained data rate is limited by the I/O bandwidth of the SSD (145 MB/s). If the data analysis resulted in $r = 0.5$ data reduction, then only half of the input data size is written to the SSD, resulting in a higher data generation rate of about 260 MB/s which can be sustained. If the data analysis reduced the data considerably, by $r = 0.9$, the I/O traffic is much decreased, and the computation part of the data analysis on the SSD becomes the limiting factor (bottleneck 1 in Figure 5), at 390 MB/s (i.e. the computation throughput of the data analysis job running on the controller).

For the *idle-time* data analysis policy, the maximum sustained data generation rate ranges from 75 MB/s to 123 MB/s, increasing with data reduction (Figure 6). However, with this scheduling policy, the entire data is first written to the SSD, which reduces the maximum sustained rate below the I/O bandwidth of the SSD. Other factors that contribute to the smaller sustained data generation rate of the idle-time policy compared to the on-the-fly policy (for I/O bound data analysis) are: additional background I/O traffic necessary to read the data back to the controller and then write the results of data analysis, and restricting data analysis to idle times only.

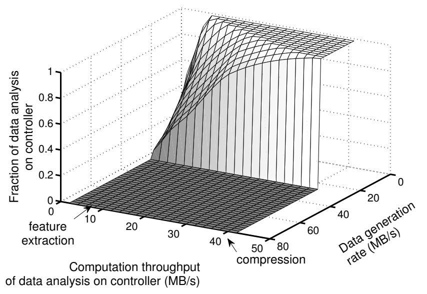
Case 'full' (intensive GC): If we start without any free space (Figure 6), intensive space cleaning is required to bring the minimum number of free blocks above the minimum limit. Due to garbage collection, the maximum sustained data generation rate for 1 GB of data drops to 25 MB/s regardless of the value of data reduction ratio (bottleneck 2 in Figure 5).

Computation bound data analysis: We studied the maximum sustained data generation rate of the scientific application of computation bound analysis (bottleneck 1 in Figure 5), for on-the-fly and idle-time scheduling policies. The entire data analysis was performed on the SSD controller ($f = 1$ in the hybrid model discussed in Section III) by the time the host-resident application finished generating data. Since in this case the computation was the limiting factor, data reduction did not have a significant effect on the results and was set to 0.5.

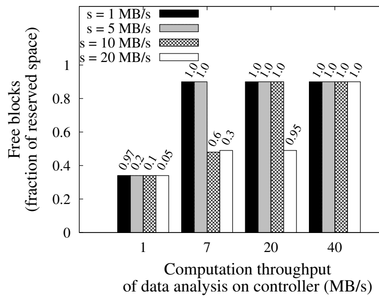
Figure 7 shows the maximum sustained data generation rate depending on the computation throughput of controller-



(a) Fraction of data analysis run on controller, 'free' start state (no GC)



(b) Fraction of data analysis run on controller, 'full' start state (intensive GC)



(c) Data analysis with Garbage Collection management, 'full' start state (intensive GC)

Fig. 8: (a), (b) - In-situ data analysis during idle times in hybrid schemes. Fraction of the data analysis which can be accommodated on the controller, while being able to sustain a specific data generation rate. In (b), intensive GC saturates the SSD at about 25 MB/s. (c) - Garbage collection is pushed earlier during extra available idle times. The bar labels represent the fraction of data analysis accommodated on the controller. The y-axis shows fraction of reserved blocks that were clean at the end of each experiment (which started with no reserved blocks free), for different data generation rates 's'.

resident data analysis. As concrete examples, we pinpoint on the figure the computation bound data analysis applications whose performance was measured in Section IV (feature extraction, i.e. edge/heartbeat detection running at 7 MB/s, and compression running at about 41 MB/s on the controller).

Case 'free' (no GC): When data analysis has the entire reserved space free initially, no garbage collection is required to process the 1 GB of data. The maximum data generation rate is dictated by the computation throughput of the data analysis. Both on-the-fly and idle-time strategies show a linear increase with the computation throughput on the controller.

Case 'full' (intensive GC): High GC activity was required when data analysis was started with no free reserved space. The maximum data generation rate increased linearly with the computation throughput of in-storage data analysis up to 20 MB/s, after which the background GC and related I/O activity become the limiting factor (bottleneck 2 in Figure 5).

Data analysis in hybrid schemes: In the results above, we investigated the maximum sustained data generation rate of the scientific application to accomplish the *entire* data analysis on the controller ($f = 1$ in the hybrid model from Section III). Here we examine the case where only a fraction $f < 1$ of the data analysis is offloaded to the storage controller with the rest carried out on the host CPU.

The hybrid model works best with the idle-time scheduling policy, based on the following considerations. For the on-the-fly policy, generated data is stored in the DRAM residing on the SSD and the data analysis job running on the controller reads the data from DRAM for processing. The size of the DRAM incorporated in the SSD restricts the amount of data that can be stored for analysis. Once the DRAM becomes full, the data analysis needs to keep up with the main application. With the idle-times policy, the generated data is stored on the SSD, and, depending on the availability of idle times, a portion of the data is processed by the data analysis job running on the controller. Thus, higher data generation rates (having fewer idle time periods) can also be sustained, however, in that case, only a part of the data analysis can be accommodated on the controller, and the rest will be carried out on the host CPU. Scheduling analysis during idle times allows for trading part of the active data analysis for a higher data generation rate.

Figures 8a and 8b examine this tradeoff—the fraction of data analysis possible on the controller versus the host-resident application data generation rate— for different data analysis speeds, ranging from 2 MB/s up to 43 MB/s. Since these values are smaller than the maximum sustained data generation rate for I/O bound data analysis ('idle') illustrated in Figure 6 (i.e. 75-125 MB/s depending on data reduction), the data analysis in these experiments is computation bound (bottleneck 1 in Figure 5). Next we discuss the results illustrated in Figures 8a and 8b for the data analysis examples described in Section IV, i.e. compression and feature extraction applications.

Case 'free' (no GC): The *entire* data analysis can be carried out on the controller at data generation rates smaller or equal to the data analysis computation throughput on controller (i.e. 41 MB/s for compression and about 7 MB/s for feature extraction), as was previously discussed (see the computation bound data analysis section). For feature extraction, when the data generation rate is increased from 7 MB/s to 25 MB/s, the controller can still handle 0.3 of the entire data analysis during idle times, and a further increase to 60 MB/s of data generation rate drops this fraction to 0.1. For compression, when the data generation rate is increased from 41 MB/s to 60 MB/s, the controller can still handle 0.7 of the data analysis.

Case 'full' (intensive GC): The impact of intensive GC is shown in Figure 8b. At 25 MB/s data generation rates, the controller can handle a fraction of 0.28 for feature extraction analysis, while compression is able to run to completion. For data generation rates higher than 25 MB/s, intensive cleaning eventually saturates the SSD (bottleneck 2 in Figure 5).

Data analysis with Garbage Collection management: Previous results showed the high impact of GC on SSD performance. The third scheduling policy proposed here addresses this concern by tuning GC to take advantage of idle time availability in application workloads.

The default GC mechanism implemented in the SSD simulator triggers GC when the number of free reserved blocks drops under a hard (low) threshold (Table IV). We introduced an additional soft (high) threshold (Table V) to coordinate the idle-time GC activity. Thus, when the number of free blocks is between the two thresholds, we push the GC earlier during idle times, given that there is no data to be processed on the controller at that moment.

The experiments started without any free space (*Case 'full'*), to trigger intensive GC (bottleneck 2 in Figure 5), and Figure 8c shows the fraction of reserved blocks that are clean at the end of the experiments. The bars in the figure are labeled with the fraction of data analysis that the controller was able to accommodate during idle times. The results are illustrated for different computation throughputs of data analysis, and various application data generation rates.

In all cases, GC needs to raise the number of free blocks at least up to the low threshold (cleaning 0.33 of the entire reserved space). For slow data analysis (1 MB/s), this is the most it can do. Since the computation takes long, data analysis on the controller monopolizes idle times. For faster data analysis, such as 7 MB/s in case of feature extraction applications, and small data generation rates (1 MB/s, 5 MB/s), GC is able to raise the number of free blocks up to the high threshold (cleaning 0.9 of the reserved space), while performing the entire data analysis on the controller. Sustaining higher data generation rates is possible with faster data analysis, e.g. GC cleans 0.9 of the reserved blocks during compression (running at 41 MB/s) of data generated at a rate of 20 MB/s.

D. Discussion

These results indicate that the *on-the-fly* policy offers the advantage of significantly reducing the I/O traffic, while the *idle-time* policy maximizes storage resource utilization by carrying out data analysis during low-activity periods. Also, idle-time scheduling offers flexibility: it permits sustaining the desired (high) application data generation rate, when only part of the data analysis is performed on the controller, and the rest on the host CPU (the hybrid Active Flash model).

Multiple factors affect the maximum sustained data generation rate, depending on the type of data analysis. For I/O bound data analysis, the data reduction size obtained from running the analysis has a major impact. This is not the case for computation bound data analysis, where the computation throughput of data analysis determines the maximum sustained data generation rate.

Garbage collection activity significantly affects performance. *GC tuning during idle times* proposed in the third policy can be a valuable resource. Consider a sequence of application workloads that generate data at different rates. We can take advantage of the extra idle times in slower applications, to clean the invalid blocks on the SSD and thus be able to accommodate the other faster applications in the workload as well.

VI. Related Work

Active storage techniques that move computation closer to data have a history dating back to the Gamma database machine [12], but the first proposals for shifting computation to the controller for the storage device itself—Active Disk [43], IDISK [22], and others—were prompted by the shift to 32-bit controllers for disk drive electronics in the 1990s. Although data computation tasks such as filtering, image processing, etc. were demonstrated in this environment, the computational power of disk-resident controllers has not increased significantly since then, while the real-time demands of mechanical control make for a poor environment for user programming. More recently, active storage concepts have also been pursued in the context of parallel file systems [38], [46], harnessing the computing power of the *storage nodes*, or hosts dedicated to disk management and data transfer (e.g. Jaguar's Lustre parallel file system at ORNL uses 192 dual-socket, quad-core, 16 GB RAM I/O servers [2]).

In contrast to hard disk drives (HDDs), semiconductor-based devices such as NAND flash SSDs show far more promise as platforms for computation as well as I/O. While flash translation layer algorithms are often complex, they have no real-time requirements—unlike e.g. disk head control, there is no risk of failure if an operation completes too late or too early. Unlike HDDs, with a single I/O channel to the media, internal I/O bandwidth inside SSDs continues to increase as channel counts increase and new interface standards are developed [4], [23], [44]. High-performance control processors are used in order to handle bursty I/O interrupts and reduce I/O latency [23], but these latency-sensitive operations account for only a small amount of total time, especially if handled quickly, leaving resources available to run other tasks such as post-processing, data conversion and analysis.

Recent work by Kim et al. [23] has studied the feasibility of SSD-resident processing on flash in terms of performance-power tradeoffs and showed that off-loading index scanning operations to an SSD can provide both higher performance and lower energy cost than performing the same operations on the host CPU. Our research extends this work with a general analytical model for performance-power and computation-I/O trade-offs, as well as experimental verification of performance and energy figures. In addition, simulation of the SSD and I/O traffic is used to explore I/O scheduling policies in detail, examining contention between I/O and computation and its result on application throughput.

VII. Concluding Remarks

As HPC clusters continue to grow, relative performance of centralized storage subsystems has fallen behind, with state-of-the-art computers providing an aggregate I/O bandwidth of 1 MB/s per CPU core. By moving solid-state storage to the cluster nodes themselves, and utilizing energy-efficient storage controllers to perform selected *out-of-core* data analysis applications, Active Flash addresses both I/O bandwidth and system power constraints which limit the scalability of today's HPC systems. We examine the energy-performance trade-offs of the Active Flash approach, deriving models that describe the regimes in which Active Flash may provide improvements in energy, performance, or both. Measurements of data analysis throughput and corresponding power consumption for actual HPC algorithms show that out-of-core computation using Active Flash could significantly reduce total energy with little performance degradation, while simulation of I/O-compute trade-offs demonstrates that internal scheduling may be used to allow Active Flash to perform data analysis without impact on I/O performance.

ACKNOWLEDGEMENTS

This work was sponsored in part by ORNL, managed by UT Battelle, LLC for the U.S. DOE (Contract No. DE-AC05-00OR22725), and in part by an IBM Faculty Award.

REFERENCES

[1] "Lustre DDN tuning," http://wiki.lustre.org/index.php/Lustre_DDN_Tuning.
[2] "Spider," http://www.nccs.gov/2008/06/30/nccs-launches-new-file-management-system/, 2008.
[3] "Supercomputer uses flash to solve data-intensive problems 10 times faster," http://www.sdsc.edu/News%20Items/PR110409_gordon.html, 2009.
[4] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *USENIX ATC*, 2008, pp. 57–70.
[5] S. Al-Kiswany, M. Ripeanu, and S. S. Vazhkudai, "Aggregate memory as an intermediate checkpoint storage device," Oak Ridge National Laboratory, Oak Ridge, TN, Technical Report 013521, Nov. 2008.
[6] J. Ansel, K. Arya, and G. Cooperman, "DMTCP: Transparent checkpointing for cluster computations and the desktop," in *IPDPS*, 2009.
[7] E. Billauer, "Peak detection," http://billauer.co.il/peakdet.html, 2011.
[8] J. S. Bucy, J. Schindler, S. W. S. G. R. Ganger, and Contributors, "The DiskSim Simulation Environment Version 4.0 Reference Manual," Tech. Rep., 2008.
[9] E. Budilovsky, S. Toledo, and A. Zuck, "Prototyping a high-performance low-cost solid-state disk," in *SYSTOR*, 2011, pp. 13:1–13:10.
[10] "CISL Research Data Archive. CORE.2 Global Air-Sea Flux Dataset," http://dss.ucar.edu/dsszone/ds260.2.
[11] "Cortex-A9 Processor," http://www.arm.com/products/processors/cortex-a/cortex-a9.php.
[12] D. J. DeWitt and P. B. Hawthorn, "A Performance Evaluation of Data Base Machine Architectures," in *VLDB*, 1981, pp. 199–214.
[13] "ECL Dhrystone Benchmark," www.johnloomis.org/NiosII/dhrystone/ECLDhrystoneWhitePaper.pdf, White Paper.
[14] "ECGSYN: A realistic ECG waveform generator," http://www.physionet.org/physiotools/ecgsyn/.
[15] W. A. H. Englese and C. Zeelenberg, "A single scan algorithm for QRS detection and feature extraction," in *IEEE Computers in Cardiology*, 1979, p. 3742.
[16] "European Bioinformatics Institute. Unigene Database," ftp://ftp.ebi.ac.uk/pub/databases/Unigene/.
[17] "GMS/GOES9/MTSAT Data Archive for Research and Education," http://weather.is.kochi-u.ac.jp/archive-e.html.
[18] A. L. Goldberger, L. A. N. Amaral *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.
[19] T. Hatazaki, "Tsubame-2 - a 2.4 PFLOPS peak performance system," in *Optical Fiber Communication Conference*, 2011.
[20] HDF Group, "Hierarchical data format, version 5," http://hdf.ncsa.uiuc.edu/HDF5.
[21] T. M. John, A. T. Ramani, and J. A. Chandy, "Active storage using Object-Based devices," in *HiperIO*, Tsukuba, Japan, 2008.
[22] K. Keeton, D. A. Patterson, and J. M. Hellerstein, "A Case for Intelligent Disks (IDISKs)," in *SIGMOD Record*, vol. 27, 1998, pp. 42–52.
[23] S. Kim, H. Oh, C. Park, S. Cho, and S.-W. Lee, "Fast, Energy Efficient Scan inside Flash Memory SSDs," in *ADMS*, 2011.
[24] Y. Kim, R. Gunasekaran, G. M. Shipman, D. Dillow, Z. Zhang, and B. W. Settlemyer, "Workload characterization of a leadership class storage," in *PDSW*, 2010.
[25] Y. Kim, S. Oral, G. M. Shipman, J. Lee, D. A. Dillow, and F. Wang, "Harmonia: A globally coordinated garbage collector for arrays of solid-state drives," in *MSST*, 2011, pp. 1–12.
[26] J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim, "A semi-preemptive garbage collector for solid state drives," in *ISPASS*, 2011, pp. 12–21.
[27] M. Li, S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. M. Shipman, "Functional Partitioning to Optimize End-to-End Performance on Many-core Architectures," in *SC'10*, 2010.
[28] C.-T. Lu and L. R. Liang, "Wavelet fuzzy classification for detecting and tracking region outliers in meteorological data," in *GIS*, 2004, pp. 258–265.
[29] "LZO real-time data compression library," http://www.oberhumer.com/opensource/lzo/.
[30] K.-L. Ma, "In situ visualization at extreme scale: Challenges and opportunities," *IEEE Comput. Graph. Appl.*, vol. 29, no. 6, pp. 14–19, 2009.
[31] N. Mi, A. Riska, X. Li, E. Smirni, and E. Riedel, "Restrained utilization of idleness for transparent scheduling of background tasks," in *SIGMETRICS/Performance*, 2009, pp. 205–216.
[32] N. Mi, A. Riska, Q. Zhang, E. Smirni, and E. Riedel, "Efficient management of idleness in storage systems," *Trans. Storage*, vol. 5, pp. 4:1–4:25, 2009.
[33] G. B. Moody and R. G. Mark, "The impact of the MIT-BIH Arrhythmia Database," *IEEE Eng in Med and Biol*, vol. 20, pp. 45–50, 2001.
[34] "The Netezza Data Appliance Architecture: A Platform for High Performance Data Warehousing and Analytics," White Paper, 2010.
[35] "OCZ RevoDrive PCI-Express SSD Specifications," http://www.ocztechnology.com/ocz-revodrive-pci-express-ssd.html.
[36] U. D. of Energy, "DOE exascale initiative technical roadmap," December 2009, http://extremecomputing.labworks.org/hardware/collaboration/EI-RoadMapV21-SanDiego.pdf.
[37] "The Pandaboard Development System," http://pandaboard.org/.
[38] J. Piernas, J. Nieplocha, and E. J. Felix, "Evaluation of active storage strategies for the Lustre parallel file system," in *SC*, 2007, pp. 28:1–28:10.
[39] B. J. Pope, B. G. Fitch, M. C. Pitman, J. J. Rice, and M. Reumann, "Performance of hybrid programming models for multiscale cardiac simulations: Preparing for petascale computation," *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 10, pp. 2965–2969, 2011.
[40] R. Prabhakar, S. S. Vazhkudai, Y. Kim, A. R. Butt, M. Li, and M. Kandemir, "Provisioning a Multi-tiered Data Staging Area for Extreme-Scale Machines," in *ICDCS'11*, 2011.
[41] R. Rew and G. Davis, "NetCDF: an interface for scientific data access," *IEEE Comput. Graph. Appl.*, vol. 10, no. 4, pp. 76–82, 1990.
[42] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagle, "Active disks for large-scale data processing," *Computer*, vol. 34, pp. 68–74, 2001.
[43] E. Riedel, G. A. Gibson, and C. Faloutsos, "Active storage for large-scale data mining and multimedia," in *VLDB*, 1998.
[44] J.-Y. Shin, Z.-L. Xia, N.-Y. Xu, R. Gao, X.-F. Cai, S. Maeng, and F.-H. Hsu, "FTL design exploration in reconfigurable high-performance SSD for server applications," in *ICS*, 2009, pp. 338–349.
[45] S. Smith and J. Brady, "Susan - a new approach to low level image processing," *Int'l Journal of Computer Vision*, vol. 23, pp. 45–78, 1997.
[46] S. W. Son, S. Lang, P. Carns, R. Ross, R. Thakur, B. Ozisikyilmaz, P. Kumar, W.-K. Liao, and A. Choudhary, "Enabling active storage on parallel I/O software stacks," in *MSST*, 2010, pp. 1–12.
[47] "Top500 supercomputer sites," http://www.top500.org/.
[48] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick, "System noise, OS clock ticks, and fine-grained parallel applications," in *ICS*, 2005, pp. 303–312.
[49] W. Wang, Z. L. W. Tang *et al.*, "Gyrokinetic Simulation of Global Turbulent Transport Properties in Tokamak Experiments," *Physics of Plasmas*, vol. 13, 2006.
[50] B. Welton, D. Kimpe, J. Cope, C. M. Patrick, K. Iskra, and R. B. Ross, "Improving i/o forwarding throughput with data compression," in *CLUSTER*, 2011, pp. 438–445.
[51] H. Xu, "Peak detection in a wave data, C source code," https://github.com/xuphys/peakdetect, 2011.
[52] C. Yang, Z. He, and W. Yu, "Comparison of Public Peak Detection Algorithms for MALDI Mass Spectrometry Data Analysis," *BMC Bioinformatics*, vol. 10, 2009.
[53] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "Predata - preparatory data analytics on peta-scale machines," in *IPDPS*, 2010.