Active Flash: Towards Energy-Efficient, In-Situ Data Analytics on Extreme-Scale Machines

Devesh Tiwari [1]*, *Sudharshan S. Vazhkudai* [2], *Youngjae Kim* [2], *Xiaosong Ma* [1,2], *Simona Boboila* [3], *and Peter J. Desnoyers* [3]
[1]*North Carolina State University,* [2]*Oak Ridge National Laboratory,* [3]*Northeastern University*
{devesh.dtiwari, ma}@ncsu.edu, {vazhkudaiss, kimy1}@ornl.gov, {simona, pjd}@ccs.neu.edu *North Carolina State University*
{devesh.dtiwari}@ncsu.edu

Abstract

Modern scientific discovery is increasingly driven by large-scale supercomputing simulations, followed by data analysis tasks. These data analyses are either performed offline, on smaller-scale clusters, or in-situ, on the supercomputer itself. Both of these strategies are rife with storage and I/O bottlenecks, energy inefficiencies due to increased data movement, and increased time to solution. We propose Active Flash, a novel approach to in-situ, scientific data analysis, wherein data analysis is conducted on the solid-state device (SSD), where the data already resides. Our performance and energy models show that Active Flash has the potential to address many of the aforementioned concerns. In addition, we demonstrate an Active Flash prototype built on a commercial SSD controller, which further reaffirms the viability of our proposal.

1 Introduction

High performance computing (HPC) simulations on large-scale supercomputers (e.g., the petascale Jaguar machine, No. 6 on the Top500 list [41]) routinely produce vast amounts of result output data [1, 33]. Examples of such applications include astrophysics (Chimera, Vulcan/2D), climate (POP), combustion (S3D), and fusion (GTC and GYRO) (Table 1). Deriving insights from these simulation results often involves performing a sequence of *data analysis* operations.

Traditionally, the simulation jobs and data analysis of the simulation outputs are conducted on separate computing resources. Data analysis tasks are run *offline*, on smaller clusters, *after* the completion of the simulation job, as shown in Figure 1. The high-end computing (HEC) machine and the analysis clusters tend to share a high-speed scratch parallel file system (PFS) to read/write input/output data, (e.g., the Lustre storage system [9] that connects Jaguar with several analysis clusters in the machine-room.) The reason for using *offline* processing for analysis is that CPU hours are expensive on HEC machines like Jaguar. Therefore, HPC users generally utilize the allocated CPU hours for FLOP-intensive codes such as the simulation job, instead of data analysis tasks.

Unfortunately, this traditional offline approach suffers from both performance and energy inefficiencies. It requires redundant I/O (simulations write, analyses read), resulting in excessive data movement across the compute and storage subsystems. As we transition from petaflop to exaflop systems, the energy cost due to data movement is predicted to be comparable, if not more than the computing cost [28]. At the same time, technology projections indicate that energy efficiency will become the primary metric for system design, as compute power is expected to increase by 1000× in the next decade with only a 10× increase in power envelope [31]. Therefore, an urgent challenge is to expedite data analysis in an energy-efficient manner, without degrading or interfering with the main simulation computation.

A promising solution is to perform *"in-situ"* analysis [22] on in-transit output data, before it is written to the PFS. Although this eliminates redundant I/O, it uses expensive compute nodes for the relatively less FLOP-

Application	Analysis data generation rate (per node)	Checkpoint data generation rate (per node)
CHIMERA	4400 KB/s	4400 KB/s
VULCUN/2D	2.28 KB/s	0.02 KB/s
POP	16.3 KB/s	5.05 KB/s
S3D	170 KB/s	85 KB/s
GTC	14 KB/s	476 KB/s
GYRO	14 KB/s	11.6 KB/s

Table 1: Output characteristics of parallel simulations on Jaguar, amortized over the entire run. Outputs comprise both result (analysis) and recovery (checkpoints) data.

*This work was performed while the author was at Oak Ridge National Lab on an ASTRO fellowship.

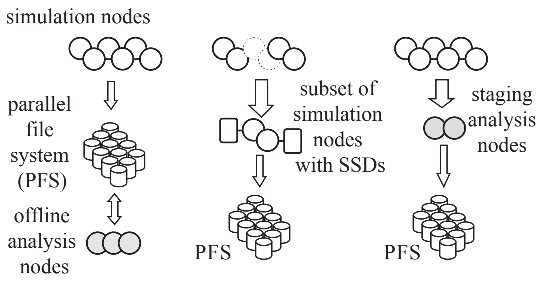


Figure 1: Three approaches to data analysis: (1) *offline* (leftmost), with another cluster for analysis, (2) *active flash* (middle), with analysis on a subset of the simulation nodes with SSDs, and (3) *analysis node* (rightmost), with an extra set of nodes on the same machine for processing.

intensive data analysis tasks. Moreover, it may even slow down the simulation job due to interference, resulting in inefficient use of expensive resources. As in-situ analysis gains interest, we may begin to see dedicated *analysis nodes*, much like service or I/O nodes.

In this paper, we propose a novel approach to in-situ data analysis. Emerging storage devices such as Solid-State Disk devices (SSDs) are being deployed on large-scale machines such as Tsubame2 [41] and Gordon [29], due to their higher I/O throughput, and are likely to be an integral part of the storage subsystem in next-generation machines. They also have significant compute power on the storage controllers [6], a large portion of which remains unused in typical storage environments. We propose *active flash*, a means to exploit the compute power in SSDs for in-situ data analysis. This approach also has the potential to enable energy-efficient data analysis, as the SSDs are equipped with low-power, ARM-based, multi-core controllers. Finally, performing computation where data is stored reduces transfer costs, yielding both performance and energy savings.

The active flash approach resembles prior work in active disks [21, 18], embedding computation in storage devices. The specifics differ greatly, however, due to differences not only in the underlying technology (flash vs. disk), but also in application needs. Further, active disk predates the evolution of the storage bandwidth bottlenecks and data movement costs in today's post-petascale HPC environments.

Contributions: The contributions of this work are as follows: (1) We present detailed energy and performance models for the active flash, offline, and analysis nodes strategies (the three modes shown in Figure 1). We evaluate them using realistic measurements both from Jaguar and data rates from several leadership applications. We model the conditions under which it is feasible to offload data analytics to SSD controllers, showing that active flash can execute many popular data analytics kernels in the background without degrading simulation performance, and provides better performance and energy consumption than alternative architectures. (2) We have built an active flash prototype on the Jasmine OpenSSD development platform [26], extending the OpenSSD flash translation layer (FTL) with data analysis functions, demonstrating the feasibility of adding active flash functionality with modest changes to current hardware. (3) Via modeling, simulation-based verification, and our prototype, we have shown that active flash is a cost-effective and energy-efficient execution model for in-situ scientific data analysis.

2 Background

SSD-based Staging Area for HPC Simulation Output: HPC simulations produce tens of TBs of output data, composed of *analysis data* and *checkpoint data*, both written at periodic intervals. Analysis data forms the input to post-processing, analytics, and visualization. Checkpoint data stores a snapshot of the execution's state, for future restart purposes. Both analysis and checkpoint data are saved at periodic intervals and create major I/O traffic on supercomputers. For instance, a GTC application (Table 1) run using 18,000 compute nodes (225,000 cores) on Jaguar produces 30TB of output in just one checkpoint.

Exascale projections [31] indicate that emerging application data rates will exert a tremendous amount of pressure on the PFS, exposing the storage bottleneck dramatically [1, 31]. Consequently, the HPC community has been investigating SSD-based solutions to help alleviate this I/O bandwidth bottleneck. Several studies have proposed the use of compute node-local SSDs as an intermediate storage buffer that can store the output data, and help recover from a failed state [11, 37, 12, 15, 34]. We refer to this collective SSD space as the *staging area*.

In this paper, we argue that performing on-the-fly data analysis on the *SSDs in the staging area* will be cost-effective, and energy-efficient. Given the massive scale of modern supercomputers[1], it may not be cost-effective to deploy SSDs on each and every compute node. A more realistic assumption is that they can be found on a subset of the nodes. In the next section, we will explore the following questions: how many SSDs should be deployed in the staging area to meet the data requirements of applications, and are they sufficient to perform active processing without hindering the main simulation?

Enabling Trends for Active Flash: While SSDs are being deployed on HEC machines to cater to growing I/O demands, we would like to explore the trends that make it amenable for active processing.

High I/O throughput and internal bandwidth: SSDs offer high I/O throughput and internal bandwidth due to

[1]E.g., Jaguar has 18,000 compute nodes; Sequoia, No. 1 on Top500, has 98,304 nodes; an exaflop machine could have a million [31].

interleaving techniques over multiple channels and flash chips; this bandwidth is likely to increase using more channels (typically 8 on consumer devices to 16 or more on high-end ones) or flash chips with higher-speed interfaces. Active computation may benefit from this high internal bandwidth.

Availability of idle times in workloads: HPC applications are bursty, with distinct compute and I/O phases. Further, the allowable I/O time for HPC applications is likely to be low (e.g., within 5% [1]). Consequently, embedded CPUs in an SSD are likely to be idle for long durations between application output phases.

Multi-core embedded processors: Recently marketed SSDs are equipped with fairly powerful mobile cores, and in some cases even multi-core controllers (e.g. a 4-core 780 MHz controller on the OCZ RevoDrive X2 [30]). As such multi-core SSD controllers become more prevalent, idle bandwidth on SSD CPUs will increase, holding promise for performing non-FLOP-intensive data analysis tasks.

Analysis Nodes Approach: While active flash is the approach we advocate, we would like to compare it against other variants of in-situ analysis. One can argue that analysis could be performed on dedicated low-power ARM cores, alongside the simulation. However, this is exactly what active flash is (less the storage capability), as the low-power cores are already available as part of the deployed SSDs. Another alternative, potentially most promising, is the *analysis nodes* approach. In Section 1, we described why performing analysis on a subset of the compute nodes that runs the HPC simulation (*simulation nodes*) is not desirable. Therefore, we model a more generalized analysis node approach in this paper, where we allocate a set of dedicated high performance compute nodes solely for data analysis purposes. We compare this approach against active flash, and offline techniques.

3 Modeling Data Analysis

In this section, we study the performance, and energy tradeoffs of *active flash*, *offline*, and *analysis node* techniques. In the *active flash* approach, we propose to conduct data analysis on the storage device controller itself (SSD controller.) With the traditional *offline* approach, data analysis or post-processing is performed on a set of compute nodes after the simulation finishes on a supercomputer. These *postprocessing compute nodes* can be allocated from the same supercomputer or from an offline cluster that shares a common PFS with the supercomputer. Finally, with the *analysis node* approach, data analytics is performed on a set of compute nodes, on the same simulation machine, while the output data is in-transit from simulation nodes to the PFS.

3.1 Active Flash

In active flash, we perform analysis on the output data, utilizing the idle cycles of the SSD controller to operate on temporary, SSD-resident data. We assess the feasibility of this new execution paradigm by developing an analytical model to examine the aggregate processing power and energy consumption of distributed SSD devices.

3.1.1 SSD Staging Ratio

We assume that SSDs are provisioned in HEC machines based on typical I/O demands and not based on data analysis requirements. These include capacity, performance, write durability, and peak output data volume of the HPC application simulation.

With this assumption, the total SSD storage requirement can be expressed in terms of the following parameters: (1) the per-compute-node data production rate for both analysis data and checkpoint data, denoted as λ_a and λ_c respectively, (2) the length of an output iteration (the time between two periodic output operations), t_{iter}, (3) the total number of compute nodes in the system, N, and (4) the number of checkpoints one would like to keep in the staging area, num_{chkpts}.

Capacity: The aggregate SSD space needs to be large enough to accommodate the temporary analysis and checkpoint data. Taking into account a certain over-provisioning factor, f_{op}, for the SSD space, the total SSD storage requirement at any given time can be estimated as $f_{op} \cdot (\lambda_a + num_{chkpts} \cdot \lambda_c) \cdot N \cdot t_{iter}$. The number of SSDs is the total SSD storage requirement divided by the capacity of one SSD, C_{SSD}. Consequently, the *maximum staging ratio* (the maximum number of compute nodes sharing one SSD) can be calculated as:

$$R_{capacity} = \frac{C_{SSD}}{f_{op} \cdot (\lambda_a + num_{chkpts} \cdot \lambda_c) \cdot t_{iter}} \quad (1)$$

Performance: There are two kinds of performance constraints. First, the aggregate bandwidth to all SSDs must be at least that of the PFS itself, otherwise the application will invariably suffer a slowdown. If the interface bandwidth between the simulation node and SSD is $BW_{sim-SSD}$, the maximum staging ratio is as follows:

$$R_{bandwidth} = \frac{N}{BW_{PFS}} \cdot BW_{sim-SSD} \quad (2)$$

Second, the primary objective of SSD deployment in HPC is to expedite application recovery by enabling faster reads of the checkpoint data. Therefore, the compute node should be able to read the checkpoint snapshot data from the SSD within a certain duration, compared to the output frequency. We refer to this as the *"restart I/O threshold fraction"*, ($f_{restart}$), which expresses the desired I/O time for recovery, as a fraction of the checkpoint interval; typical values range from 0.03 to 0.05 [1].

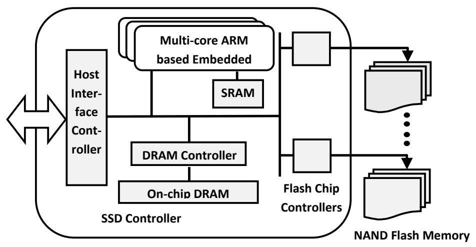


Figure 2: Detailed view of the SSD controller

The resulting constraint on the staging ratio is thus:

$$R_{restart} = \frac{f_{restart} \cdot BW_{sim-SSD}}{\lambda_c} \quad (3)$$

Write-endurance: An additional constraint is imposed by the finite write endurance of SSDs. Large-scale HPC systems are typically upgraded at regular intervals, and the minimum lifetime of any system component is dictated by the interval between two system upgrades (U_{time}). If we measure the write endurance $W_{endurance}$ of an SSD by the maximum number of bytes which may be safely written over its lifetime, then the write endurance limit on staging ratio is:

$$R_{endurance} = \frac{W_{endurance}}{(\lambda_a + \lambda_c) \cdot U_{time}} \quad (4)$$

Considering the capacity, performance, and write-endurance constraints, the final staging ratio is determined as follows:

$$R_{SSD} = \min(R_{capacity}, R_{bandwidth}, R_{restart}, R_{endurance}) \quad (5)$$

Observation 1 *Deciding staging ratios for SSD deployment in HEC settings is non-trivial, even without active flash computation. The ratio is limited (more SSDs are required) when output data production rate, PFS bandwidth, and system upgrade time are high, while higher write-endurance, SSD bandwidth, and allowable I/O overhead allow higher staging ratios (fewer SSDs).*

3.1.2 Performance Model

Given an SSD deployment with a staging ratio determined by the above constraints, we derive a performance model to study which data analytics kernels can be offloaded to SSD without degrading performance of the main simulation.

Once the analysis data arrives at the SSD (attached to a staging simulation node), active computation involves the following data transfers: (1) from the flash memory to the flash chip controller at the SSD internal bandwidth BW_{fm2c}, and (2) from the flash chip controller to the on-device DRAM at bandwidth BW_{c2m} (see Figure 2 for a detailed view of the SSD controller.) We use T_{SSD_k} to denote the data processing throughput of a single active flash device, running the data analysis kernel k. Certain data analytics kernels (e.g., statistical summaries) may even reduce the analysis data by some factor α. The SSD is responsible for writing the analysis data (possibly reduced) and checkpoint data to the PFS, consuming an appropriate share of the aggregate file system bandwidth BW_{PFS}. Processing plus output time for analysis results per checkpoint interval is thus:

$$t_a = R_{SSD} \cdot \lambda_a \cdot \left(\frac{1}{BW_{fm2c}} + \frac{1}{BW_{c2m}} + \frac{1}{T_{SSD\_k}} + \frac{\alpha}{\frac{R_{SSD}}{N} \cdot BW_{PFS}}\right) \cdot t_{iter} \quad (6)$$

and the checkpoint data output time is as follows:

$$t_c = \frac{R_{SSD} \cdot \lambda_c}{\frac{R_{SSD}}{N} \cdot BW_{PFS}} \cdot t_{iter} \quad (7)$$

While the draining of the staged checkpoint data to the PFS can overlap with the processing of analysis data, we conservatively assume that these two tasks are carried out sequentially on the same SSD controller core. We assume without loss of generality that the result of data analysis is smaller than the output data itself. (i.e. α cannot be larger than one.)

In Equation 7 we assume output data is written to the PFS at peak bandwidth, BW_{PFS}; however such speeds may not be realized under contention. To evaluate our model under realistic scenarios we use a fraction of peak bandwidth for this value, as determined by real-system measurements described in Section 5 below.

To account for the overhead of transferring data over the interconnect from SSDs to I/O nodes, we next model interconnect transfer time. We assume that the mean distance from a staging node to the nearest I/O node is hop distance d, and that the mean distance from a simulation node to the nearest I/O node is identical, as staging nodes are a subset of simulation nodes.

Assuming wormhole routing [16]) with packets segmented into m-byte *flits*, and interconnect bandwidth (with no contention) of $BW_{idlelink}$, we can express the total data transfer time within a single output interval t_{iter} as:

$$t_i = \frac{m}{BW_{idlelink}} \cdot d + \frac{R_{SSD} \cdot (\alpha \cdot \lambda_a + \lambda_c)}{BW_{idlelink}} \cdot t_{iter} \quad (8)$$

$BW_{idlelink}$ is an optimistic bandwidth estimate for the interconnect latency, which may potentially favor our active computation approach; we thus model contention in order to provide a conservative transfer time estimate.

The interconnect contention ratio is the mean number of nodes sharing one link. Assuming in the worst case that each node sends a message to its farthest neighbor in the interconnection network, we have a total of $N \cdot d_{max}$ messages, where N is the total number of nodes and d_{max} is the network diameter. If the total number of links is L, then per-link contention is $\frac{N \cdot d_{max}}{L}$. Thus, the effective interconnect bandwidth can be expressed as $\frac{BW_{idlelink} \cdot L}{N \cdot d_{max}}$, and Equation 8 can be re-written as follows:

$$t_i = \frac{1}{BW_{busylink}} \cdot (m \cdot d + R_{SSD} \cdot t_{iter} \cdot (\alpha \cdot \lambda_a + \lambda_c)) \quad (9)$$

where

$$BW_{busylink} = \frac{BW_{idlelink} \cdot L}{N \cdot d_{max}}$$

The diameter, d_{max}, of an interconnect is easily determined; similarly, the total number of links is determined by its topology. For example, in a 3D-torus as used in the Jaguar supercomputer, each node has 6 outgoing links and each link is shared by two nodes, giving $3N$ total links, and an effective link bandwidth of $\frac{3 \cdot BW_{idlelink}}{d_{max}}$.

For the active flash approach to be feasible, the device needs to process and output the data from the S simulation nodes before the next I/O iteration, i.e., consuming data at a rate faster than its generation:

$$t_a + t_c + t_i < t_{iter} \quad (10)$$

Solving the inequality 10, we arrive at the minimum processing throughput required for an analytics kernel k placed on the flash device:

$$T_{SSD\_k} > \quad (11)$$
$$\frac{\lambda_a \cdot R_{SSD}}{1 - \lambda_a \cdot R_{SSD} \cdot (\frac{1}{BW_{fm2c}} + \frac{1}{BW_{c2m}}) - \frac{N \cdot (\alpha \cdot \lambda_a + \lambda_c)}{BW_{PFS}} - \frac{t_i}{t_{iter}}}$$

In Section 5 below we use this equation to evaluate the performance feasibility of active flash for a set of representative large-scale applications and data analytics kernels.

Observation 2 *Multiple factors determine which kernels may be offloaded to the SSD without degrading simulation performance. Increases in either PFS or internal SSD bandwidth allow more computationally-intensive kernels to be offloaded, while higher staging ratios and data production rates decrease the amount of computation which may be offloaded. In Inequality 12 we see the effect of each factor x is proportional to $\frac{x}{1-x}$, rising exponentially as $x \to 1$.*

3.1.3 Energy Model

Total energy consumed by active flash approach ($E_{active-flash}$) can be expressed as summation of data movement energy expense in the interconnect ($E_{interconnect}$), and energy consumed by all SSDs in the staging area (E_{SSD}), for the entire duration of the application run:

$$E_{active-flash} = E_{interconnect} + E_{SSD} \quad (12)$$

First, we model the interconnect energy as the data movement energy overhead for a given number hops. If $e(h)$ represents the energy to transfer a unit amount of data across h hops, then we can express the total data movement energy cost as:

$$E_{interconnect} = t_{sim} \cdot N \cdot (e(d_s) \cdot (\lambda_a + \lambda_c) + e(d) \cdot (\alpha \cdot \lambda_a + \lambda_c)) \quad (13)$$

where d_s is the number of hops from the simulation node to its nearest staging node (as opposed to d, the distance from a staging node to the nearest I/O node.).

The energy consumed by the SSDs is:

$$E_{SSD} = E_{busySSD} + E_{idleSSD} - E_{iosaving} \quad (14)$$

where $E_{busySSD}$, energy consumed during SSD busy time, is the sum of (1) energy to transfer data from simulation nodes to SSDs ($E_{sim-SSD}$), (2) energy used processing analysis data ($E_{activeSSD}$), and (3) energy to transfer data to the PFS ($E_{SSD-PFS}$).

$E_{idleSSD}$, in turn, is the energy consumed by the SSD in the idle state. Finally, higher I/O bandwidth due to SSDs may reduce time spent waiting for I/O at the simulation nodes; the resulting reduction in energy consumption at simulation nodes is denoted $E_{iosaving}$.

Next, we estimate each component of $E_{busySSD}$—SSD busy time—individually. Let P^{SSD}_{busy} and P^{SSD}_{idle} be the SSD busy and idle power level, respectively, and t_{sim} the total simulation computation time for a single application run.

$E_{sim-SSD}$ is the total time ($t_{transfer}$) to transfer data from simulation nodes to SSDs in the staging area, times the power P^{SSD}_{busy}. Total data transferred is

$$data_{total} = N \cdot (\lambda_a + \lambda_c) \cdot t_{sim}$$

Then, the $t_{transfer}$ is equal to

$$(\frac{1}{BW_{sim-SSD}} + \frac{1}{BW_{busylink}}) \cdot data_{total}$$

The total energy consumed by SSDs during the data transfer from simulation nodes to SSDs can then be expressed as:

$$E_{sim-SSD} = P^{SSD}_{busy} \cdot t_{transfer} \quad (15)$$

Similarly, the time taken to post-process, $t_{activecompute}$, is equal to

$$N \cdot \lambda_a \cdot t_{sim}(\frac{1}{BW_{fm2c}} + \frac{1}{BW_{c2m}} + \frac{1}{T_{SSD\_k}})$$

and the total energy consumed by SSDs during data analysis is:

$$E_{activeSSD} = P^{SSD}_{busy} \cdot t_{activecompute} \quad (16)$$

After data analysis tasks, both checkpoint and analysis data are written to the PFS by SSDs. Accounting for both PFS transfer and interconnect latency, energy cost is:

$$E_{SSD-PFS} = \quad (17)$$
$$P^{SSD}_{busy} \cdot data_{total} \cdot (\frac{1}{BW_{PFS}} + \frac{1}{BW_{busylink}})$$

Finally, the idle energy consumption can be calculated from estimated total idle time, utilizing the busy time estimate above:

$$E_{idleSSD} = \quad (18)$$
$$P^{SSD}_{idle} \cdot (\frac{N}{R_{SSD}} \cdot t_{sim} - \frac{E_{activeSSD} + E_{SSD-PFS}}{P^{SSD}_{busy}})$$

The busy time involved in $E_{sim-SSD}$ is not subtracted in the equation above, unlike the other two components.

This is because both CPUs and SSDs are involved in the data transfer, which is not a part of t_{sim}, and does not perform data generation.

We estimate the per-node idle time reduction as

$$T_{iosaving} = \frac{N \cdot (\lambda_a + \lambda_c)}{BW_{PFS}} - (R_{SSD} \cdot (\lambda_a + \lambda_c)) \cdot (\frac{1}{BW_{sim-SSD}} + \frac{1}{BW_{busylink}})$$

Therefore, the energy savings at all simulation nodes from idle time reduction, over the entire application run, is:

$$E_{iosaving} = N \cdot T_{iosaving} \cdot t_{sim} \cdot P^{server}_{idle} \quad (19)$$

Observation 3 *Energy spent during SSD busy time is dependent on the total amount of work performed and/or data transferred, and thus independent of staging ratio. In contrast, idle SSD energy costs and simulation node energy use vary with staging ratio.*

3.2 Modeling Offline Processing

With the offline approach, compute nodes on the analysis cluster need to read only the analysis data and write the potentially reduced (by a factor of α) analysis data according to their share of the whole data, i.e., $\frac{N \cdot \lambda_a}{M}$, assuming M nodes are used to perform offline analysis. Similarly, each of the M nodes need to process the data at the processing rate, T_{server_k}, for a given kernel k. Again, we assume that each node gets its appropriate share of the PFS bandwidth $\frac{1}{N} \cdot BW_{PFS}$. The interconnect latency would be equal to $\frac{N \cdot \lambda_a}{M \cdot BW_{busylink}}$. The following equation captures the runtime of the offline analysis:

$$T_{offline} = \frac{N \cdot \lambda_a \cdot t_{sim}}{M \cdot T_{server\_k}} + (1+\alpha) \cdot (\frac{N \cdot \lambda_a \cdot t_{sim}}{M}) \cdot (\frac{N}{BW_{PFS}} + \frac{1}{BW_{busylink}}) \quad (20)$$

In some cases, analysis data is transferred over the wide area network for processing, resulting in more performance and energy penalty than what our optimistic model estimates.

Next, we model the energy cost of offline processing. We charge only idle power for the compute servers while they read and write the analysis data, and account for busy power during the data analysis. We can obtain this by multiplying Equation 20 by P^{server}_{idle} for the reading and writing part of the process, and multiplying by P^{server}_{busy} for the analysis part of the process.

There are various topologies which could be used for the offline approach, making estimates of interconnect energy cost difficult. We therefore conservatively ignore the cost of moving data to an offline compute cluster when comparing to our active flash approach, giving a total energy cost of:

$$E_{offline} = P^{server}_{idle} \cdot (1+\alpha) \cdot (\frac{N^2 \cdot \lambda_a \cdot t_{sim}}{BW_{PFS}} + \frac{N \cdot \lambda_a \cdot t_{sim}}{BW_{busylink}}) + P^{server}_{busy} \cdot \frac{N \cdot \lambda_a \cdot t_{sim}}{T_{server\_k}} \quad (21)$$

Observation 4 *The energy cost of offline processing does not depend on the number of offline nodes, but only on the total amount of data to be read and processed.*

3.3 Modeling the Analysis Node Approach

Much like the case of SSD deployment, we begin by determining the staging ratio for analysis node deployment, based on constraints of capacity and bandwidth. The capacity constraint here refers to the memory capacity (C_{mem}) on the analysis node, and the bandwidth constraint corresponds to the memory bandwidth (BW_{mem}) instead of the host to SSD bandwidth.

We assume that all simulation output—both checkpoint data and analysis input—is transferred to analysis nodes, as is done by libraries such as ADIOS [22], as the resulting increase in checkpoint write bandwidth allows the simulation nodes to progress faster.

Equations 1, 2, and 5 are thus changed as follows:

$$R_{capacity} = \frac{C_{mem}}{f_{op} \cdot (\lambda_a + num_{chkpts} \cdot \lambda_c) \cdot t_{iter}} \quad (22)$$

$$R_{bandwidth} = \frac{N}{BW_{PFS}} \cdot BW_{mem} \quad (23)$$

$$R_{a\_node} = \min(R_{capacity}, R_{bandwidth}) \quad (24)$$

where subscript a_node refers to analysis node approach. Similarly, Equations 6, 7, and 9 can be modified to estimate the data analysis time, output time, and the interconnect latency:

$$t_a = R_{a\_node} \cdot \lambda_a (\frac{1}{T_{a\_node\_k}} + \frac{\alpha}{\frac{R_{a\_node}}{N} \cdot BW_{PFS}}) \cdot t_{iter} \quad (25)$$

where $T_{a\_node\_k}$ is the throughput required to run the data analysis kernel, k on the analysis node.

$$t_c = \frac{R_{a\_node} \cdot \lambda_c}{\frac{R_{a\_node}}{N} \cdot BW_{PFS}} \cdot t_{iter} \quad (26)$$

$$t_i = \frac{1}{BW_{busylink}} \cdot (m \cdot d + R_{a\_node} \cdot t_{iter}(\alpha\lambda_a + \lambda_c)) \quad (27)$$

Accordingly, the feasibility constraint for what data analysis kernels can be run on the analysis node without slowing down the simulation can be expressed as:

$$t_a + t_c + t_i < t_{iter} \quad (28)$$

Solving the inequality 28, we derive the minimum throughput required for the analytics kernels that can be placed on these analysis nodes:

$$T_{a\_node\_k} > \frac{\lambda_a \cdot R_{a\_node}}{1 - \frac{N \cdot (\alpha \cdot \lambda_a + \lambda_c)}{BW_{PFS}} - \frac{1}{BW_{busylink}}(\frac{m \cdot d}{t_{iter}} + R_{a\_node} \cdot (\alpha\lambda_a + \lambda_c))} \quad (29)$$

Next, we account for the energy overhead of the different components, starting with data movement costs, which are:

$$E_{interconnect} = t_{sim} \cdot N \cdot (e(d_s^{a\_node}) \cdot (\lambda_a + \lambda_c) + e(d) \cdot (\alpha \cdot \lambda_a + \lambda_c)) \quad (30)$$

where $d_s^{a\_node}$ is the average number of hops from the simulation nodes to the analysis nodes.

Next, we account for the energy spent when analysis nodes are busy. This consists of three components: (1) transferring data from the simulation nodes to the data analysis node (E_{sim-a_node}), (2) processing the analysis data (E_{a_node}), and (3) transferring the data to the PFS ($E_{a\_node-PFS}$).

Let $P_{busy}^{a\_node}$ and $P_{idle}^{a\_node}$ be the data analysis node's busy and idle power level, respectively. Then, we can modify Equations 15 to 19 to get the corresponding equations for data analysis energy overheads:

$$E_{sim-a\_node} = P_{busy}^{a\_node} \cdot (\frac{1}{BW_{mem}} + \frac{1}{BW_{busylink}}) \cdot data_{total} \quad (31)$$

$$E_{a\_node} = \frac{P_{busy}^{a\_node} \cdot N \cdot \lambda_a}{T_{a\_node\_k}} \cdot t_{sim} \quad (32)$$

$$E_{a\_node-PFS} = P_{busy}^{a\_node} \cdot data_{total} \cdot (\frac{1}{BW_{PFS}} + \frac{1}{BW_{busylink}}) \quad (33)$$

$$E_{idle-a\_node} = P_{idle}^{a\_node} \cdot (\frac{N}{R_{a\_node}} \cdot t_{sim} - \frac{E_{a\_node} + E_{a\_node-PFS}}{P_{busy}^{a\_node}}) \quad (34)$$

Finally, we estimate the per simulation node idle time reduction due to analysis nodes as

$$T_{iosaving-a\_node} = \frac{N \cdot (\lambda_a + \lambda_c)}{BW_{PFS}} - (R_{a\_node} \cdot (\lambda_a + \lambda_c)) \cdot (\frac{1}{BW_{mem}} + \frac{1}{BW_{busylink}})$$

and total energy savings for this approach is:

$$E_{iosaving-a\_node} = N \cdot T_{iosaving-a\_node} \cdot t_{sim} \cdot P_{idle}^{sim} \quad (35)$$

4 Experimental Methodology

We evaluate our models using the data production rates from "hero" applications on the Jaguar machine at ORNL (Table 1).

Our model is generic and applies to common supercomputer configurations seen today. Our evaluation is driven by parameters from the Cray XT5 Jaguar supercomputer [8], as shown in Table 2. In the current SSD landscape, there is no support for active computation on the device. To study the viability of active flash, we model after a contemporary SSD such as Samsung PM830, which has a multicore SSD controller based on the ARM processor [6]. Although such a controller has three ARM-based cores, we adopt a conservative approach, and propose to use only one core for active computation, while leaving the other cores free for typical SSD activities (e.g. error-checking, garbage collection etc.). In the future, more cores are likely to be placed on the same chip, making active computation more promising than what is projected in this study.

No. of compute nodes (N)	18,000
PFS bandwidth (BW_{PFS})	240 GB/s [32]
Output frequency (t_{iter})	1 hour
Simulation duration (t_{sim})	24 hours
Overprovisioning factor (f_{op})	1.50
Data reduction factor (α)	1 (no reduction)
Time between upgrades (U_{time})	2 years
Restart I/O threshold ($f_{restart}$)	0.05 [1]
Interconnect bandwidth ($BW_{busylink}$)	814 MB/s
Flit size (m)	32 bytes
Hop count (d, d_s, $d_s^{a\_node}$)	3 hops, 1.5 hops, 1.5 hops
$e(d)$, $e(d_s)$, $e(d_s^{a\_node})$	10 pJ/bit, 5 pJ/bit, 5 pJ/bit [10, 14]
SSD model	Samsung PM830
$BW_{host2ssd}$	400 MB/s [19]
BW_{fm2c}	320 MB/s [19]
BW_{c2m}	3.2GB/s [4]
SSD power (P_{busy}, P_{idle})	3 W, .09 W [6]
SSD write endurance ($W_{endurance}$)	300 TB written [6]
Offline model / analysis node model	2 Quad-core Intel Q6600 [3]
BW_{mem}	30 GB/s
Server / node power (P_{busy}, P_{idle})	210 W, 70 W
Price: SSD (64 GB), node (2×4 cores), DRAM (64 GB)	\$ 100 [7], \$ 270 [5], \$ 400 [2]

Table 2: Parameters for performance and energy models.

To emulate the computing speed of the ARM-based SSD controller, we use an ARM Cortex-A9 processor in the Pandaboard mobile software development platform [35]. We model the data transfer times as follows. We assume 8 flash memory channels, each with 40MB/s bandwidth, to transfer data from the NAND Flash to the chip controller (BW_{fm2c}) [19]. Our numbers are conservative, and modern devices usually have higher channel bandwidth or more channels. Similarly, while the DDR2 SDRAM cache used in these SSDs may have a bandwidth of up to 5.3GB/s [4], we conservatively set BW_{c2m} to 3.2GB/s.

For a comparison with the offline and analysis nodes models, we use two Intel Core 2 Quad 6600 processor [3] as the data processing node. We use an input file of 100MB to measure the analysis throughput.

The chosen data analytics kernels cover a wide variety of representative analytics operations on scientific data, including pattern matching, clustering, changing data layout, and compression [23, 24, 20]. We mea-

Data Analytics Kernels	ARM Cortex-A9	Intel Q6600 server
Statistics (mean)	416 MB/s	17.7 GB/s
Pattern Matching (grep)	123 MB/s	4.34 GB/s
Data formatting (transpose)	76 MB/s	1.95 GB/s
Dim. reduction (PCA)	10.2 MB/s	80 MB/s
Compression (gzip)	4.1 MB/s	216 MB/s
Dedup. (Rabin fingerprint)	1.81 MB/s	106 MB/s
Clustering (k-means)	0.91 MB/s	13.7 MB/s

Table 3: Processing throughput of common data analytics kernels on different devices. Our offline compute node consists of two Intel Q6600 Quad-core machines.

sured their processing throughput on both the ARM-based SSD controller and the Intel server (Table 4). Note that the throughput of the analysis kernels may be input-dependent as well (e.g. number of clusters, dimensions, search expressions etc.). Therefore, choosing a wide variety of kernels, whose throughput varies from less than one MB/s to a few GB/s helps better understand the limits, and the potential of active flash.

For energy calculations we use the thermal design power (TDP) of 210W (2 $\times$ 105W per quad-core machine) as the busy-state power for each offline node; we conservatively estimate idle power at one third of TDP and cooling cost as zero.

In addition we ignore power required for memory and network traffic for the offline approach, which is clearly higher compared to active flash due to additional reading of the analysis data.

We computed the hop counts, d for Jaguar as follows. Jaguar has 200 cabinets in 8 rows and 25 columns, with 3 cages per cabinet, 8 blades per cage, and four compute nodes per blade. Every node is connected via the 3D torus Cray Gemini network, with routing on X, Y, Z directions [17]: X is across cabinets within a row, Y is across cabinets within a column, and Z is across nodes within a cabinet. Average cable lengths are 1.8 m and 7.5 m for the X and Y directions respectively. Based on the coordinate information, we can infer node-to-node hop count in distance, and also determine the number of cabinets traversed. To account for the energy cost in our experiments, we are particularly interested in the hop distance from the compute nodes to the I/O nodes and consequently, the number of cabinets traversed. We estimate d_s and $d_s^{a\_node}$ to be half of d (Table 2). This, however, is a conservative estimate since the number of analysis nodes or SSDs (e.g., at a staging ratio of 10) is approximately 9 times the number of I/O nodes (192) in Jaguar.

We have validated the busy link bandwidth model, $BW_{busylink}$, against the actual link bandwidth measurement on Jaguar by performing all-to-all MPI communication between N node pairs; measured busy link bandwidth based on 1024 nodes is 814 MB/s.

Active Flash Model						
	CHIMERA	VULCAN	POP	S3D	GTC	GYRO
$R_{capacity}(32\,GB)$	**1**	2571	233	**18**	**6**	166
$R_{capacity}(64\,GB)$	1	4500	461	36	12	333
$R_{bandwidth}$	29	**29**	**29**	29	29	**29**
$R_{endurance}$	1	2268	245	20	10	204
$R_{restart}$	4	896218	4054	240	42	1758
Analysis Node Model						
	CHIMERA	VULCAN	POP	S3D	GTC	GYRO
$R_{capacity}(16\,GB)$	**1**	**1285**	**117**	**9**	**3**	**83**
$R_{capacity}(32\,GB)$	1	2571	233	18	6	166
$R_{capacity}(64\,GB)$	1	4500	461	36	12	333
$R_{bandwidth}$	2250	2250	2250	2250	2250	2250

Table 4: Staging ratio derived from capacity, bandwidth, and endurance and restart (flash only) constraints for applications on Jaguar. The most restrictive ratio is shown in bold.

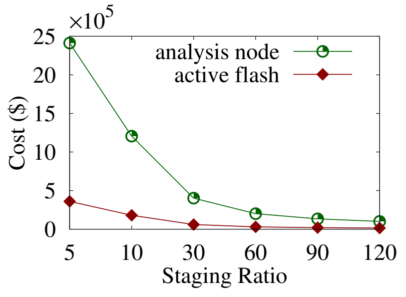
5 Evaluation

Our evaluation aims to answer three questions: (1) What is the staging ratio for SSD provisioning, based on different constraints for representative applications? (2) Will these staging ratios support *in-situ* data processing on SSDs? (3) If so, what energy savings may be achieved with active flash?

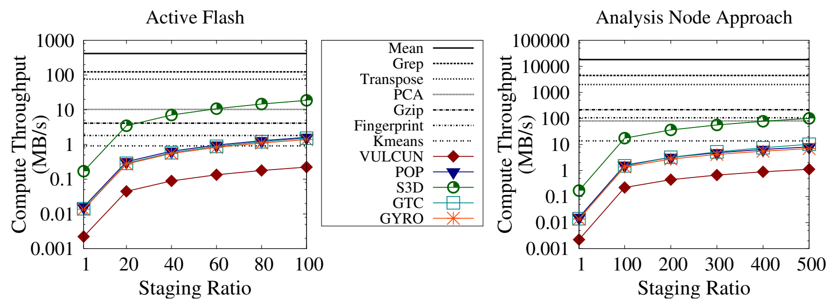
Staging ratio: Table 4 shows the staging ratio for the active flash, and the analysis node models, based on constraints from Equations 1-4 and 22-23; higher staging ratios correspond to more compute nodes per SSD or analysis node. Capacity constraints are identical for both models, while bandwidth constraints differ due to higher memory bandwidth (for analysis nodes) than SSD bandwidth (for active flash), and are application independent constants of 2250, and 29, respectively. For active flash the bandwidth constraint is most restrictive for applications with high data rates relative to overall data volume, capacity is not restrictive for 64GB SSDs, and write endurance is a limitation for applications like *S3D*, and *GTC* with large output volumes.

Inference: *Bandwidth appears to be the predominantly limiting factor for active flash, whereas capacity is the overriding concern for analysis nodes.*

Infrastructure cost: As stated above (Section 2), SSDs are currently deployed in HEC systems as burst buffers to alleviate I/O bandwidth bottlenecks when checkpointing. As active flash piggybacks computation on this *existing infrastructure*, we argue that its cost may be ignored, unlike that of analysis nodes; however to be conservative our analysis considers both costs in Figure 3(a) (cost details in Table 2). At low staging ratios the analysis node approach is rather expensive—e.g. \$1,206,000 to provision Jaguar with 64 GB DRAM analysis nodes at a staging ratio of 10, vs. \$180,000 for SSDs at the same ratio. However, analysis nodes might enjoy a higher staging ratio in certain cases due to its higher memory bandwidth.



(a) Infrastructure Cost



(b) Feasibility Analysis

Figure 3: (a) Cost of provisioning SSDs, and analysis nodes. The SSD capacity is 64GB; the analysis node is a dual quad-core machine with 64GB of DRAM. (b) Feasibility of running analysis kernels using different staging ratios. A kernel line *higher* than a dot point on the application's slope line is *suitable* for active flash (left figure), and the analysis node approach (right figure).

One example is the application POP, where the staging ratio is limited to 29 (by bandwidth for active flash, but is capacity-limited to 461 for analysis nodes (Table 4). Provisioning cost is estimated at roughly \$62,000 and \$25,605 for active flash and analysis nodes, respectively.

Inference: *In general, analysis nodes can sustain a higher staging ratio at higher capacities, which can help reduce the infrastructure costs. However, as active flash piggybacks on existing infrastructure, its cost is already paid for.*

Feasibility analysis: A higher staging ratio lowers the infrastructure cost, but not all analysis kernels are feasible above certain staging ratios. Figure 3(b) shows the computational throughput of the analysis kernels as flat horizontal lines and the application throughput requirement as sloped lines with dots. A data analysis kernel is able to run on the SSD without any penalty to the simulation performance if its computational throughput is higher than the required throughput of the application (T_{SSD_k} in Equation 12, and $T_{a\_node\_k}$ in Equation 29). As the staging ratio increases, the processing requirements for both active flash devices and analysis nodes increase.

For example, all of the data analysis kernels can be run on active flash (left graph in Figure 3(b)) for the fusion application *S3D*, when the staging ratio is 5, i.e. 3,600 SSDs for 18,000 compute nodes. As we increase the staging ratio to 50, the `gzip`, `fingerprinting`, and `kmeans` analysis kernels cannot be run on the SSD for this application's output, as the threshold throughput of *S3D* is *higher* than the compute throughput of these kernels. Figure 3(b) shows that a staging ratio as high as 50 can accommodate all data analysis kernels for most applications, with the exception of the S3D application.

For the analysis node model with 64GB of DRAM, a staging ratio of 10 (costing \$1.2M as calculated above) can accommodate all of the kernels for all of the applications, but incurs \$1.2M infrastructure cost. To reduce provisioning costs the staging ratio may be increased beyond this point, reducing the number of analysis kernels which may be used with certain applications. For example, if the staging ratio is increased to 30, the cost falls 67%, to \$402,000, while eliminating support for running data analysis kernels on the output of *GTC* (Table 4).

Inference: *Both active flash and analysis node techniques can support most of the example kernels and applications at low staging ratios, but at a higher infrastructure cost. High infrastructure cost can be reduced by increasing the staging ratio, but that scarifies the analysis kernels that can be run on active flash or analysis nodes.*

Energy consumption: Next, we discuss the energy consumption of the models, and its effect on the overall cost. Figure 4 shows the energy costs of active flash, offline, and analysis nodes. We assume a staging ratio of 10, allowing almost all application/analysis kernel pairings to be executed except for *CHIMERA* (Figure 3(b)). For this ratio we need 1800 staging or analysis nodes, each with 64GB of SSD or DRAM, to accomodate the next most restrictive application, *GTC*.

We define the configuration $baseline_{PFS}$ ($y = 0$) to be the case where the simulation is run without SSDs: all checkpoint and output data are written to the PFS, and no further data analysis is performed after simulation. Results are given as the difference between energy usage for a configuration vs. that for $baseline_{PFS}$; negative numbers indicate energy savings. We observe that deploying SSDs just for higher I/O throughput ($baseline_{SSD}$, the dotted horizontal line in Figure 4), saves significant energy by shortening application run time.

It is not surprising that active flash consumes extra energy compared to $baseline_{SSD}$; however, it still results in savings compared to $baseline_{PFS}$ for almost all application / kernel pairs (except *VULCUN*, which performs little I/O). In contrast, the offline model consumes more energy due to the I/O wait time on the offline compute nodes. For example, for *S3D* with *fingerprinting*, active flash conserves up to 968 kWh per simulation run vs. offline processing, due to the volume of analysis data.

Active flash also results in significant energy savings over use of analysis nodes. For the same *S3D* with *finger-*

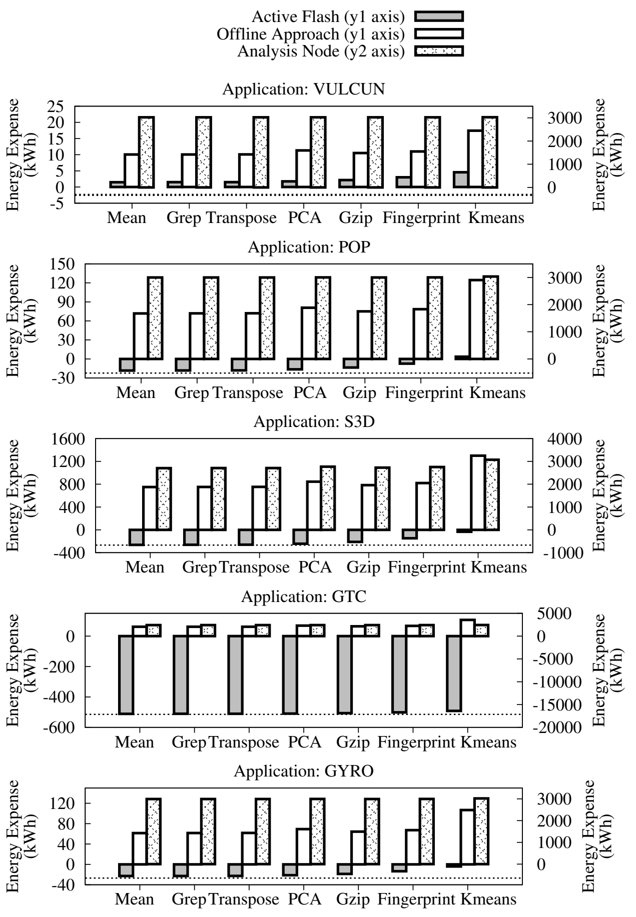


Figure 4: Comparing the energy expenses of all applications for Active Flash, offline, and analysis nodes techniques. Expenses are w.r.t. running only the simulation job using PFS, without SSDs ($baseline_{PFS}$.) Dotted line denotes energy savings due to running only simulation using SSDs without active computation ($baseline_{SSD}$.) Active flash and offline are plotted on the primary y-axis (y1), and the analysis nodes model is plotted on the secondary y-axis (y2).

printing, we observe up to 2900 kWh of energy savings per run. Note that these energy savings are per simulation run, and will compound over the lifetime of the machine, potentially helping to defray the cost of the initial deployment. Although analysis nodes offer increased performance, they are seen to consume more energy at low staging ratios than offline processing (up to 1932 kWh), as they spend significant time idle; this idle time (and energy expenditure) is reduced at higher staging ratios.

Inference: *Overall, active flash makes data analysis virtually "free" in most cases when it is piggybacked on the SSDs, promising significant energy and performance savings vs. the offline and analysis node techniques.*

Overall cost: To better understand the overall costs we examine active flash and analysis nodes (both of which offer better performance than offline) at various staging ratios (Table 2) across our portfolio of HPC applications, comparing the up-front (capital) costs as well as the energy expenditure over time. We study the trade-off between these costs, displaying infrastructure and 2-year energy costs in Table 5 for running our application portfolio continuously at staging ratios of 10, 30, and 300.

At a staging ratio of 10, both active flash, and analysis nodes can support all of the application/kernel combinations. In Table 5 we see that active flash is the most energy-efficient, saving \$22,834 (at \$0.30 per kWh) to support the five applications for a period of two years and thereby *defraying almost 13% of the deployment cost of \$180,000.* In contrast, the analysis node approach results in energy costs of \$639,290, or an additional 53% on top of the initial provisioning cost of \$1,206,000.

At staging ratios of 30 and 300 active flash cannot be used with any applications, while analysis nodes are more cost effective, while failing to run *GTC* (at 30 and 300) and *S3D* (at 300). At a ratio of 30 the infrastructure and energy expenses decrease to roughly \$402K and \$185K, respectively; at 300 the infrastructure and energy expenses decrease even further to roughly \$40K and \$36K, respectively.

Inference: *Although energy costs for analysis nodes decrease at higher staging ratios, at the expense of failing to support certain applications, unlike active flash the cost reduction fails to defray any of the initial infrastructure expense.*

Staging Ratio	Infrastructure Cost (\$)	Energy Bill (\$)	Total Cost (\$)	Feasible Applications
Active Flash Model				
10	180,000	−22,834	157,165	all
30 & 300	–	–	–	none
Analysis Node Model				
10	1,206,000	639,289	1,845,289	all
30	402,000	185,220	506,820	all, w/o GTC
300	40,200	36,044	60,164	all, w/o GTC, S3D

Table 5: Capital and energy costs to support a portfolio of five leadership applications for 2 years. Assumptions: continuous usage, equal number (146) of 24-hour simulation runs per application, electricity \$0.30 per kWh.

Summary of results: In summary, for most of the applications, and all of the analysis kernels, active flash is effective at low staging ratios, while being extremely efficient in terms of both deployment cost and energy; however for higher staging ratios it is unable to support certain applications. Active flash is limited by the bandwidth and write endurance constraints. In contrast, analysis node performs well at higher staging ratios, especially for compute-intensive analytics kernels; however it is limited by capacity constraints on DRAM and suffers from higher cost and energy consumption. A hybrid approach, deploying analysis nodes with active flash might be feasible, but is beyond the scope of this paper.

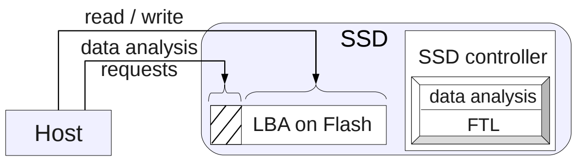


Figure 5: Active Flash prototype.

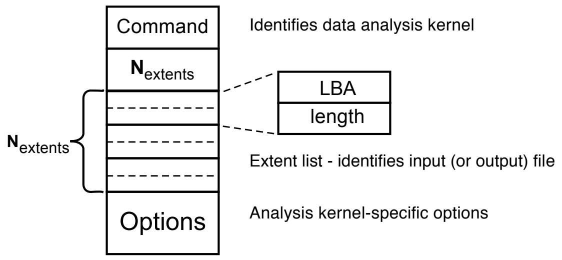

Command
Identifies data analysis kernel
$N_{extents}$
LBA
length
$N_{extents}$
Extent list - identifies input (or output) file
Options
Analysis kernel-specific options

Figure 6: Data analysis command format.

6 Active Flash Prototype

To demonstrated feasibility of the active flash approach, we have implemented a prototype on the Jasmine OpenSSD development platform [26], extending the OpenSSD flash translation layer (FTL) with data analysis functions. This platform uses the Indilinx Barefoot controller, an ARM-based SATA controller with parameters shown in Table 6.

Parameter	Value
Controller	ARM Indilinx Barefoot™ at 87.5 MHz
Host Interface	SATA 2 at 3 Gbps
SDRAM	64 MB
Flash Memory	64 GB

Table 6: SSD parameters on the OpenSSD platform.

An overview of our implementation is shown in Figure 5. Active flash functionality is requested via out-of-band commands, distinguished by LBA range. An active flash operation comprises: **(a) write:** analysis input data is written by the host to the SSD, e.g. as part of a checkpoint operation, **(b) request:** data analysis request sent in the form of write to a reserved LBA, **(c) process:** the analysis kernel running on the SSD controller, reading input data from internal flash (where it was written in step (a)), **(d) result:** output data is written to internal SSD storage, and the host notified via polling or completion of a pending command.

6.1 Data analysis commands

The command format is simple and general, as shown in Figure 6, specifying an operation, a list of LBA extents, and operation-specific options.

Data for analysis is transferred to the controller by (a) writing to a file in the host file system, (b) flushing data from RAM to storage, (c) translating offsets within the file into physical LBAs, and (d) passing the sequence of LBA extents in the analysis command. Analysis results are returned by creating and pre-allocating a file for analysis output, then again translating file offsets to LBAs and passing an LBA extent list to the controller.

Simple analysis requests such as the statistical kernels described below (mean, max, standard deviation, linear regression) are fully specified by the command type; input is retrieved from the locations identified in the extent list. A variable-length options field is available for more sophisticated kernels like K-means clustering, which may require additional parameters.

The analysis input typically represents a multi-dimensional numeric array; in the current implementation this data layout is either known implicitly by the analysis kernel or is specified in the Options field. In practice this information is expected to be conveyed via the use of self-describing data formats such as NetCDF [38] or HDF5 [25]: metadata in these scientific data storage formats include information such as array shape and precision needed to interpret the raw data.

6.2 Scheduling I/O and data analysis tasks

As part of servicing read and write requests, SSD controllers must perform a number of overhead functions, such as garbage collection, wear leveling, bad block management, and error correction-related tasks. Some of these are performed at request time, while others may be deferred to background processing. In either case, such tasks will compete for CPU time with analysis tasks. To minimize impact on SSD performance on our single-core prototype, we implement a preemption-based scheme: data analysis is interrupted when I/O requests arrive, resuming after they finish.

The OpenSSD-based prototype implements a simple event loop, with CPU-bound tasks processed to completion. This preemption is performed by polling for requests at periodic intervals, implemented by processing B bytes at a time between checks. I/O requests may thus have to wait until the current interval is finished, incurring a mean delay of half that interval. Suppose T_{SSD_k} is the throughput of the data analysis kernel k, this delay is:

$$Delay\_k = \frac{1}{2} \cdot \frac{B}{T_{SSD\_k}}$$

In our prototype B is currently set to 32 KB, the hardware flash controller read size; for the analysis kernels shown in Table 7 below, this results in a mean delay of 3 to 8 ms. When I/O service requests are received in bursts, this delay should only be incurred once, achieving an acceptably low level of interference; on multi-core SSDs it may be avoided entirely.

	Throughput (MB/s)		Time (s)				Energy (Joules)	
Data analysis kernel	**Active Flash**	**Host**	**Active Flash I/O**	**Active Flash compute**	**Host I/O**	**Host compute**	**Active Flash**	**Host**
Max	4.5	33.3	2.4	19.8	2.6	0.4	31	288
Mean	4.0	33.1	2.4	22.1	2.6	0.4	34	290
Standard Deviation	3.3	32.0	2.4	27.7	2.6	0.5	42	300
Linear Regression	1.9	26.8	2.4	49.6	2.6	1.1	73	357

Table 7: Performance of data analysis kernels on active flash controller and host: throughput, I/O and computation time, and energy consumption. Input size = 100 MB; host CPU = AMD 2200 MHz, active flash CPU = ARM 87.5 MHz

6.3 Results

The host-controller communication mechanism described above, plus four data analysis kernels, have been implemented in our prototype as part of the SSD firmware. The kernels represent statistical computation common for scientific data processing: max, mean, standard deviation and linear regression. Input for all experiments consisted of 100 MB of 32-bit integers, stored on the SSD in binary. For comparison, each kernel computation was performed on the host CPU, an AMD Phenom 9550 quad-core at 2200 MHz with 2 GB of DRAM.

Throughput: Analysis throughput is shown in Table 7. We see that the statistical data analysis kernels achieve about 2–4.5 MB/s on the SSD controller (second column), and run about 7–14 times faster on the host CPU (third column). The OpenSSD CPU runs at only 87.5 MHz; active flash performance is expected to scale with faster cores in today's higher-end SSDs (e.g. the OCZ RevoDrive X2, with four 780 MHz Tensilica cores).

Division of I/O and computation time: Analysis run time consists of I/O and computation:

$$t = t_{io} + t_{comp} \tag{36}$$

For active flash, $t_{io} = t_{fm2c} + t_{c2m}$ represents the time to read the data from flash to the controller's DRAM. For host–resident data analysis, $t_{io} = t_{fm2host}$ is the time to read the data from flash to the DRAM on host.

In Table 7 I/O and computation time is presented in each case for 100 MB of input data; as expected, with active flash, t_{comp} is dominant, while the host–resident analysis time is dominated by t_{io}. Since the controller is optimized to deliver full internal flash bandwidth to the host, the two cases generated very similar I/O time with 32 KB reads. However, if less well-aligned read sizes are used by the host application, the SSD-to-host I/O time will increase (e.g., to 9 seconds with 4 K reads).

Power and energy consumption: The active flash prototype and host were measured in idle (1.35 W, 79 W), sustained write (1.5 W, 80 W) and data analysis (1.4 W, 96 W) states. Note that the relatively flat power consumption of OpenSSD is typical of low-end CPUs; higher-end embedded cores may incorporate power-saving idle modes similar to those of the host CPU. Energy consumption for processing 100 MB of data is also shown in Table 7; the active flash prototype is seen to use 5 to 9 times less energy than host-resident processing.

7 Related Work

Active storage refers to performance of computation within storage elements, for purposes such as increased performance or decreased energy cost. Active Disk [39] and IDISK [27] were early proposals for shifting computation to the disk-resident CPU, with target applications including image processing and data filtering. However increasing real-time demands on disk controllers and modest growth of disk bandwidth have limited the benefits possible with this approach. Active storage concepts have been explored in the context of parallel file systems [36, 40], with computation being performed on file system I/O nodes. Boboila et al. have proposed *Active Flash* [13], which takes advantage of the higher internal bandwidth, lower power, and greater flexibility of the SSD platform vs. magnetic disk; Boboila's work, however, models a single active flash device in isolation rather than an entire HEC workflow.

Kim et al. [19] studied the feasibility of offloading database scanning operations on the SSD controller. In contrast, we evaluate and model the active computation paradigm on SSDs in an extreme-scale computing environment for a class of high-end simulation jobs and data analysis kernels. Also, in contrast to previous work, we are unique in considering a case where data output rates impose real-time constraints on active computation to avoid overall performance degradation.

8 Conclusion

We have proposed active flash, an approach to piggy-back in-situ scientific data analysis on SSDs, in HEC machines. We argue that extant approaches to scientific data analysis (e.g., offline, and analysis nodes) are stymied by several inefficiencies in data movement, and energy consumption that results in sub-optimal performance. We have put forth detailed performance and energy models for active flash, offline, and analysis nodes, and studied their provisioning cost, performance, and energy consumption. Our modeling and simulation efforts indicate that active flash is better than either approaches for all of the aforementioned metrics. In fact, our results suggest that active flash can even help defray part of the capital expenditure through energy savings. Further, we have demonstrated the feasibility of active flash through the construction of a prototype, based on the OpenSSD development platform, extending the OpenSSD FTL with data analysis functions. Finally, our experience suggests that active flash is a viable approach for future scientific data analysis.

References

[1] Computational science requirements for leadership computing, 2007, http://tinyurl.com/nccs2007.

[2] DDR3 Memory price 64GB(8x8). http://tinyurl.com/dramprice.

[3] Intel core2 quad processors. http://tinyurl.com/q6600core.

[4] Micron dram datasheet. http://tinyurl.com/drambw.

[5] Processor price (Intel Core 2 Quad Q6600). http://tinyurl.com/c497wqb.

[6] Samsung pm830 datasheet. http://tinyurl.com/co9zyq7.

[7] SSD price (Samsung 830 Series 64GB). http://tinyurl.com/9cldame.

[8] National Center for Computational Sciences. http://www.nccs.gov/, 2008.

[9] Spider. http://tinyurl.com/spiderfsms, 2008.

[10] Scientific Grand Challenges. Architectures and Technology for Extreme Scale Computing. Technical report, Dec 2009.

[11] Samer Al-Kiswany, Matei Ripeanu, Sudharshan S. Vazhkudai, and Abdullah Gharaibeh. stdchk: A Checkpoint Storage System for Desktop Grid Computing. In *ICDCS'08*.

[12] Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, and Satoshi Matsuoka. Fti: high performance fault tolerance interface for hybrid systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 32:1–32:32, New York, NY, USA, 2011. ACM.

[13] Simona Boboila, Youngjae Kim, Sudharshan Vazhkudai, Peter Desnoyers, and Galen Shipman. Active flash: Performance-energy tradeoffs for out-of-core processing on non-volatile memory devices. In *Proceedings of the 2012 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '12, April 2012.

[14] Shekhar Borkar. Technology and Design Challenges to Realize Exascale. Intel Corp. http://www.orau.gov/archI2011/presentations/borkars.pdf, Aug 2011.

[15] G. Bronevetsky and A. Moody. Scalable i/o systems via node-local storage: Approaching 1 tb/sec file i/o. LLNL Technical Report LLNL-TR-415791, Lawrence Livermore National Laboratory, 2009.

[16] William James Dally and Brian Towles. Principles and Practices of Interconnection Networks. *Morgan Kaufmann Publishers, Inc. ISBN 978-0-12-200751-4*, 2004.

[17] David Dillow, Galen M Shipman, Sarp Oral, Zhe Zang, and Youngjae. Kim. Enhancing i/o throughput via efficient routing and placement for large-scale parallel file systems. In *Proceedings of the 30th IEEE Int'l Performance Computing and Communications Conference*, IPCCC '11, pages 1–9, 2011.

[18] Acharya et al. Active disks: programming model, algorithms and evaluation. *In ASPLOS*, 1998.

[19] Kim et al. Fast, Energy Efficient Scan inside Flash Memory Solid-State Drives. *In the International Workshop on Accelerating Data Management Systems (ADMS) with VLDB*, 2011.

[20] Ranger et al. Evaluating mapreduce for multi-core and multiprocessor systems. *In HPCA*, 2007.

[21] Riedel et al. Active storage for large scale data mining and multimedia applications. *In VLDB*, 1998.

[22] Zheng et al. Predata - preparatory data analytics on peta-scale machines. In *IPDPS*, 2010.

[23] Gnu grep, `http://www.gnu.org/software/grep/`.

[24] Gnu zip, `http://www.gzip.org/`.

[25] HDF Group. Hierarchical data format, version 5. `http://hdf.ncsa.uiuc.edu/HDF5`.

[26] Jasmine OpenSSD Platform. `http://www.openssd-project.org`.

[27] Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. A case for intelligent disks (idisks). *SIGMOD Rec.*, 27(3):42–52, 1998.

[28] ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, Peter Kogge, Editor and Study Lead, 2008.

[29] Michael L. Norman and Allan Snavely. Accelerating Data-intensive Science with Gordon and Dash. In *TG'10*.

[30] OCZ RevoDrive PCI-Express SSD Specifications. `http://www.ocztechnology.com/ocz-revodrive-pci-express-ssd.html`.

[31] U.S. Department of Energy. DOE exascale initiative technical roadmap, December 2009. `http://extremecomputing.labworks.org/hardware/collaboration/EI-RoadMapV21-SanDiego.pdf`.

[32] Sarp Oral, Feiyi Wang, David Dillow, Galen Shipman, Ross Miller, and Oleg Drokin. Efficient object storage journaling in a distributed parallel file system. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[33] Preparing for Exascale: ORNL Leadership Computing Facility Application Requirements and Strategy, 2009, `http://tinyurl.com/nccs2009`.

[34] X. Ouyang, S. Marcarelli, and D. Panda. Enhancing Checkpoint Writing with Staging IO and SSD. In *In Workshop on Storage Network Architecture and Parallel I/Os (SNAPI'10)*, May 2010.

[35] The Pandaboard Development System. `http://pandaboard.org/`.

[36] Juan Piernas, Jarek Nieplocha, and Evan J. Felix. Evaluation of active storage strategies for the lustre parallel file system. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 28:1–28:10, 2007.

[37] Ramya Prabhakar, Sudharshan S. Vazhkudai, Youngjae Kim, Ali R. Butt, Min Li, and Mahmut Kandemir. Provisioning a multi-tiered data staging area for extreme-scale machines. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS '11, pages 1–12, 2011.

[38] R. Rew and G. Davis. NetCDF: an interface for scientific data access. *IEEE Comput. Graph. Appl.*, 10(4):76–82, 1990.

[39] Erik Riedel, Garth A. Gibson, and Christos Faloutsos. Active storage for large-scale data mining and multimedia. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 62–73, 1998.

[40] Seung Woo Son, Samuel Lang, Philip Carns, Robert Ross, Rajeev Thakur, Berkin Ozisikyilmaz, Prabhat Kumar, Wei-Keng Liao, and Alok Choudhary. Enabling active storage on parallel i/o software stacks. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–12, Washington, DC, USA, 2010. IEEE Computer Society.

[41] Top500 supercomputer sites. `http://www.top500.org`.