Provisioning a Multi-Tiered Data Staging Area for Extreme-Scale Machines

Ramya Prabhakar[1], Sudharshan S. Vazhkudai[2], Youngjae Kim[2], Ali R. Butt[3], Min Li[3], and Mahmut Kandemir[1]

[1]Pennsylvania State University, [2]Oak Ridge National Laboratory, [3]Virginia Tech

{rap244, kandemir}@cse.psu.edu, {vazhkudaiss, kimy1}@ornl.gov, {butta, limin}@cs.vt.edu

***Abstract*—Massively parallel scientific applications, running on extreme-scale supercomputers, produce hundreds of terabytes of data per run, driving the need for storage solutions to improve their I/O performance. Traditional parallel file systems (PFS) in high performance computing (HPC) systems are unable to keep up with such high data rates, creating a *storage wall*. In this work, we present a novel multi-tiered storage architecture comprising hybrid node-local resources to construct a dynamic data staging area for extreme-scale machines. Such a staging ground serves as an *impedance matching* device between applications and the PFS. Our solution combines diverse resources (e.g., DRAM, SSD) in such a way as to approach the performance of the fastest component technology and the cost of the least expensive one. We have developed an automated provisioning algorithm that aids in meeting the checkpointing performance requirement of HPC applications, by using a least-cost storage configuration. We evaluate our approach using both an implementation on a large scale cluster and a simulation driven by six-years worth of Jaguar supercomputer job-logs, and show that our approach, by choosing an appropriate storage configuration, achieves 41.5% cost savings with only negligible impact on performance.**

I. INTRODUCTION

More than ever before, scientific discovery is dependent on the availability of computational power. Designing and developing software systems for emerging massive-scale High Performance Computing (HPC) machines, especially for achieving performance beyond the petascale, is a daunting task. In such post-petascale era, the imbalance between storage system performance and other system components, such as memory and compute, can significantly hinder critical scientific HPC applications. Emerging applications run on hundreds of thousands of cores for hours on end and conduct intense I/O, producing terabytes of data. An integral part of the application run is checkpointing, the process of saving a snapshot of the system state (application's in-memory data structures) to the parallel file system (PFS) periodically to protect against failure (as well as for inspection of application progress). Mean time to failure (MTTF), which is a week to a day in today's petascale systems will exacerbate to a day to an hour at exascale [20]. Thus, checkpointing is an indispensable I/O activity in HPC applications. For instance, one checkpoint timestep for a 100,000-core run of GTS [31] (a fusion simulation) on the Jaguar supercomputer [1] at Oak Ridge National Laboratory (No. 2 in Top500 [29]), produces 22 TB of data. To put this in perspective, consider that simply writing ten such checkpoints (one every hour) to storage at current I/O rates can consume up to 30% of the runtime. This is despite the fact that the PFS on such supercomputers can offer tens of GB/s of I/O throughput. Exascale projections [20] indicate that the many-cores and the data they can produce will expose the storage performance gap dramatically, suggesting that "storage wall", rather than memory wall, will be the limiting factor in such systems.

To mitigate the I/O bottleneck in checkpointing to a PFS, in-memory checkpointing techniques are often used. Solutions that checkpoint to a peer's node-local memory have been studied as a means to expedite checkpoint time and to return the application to conducting useful work [23]. Our own prior work used a percentage of the application's allocation (cores and their memory) to create an aggregate staging area, which can absorb checkpoint data [15]. The aggregate space is essentially a distributed memory buffer or a distributed ramdisk-based storage that intercepts checkpoint data and eventually drains it to PFS. While these approaches can reduce the checkpoint overhead, they use extra memory resources.

DRAM is an expensive resource in the HPC landscape for the following reasons. (i) Main memory provisioning is a significant portion of the multi-million dollar supercomputer budget. (ii) While it may seem that large-scale machines have a lot of memory, often to the tune of hundreds of TBs (e.g., Jaguar has 360TB of memory), modern HPC applications are equally memory hungry. This is because, such applications typically perform "in-core" computations, loading entire input datasets into main memory to avoid going to the disk often. The memory-to-FLOP ratio has been steadily declining, from 0.85 for the No. 1 machine on Top500 in 1997 to 0.01 for the projected exaflop machine in 2018 [29]. The exaflop machine is projected to host 60 petabytes of main memory. Even so, memory will be scarce. (iii) DRAM is a significant contributor to the power budget of the machine. Consequently, it is desirable to minimize the impact due to using memory for purposes other than computation. With reference to the staging approach outlined above, the traditional thinking has been to leave it to the application user to determine how much memory and cores to allocate in order to alleviate checkpointing I/O pressure. Thus, a dedicated center-wide partition of a memory-based staging area for checkpointing, while desirable, is not economically viable.

The advent of non-volatile memory devices (e.g., solid state devices SSDs) offers a tremendous opportunity in such a

setting. It is widely believed that SSDs or flash technology will serve to bridge the I/O performance gap between DRAM and disk [18], [25], [13], offering several desirable properties. First, SSDs provide superior throughput and lower access latency when compared to disks. Second, due to the lack of mechanical moving parts, SSDs are less failure prone so as to be adopted as node-local devices within supercomputer compute nodes. And finally, SSDs consume less power when compared to DRAM, albeit at the cost of an order of magnitude higher latency.

The logical question that arises in this context is how to use these *storage and memory technologies in concert and bring them to bear on the urgent problem of HPC checkpointing*. In this paper, we propose a multi-layered storage system for the HPC I/O hierarchy, using memory, SSD and disk. We envision an environment in future extreme-scale systems where a partition of compute nodes, along with their node-local main memory and any potential non-volatile memory, is dedicated to alleviating the I/O bandwidth bottleneck in checkpointing and writing output data. Such a storage system could be HPC center-wide, servicing all in-coming jobs or could be built from a percentage of a job's own core-allocation as mentioned earlier. The conjoined use of memory technologies makes it feasible to have a dedicated data staging area. SSD technologies offer larger capacity (hundreds of GBs) at a much cheaper price when compared to DRAM. For instance, the No. 4 machine in Top500 (Tsubame2 [30]) has around 173 TB of total node-local SSD storage. Currently, a high-end Fusion I/O PCIe MLC SSD card (io Drive Duo) at 640 GB is priced around $15K. Much like disk storage, SSD storage is increasing in capacity and decreasing in cost. Thus, growth in SSD space is currently outpacing memory increases. A concerted use of these layers can help provide larger capacity than what memory-alone can offer, while also reducing the cost required to provision such a staging storage system.

A. Contributions

Staging Area: We have proposed, built and evaluated several scenarios for an intermediate data staging device for extreme-scale machines. Our work illustrates how such a device can be positioned in the HPC center as a center-wide or as a per-job dedicated resource and absorb intense checkpoint data.

Multi-Tiered, Hybrid Architecture from Node-local Resources: We have put forth a novel multi-tiered storage architecture for the staging area, using node-local resources such as DRAM and SSD. Our solution is able to seamlessly use these devices under different resource contribution constraints and offer excellent I/O throughput to checkpointing applications. We particularly shed light on the utility of the emerging SSDs in the HPC I/O landscape.

Provisioning and Cost/Performance Model: We have proposed a cost-aware provisioning algorithm that chooses the best (least-cost) storage configuration from a set of candidate configurations that meets the checkpointing performance requirement. Our model illustrates how using a combination of DRAM and SSD contributions can help optimize the total cost of provisioning of the staging area for a given performance goal.

Evaluation: Our evaluation was performed using both a large-scale (2400-core) machine and a simulation based on six years worth of Jaguar supercomputer job logs. The experimental results show that the multi-tiered hybrid staging area is able to scale to thousands of client application processes. Our simulation study shows that up to 41.5% and 36.3% cost savings can be achieved by constructing the staging area using our scheme, compared to memory-based and disk-based checkpointing, respectively, while incurring negligible impact on performance. Furthermore, using SSDs in the center-wide staging area results in 59.25% higher I/O throughput, compared to disk-based checkpointing without SSDs without any additional costs.

II. Design and Implementation

In this section, we first present the motivation and rationale behind our system design. Then, we describe the system architecture, followed by a discussion on how we have realized the system through a practical implementation.

A. Rationale

Our target systems are extreme-scale multi-petaflop machines. Current petascale machines comprise of thousands of compute nodes (e.g., Jaguar [1] with 18,000+ nodes and Kraken [2] with 8,000+ nodes), amounting to O(100,000) cores. The international exascale roadmap projects that by 2015, a 100-300 petaflop machine will host O(100,000) nodes with O(1M) cores and a tenfold increase for the exaflop machine by 2018 [20]. System memory size, which is at 0.3 PB in the 2+ petaflop Jaguar machine (No. 2 in Top500), will rise to 5 PB and 60 PB in the 100-300 petaflop and exaflop machine by 2015 and 2018 [20]. Fault tolerance techniques such as checkpointing will be a major challenge in such systems. A critical question facing the community is: *How to take a snapshot of the entire application memory within a matter of minutes?* Such questions form the basis for designing the storage subsystem: how long it takes to drain the system memory of the entire machine is a key factor in deciding how much I/O throughput the storage subsystem should provide. For example, if 5 PB of memory in the 100-300 petaflop is to be drained to the PFS in 10 minutes, then the storage system needs to support an I/O throughput of approximately 8-10 TB/s. The research challenge lies in answering whether a PFS alone can support such extreme speeds, specifically, whether an entirely disk-based network-attached storage substrate can offer such throughput.

Addressing the above challenge requires investigating the impact of several factors, including the usual suspects of cost, power, etc. To this end, one technology that is expected to play a significant role in extreme-scale machines in helping to alleviate the checkpoint overhead, is non-volatile memory devices or SSDs. SSDs posses several desirable qualities in terms of capacity, cost, performance, persistence and power consumption. However, given that HPC systems are only

barely beginning to scratch the surface on the utility of SSDs in the I/O hierarchy, it is not always clear where one would place such a device. Similarly, it is not clear whether SSDs would be local to the compute nodes, be on a separate partition, or on dedicated I/O nodes. While these aspects still need to be explored thoroughly, it is increasingly becoming clear that future multi-petaflop and exaflop systems will be essentially required to exploit such devices in an intelligent fashion to deliver the needed high I/O rates. The combination of memory, SSD and disk promises to be a highly potent solution, if used in a concerted fashion. Therefore, our goal of studying these technologies in a multi-tiered storage architecture for the HPC I/O hierarchy is very timely and can provide formative feedback to HPC system designers.

B. Resource Aggregation Scenarios

We envision two primary candidate target scenarios for the multi-tiered storage architecture (Figure 1). These present us with different resource aggregation use-cases.

1) Dedicated Center-wide Partition: In this scenario, a set of "fat", system nodes are dedicated as an HPC center-wide resource to address the I/O bandwidth bottleneck problem. These fat nodes are essentially no different than the compute nodes, but for perhaps having more DRAM (when possible) and non-volatile memory devices. Given the provisioning budget, only a subset of nodes may be equipped with SSDs. Thus, the nodes are likely to be a mix of resources that contribute memory-only, memory and SSD, or both. We also envision that such a partition would be well connected to the main compute nodes. As such separate *center-wide staging resource pools* gain popularity, one can even imagine scheduling its usage in the future. For instance, users could request for nodes in the staging pool, much like how compute nodes are requested through a scheduler (e.g., 100,000 nodes for compute and 500 nodes from the staging pool.) The advantage with the center-wide pool is that the resource can be scheduled and optimized across all jobs. However, a significant challenge would be to deliver the desired throughput on a per-application/job basis as it is now a shared resource. Finally, the I/O nodes in an HPC machine can even potentially double as a staging pool when equipped with SSDs.

2) Partition of Job Allocation: Alternatively, a percentage of the user's job allocation (e.g., 1% of 200,000 cores requested by the application) can be dedicated for the application's own I/O activities. This *in-job staging pool* would be dedicated to the application and, consequently, there will be no interference with other jobs. Several resource contribution models are feasible here, for example: (i) the staging pool is from dedicated compute nodes within the job allocation; and (ii) the staging pool is built from one or more cores and their memories (DRAM as well as non-volatile) spread across the compute nodes. The disadvantage of the in-job staging approach is that it potentially takes cores and memory away from computation. However, we argue that the application user should consider the entire application turnaround (and not just the computation time), and checkpointing time is a significant factor therein. Dedicating a staging pool only expedites the user's own application turnaround time.

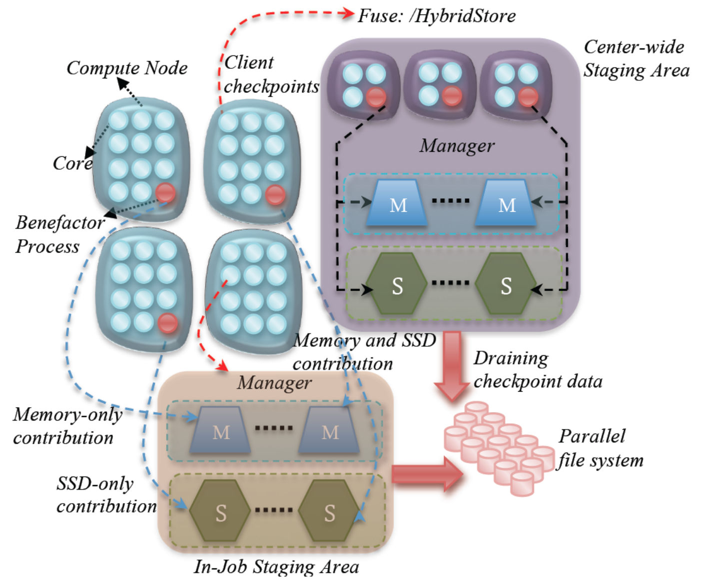

Fig. 1. High-level view of the multi-tiered staging area that is built by aggregating node-local resources such as memory and SSD. The figure depicts different resource aggregation scenarios such as center-wide staging area and in-job staging area. In addition, the figure depicts different benefactor contribution usecases.

C. Usage Models

The multi-tiered storage built using the aforementioned resource aggregation scenarios can be potentially used for both reading and writing large input and output data.

1) Checkpointing and Result Data Outputs: The primary usage model for the multi-layered storage is absorbing large, intensive checkpoint I/O. The storage tiers will be used to buffer the data as it flows down the I/O hierarchy from memory and SSD layers to the PFS. The checkpoint data is written once and read only in the case of failure. Thus, a key goal for this usage model is to optimize the multi-layered system for performance, in order to quickly digest the data emanating from the multicore compute nodes.

2) Reads and Prefetching: Another potential use of the tiered storage is to use it to perform prefetching or read-ahead caching from the PFS to the SSD and memory tiers so data can be brought closer to the application processes. While a potential usage scenario, this is beyond the scope of this paper.

D. Hybrid Storage Architecture

In this section, we describe the different components of the proposed heterogeneous staging storage architecture for the HPC I/O hierarchy in Figure 1. Our architecture is derived from our prior work [15] that amasses node-local disk or SSD contributions. In this work, we significantly extend the prior work to seamlessly aggregate both DRAM buffers (not ramdisks) as well as SSD contributions and build a tiered architecture. Our design comprises of *benefactor* processes,

contributing node-local resources, i.e., memory buffers, SSD or both, to a *manager* process that aggregates these contributions and presents a collective front to client applications. In the case of a center-wide partition resource aggregation model, several compute nodes, in their entirety, can be dedicated to the staging pool. Here, the entire memory and SSD per node is contributed to the aggregate store. Alternatively, in the job-allocation partition model, a core within each compute node runs the benefactor process and contributes a portion of the memory and a partition of the SSD to the manager.

The manager process keeps track of different classes of benefactors: those that contribute (i) DRAM only (*m-benefs*); (ii) SSD only (*s-benefs*); and (iii) DRAM and SSD (*ms-benefs*). *ms-benefs* are more likely to be found in the center-wide staging pool model, while *s-benefs* can result primarily from the in-job partition model, wherein compute nodes only contribute a portion of their SSDs and are unwilling to part with any available memory. The manager uses these lists of benefactors and their available storage to satisfy different incoming client requests to store checkpoint data. In addition to keeping track of available space, the manager also maintains the status of the benefactors and the datasets stored in them. Thus, the manager serves to maintain the metadata information about the benefactor storage nodes and also maintains a mapping of benefactor to datasets. Further, it runs key provisioning and striping algorithms to determine which classes of benefactors to store datasets on, so as to maximize application perceived performance and also optimize the storage system cost.

Clients are typically application processes running on the thousands of cores of the job's compute node allocation. Each process of a parallel job is a client to the hybrid store, checkpointing its memory/core. Thus, the hybrid storage system will need to cater to thousands of requests at the same time, even if it is setup as an in-job staging area. This is because, parallel applications typically involve the processes invoking a barrier and then checkpointing. Clients contact the manager to obtain a plan (benefactor list) for their checkpoint data. The dataset is chunked (chunk size of 1 MB) and striped across the benefactors. Clients contact the benefactors directly, in parallel, to store the chunks, achieving high aggregate throughput. The benefactors are responsible for storing the chunks on their respective storage media (DRAM or SSD.) The aggregate memory or SSD store itself can be accessed through a FUSE [8] mount point so that client accesses can follow regular POSIX semantics.

E. Hybrid Storage as Multiple Tiers

The manager organizes the benefactor resources into an aggregate memory buffer and an aggregate SSD pool and further positions them into hierarchal layers, with memory as the top tier, followed by the aggregate SSD layer. These two tiers are followed by the HPC center's PFS. The stacked approach allows us to efficiently exploit the available resources in a concerted fashion with the checkpoint data trickling down from the aggregate memory to aggregate SSD and eventually to the PFS for stable storage. The entire process of the checkpoint data flowing through the tiered storage system is transparent to the client, with it being hidden behind an elegant file system interface.

Upon a client request to store the checkpoint data, the manager provides a set of benefactor storage nodes based on a provisioning algorithm (discussed in the next section.) The striping plan may contain a combination of benefactors from the *m-benefs*, *s-benefs* or *ms-benefs*, depending on availability and whether a desired performance criteria can be met. Data is written to the memory buffers of the benefactors in the aggregate memory pool, if such nodes from the *m-benefs* list have been allocated as part of the striping plan. DRAM is likely limited in the staging area, and one of our key goals is to study if having SSDs can help substitute for more nodes needing to contribute memory, and therefore, reduce the provisioning budget. To this end, the multi-tiered architecture seamlessly drains the data to the SSD tier, if one such has been allocated. If there are no memory benefactors, then checkpointing automatically begins at the SSD tier and gets drained to PFS. While the aggregate SSD tier likely has more capacity, it might still not be sufficient to hold the snapshot data from O(100,000) cores, each with say 2 GB/core. Consequently, draining the data to the PFS is a desirable feature for persistence. The key is to perform these operations asynchronously, in a way that does not impact application perceived throughput. Thus, the multi-tiered hybrid store should be viewed as an intermediate device to absorb checkpoint data and is primarily an *impedance matching* mechanism between the application and the PFS.

In cases where the SSD tier can hold the checkpointing data in its entirety, the draining to PFS can even be conducted in an out-of-band fashion, between two checkpoint timesteps. Leaving the checkpoint data stored in the aggregate SSD tier (if space is not an object) may also have certain other benefits in terms of faster access to restart files and input data for dependant jobs.

To ensure faster performance and reduce stalls, each benefactor uses a triple-buffer design to store and transfer the checkpoint data from memory to SSD or to stable storage. SSD layers tend be larger and, therefore, draining can easily be achieved without application perceived performance hit. Draining from memory in a timely fashion is critical due to its potentially limited size. Upon start-up of the benefactor component, a pre-specified amount of memory is allocated for managing the checkpoints. The memory is further divided into three equally sized buffers that are managed using the state diagram shown in Figure 2. This ensures that while a buffer is being drained by the draining agent, another buffer is used to write the next checkpoint chunk. The goal is to completely overlap the writing of buffers in the local memory and draining them to SSDs or to the PFS. Each benefactor runs an asynchronous dedicated thread, the draining agent, which writes the checkpoint data to the SSD tier or to PFS. In our design, we have employed one dedicated queue for each of the three buffers to ensure storing metadata such as

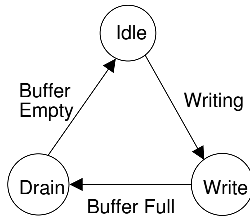

Fig. 2. State machine for managing benefactor memory buffers.

addresses and sizes of the checkpoint data chunks. However, each benefactor has only one draining agent that monitors the queues by continuously polling them to check if another buffer has become full for draining, which is then written to the next tier. After the data has been successfully drained, the buffer is released for handling the next set of chunks.

An interesting aspect here is the interaction between the memory and SSD tiers. While we view the memory and SSD layers as two aggregate storage systems, draining from the memory tier to the SSD tier does not necessarily mean always writing to an aggregate SSD abstraction. For instance, since a given striping map can potentially have nodes from the *ms-benefs* list, it makes sense for the draining operations to occur locally when feasible. Therefore, for benefactor nodes that contribute both memory and SSD, draining data is a local operation. However, when a node only contributes memory, draining involves writing from the benefactor memory to another benefactor's SSD. In the provisioning discussion, we will highlight how storage provisioning enables the selection of these nodes.

Another factor to consider in the multi-tier model is if a strict trickle-down approach always makes sense. For instance, given the availability of the benefactors or due to budget provisioning, there may be a disproportionate number of *m-benefs*, compared to *s-benefs*. Or, simply the amount of memory may be limited. In such cases, it does not make sense to throttle the checkpointing and have it all flow through the aggregate memory tier, which will be narrow. Instead, we should be able to stripe widely enough, across *m-benefs* and *s-benefs*, so as to maximize throughput. Such a scenario implies that client writes are performed in parallel to both memory as well as SSD benefactors. Within a single *ms-benef*, however, preference is always given to a memory write.

III. Provisioning the Memory and SSD Tiers

In this section, we discuss the allocation of available memory and SSD resources to different jobs, and present an algorithm for provisioning the resources with the goal to improve end-to-end application performance.

A. A Motivating Example

Consider the following example usage scenario that motivates the need for efficient provisioning of the hybrid storage tiers. We conducted a simple experiment to perform cost-performance analysis for different configurations of aggregate memory and SSD allocations. We use an MPI application running on 8 nodes, which checkpoints 2 GB per node to the multi-tiered storage. Figure 3 plots the variation in execution time and corresponding cost (in dollars) for provisioning a combination of memory and SSD resources for the application. We used the cost numbers from [26] for memory, SSD and disk. Comparing the two configurations, one with 8 GB memory and 8 GB disk ($msd-8-0-8$) and the other with 3 GB memory, 8 GB SSD, and 5 GB disk ($msd-3-8-5$), we see that there is a 40.9% drop in cost for only a 2.5% increase in execution time. Different configurations have their corresponding cost-performance tradeoff points, thereby motivating the use of SSDs in the storage hierarchy to achieve similar application performance at a lower provisioning cost. Also, between the configurations $msd-2-6-8$ and $msd-4-0-12$, we see that $msd-2-6-8$ achieves higher performance at a lower cost. This is because, in $msd-4-0-12$ configuration, the cost of 4 GB memory dominates, while the application performance degrades because of draining the remaining 12 GB of data to disk in the critical path of execution. Both cost and execution time overheads are amortized in $msd-2-6-8$ configuration due to the introduction of 6 GB of SSD and associated reduction in memory provisioning. This motivates the need for effective provisioning techniques to achieve the target performance from a cost-effective configuration.

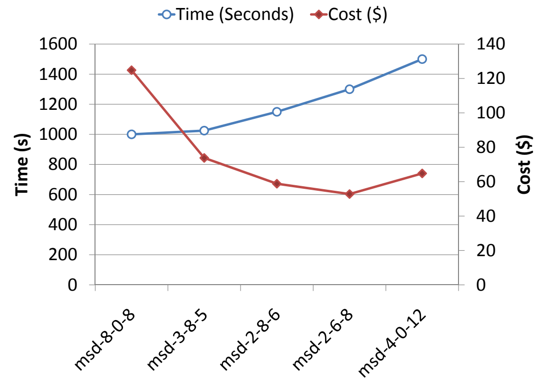

Fig. 3. Cost-performance analysis of different configurations of hybrid storage hierarchy. X-axis denotes storage capacities (GB) of each layer in the hierarchy formed of memory(m), SSD(s) and disk(d) in order.

B. Overview of the Provisioning Algorithm

The primary goal of the provisioning algorithm is to find the capacity to be provisioned for each tier (memory and SSD) in the hierarchical hybrid storage architecture to meet the checkpointing performance requirement with a least cost configuration. To this end, the algorithm strives to provide the following capabilities:

- It provides a guideline for system administrators to statically provision the staging area–either at the acquisition phase or during its instantiation–based on a prior history of jobs' checkpointing needs. The guideline helps decide the number of staging nodes and their memory, and SSD capacities to be provisioned, under given budget constraints, while still achieving a specified performance target (e.g., checkpoint in

10 minutes, etc.)

• The algorithm also aids the manager process in further optimizing the hybrid store's performance by guiding how the provisioned resources (available memory and SSD capacities) are to be distributed across the nodes allocated to applications.

The cost-performance analysis performed in the proposed algorithm is based on the requirement for every application individually. It is assumed that system administrators have access to a potential target workload for which they are trying to provision the storage.

1) Terminology: Let T_{app} be the performance requirement of the application. For example, the performance requirement can be specified using I/O throughput (GB/s). Let $Benef_{mem} = m_1, m_2, ..., m_i$ be the theoretical set of memory contributing benefactors with sizes $m_1, m_2, ..., m_i$. Similarly, let $Benef_{ssd} = s_1, s_2, ..., s_j$ be the theoretical set of SSD contributing benefactors with sizes $s_1, s_2, ..., s_j$. These represent the granularity of control for determining the resources required to meet the specified performance target. Recall that a hierarchical storage system can be viewed as two sets of memory and SSD contributing nodes. We refer to a *benefactor* as the process executing on a resource contributing node, which donates the specified amount of resources for the purposes of checkpointing. Benefactors can donate different amounts of memory and SSD resources. The number of all possible combinations of these benefactors for a hybrid staging area (All_Config) is the cross product of the sets $Benef_{mem}$ and $Benef_{ssd}$, i.e., $All_Config = Benef_{mem} \times Benef_{ssd}$. Each element in All_Config is denoted by $Hybrid(i, j)$. For example, $Hybrid(16, 32)$ denotes a configuration with 16 GB memory and 32 GB SSD. The configurations in All_Config can also denote only-memory or only-SSD configurations, where the other element in the tuple is zero.

2) Provisioning Procedures: The provisioning algorithm (Figure 4) is composed of the following five steps:

Step 1: For each combination of any two benefactors ($Hybrid(i, j)$), we determine its average throughput denoted by $T_{Hybrid(i,j)}$.

Step 2: We compute the number of instances of $Hybrid(i, j)$ required to meet the performance requirement of application T_{app}. The number of instances $N_{Hybrid(i,j)}$ of $Hybrid(i, j)$ can be modeled as:

$$N_{Hybrid(i,j)} = \frac{T_{app}}{(T_{Hybrid(i,j)} - \frac{(N-1)}{D_{link}})}, \quad (1)$$

where D_{link} is the loss in I/O throughput of every link access. The above model is for an in-job staging area. The model for a center-wide staging area would be N instead of $N - 1$ in the right operand of the denominator, denoting that all nodes in the staging area are remote to the clients. At the end of this step, we would have the aggregate memory and SSD resources required to meet target performance T_{app}. For example, if we deduce that we need four instances of 2 GB memory and 4 GB SSD, it translates to an aggregate memory of 8 GB and an aggregate SSD of 16 GB to be provisioned to achieve the application specified checkpoint throughput.

Input :
T_{app}: Target checkpointing I/O throughput (GB/s)
M' represents set of theoretical memory sizes,
M':$\{m_1, m_2, ..., m_i\}$,
S' represents set of theoretical SSD sizes,
S':$\{s_1, s_2, ..., s_j\}$
Output :
$Best_Config(K, M, S)$: A set of K nodes configured with a total of memory (M) and SSD (S) capacities $\rightarrow T_{app}$
Additionally, $Best_Config(K, M, S)$ minimizes cost

Algorithm :
System Initialization :
Determine the set of all possible hybrid configurations
$All_Config : M' \times S' = \{(m_1, s_1), (m_1, s_2), ..., (m_i, s_j)\}$.
while (true)
Iterative Performance Annotation:
$\forall j \in All_Config$,
Compute $T_j \leftarrow$ maximum achievable I/O throughput for instance j.
Determining Instance Set:
$\forall j \in All_Config$,
Compute $Num_j \leftarrow$ (Num) number of instances of j required to achieve T_{app} using,
$Num_j = \frac{T_{app}}{(T_j - \frac{(Num-1)}{D_{link}})}$,
where D_{link} is the loss in I/O bandwidth for every link
Add (Num,j) to instance set.
Selecting Least-cost Configuration :
$\forall (Num, j) \in instanceset$,
Compute cost([Num,j]) = installation + operation cost
Select the least-cost configuration.
Performance Optimization:
The least-cost configuration with (Num) instances of j is provisioned from the resource contributing nodes allocated to applications by prioritizing nodes in the order of nodes contributing both memory and SSD, followed by only memory nodes, and then only SSD nodes.

Fig. 4. Details of our provisioning algorithm for data staging area.

Step 3: Next, we determine the associated costs of provisioning resources. Here, the cost function of each resource can be defined as a function of resource installation cost ($Cost_{installation}$), and its operational cost ($Cost_{operation}$). The installation cost is a function of the device unit cost, and the operational cost is a function of the power consumed followed by the cooling cost. Since the resource installation cost is much higher than its operational cost under the current memory market prices, we mainly consider the device installation cost in our provisioning model. That is, cost of $Hybrid(i, j)$ is a function of the purchasing cost of memory and SSD of sizes i and j, respectively. The total cost of provisioning is therefore, $N_{Hybrid(i,j)} \times (Hybrid(i, j)$. We use the cost numbers from [26] for memory, SSD and disk. At the end of this step, of the different configurations that meet the target performance, we choose the least-cost configuration. It is important to note that our primary objective is to provision resources to meet applications' performance goals. Our algorithm chooses the least-cost configuration only when there is more than one configuration yielding the same performance benefits. Another important decision to be made is to determine how the aggregate memory and SSD resources, determined in the previous steps, are allocated at individual nodes so that applications derive maximum performance benefits.

TABLE I
EXPERIMENTAL SETUP.

Parameter	Value
Number of Processing Nodes	300
Storage per Node	320GB
Network Interconnect	Infiniband QD 40 Gb/s
HDD Model	WD3200AAJS-41VWA0
Cores per Node	8
Memory per Node	8GB
Maximum Available Cores	2400

TABLE II
INTEL X25-E SSD SPECIFICATIONS [11].

Parameter	Value
Model	Intel X25-E Extreme
Features	SATA-II SLC Flash Technology
Capacity	32GB
Sequential Read Bandwidth	250MB/s
Sequential Write Bandwidth	175MB/s
Random 4KB reads	35K IOPS
Random 4KB writes	3.3K IOPS

Step 4: In order to maximize parallel access to memory and SSD resources for checkpointing, it is important that the determined aggregated memory and SSD capacity is allocated across multiple nodes. However, the number of nodes across which the resources are allocated should not exceed $N_{Hybrid(i,j)}$, determined in Step 2. This is because the memory and SSD capacity to be provisioned to meet the target performance was determined factoring the link delay, D_{link}. Therefore, distributing resources to more than $N_{Hybrid(i,j)}$ may not yield the specified performance. Although, it appears that minimizing the number of nodes, where memory and SSD resources are allocated, is always beneficial because of reduced link delays, the performance benefits obtained by allocating resources across multiple nodes due of parallel access, amortizes the loss in I/O throughput due to extra link accesses.

Step 5: In order to further optimize performance when allocating memory and SSD resources, the type of benefactors ($m - benef$, $s - benef$, $ms - benef$) used for allocation is chosen carefully. Of these three types of nodes, in order to derive maximum performance, when available, it is beneficial to allocate nodes that contribute both memory and SSD ($ms - benef$.) This is because, such nodes have access to local SSD resources to which data can be drained without incurring network access latency. Our proposed provisioning algorithm performs this optimization, whenever feasible, after choosing a least-cost configuration.

IV. EXPERIMENTAL SETUP AND EVALUATION

We have evaluated our multi-tiered hybrid store along with the provisioning algorithm using both trace-driven simulations and an implementation running on a large-scale machine comprising 2400 cores. We first present the experimental setup and evaluation based on our implementation, and then describe the simulation-based results.

A. Evaluation at Scale Using the SystemG Machine

1) Setup: For our experiments, we used the 2400-core systemG machine at Virginia Tech. Table I shows the detailed configuration of the testbed. Each node is also equipped with an SSD in our evaluation setting, using an emulated SSD device driver that has been validated against a real product (specifications shown in Table II) for sequential I/O throughputs within an error margin of 0.57%. The emulated device uses DRAM for storage and emulates a real SSD by introducing artificial delays, based on our previous work on simulating SSDs [14]. In our testbed, we use node-local disks as the last level in the storage hierarchy to provide data persistence. We use a synthetic checkpointing application that generates a uniform dataset as checkpoint data on every time step. Specifically, we created an MPI program that has a configurable number of processes, with each process writing 0.25 GB of data per checkpoint.

2) Large-scale System Results: In our first set of experiments, we evaluated the impact of varying memory, SSD and disk allocations to the checkpointing application. We consider the following three cases. The first two depict a center-wide staging scenario, wherein each node contributes both memory as well as SSD. The latter depicts an in-job staging area with non-uniform contributions.

Aggregate Memory and Disk Tiers: The experiments were run with 1800 and 1200 parallel application client processes that checkpoint 450 GB and 290 GB of data, respectively, to an aggregate memory device in each time step, with the data drained to disk storage. The total memory contributions range from 50 GB to 400 GB, coming equally from all 300 nodes i.e., memory contribution per node ranges from 0.16 GB to 1.33 GB. Figure 5 shows the I/O throughput obtained under this test. As expected, the checkpointing throughput of the applications are highly sensitive to memory allocation with the throughput ranging from 31.0 GB/s to 51.8 GB/s on 1800 processes, and varying from 16.2 GB/s to 43.7 GB/s on 1200 processes.

Aggregate Memory, SSD and Disk Tiers: In this test, application processes checkpoint (450 GB of data) to the aggregate memory, aggregate SSD and disk tiers. Data is drained to SSDs (with aggregate SSD contributions ranging from 150 GB to 250 GB, coming equally from all nodes i.e., SSD contribution per node ranges from 0.5 GB to 0.8 GB). Any overflow data is also drained to disks. We used 1800 parallel application client processes. Figure 6 shows the I/O throughput obtained when both memory and SSD contributions are varied. Comparing Figure 5 and Figure 6 reveals some interesting cost/performance trade-offs in provisioning storage systems. For instance, if the cost is fixed as that of provisioning 150 GB of memory, then adopting a mix of 100 GB memory and 250 GB SSD can provide an I/O throughput improvement of 5.2 GB/s. Conversely, if the desired I/O throughput is fixed as that obtained using 250 GB of aggregate memory, then

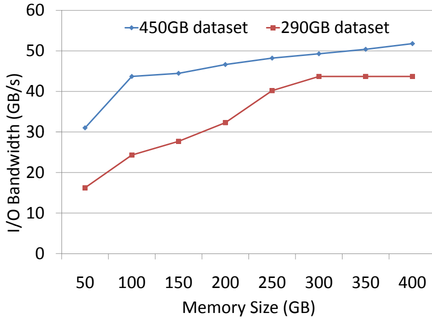

Fig. 5. Sensitivity to different memory–disk configurations using 1800 and 1200 client processes.

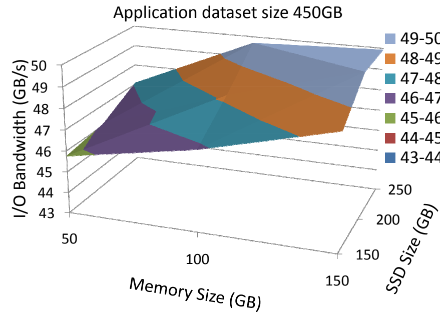

Fig. 6. Sensitivity to different memory-ssd-disk configurations using 1800 client processes.

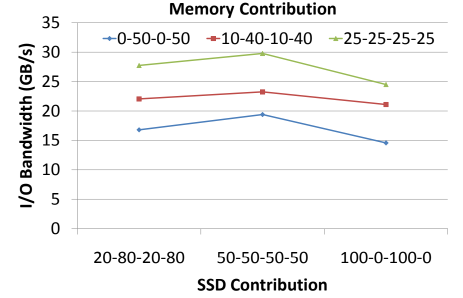

Fig. 7. Evaluating memory-ssd-disk configuration with non-uniform resource contributions from nodes using 1200 client processes.

adopting a 100 GB memory contribution along with a 200 GB SSD contribution can yield a cost reduction of more than 44%. This suggests that a multi-tiered staging area with careful allocation of DRAM and SSD tiers can help amortize the cost of provisioning, while also achieving high I/O throughput.

Non-Uniform Contributions: In this test, the application checkpoints (290 GB of data) to the aggregate memory/SSD and disk tiers. We used non-uniform resource contributions, with nodes contributing different amounts of memory and SSD storage. The nodes are divided into 4 sets with 50 nodes (400 cores) in each set. All the nodes in a particular set contribute equally, but the contributions of the sets are non-uniform. Any overflow data is also drained to disks. Figure 7 shows the I/O throughput in this heterogeneous resource donation case. The x-axis in Figure 7 represents three cases, where the notation $(s_1 - s_2 - s_3 - s_4)$ represents the total SSD contribution of the nodes in each of the 4 sets, with s_i GB SSD contribution from each set i (e.g., each of the 50 nodes together contribute s_i GB of SSD.) Similarly, three types of memory configurations are studied, with each line in the graph representing a case with different memory contributions by different sets. A notation $(m_1 - m_2 - m_3 - m_4)$ is used to represent the total memory contribution of the nodes in each of the 4 sets, with m_i GB memory contribution from each set i. This heterogeneous scenario tests an in-job staging area scenario, wherein all compute nodes may not be able to contribute equal amounts of memory or SSD. All nodes in the job allocation may not be equipped with an SSD unlike the center-wide staging area usecase. The key observations from these results are that the available parallelism in checkpointing data is critical to the I/O throughput obtained. For example, when all 4 sets (of 50 nodes each) contribute both memory (total of 100 GB) and SSD (a total of 200 GB), the throughput is observed to be higher than when some of the sets contribute either memory only or SSD only, although the overall contributions remain the same. This is because, such nodes have access to local SSD resources to which data can be drained without incurring network access latency.

Using these evaluations at scale from a real-world machine, we have described key insights learned on the design parameters that influence provisioning. Further, we were able to obtain realistic values for parameters that we then used in our job-logs based simulations.

TABLE III
JAGUAR JOB LOGS STATISTICS.

Parameter	Value
Duration	443,803 Hrs
Number of jobs	304,361
Job execution time	30 s to 7,394.08 hrs, average 1.45 hrs
Data size	2.28 MB to 291.27 TB, average 2.8 TB

B. Simulation

We now describe the large-scale job traces used in the simulation, the simulator setup and our analysis of the proposed provisioning algorithm using the job traces.

1) Traces: The simulations were driven using six-year job traces from the Jaguar supercomputer at Oak Ridge National Lab. The job traces provide the following information for each job: *arrival time, start time, total job execution time, total amount of memory and compute resources used*. Since the logs are devoid of output data sizes, we used the product of the memory requested per core and the number of cores requested as an approximation of the checkpoint data size. For example, if the job used 1,000 cores and 2 GB of memory per core, we assume its output data size to be 2000 GB. This is a very reasonable assumption given that many data-intensive applications' checkpoint or restart output datasets cannot be larger than their total memory usage. The output data of real applications can run in the hundreds of GBs and TBs for leadership simulations on Jaguar. Each job in the trace corresponds to a job executing in our simulator. The output from the simulator contains the overall resource usage in terms of memory, SSD and disk space used, and the time taken to write the required data for a given job. This information is used for computing the cost for provisioning those resources and the job's I/O throughput. We used cost estimates for memory, SSD and disk from the rates presented in [26].

Table III shows some relevant characteristics of the logs. Also, note the large variance in both the duration of the jobs (from a few seconds to over a day) and the amount of data they access (from a few MBs to several TBs). Clearly, a single provisioning method to accommodate such a large variety of

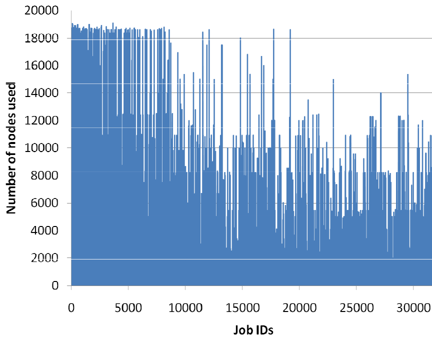

Fig. 8. Distribution of the number of nodes used by different jobs.

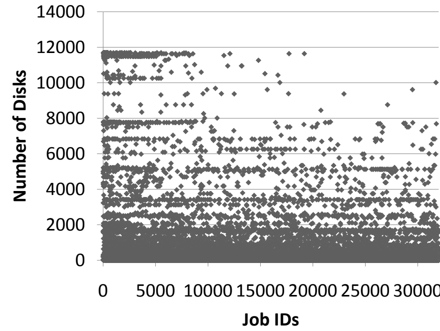

Fig. 9. Number of disks required in disk-based checkpointing to achieve target performance.

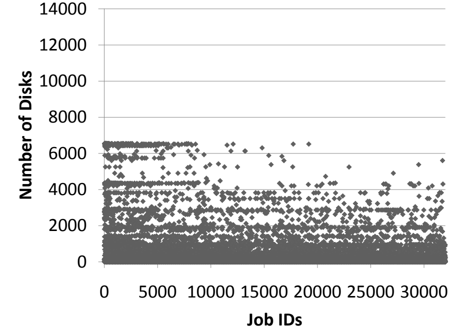

Fig. 10. Number of disks required in ssd-disk configuration to achieve target performance.

jobs on HPC systems is not efficient. Understanding the cost-performance tradeoffs with these job logs from real execution, enables better system-wide provisioning decisions.

2) Simulator Details: We performed a detailed cost-performance analysis of our provisioning algorithm using six-years of Jaguar job logs with 304,361 jobs. An overview of a representative set of 32,000 jobs[1] in terms of the number of nodes used by each job is shown in Figure 8. Each node in this case (in Jaguar) has dual-hex cores (12 cores) and 16 GB memory. The number of nodes used among the jobs varies all the way from 1 node to more than 18,000 (representing 225,000+ cores.) With these jobs, and some "hero runs" (applications that use large number of cores and have large datasets) we selected from the same, we now report the results of the analysis. The simulator faithfully models the impact of both checkpoint staging area model and in-job allocation model by providing the flexibility to change both memory and SSD provisioning in either of these models. It takes as parameters, the fraction of cores allocated for in-job allocation in addition to memory and SSD sizes used for staging area. Simulation of draining between memory and SSD tiers is also validated against our real system emulation results by capturing the draining rates from each tier in an equivalent real system and using them in simulations.

3) Comparing Disk-based and Staging-Area-based Checkpointing: In provisioning disks to typical HPC workloads, we are often constrained not by disk drive capacity, but by the ability to sustain throughput (IOPS or GB/s). Since capacity per device is increasing exponentially and I/O throughput per device is increasing linearly (sublinearly), storage provisioning for HPC systems will have to start purchasing performance rather than capacity (if not already). Keeping this in mind, we analyzed the number of parallel disks our system would need, in order to provide a sustained I/O throughput to all the jobs on the Jaguar logs, such that their checkpointing requirements can be met within 5 minutes per hour of execution. The number of parallel disks required to be provisioned are shown in Figure 9. We can see that the maximum value is 11,696 disks in parallel to sustain the I/O throughput requirements. However, by introducing node-local SSDs into slightly more than 10% of the nodes in the system, the staging area can not only meet the sustained throughput requirements of all applications, but also minimize the overall cost of provisioning the system. For example, the reduced number of disks required when SSDs are introduced is shown in Figure 10. We can see that the maximum number of disks required reduces to 6550 disks (a reduction of 5,196 Disks, i.e., 44%). Therefore, it is extremely beneficial to introduce SSDs in the I/O stack, to improve cost per unit performance of the system.

4) Deploying SSDs in Center-Wide Staging Model: We now evaluate the benefits of deploying SSDs in a center-wide staging model and show the benefits of using our provisioning algorithm to strike the right balance between cost and performance in provisioning systems. We provide a detailed cost-performance analysis for the "hero applications" selected from Jaguar job logs. There are 515 such hero applications in the logs. However, we present data for 5 representative applications (DS199TB, DS41TB, DS109TB, DS298TB, DS260TB) with data set sizes ranging from 41 TB to 298 TB, that exhibit varying characteristics in the range of number of nodes used and the amount of checkpointing performed. Figure 11 shows the cost savings achieved as analyzed by our provisioning algorithm to minimize the cost of provisioning the system for a sustained throughput of checkpointing within 5 mins for every hour. The results are shown for three scenarios in the center-wide staging model where the ratio between the number of computing nodes to nodes in the staging area varies between (64:1), (128:1) and (256:1) (much like compute nodes to I/O nodes ratios in HPC centers). The bars for cost without SSD represent a scenario where memory in all nodes of the center-wide staging area is used for checkpointing purposes as well. The bars for cost with SSD show the cost of the minimum cost configuration with the introduction of SSD as analyzed by our provisioning algorithm. We observe up to a 41.5% reduction in cost that can be captured by our provisioning algorithm. Similarly, Figure 12 shows the maximum throughput for a given cost of $90,000 that can be achieved as pointed out by our provisioning algorithm when using SSDs. We can see that up to 59.3% improvement in throughput can be obtained without incurring any additional costs by using SSDs in the

[1]Although we analyzed the logs using all 304,361 jobs, we present a representative set of 32,000 jobs due to limitations in the plotting tool we used to show more than 32,000 data points

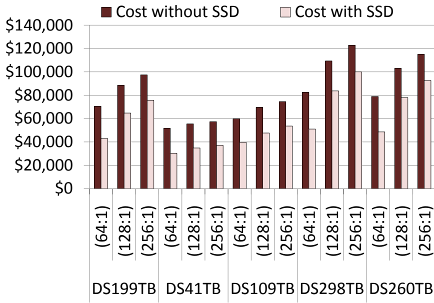

Fig. 11. Cost savings achieved by provisioning SSDs to meet target checkpointing performance of 5 minutes checkpointing time per hour of execution.

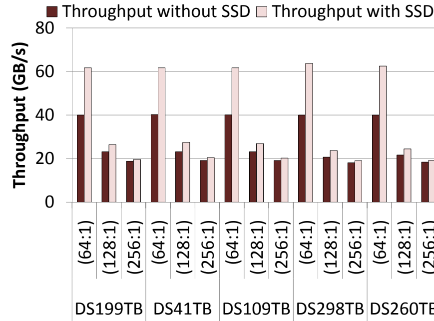

Fig. 12. Performance improvements achieved by provisioning SSDs under given cost constraints of $90,000.

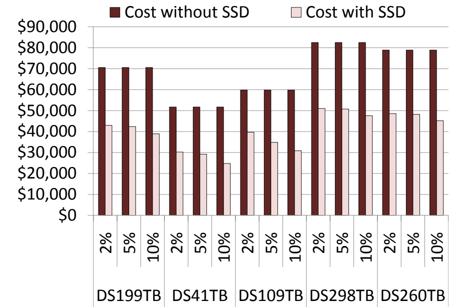

Fig. 13. Variation in cost of provisioning for a fraction of loss in target checkpointing performance.

center-wide staging area. These are the two extreme cases of provisioning in the center-wide staging model. However, if the system is willing to tolerate a trade-off in performance, ranging from 2 to 10% (corresponding to an increase in checkpointing time by 6 to 30 seconds), we can discover opportunities for significant reductions in cost. Figure 13 shows these trade-offs. We can find up to 52.1% reduction in cost for a 30 sec (10%) increase in checkpointing time.

5) Sensitivity to In-Job Staging Area: Finally, we also evaluated the trade-offs involved in using an in-job staging area model. In the in-job staging area, we used a fraction of the computing cores to execute the benefactors, thereby contributing their memory and associated SSD space for checkpointing. We show the analysis for a “hero application” that uses all the cores (more than 224,000) on Jaguar and executes for 14,440 seconds (as indicated by the dotted line). The increase in execution time of this application when it parts with 1% to 2% of the cores is shown in the first bar of Figure 14. However, using these 1% to 2% of the cores for an in-job staging area and utilizing both memory and SSDs available on it, not only contributes to an improvement in the checkpointing performance, but also improves the overall application execution time by up to 3%. This experiment illustrates the benefit of the in-job staging area model. We also experimented with the case when only SSD resources are utilized from the in-job staging area. In such a scenario, in order to achieve the performance target of 14,440 seconds, we would need more than 4% of the cores and associated resource allocation. Hence, to keep comparisons uniform, we report results with memory-disk and memory-SSD-disk configurations.

In summary, our simulation results suggest that the introduction of SSDs into the data staging area can not only reduce the cost of provisioning the tiered storage system, but also improve throughput and checkpointing time. Our analysis further suggests that the tiered staging area is a viable approach in both a center-wide as well as an in-job resource aggregation scenario. Finally, the results also indicate that the provisioning algorithm can help discover and highlight potential tradeoffs in cost and performance.

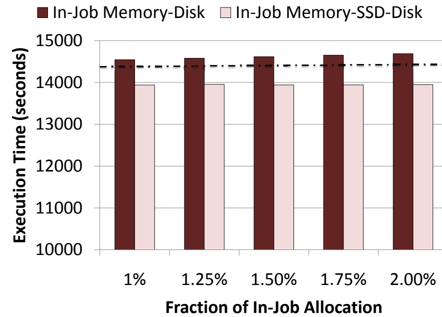

Fig. 14. Sensitivity to fraction of in-job checkpoint node allocation.

V. Related Work and Discussion

In this section, we discuss related efforts that explore the use of SSDs in the storage hierarchy. We also discuss the state-of-the-art in storage system provisioning and explain their shortcomings when applied to situations that require provisioning heterogeneous resources (memory, SSD and disks) from several storage nodes in a distributed storage system.

A. SSDs in the Storage Hierarchy

There has been considerable amount of work on using SSDs as part of the storage hierarchy in the traditional operating systems community. Some of these efforts focus on integrating flash as a persistent storage in file systems, either by replacing hard disks or by augmenting it with existing disk drives. In [18], the authors show that for large server-side workloads, replacing disks with SSDs is not a cost effective option at today’s prices. Depending on the workload, the capacity/dollar of SSDs needs to improve by a factor of 3 to 3000 for SSDs to replace disks. However, they do acknowledge the benefits of using SSDs as an intermediate caching tier, alongside disks. Several efforts [17], [7], [21] employ SSDs as a cache atop hard disk to improve read performance. For example, Intel’s Turbo Memory [17] uses NAND-based non-volatile memory as an HDD cache. Operating system technologies such as Windows ReadyBoost [7] uses flash memory, for example in the form of USB drives, to cache data that would normally be paged out to an HDD. Windows Ready-Drive [21] works on hybrid ATA drives with integrated flash memory, which allow reads and writes even when the HDD is spun down.

Several other efforts [25], [12] show how Flash can be used in today's server platforms as a disk cache. Combo Drive [24] is a heterogeneous storage device in which sectors from the SSD and the HDD are concatenated to form a continuous address range, where data is placed based on heuristics. In [27], the authors propose to use hard disks as a write-cache to SSDs to increase SSD lifetime. In [16], the authors have designed and developed a hybrid device driver to use SSD as an I/O cache. In [13], the authors provided capacity planning techniques to administrators with the overall goal of operating within cost-budgets and performance/lifetime guarantees during episodes of deviations from expected workloads. However, their work is limited by focusing on building back-end storage systems and by not considering the HPC domain.

B. SSDs in HPC Systems

The adoption of Flash in HPC systems, however, has been slow and is only now gaining momentum. Recently, several NSF and DOE supercomputers are being equipped with SSDs on each of the compute nodes, offering the application processes a very desirable node-local storage alternative to the ever-crowded PFS. For instance, the No. 4 machine in Top500 (Tsubame2 [30]) has around 173 TB of total node-local SSD storage. NSF's 64-node DASH machine [10] at SDSC (precursor to Gordon [19], a 1024 node cluster) has 4 TB of flash memory compared to its 3 TB of DRAM. It is widely expected that flash devices will be used in some form—either as a bleed-down buffer or as a memory extension—for future extreme-scale machines. HPC I/O operations such as reading and writing of large input and output datasets and checkpointing intermediate snapshots of data are extremely I/O intensive. These operations can benefit immensely by smart placement of SSDs in the data path (e.g., on the compute nodes, staging nodes, I/O nodes or on the PFS). While we are beginning to see studies that evaluate scientific application data accesses on SSDs [22], there is not much work exploring the various ways in which such devices can be used for HPC I/O. Even the aforementioned machines that are equipped with SSDs leave it upto the applications to use them as simply a node-local resource, without any coherent solution. Our own prior work [15] in this area explored the construction of an aggregate store of SSD devices as an intermediary between applications and the PFS. In this paper, however, we extend this work to position SSDs as the second tier in a multi-tiered staging storage system.

C. Staging Storage in HPC Systems

Using a set of staging nodes, from within a job allocation to conduct in-situ execution is gaining popularity in HPC [32]. Staging areas serve dual purposes: accelerating periodic I/O and performing data analysis/reduction on-the-fly, while the job is running. PreDatA [32] showed that with a rather small extra set of staging nodes (1.5% of the original allocation), important yet relatively unscalable common data processing tasks such as sorting can be carried out on these nodes while the normal compute nodes proceed with their computation. Our own prior work explored the in-job staging approach to dedicate entire compute nodes (up to 1% of the allocation) to perform checkpointing therein [3]. All of the above solutions simply use the DRAM available to the compute node to build a staging area. PreDatA does not aggregate memory like our approach either. The fundamental drawback of these techniques is their reliance on DRAM for the staging area. While the staging ground can offer excellent performance, DRAM (as discussed earlier) is an expensive resource in HPC systems. Memory is only becoming much more scarce in extreme-scale machines. In this paper, however, we explore different scenarios for the staging ground (including a center-wide staging storage) and build it using a hierarchical model with hybrid resources. Our hybrid approach preserves the desired performance of the staging area, while also reducing the cost of provisioning such a staging storage in HPC systems. Cost of provisioning is a key factor as the staging storage is proposed as a resource in addition to the traditional PFS, which itself is a substantial fraction of the HPC center's acquisition and operations budget.

D. Storage System Provisioning

Storage systems can be complex to manage. Management consists, among many other tasks, allocating resources for each storage node, and mapping application workloads to these nodes. Unfortunately, the state-of-the-art in storage management requires much of this to be done manually. Often, storage administrators use rules-of-thumb to determine the appropriate sizes, and workload-device mappings. This can lead to suboptimal performance and wasted resources.

Storage provisioning and configuration tools, including Minerva [4], Ergastulum [6], and the Disk Array Designer [5], are targeted at creating minimum cost designs that satisfy some fixed level of performance and data protection. Meisner et. al. [9] argue that with the increasing heterogeneity in storage systems, intelligent provisioning of resources is difficult without end-to-end performance specification. In [28], the authors propose the use of utility functions to enable cost-benefit structure to be conveyed to an automated provisioning tool, enabling the tool to make appropriate trade-off decisions. However, the various system models for performance, power and availability mentioned in [28] are devised considering only hard-disks as storage system resources. To the best of our knowledge, none of the previous storage provisioning research considers heterogeneity in distributed storage systems while modeling their system parameters. It is therefore important to explore the storage system provisioning techniques in the context of heterogeneous distributed storage systems.

VI. CONCLUSION

This paper presents a novel multi-tiered data staging area for large-scale machines, which acts as an intermediate storage device between applications and the PFS. The staging device absorbs application checkpoint data and seamlessly drains it through its tiers to the PFS. Our approach exploits diverse node-local resources (e.g., DRAM, SSD) to provision a hybrid

storage that is less expensive than an entirely memory-based staging ground and yet is able to meet application checkpointing performance needs. We argue that reconciling these tradeoffs between different system metrics is crucial in future extreme-scale machines. Such staging device also alleviates the pressure on the PFS. Our evaluation, based on a large-scale testbed and Jaguar job-log simulation, suggests our approach is viable, and is promising in mitigating the storage wall faced in emerging large-scale machines. Our future work is exploring the design of efficient caching and prefetching strategies to utilize the aggregated staging area more effectively.

ACKNOWLEDGMENTS

This work was sponsored in part by the LDRD program of ORNL, managed by UT-Battelle, LLC for the U.S. DOE (Contract No. DE-AC05-00OR22725), and by the U.S. NSF Awards CCF-0937827, CCF-0746832, CNS-1016793, CCF-0937949, CCF-0621402, OCI-0724599, OCI-0821527, and CCF-0833126. Ramya Prabhakar was an intern at ORNL during the summer of 2010, when this work was performed.

REFERENCES

[1] Jaguar. http://www.nccs.gov/computing-resources/jaguar/, 2010.

[2] National Institute for Computational Sciences. http://www.nics.tennessee.edu/computing-resources/kraken, 2010.

[3] Samer Al-Kiswany, Matei Ripeanu, and Sudharshan S. Vazhkudai. Aggregate memory as an intermediate checkpoint storage device. Technical Report 013521, Oak Ridge National Laboratory, Oak Ridge, TN, November 2008.

[4] Guillermo A. Alvarez, Elizabeth Borowsky, Susie Go, Theodore H. Romer, Ralph Becker-Szendy, Richard Golding, Arif Merchant, Mirjana Spasojevic, Alistair Veitch, and John Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Trans. Comput. Syst.*, 19:483–518, November 2001.

[5] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair Veitch. Hippodrome: Running circles around storage administration. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[6] Eric Anderson, Mahesh Kallahalla, Susan Spence, Ram Swaminathan, and Qian Wang. Ergastulum: Quickly finding near-optimal storage system designs. In *Technical Report HPLSSP 200105, Hewlett-Packard Laboratories,*, 2001.

[7] Microsoft Corporation. Microsoft windows ready-boost. http://www.microsoft.com/windows/windows-vista/features/readyboost.aspx.

[8] FUSE Development. Fuse: Filesystem in user space. http://sourceforge.net/apps/mediawiki/fuse/index.php.

[9] M. Meisner et al. Making the most of your ssd: a case for differentiated storage services. In *FAST, Work in Progress*, 2009.

[10] Jiahua He, Jeffrey Bennett, and Allan Snavely. Dash-io: an empirical study of flash-based io for hpc. In *Proceedings of the 2010 TeraGrid Conference*, TG '10, pages 10:1–10:8, New York, NY, USA, 2010. ACM.

[11] Intel. Intel x25-e extreme sata solid-state drive. http://www.intel.com/design/flash/nand/extreme/index.htm.

[12] Taeho Kgil, David Roberts, and Trevor Mudge. Improving nand flash based disk caches. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 327–338, Washington, DC, USA, 2008. IEEE Computer Society.

[13] Youngjae Kim, Aayush Gupta, and Bhuvan Urgaonkar. Mixedstore: An enterprise-scale storage system combining solid-state and hard disk drives. Technical Report CSE-08-017, Dept. of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, August 2008.

[14] Pavan Konanki and Ali R. Butt. An exploration of hybrid hard disk designs using an extensible simulator. In *Masters Thesis, Virginia Tech*, 2008.

[15] Min Li, Sudharshan S. Vazhkudai, Ali R. Butt, Fei Meng, Xiaosong Ma, Youngjae Kim, Christian Engelmann, and Galen M. Shipman. Functional partitioning to optimize end-to-end performance on many-core architectures. In *Proceedings of the 2010 ACM/IEEE conference on Supercomputing*, SC '10, 2010.

[16] Thanos Makatos, Yannis Klonatos, Manolis Marazakis, Michail D. Flouris, and Angelos Bilas. Using transparent compression to improve ssd-based i/o caches. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 1–14, New York, NY, USA, 2010. ACM.

[17] Jeanna Matthews, Sanjeev Trika, Debra Hensgen, Rick Coulson, and Knut Grimsrud. Intel turbo memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems. *Trans. Storage*, 4:4:1–4:24, May 2008.

[18] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to ssds: analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys '09, pages 145–158, New York, NY, USA, 2009. ACM.

[19] Michael L. Norman and Allan Snavely. Accelerating data-intensive science with gordon and dash. In *Proceedings of the 2010 TeraGrid Conference*, TG '10, pages 14:1–14:7, New York, NY, USA, 2010. ACM.

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

[21] Ruston Panabaker. Hybrid hard disk and readydrive technology: Improving performance and power for windows vista mobile pcs. http://www.microsoft.com/whdc/system/sysperf/accelerator.mspx.

[22] Stan Park and Kai Shen. A performance evaluation of scientific i/o workloads on flash-based ssds. In *Workshop on Interfaces and Architectures for Scientific Data Storage (IASDS'09)*, 2009.

[23] James S. Plank, Kai Li, and Michael A. Puening. Diskless checkpointing. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):972–986, October 1998.

[24] Hannes Prayer, Marco A.A. Sanvido, Zvonimir Z. Bandic, and Christoph M. Kirsch. Combo drive: Optimizing cost and performance in a heterogeneous storage device. In *First Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, March 2009.

[25] David Roberts, Taeho Kgil, and Trevor Mudge. Integrating nand flash devices onto servers. *Commun. ACM*, 52:98–103, April 2009.

[26] Mohit Saxena and Michael M. Swift. Flashvm: virtual memory management on flash. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 14–14, Berkeley, CA, USA, 2010. USENIX Association.

[27] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Extending ssd lifetimes with disk-based write caches. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, Berkeley, CA, USA, 2010. USENIX Association.

[28] John D. Strunk, Eno Thereska, Christos Faloutsos, and Gregory R. Ganger. Using utility to provision storage systems. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 21:1–21:16, Berkeley, CA, USA, 2008. USENIX Association.

[29] Top 500 Supercomputers. http://www.top500.org.

[30] TSUBAME2. http://www.gsic.titech.ac.jp/en/tsubame2.

[31] W.X. Wang, Z. Lin, W.M. Tang, W.W. Lee, S. Ethier, J.L.V. Lewandowski, G. Rewoldt, T.S. Hahm, and J. Manickam. Gyrokinetic simulation of global turbulent transport properties in tokamak experiments. *Physics of Plasmas*, 13, 2006.

[32] Fang Zheng, Hasan Abbasi, Ciprian Docan, Jay Lofstead, Qing Liu, Scott Klasky, Manish Parashar, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. Predata - preparatory data analytics on peta-scale machines. In *24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2010.