Group based energy adaptation for storage systems

Muthukumar Murugan
HP Storage[*]

Krishna Kant
Temple University

Ajaykrishna Raghavan, David H.C. Du
University of Minnesota

Abstract

Storage systems play a significant role in data centers and there is an urgent need to efficiently store, retrieve and manage the ever increasing volume of data required by a variety of applications in the data center. Much of the stored data often contains a lot of redundancies at the block level that can be removed via de-duplication. The performance and fault-tolerance requirements also need explicit replication of data, but more copies mean higher storage system energy consumption. In our previous work we proposed a flexible storage infrastructure called flexStore that can dynamically control the replication of de-duplicated data based on changing energy budgets for the storage subsystem. In this paper we extend this mechanism with storage policies that allow for differentiated treatment of various applications. In particular, we consider replication of virtual machines belonging to different application groups that are managed independently with respect to both de-duplication and replication. We have built a prototype of the storage system and evaluate the proposed system on an Amazon EC2 cluster. Through this prototype we study the benefits of group based replication both on storage node and on the host side in a data center.

1 Introduction

The storage needs of data centers is increasing at a fast pace and efficiently managing this explosive growth of data is extremely important from multiple perspectives including cost, performance, and power consumption. A large data center may host a diverse set of applications with different service level agreements (SLA's), QoS requirements, and data access characteristics. The storage system should be able to cater to the storage needs of these applications and deliver the desired performance and availability. For instance, the storage system should be able to guarantee low latency to a high priority application that serves real time queries as compared to a lower priority application running in the background, even if the former has less favorable data access characteristics than the latter (e.g., random vs. sequential data access pattern).

While performance is an important aspect of storage systems, the rapidly increasing data volume also requires a close attention to the power consumption. As the CPUs become more energy efficient and better power managed, the fraction of data center energy consumed by the storage continues to go up. Reference [7] suggests that the storage system power consumption may be around 40% of the total data center power consumption. Sustainability is also a key factor in the design of modern datacenters. Recently, there have been significant efforts by the industry to use renewable energy sources to power data centers (e.g., [4]). The use of renewable energy sources entails variability in available energy and the different subsystems in the datacenter have to be able to adapt to the energy variations. The goal of the adaptation techniques is to limit the power consumption of the datacenter during energy constrained situations while minimizing the impact on the guaranteed performance. Rightsizing of the power and cooling infrastructure – as opposed to the traditional overdesign – is also a crucial aspect of sustainability and leads to similar issues.

In our previous works [1, 6, 8], we introduced a new paradigm called Energy Adaptive Computing (EAC). The primary objective of EAC is to provide flexibility of operation by providing the right interfaces and mechanisms for the data center infrastructure to dynamically adapt to the variations in energy availability or variations in the ability to consume energy (due to power circuit, thermal or cooling limitations) while ensuring that the performance degrades minimally and gracefully according to specified QoS requirements.

In this paper we propose and demonstrate the working prototype of an energy adaptive storage system which dynamically adapts to the variations in available energy via group specific adaptive replication. Replication, i.e.,

[*]Work done as part of Center for Research in Intelligent Storage (CRIS)

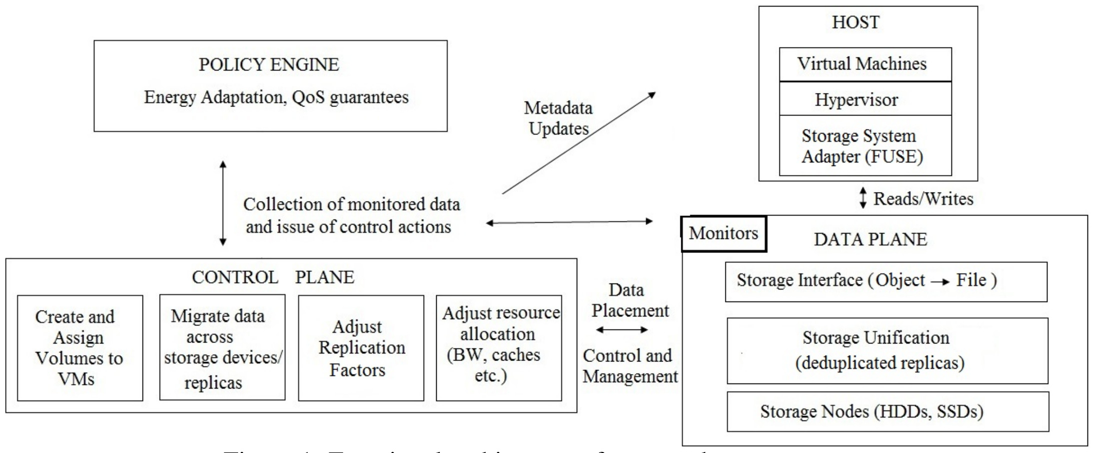
POLICY ENGINE
Energy Adaptation, QoS guarantees
Metadata Updates
HOST
Virtual Machines
Hypervisor
Storage System Adapter (FUSE)
Reads/Writes
Collection of monitored data and issue of control actions
Monitors
DATA PLANE
Storage Interface (Object → File)
Storage Unification (deduplicated replicas)
Storage Nodes (HDDs, SSDs)
CONTROL PLANE
Create and Assign Volumes to VMs
Migrate data across storage devices/ replicas
Adjust Replication Factors
Adjust resource allocation (BW, caches etc.)
Data Placement
Control and Management

Figure 1: Functional architecture of proposed storage system

maintaining multiple copies of data, is widely used in many scale–out storage systems (e.g., Network RAID 10 in [5]] both to improve performance and to increase resilience in the face of failures or data corruption. Most workloads tend to have a much higher percentage of read IO than write IO, and the read performance scales almost linearly with the number of replicas. However, multiple copies of the data implies that more storage nodes need to be kept powered on, which in turn consumes more energy. The proposed storage system dynamically controls the replication factors i.e., the number of replicas and the association of replicas to virtual machines, in order to adapt to the variations in available energy. The proposed storage system also provides interfaces to specify performance requirements of virtual machines and allows grouping of VMs based on their priority levels, dataset characteristics (e.g., de-duplication ratio), workload patterns, etc. The proposed storage system factors in these performance considerations in addition to the energy constraints when making the adaptation decisions. We demonstrate the adaptation mechanisms that the storage system provides and the performance of the storage system with a prototype implementation.

2 System Design and Architecture

Figure 1 shows the functional components of the proposed storage system. The virtual machines run on physical hosts on a hypervisor Xen (see www.xenproject.org). Each VM belongs to an application group. In this paper we group the application VMs based on (1) priorities and (2) de-duplication ratios. We evaluate the ability of the storage system to provide differentiated storage QoS to different application groups. The storage system maintains data in the de-duplicated format. That is, each VM image is divided up into blocks of equal size, called "chunks". Since VM images often have a lot of redundancy, many chunks have identical content which can be squeezed out by simply storing one copy of the chunk, and having others simply point to it. As an example, a study by Clements et al., [2] shows that in virtualized enterprise environments de-duplication can lead to up to 80% storage savings. For the purposes of this paper, we do de-duplication on a per-group basis. Following the de-duplication, we create multiple replicas of the entire chunk-set for the group. The number of replicas that an application group uses is dependent on predefined QoS and energy policies. In effect, the scheme eliminates uncontrolled redundancy and then reintroduces it in a controlled and adaptive fashion. Doing this on a group-basis is not as storage efficient as doing it across all VMs, but the independent management of each group provides flexibility in QoS management.

Since there is only one copy of any chunk in a deduplicated storage system, when multiple VMs try to access the same chunk, there is resource contention. Having multiple replicas improves performance because of the availability of more resources and also increases availability since when a replica fails, the data is available on other replicas. The replicas provide an object interface. The virtual machine disks are stored as files in the file system mounted on the host. We built a file system using FUSE (Filesystem in User Space, see http://fuse.sourceforge.net). The FUSE layer basically captures the POSIX calls (read, write etc.) and transforms them into object calls (GET, PUT etc.) which are then sent to the corresponding replica assigned to the VM. Each host mounts this file system and the virtual disks are stored at that mountpoint.

As shown in Figure 1 the design has separate "control" and "data" planes. The control plane decides what storage nodes or replicas the data is placed on or retrieved from. The data plane includes the storage nodes (replicas) where the data is actually read from and written to. This separation provides the capability to implement policy based adaptation mechanisms in the proposed storage system. Such a separation is an essential component of the emerging concept of software defined storage (SDS) (see [10]).

The policy engine is a distributed module that runs on each host, collects data from monitoring daemons on replica nodes, and initiates the control actions. The control actions include deciding how many replicas need to be kept powered on, assigning VMs to replicas based on the IO load on the replicas, and synchronizing across the replicas. Each VM normally reads from and writes to only one replica (the one it is assigned to). The only exception is the transition phase (during adaptation actions) when the VM is reassigned to a new replica in which case it writes to the new replica until its reassignment is complete. Hence there is no split–brain scenario where inconsistencies occur due to a single VM reading from two different copies of data.

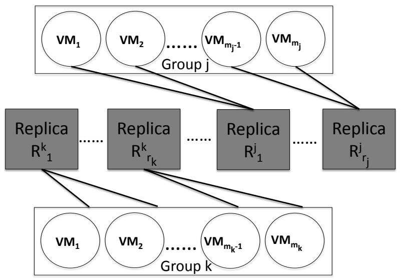

Figure 2: VM-Replica Association in each group

3 Energy Adaptive Replica Management

3.1 Dynamic Replication and Adaptive Consistency

Let the number of application groups be k. Let the total number of VMs in the i^{th} group, g_i be m_i. Each group is assigned a certain number of replicas. Let the number of replicas assigned to group g_i be r_i. The r^{th} replica of group g_i is denoted by R^i_r. Figure 2 shows the association of VM groups to replicas. We assume that each VM in a group does roughly the same number of IO's per second (IOPS), but the IOPS may vary significantly across groups. In this paper we measure the average IO latency across the entire group as the QoS measure. In real world, these assumptions could mean that all VMs in a group perform same or similar tasks and thus have similar behavior and handling requirements.

During energy plenty situations, we maintain at least the minimum number of replicas that are required to guarantee the latency requirements of each group. The VMs belonging to each group are assigned to the replicas in such a way that the load on each replica is about the same. As mentioned before, each VM reads from and writes to only one replica during normal operations. The replicas do diverge from one another as write operations generate new chunks in the system. Hence the replicas need to be synchronized periodically. Many synchronization protocols have been examined in the literature and they offer different levels of consistency in distributed systems. A *Weak Consistency* model refers to update only to the requested replica – other replicas are synchronized to it only when the VM is assigned to a different replica. A *Strong Consistency* model refers to immediate update of all the replicas so that they stay synchronized.

In flexStore, we adaptively perform replica synchronization in the background (by copying updated chunks across replicas) so that the impact on foreground read and write operations remains minimal. These background operations consume additional energy as well and hence need to be scheduled carefully. This adaptive consistency protocol, referred to as *flexStore consistency* protocol, is described in detail [9]. Briefly, the flexStore consistency protocol tries to minimize the number of chunks that are different between any two replicas of the same group of VMs. It does so under the given constraints of bandwidth and energy used for the replica synchronization.

During energy constrained situations, the replicas of lower priority VMs are progressively powered down and the VMs that were served by these replicas are reassigned to other replicas associated with the respective group to which they belong. This increases the load on the replicas serving the low priority VMs but the replicas serving higher priority VMs are not impacted until there is a large power constraint.

3.2 Application Groups, Flexibility and Storage Power

The replicas of each group of VMs are independent and hence having multiple application groups increases the flexibility of management. When the group size (i.e., the number of VMs per group) is small, the latency requirements of VMs can be managed at a finer granularity and the energy adaptation operations also can be done more efficiently within smaller groups. However if the number of VM groups is large, the storage resources required and hence the storage power consumption becomes high. In order to avoid having too many replicas in the system, we start with an original larger set of groups and then adaptively "combine" groups into one. As usual, the combined group is served by the same set of replicas and thus requires less storage and energy.

When the VMs are grouped based on the de-duplication potential (i.e., ability to reduce storage consumption via de-duplication), the number of groups significantly impacts the total de-duplication potential. Since de-duplication is restricted to within a group, the common chunks across different groups are not eliminated and hence the overall storage requirements can increase if we have many small groups. However, if the grouping is done intelligently in such a way that the de-duplication potential within a group is really high, the overall space requirements might actually decrease. Ideally, we need to have minimum number of groups of VMs and replicas where managing the QoS at the granularity of groups of VMs does not impact the QoS of individual VMs too much and also the deduplication potential within a group is high. This is a multi–dimensional optimization problem and we plan to address this in our future work.

3.3 Defining Policies

The policy engine shown in Figure 1 is the component that provides interfaces to define storage system policies. The QoS policies and energy policies are defined in the policy engine and are enforced in the control plane. An example QoS policy is as follows: if the IO latency of a VM is $> t$ ms, allocate more bandwidth for the VM to reduce the queuing delay. Similarly an example energy policy could be as follows: if the available power is $< P$ Watts, consolidate data onto a fewer nodes and shut down the rest of the nodes. The policy engine is designed as a distributed, logically centralized module and there is minimum interference with the regular read and write critical paths. It uses the resource usage collected and reported by performance monitoring daemons periodically. The policy engine enforces the required policies by communicating with the corresponding control plane entities. It dynamically adjusts (a) the allocation of VMs to the different storage systems to satisfy the performance characteristics of the VMs, and (b) the data layout (e.g.,) consolidation of data into a fewer devices, and (c) the number of replicas of the dataset based on energy availability. The policy engine also provides interfaces to associate a VM to an application group. The monitoring daemons, policy engine, and the adaptive consistency mechanisms help to achieve graceful degradation of performance during energy constrained situations.

4 Experimental Evaluation

We evaluated our storage system prototype on an Amazon EC2 cluster (http://aws.amazon.com/ec2/). Amazon instances are virtual machines that run on the Amazon servers. The different types of instances differ in the available CPU, memory, storage and networking capacities. Our experimental testbed comprises general purpose M1 type instances. We used EC2 M1.xlarge instances that have 4 vCPUs (virtual CPUs) with maximum of 15 GB of RAM as the storage nodes (i.e replicas). In all the experiments we set the amount of memory available on the storage nodes to 2 GB (by setting a boot time option in GRUB), to ensure that not all IOs are served from buffers and there is disk activity. We also used Amazon EC2 M1.large instances that have 2 vCPUs with maximum of 7.5 GB of RAM as hosts. All EC2 instances ran Ubuntu Server 14.04 as the operating system. We ran a maximum of 4 VMs on each of the M1.large instances (hosts). In order to evaluate our prototype storage system, we used *fio* [3] which is a standard benchmark to evaluate file system workloads. We used uniform access distribution in fio. Each of the virtual disks was assumed to belong to one Virtual Machine in a datacenter.

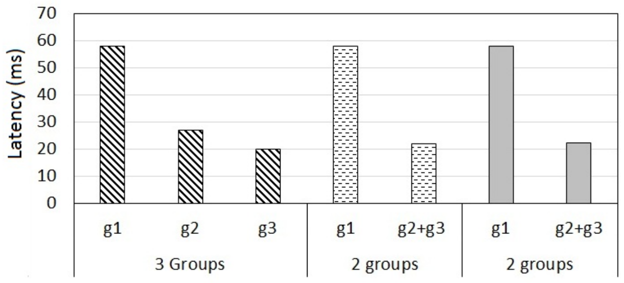

Figure 3: Latency values with different number of groups Case(1): Three groups with 25% de-duplication Case (2): Two groups each with 25% and 50% de-duplication respectively and Case (3) Two groups each with 25% and 75% de-duplication respectively

We studied a case with 32 VMs in three groups: g_1 with 22 VMs, g_2 with 6 VMs, and g_3 with 4 VMs respectively. The groups had the following latency requirements: g_1 required 60 ms latency bound, g_2 required 30 ms bound, and g_3 25 ms. We started by fixing the *de-duplication ratio* of each group to 25% (Case (1) in Figure 3). Here de-duplication ratio is defined simply as the fraction of bytes eliminated (i.e., not stored explicitly) as a result of deduplication. We controlled the deduplication ratio by changing the files that get stored in the data disks of each VM. We ran read, write, and mixed read/write workloads following uniform distribution of block numbers (average request size = 4 KB) accessed and measured the average request latency for different replica assignments to the groups, until we arrived at an assignment that satisfied the latency requirements.

We observed that when g_1 was assigned 3 replicas R_1^1, R_1^2, and R_1^3, g_2 to two replicas, R_2^1 and R_2^2 , and g_3 was assigned to one replica R_3^1, the storage system was able to provide the guaranteed latency to the VMs.

Next we combined g_2 and g_3 into a single group, and the de-duplication ratio of this combined group was set to 50% (Case (2) in Figure 3). Since VMs in g_3 require latency bound of 25 ms, we set that as the required latency bound for this new combined group. The de-duplication ratio of VMs in g_1 was still 25%. The groups were assigned replicas as follows to meet the latency bounds: g_1 was assigned R_1^1, R_1^2, and R_1^3 and the combination of g_2 and g_3 was assigned $R_{(2,3)}^1$ and $R_{(2,3)}^2$. We found that this configuration was able to meet the latency requirements under different workloads at reduced storage resources as shown in Case (2) of Figure 3.

In Case (3) we repeated the above experiment but now the de-duplication ratio of the combined group g_2 and g_3 was set to 75%. In this case, g_1 still required 3 replicas while the combination of g_2 and g_3 required only one replica. It can be seen that with higher de-duplication ratio, the available memory in the storage nodes was sufficient to cache most pages during reads and hence fewer number of replicas were adequate to satisfy the latency requirements of the VMs. The impact on latency and hence the number of replicas required to satisfy latency requirements will depend on the combination of VM groups which are assigned to use the same replicas. Generalized and effective algorithms for dynamically merging and separating groups can be quite challenging and will be addressed in future works.

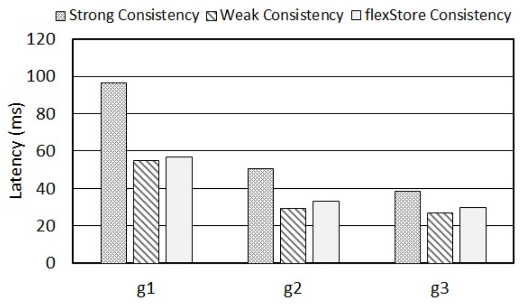

Figure 4: IO latencies with different consistency levels across replicas

In the next two experiments (Figures 4 and 5), we changed the number of VMs per group to show the impact of consistency model on the IO latencies. Here g_1 has 18 VMs, g_2 has 12 VMs, and g_3 has 6 VMs. The de-duplication ratio of each group was set to 25%. The groups had the following latency requirements: g_1 required 60 ms latency bound, g_2 required 35 ms bound, and g_3 30 ms. We observed that when g_1 was assigned 3 replicas R_1^1, R_1^2, and R_1^3, the storage system was able to provide the guaranteed latency to the VMs in g_1 for *flexStore consistency* as shown in Figure 4. Similarly when g_2 was assigned three replicas, R_2^1, R_2^2 and R_2^3, and g_3 was assigned two replicas, R_3^1, R_3^2 to meet the latency requirements for *flexStore consistency*.

We changed the consistency level to *strong consistency* and found that these latency bounds could not be honored for any group for the above mentioned replica assignment. The average latencies increased by at least 30% in case of *strong consistency*. Since both read and write operations contend for common resources, i.e memory, disk/network bandwidth, etc., read latencies are also impacted in case of *strong consistency*. In case of *weak consistency* and *flexStore consistency*, writes return immediately after being written to one replica in the group and hence write latencies are practically the same for these two cases.

In our energy adaptive storage system, the VMs in a group are assigned to replicas based on a greedy policy that balances the number of IOPS of all the replicas for that group that are powered on. VMs are re-assigned to another replica when: (1) number of IOPS is not balanced, (2) latency values prescribed by the policy is not met, and (3) when a replica needs to be turned off during energy deficient periods. In the case of *strong consistency* the re-assignment can happen immediately since all replicas in a group have the required data. In the case of *weak consistency*, the replicas have to be synchronized when a VM is re-assigned from one replica to another, which can potentially take a long time and hence require more brown energy during energy deficient periods. In the case of *flexStore*, the replica synchronization happens periodically once a threshold amount of new data has been written to the replica. This ensures that the amount of data (hence the time and the amount of brown energy) that needs to be synchronized during reassignment is bounded. Our storage system performs this periodic synchronization in a manner that the foreground (i.e client) IOs aren't significantly impacted. The adaptive consistency model helps balance the tradeoff between the penalty on the foreground IO latency due to replication and the time required to perform re-assignment.

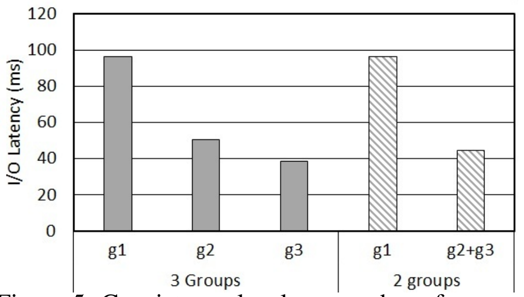

Figure 5: Consistency level vs. number of groups

Figure 5 shows the impact of de-duplication on latencies for *strong consistency*. We started with the same group and replica setup as the previous experiment. Next we combined g_2 and g_3 into one single group with a de-duplication ratio of 50%. Since VMs in g_3 require latency bound of 30 ms, we set that as the required latency bound for this new combined group. The de-duplication ratio of VMs in g_1 was still 25%. The groups were assigned replicas as follows to meet the latency bounds for *flexStore consistency*: g_1 was assigned R_1^1, R_1^2, and R_1^3, and the combination of g_2 and g_3 was assigned $R_{(2,3)}^1$ and $R_{(2,3)}^2$. We found that this configuration was able to meet the latency requirements under *flexStore consistency*. We changed the consistency level to *strong consistency* and measured the average IO latencies. We observed that the IO latency for the combined group g_2 and g_3 for *strong consistency* was less than the IO latency of g_2 (by 12%). This is because only 2 replicas were required to serve the combined group due to higher de-duplication ratio and hence the synchronization overhead was low, as opposed to 3 required to serve g_2 previously. The latency values for the combined group are higher than that of g_3 by 16% with *strong consistency*, when it was handled as a separate group, since now there are more VMs in the combined group, resulting in higher contention for shared resources. The IO latency of the combined group was nearly the same as the average of the IO latency for g_2 and g_3. Here we see that by combining the groups to achieve higher de-duplication ratio, we are able to provide similar performance levels even in case of *strong consistency* with fewer replicas. However, the QoS is managed at a coarser granularity and hence some groups may be impacted more than the others.

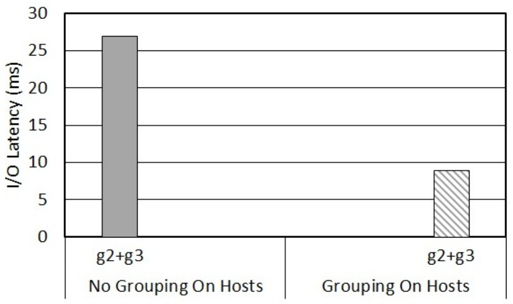

Figure 6: Impact of grouping VMs on the host side

The experiments thus far have explored grouping of VMs on the storage node side. We motivate the need for grouping of VMs on the host side with this next experiment. We started with same group and replica setup as the previous experiment. The VMs belonging to the group g_1 and the combined group g_2 and g_3 (with 50% de-duplication ratio) were distributed among the different hosts. We ran a sequential read workload on VMs belonging to the combined group g_2 and g_3 and measured the IO latencies. Next we co-located the VMs belonging to the combined group g_2 and g_3 on the hosts and ran the sequential read workload on the VMs and measured the IO latencies. We found that there was a reduction in latencies (by around 67%) in the second case. This is because in the second case the read ahead performed by the OS helps populate the read cache on the hosts. Since the co-located VMs belonging to the combined group g_2 and g_3 contain common data, the read requests from different VMs can be served from the host's read cache. Even though the host OS performs read ahead with or without co-location, non co-located VMs do not get the benefit of a warmed up cache. This suggests that fewer replicas could be used to meet a latency bound if VM grouping on the hosts could also be controlled. In future work we plan to explore techniques to group VMs dynamically on the host side depending on the workload to minimize energy required to meet latency requirements of VMs.

5 Conclusion and Future Work

In this paper we presented the extension to our flexible storage infrastructure in [9] that allows specification of storage policies for differentiated treatment of various applications. Through our evaluations we illustrated the benefits of group based de-duplication and replica management. We also motivated the advantages of grouping VMs on the hosts themselves. In the future we plan to explore dynamically changing this combination of groups which use the same replicas both on the host and storage node side.

A possible area to be investigated is the impact of the storage system adaptations on the resource allocations on the host side. For instance if the number of replicas is smaller, the VMs spend more time waiting and hence the CPUs of the hosts remain idle for longer periods. This might provide additional opportunities to do power management in the hosts as well. Defining policies and allocating storage resources in the presence of heterogeneous storage media like hard disks and SSDs is a future extension of this work.

References

[1] K. Kant, M. Murugan and D. H. C. Du . Enhancing data center sustainability through energy adaptive computing. *ACM JETC (Special Issue)* (April 2012).

[2] Clements, A. T., Ahmad, I., Vilayannur, M., and Li, J. Decentralized deduplication in san cluster file systems. In *USENIX Annual technical conference (ATC)* (2009).

[3] Flexible IO Tester. http://git.kernel.dk/?p=fio.git;a=summary.

[4] Google Green Energy Initiative. `http://www.google.com/about/datacenters/energy.html`.

[5] HP StoreVirtual. http://www8.hp.com/us/en/products/data-storage/data-storage-products.html?compURI=1225885.

[6] Kant, K., Murugan, M., and H.C.Du, D. Willow: A Control System for Energy and Thermal Adaptive Computing. In *IPDPS '11* (2011).

[7] Li, Z., Greenan, K. M., Leung, A. W., and Zadok, E. Power consumption in enterprise-scale backup storage systems. In *FAST'12* (2012).

[8] M.Murugan, K.Kant, and Du, D. Energy Adaptation for Multi-tiered Datacenter Applications. *Intel Technology Journal 16* (2012).

[9] Murugan, M., Kant, K., Raghavan, A., and Du, D. flexstore: A software defined, energy adaptive distributed storage framework. In *IEEE MASCOTS 2014* ([To Appear]).

[10] The Software-Defined Data Center (VMWare). http://www.vmware.com/software-defined-datacenter/storage.