Techniques for an Energy Aware Parallel Storage System

Cengiz Karakoyunlu, *Member, IEEE,* and John Chandy, *Member, IEEE*

Abstract—The decreasing cost of disk drives has made it possible to construct large-scale storage systems at a reasonable cost while achieving high performance for I/O demanding applications by using parallel storage architectures. Although these systems are primarily concerned with performance, energy consumption has also become an important problem since it can cancel out any performance gains. There have been several approaches both at the hardware and software level to deal with the energy consumption problem. However; these solutions suffer from issues such as being workload-dependent or requiring large data migrations. Assuming that a certain number of users usually max out the available bandwidth in a storage system, we propose restricting the I/O load to subsets of the storage system nodes in order to address the energy consumption problem. Specifically, different subsets of the storage system nodes can be assigned to each user based on a certain identifier (i.e. user id, source or destination address) and the remaining nodes can be switched to low-energy modes to save energy. We introduce and evaluate our techniques for an energy-aware parallel storage system in order to understand how energy efficiency can be improved at the storage system level while maintaining uniform system utilization.

Index Terms—Distributed file systems, Energy-aware systems, Power Management

1 Introduction

The cost of disk arrays has decreased dramatically in the last decade and many applications today use large-scale storage systems that consist of a large number of disk arrays. The primary purpose of these systems is to achieve high performance by exploiting the computing capabilities of individual nodes with parallel file systems [7], [15], [16], [17] that stripe data across storage servers. While the performance is the primary concern for many users, energy efficiency has also attracted considerable interest from storage system developers in recent years since the increasing energy consumption and maintenance costs of storage systems have started to limit any possible performance gains with increasing scale. As an example, a single high-performance 300-Watt server consumes 2628 KWh of energy per year and the total estimated energy cost of this server would be $338 per year [6], [5]. The cost of a data-center rack full of these servers would be $22,800 over ten years [5]. Therefore, it is critical to develop energy-efficient data storage and computation techniques in large-scale storage systems, in order to meet the demands of increasing number of users.

Previous research on energy saving techniques concentrated on either hardware level or software level solutions; but they were usually not at the storage system level and they also suffered from issues such as being workload-dependent or migrating big chunks of data between I/O nodes. In this work, we propose algorithms to have an energy-aware parallel storage system while at the same time having uniform system utilization. In particular, our work is driven by two key assumptions: first, the system suffers from incasting, and second a very small subset of users are actually using the system at any one time.

Incasting is a condition that occurs because of queue limitations in most network switches [13]. As a result of this incasting behavior, there is a limit to the number of storage servers across which data can be striped. Beyond this limit, I/O bandwidth no longer scales and in fact deteriorates. If m represents the number of nodes at which the performance maxes out, from a performance point of view, there is no point in using more than m nodes to increase parallelism. Ideally, we would only have m I/O nodes in the system. However, for storage capacity reasons, we may need more than m nodes. In such systems, we would need to keep these extra nodes active and thus waste energy since the network resources are maxed-out by a *subset* of the I/O nodes.

To save energy, one could turn off these extra nodes, and activate them when necessary. The difficulty comes when trying to identify when to turn on and off the various I/O nodes. Our approach is to distribute users across the I/O nodes, such that each user is allocated only m nodes - i.e. the limit at which performance is maxed out. If only one, or few users, are active at any one time, we can effectively turn off the other I/O nodes in the system. In most HPC systems, this assumption is generally true. Considering the HPC workloads we use to test our work, for the Hornet Cluster [2], on average only 11 of 203 users were active at any one time. Similarly, for the CTC IBM-SP2 Cluster [3], on average only 23 of 679 users were active at any one time. The

• *The authors are with the Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT, 06269.*
Email: {cengiz.k, john.chandy}@uconn.edu

same assumption is also true for the NFS workload we use to test our work. For the LAIR62 NFS workload [4], on average only 1 of 81 users was active at any one time. In an HPC system, because of the job structure, we can effectively turn on and off I/O nodes based on the user job submissions since we know a priori which m nodes are going to be used at any time. For a more general storage system, such as a network file system, this is not true since we do not know when a user will be using the system. However, users tend to use storage in "sessions" - i.e. use it for a short time and not use it again for a long period.

The challenge becomes how to assign users a subset of I/O nodes in order to reduce the energy footprint, while at the same time trying to preserve uniform distribution of the system resources. As an example, if we assume a storage system consists of n I/O nodes and m of them max out the incast bandwidth ($n > m$), then each user can be assigned a separate subset of m I/O nodes based on a certain identifier (i.e. user id, source or destination address) and any I/O node that is not assigned to any user at that time can be switched to low-energy modes. In this paper, we present two different approaches (sequential and distributed sequential) to map subsets of I/O nodes to different users and show their outcomes with extensive simulations using real-world workloads. These approaches are modified versions of the techniques we presented in our earlier work [1].

The rest of this paper is organized as follows; Section 2 gives a brief overview of the existing energy saving methods in the literature. Section 3 describes how the different node subset assignment methods have been implemented. Section 5 describes the evaluation environment for these methods and also gives the results of the performance and energy consumption tests. Finally, we present conclusions and possible future work in Section 6.

2 Related Work

There have been many approaches to save energy in large-scale storage systems. Massive Array of Idle Disks [9] (MAID) starts with the fact that 50% of the data in large-scale storage systems is never accessed again after being stored [12] and introduces an energy-efficient data storage technique that has acceptable data access performance. MAID separates the nodes in the storage system into two groups; *active* and *passive* nodes. Active nodes are kept on regardless of the I/O load associated with them. Whenever new client requests arrive, virtualization and cache managers forward these requests to active nodes. If the active nodes cannot satisfy the I/O requests, these requests are forwarded to the passive nodes. The disadvantage of MAID is the variation in its efficiency depending on the cache characteristics and workload of the system.

Popular Data Concentration [14] (PDC) is another energy saving method for large-scale storage systems. PDC dynamically migrates frequently accessed data to a subset of the nodes in the storage system, called *active* nodes. Active nodes may change over time as the clients manipulate the system and data access frequency varies. In order to migrate popular data to a subset of available nodes, PDC needs to predict the future load for each node at certain time intervals. If a node in the subset of active nodes is running close to its bandwidth, PDC will migrate the data of that node to the others in the subset. Therefore, PDC methods should be launched periodically in a system. PDC performs close to MAID for small workloads and it provides more consistent energy savings for large workloads. The disadvantage of PDC is the overhead due to data migration and load prediction.

Ganesh et al. [10] has shown that a filesystem which was not originally developed to reduce energy consumption, can actually be used for this purpose due to its I/O characteristics. All write requests in the Log Structured Filesystem (LFS) are forwarded to the log head on the client side; therefore it is possible to know where a write request wants to go before it is even transferred to the communication network between the server and the client. Performance tests on LFS showed that it can save energy if a proper read cache exists in the system; however the LFS log is rapidly filled up with the requests and the overhead of cleaning the log cancels out any potential energy savings.

3 Energy Aware File System Techniques

In order to tackle the energy efficiency problem of the large-scale storage systems, we propose moving I/O load to subsets of the nodes in the storage system. As described earlier, we are assuming a use scenario where a subset of the I/O nodes max out the available bandwidth of the system. Therefore, the problem we are trying to solve is how to map subsets of the I/O nodes to users. Since user utilization is well-defined in large-scale storage systems, we retrieve a random identifier (i.e. user id, source or destination address) from the user and a subset of the I/O nodes are assigned to that user based on this identifier.

In this work, we propose two different methods to assign subsets of the available I/O nodes to users; *sequential* and *distributed*. The energy-aware node assignment methods proposed here are designed to work for a traditional parallel storage system architecture, but could also work in a disk array. The node assignment methods try to save energy by reserving a subset of the nodes for each user while at the same time distributing system resources uniformly.

3.1 Subset Assignment

3.1.1 Sequential Assignment

Assuming n is the total number of I/O nodes in the system and m is the number of I/O nodes to be assigned

to a user, the sequential method first calculates an offset value for a given user identifier. If we denote this offset by p and the user identifier by u, then $p \equiv u \bmod n$. The next m I/O nodes following the p^{th} node are assigned to the user. The offset calculated in the the sequential method may be close to the last I/O node in the system; such that when m I/O nodes are assigned to a user, we may run out of I/O nodes in the system. In this case the node assignment continues with the first I/O node. Figure 1 shows sequential I/O node assignment procedure for a user with user id 508 with a total number of five nodes in the system, where three of them are assigned to each user.

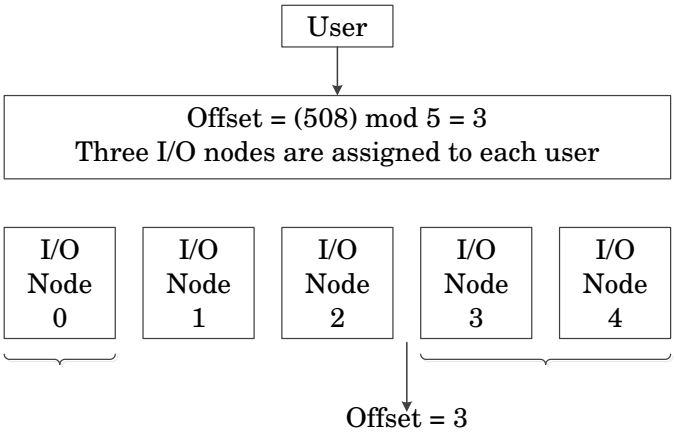

Fig. 1. Sequential node assignment where user id is 508, total number of I/O nodes is 5 and the number of nodes assigned to each user is 3

3.1.2 Distributed Assignment

Distributed node assignment method assigns the subsets of the I/O nodes to users, while at the same time trying to distribute the storage utilization uniformly. When the sequential assignment method is used, several users might be assigned a limited range of the I/O nodes in the storage system, depending on the values of the user identifiers. While this would increase the number of nodes that need to be turned off and bring down the energy consumption; the limited range of the I/O nodes might be overloaded with the user accesses. Distributed assignment method takes uniform system utilization into account by trying to use all I/O nodes in the system equally, while at the same time trying to maximize the energy savings.

In order to accomplish uniform distribution and low energy consumption goals; distributed assignment method creates *groups* of I/O nodes in the storage system. Assuming m is the number of I/O nodes to be assigned to each user, each group consists of m I/O nodes that follow each other sequentially in the storage system. In this case, once a user is assigned a group; that user is automatically assigned m I/O nodes. If there are n I/O nodes in the storage system; then there will be nm groups available.

In order to choose a group for a user, two different methods are followed. The first method calculates the modulus of the user identifier with respect to the number of groups. If we denote the user identifier by u, the number of groups by ng and the group to choose by g; then $g \equiv u \bmod ng$ in the first method. The second methods keeps track of the number of accesses in each group and chooses the one with the smallest number of accesses. Each group can be assigned to different number of users and each of these users can have different number of accesses to the system. The second method keeps track of the number of accesses in each group dynamically and assigns a user the group with the smallest number of accesses, when that user comes to the storage system for the first time. If we denote the number of accesses in a group by na and the number of groups by ng, then $g \equiv min(na_1, na_2, ..., na_n g)$. Figure 2 shows the procedure of distributed I/O node assignment by user identifiers and Figure 3 shows the procedure of distributed I/O node assignment by number of accesses, for a user with user id 508 where there are six nodes in the system and each user is assigned two of them.

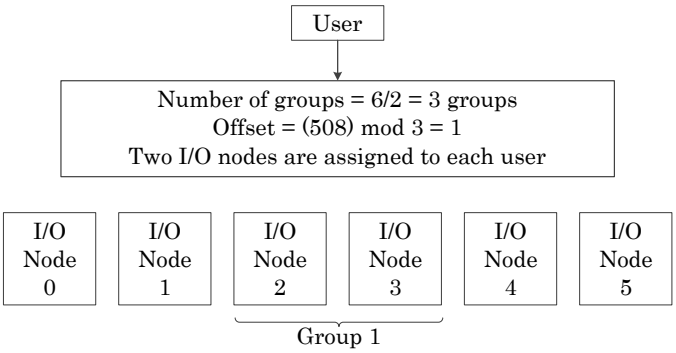

Fig. 2. Distributed node assignment by user identifiers where user id is 508, total number of I/O nodes is 6 and the number of nodes assigned to each user is 2

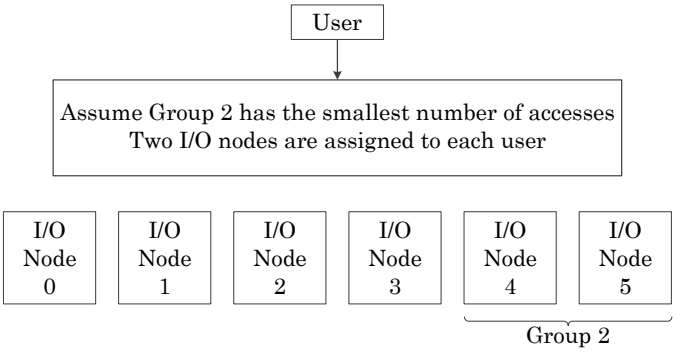

Fig. 3. Distributed node assignment by number of accesses where user id is 508, total number of I/O nodes is 6 and the number of nodes assigned to each user is 2

3.2 Inactivity Threshold

The assignment methods we are proposing try to save as much energy as possible from all the nodes in the storage system. In this respect, if a node is not assigned to any user at all; then it will always stay in a low-energy mode. On the other hand, if a node is assigned to a user; that node will eventually switch to a low-energy mode after all the jobs running on that node are completed. In an HPC system, the completion time of jobs can be predicted; thus, we can decide when to switch an I/O node to a low-energy mode. However, if we consider more general storage systems, we need to have a condition to decide when to switch a node to a low-energy mode. We define this metric as the *inactivity threshold*, which can be defined as *the period of time a node continues to work at full capacity after the completion of the most recent job*. This means that, once a node stays inactive for longer than the inactivity threshold, it can be switched to a low-energy mode.

In our work, we define the low-energy mode as the state where a node is completely turned off. However, modern hard drives have the ability to operate at various speeds [11]. Thus, in order to conserve energy it is not mandatory to completely turn off a node. Depending on how much energy saving is demanded by the user, the sequential and distributed assignments can go into low-speed modes without being turned off. We assume the worst case and turn off a node when it is in a low-energy mode. When a node is turned off, the energy conservation is at its maximum.

If a user is assigned a turned-off node as a result of the sequential or distributed assignment methods, that node needs to start operating at full capacity again. In this case, the user can not immediately run jobs on this node since it will take some time for the node to run at full capacity. We define the time it takes to start up a completely turned-off node as the *startup time*. If a node has been inactive between two jobs for longer than the inactivity threshold value, then the next job arriving to that node will have to wait for the node to start up before being executed. As an example, when there is no inactivity threshold (I/O node runs constantly), three consecutive jobs (Job A, Job B and Job C) would be executed as shown in Figure 4. If we take the inactivity threshold and the startup time into account, these jobs would be executed as shown in Figure 5; where the inactivity threshold is 20 seconds and the startup time is 30 seconds.

As we can see in Figure 5, the inactivity threshold (20 seconds) is exceeded before Job B arrives. As a result, Job B waits for a period of time equal to startup time (30 seconds) before being executed. The inactivity threshold is not exceeded between Job B and Job C and the I/O node these jobs are running on is not switched to turn-off mode. If we consider the total time the I/O node stays on, we can see the case with the inactivity threshold and the startup time consumes less energy. The inactivity threshold and the startup time are good indicators of how a real cluster is going to be affected by the node assignment techniques we are proposing.

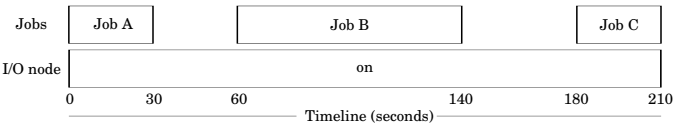

Fig. 4. Regular case with no thresholds

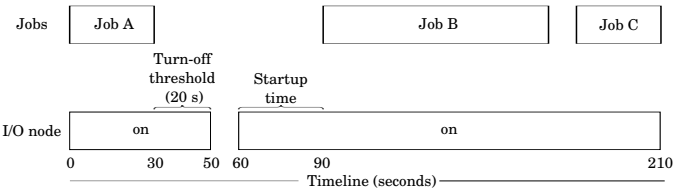

Fig. 5. Startup time (30 s) due to starting *Job B* on a node that has stayed turned-off longer than the turn-off threshold (20 s) after *Job A* was completed

3.3 Overlapping Executions

In a large-scale storage system, each I/O node may be used by multiple concurrent users for different I/O operations. Our node assignment implementation might assign a node to multiple users depending on the result of the sequential or distributed assignment method. Therefore, concurrent users can simultaneously run their jobs on an I/O node. When the execution of two or more jobs overlap on a node, we assume that node still consumes the same energy per unit time [18]. This increases energy conservation since time spent per job goes down.

4 Analytical Model

In this section, we present an analytical model that can be useful to predict the behavior of the approaches under various loads. The main parameter of the analytical model is the *probability of exceeding the inactivity threshold*. In order to calculate this probability, we need to know the cumulative distribution function (cdf) of the interarrival times in a given workload. This goal can be achieved by fitting the interarrival times of the given workload to a data distribution model. We have done so by using the *allfitdist* function [**?**] in MATLAB.

Using a workload representative of an NFS file server [4], we were able to fit it to a data distribution using MATLAB, and we concluded that Generalized Pareto Distribution is the best data model to represent our data. The cumulative distribution function of the Generalized Pareto Distribution is given below, where t is the random variable.

$$F(t) = 1 - (1 + k\frac{(t-\theta)}{\sigma})^{-1/k} \quad (1)$$

By definiton of the cumulative distribution function, $F(\tau)$ represents the probability that the random variable t is smaller than or equal to the inactivity threshold value (represented by τ) of a workload. Since we are interested in estimating the probability of exceeding the inactivity threshold, we need to formulate the probability that the random variable t is greater than the threshold value (τ). If we denote this probability by P_τ;

$$P_\tau = 1 - F(\tau) = (1 + k\frac{(\tau - \theta)}{\sigma})^{-1/k} \tag{2}$$

Now that we know the probability of the random variable t being greater than the threshold value τ, the next parameter we are interested in calculating is the average transaction service time in a workload. The probability that a transaction will experience a startup delay because of exceeding the inactivity threshold is given by P_τ. Therefore, we can calculate the average transaction service time as shown below, where μ is the average transaction time without considering any penalties, t_s is the startup time spent due to exceeding inactivity threshold τ and $\mu^{'}$ is the average transaction service time by taking the startup delays into account.

$$\begin{aligned} \mu^{'} &= (1 - P_\tau)\mu + P_\tau(\mu + delay \;\; per \;\; transaction) \\ &= (1 - P_\tau)\mu + P_\tau(\mu + \frac{t_s}{2}) \end{aligned} \tag{3}$$

It is important to show here how *delay per transaction* is derived. There may be multiple transactions submitted during the startup period and each transaction will experience a different length of startup delay depending on the time of submission. In this case, in order to estimate delay per transaction, we need to use the interarrival rate, λ. Since λ transactions arrive per second, then λt_s transactions will arrive during the startup period, t_s, with each transaction arriving $\frac{1}{\lambda}$ seconds after the previous one. In this case, the first transaction during the startup period, will experience a startup delay of $\frac{\lambda t_s}{\lambda}$ seconds; whereas the last transaction during the startup period will experience a startup delay of $\frac{1}{\lambda}$ seconds. Therefore, the total startup delay for λt_s transactions during the startup period will be given by;

$$\begin{aligned} Total \;\; startup \;\; delay &= \frac{1}{\lambda} + \frac{2}{\lambda} + ... + \frac{\lambda t_s}{\lambda} \\ &= \frac{(\lambda t_s)(\lambda t_s + 1)}{2\lambda} \simeq \frac{(\lambda t_s^2)}{2} \end{aligned} \tag{4}$$

If we divide the total startup delay by the number of transactions that arrive during the startup period (λt_s), we can find delay per transaction as given below;

$$Delay \;\; per \;\; transaction = \frac{\frac{(\lambda t_s^2)}{2}}{\lambda t_s} = \frac{t_s}{2} \tag{5}$$

Now, if we assume the total number of non-overlapping transactions in the workload is N_e, then the total time the system stays on will be given by;

$$System \;\; on \;\; time = N_e \mu^{'} \tag{6}$$

Finally we can find the system utilization with the equation below.

$$Utilization = System \;\; on \;\; time / Total \;\; time \tag{7}$$

5 Results

5.1 Experimental Setup

We conduct the experimental evaluations on the proposed node assignment methods (*sequential* and *distributed*) by using three different real-world workloads; workload from Hornet Cluster at the University of Connecticut [2], parallel workload from IBM-SP2 Cluster at Cornell Theory Center(CTC) [3] and NFS workload of a research workload from a university computer science department [4].

5.1.1 Workloads

5.1.1.1 Hornet Cluster: The Hornet Cluster is a high end cluster consisting of 64 nodes where each node has 12 Intel Xeon X5650 Westmere cores, 48 GB of RAM and 500 GB of storage. Users submit jobs on this cluster and each job is recorded in the log files. For each job, the log files provide the identifier of the user who submitted the job, in addition to the submission and completion times. Since the initialization of this cluster in August 2011, there have been around 40000 job submissions.

5.1.1.2 CTC IBM-SP2 Cluster: This workload contains parallel job information for the 512-node IBM-SP2 Cluster at Cornell Theory Center(CTC). The workload includes jobs submitted between July 1996 and May 1997 and for each job, it provides the identifier of the user who submitted the job, in addition to the submission and completion times.

5.1.1.3 LAIR62 NFS Workload: This workload includes the NFS transactions for the first nine days of the *LAIR62* workload, a research workload from a university computer science department. The workload has information about the submission and completion times of each NFS transaction, in addition to the source and destination addresses and type of each transaction (response or call).

5.1.2 Test Procedure

The first step of each test case is to retrieve the necessary information from each workload. A script that goes through all the workloads parses the user identifiers and the submission and completion times of each job in the Hornet Cluster or CTC IBM-SP2 Cluster. For the LAIR62 NFS workload, the script parses the submission and completion times of each transaction, in addition to the source and destination addresses of each transaction since the user identifier is not available.

Once the necessary information is parsed from the workloads, a simulation script starts assigning users

Test Parameter	Values
Total number of nodes	16, 32 or 64
Number of nodes assigned to each user	2, 4 or 8
Assignment methods	Sequential or Distributed
Low-energy mode	Turn Off
Inactivity threshold	Varied between 1000 and 2000000 s
Startup time	10, 60, 120 or 240 s
Power consumption per node	167, 300 W

TABLE 1
Test Parameters

to I/O nodes using the sequential or distributed node assignment methods. It is possible to change the total number of nodes, the number of nodes assigned to each user, the assignment method, the inactivity threshold and the startup time in this simulation script for various evaluation scenarios.

Once the user assignment to I/O nodes is completed, the simulation script checks whether there is any overlap between the jobs or transactions executed on an I/O node. Since we would like to know how long each I/O node is going to stay on or off, it is important to handle the overlaps between jobs or transactions. Assume Job1 and Job2 are assigned to I/O server A. If Job1 starts at time $d1$ and finishes at $d2$, and if Job2 starts at $d3$ and finishes $d4$, these two jobs will overlap in time when $d1 < d3$ and $d3 < d2$. Thus, the total time that I/O node A is kept on will be ($d2$ - $d1$) + ($d4$ - $d3$) - $overlaptime$.

Finally the simulation script calculates the output information in order to understand how well the system is doing in terms of the energy consumption and system utilization. The simulation finds the energy consumption and system utilization for various values of the inactivity threshold, startup time, total number of nodes and number of nodes assigned to each user as shown in Table 1. Power consumption per node is 167 Watts [8] for LAIR62 NFS workload and 300 Watts [6] for Hornet and CTC IBM-SP2 Clusters.

5.2 Energy consumption

We first evaluate the total energy consumption of the node assignment methods (sequential and distributed) by changing the total number of nodes (16, 32, 64) and the number of nodes assigned to each user (2, 4, 8) in the system. The total energy consumption values for different workloads are shown in Figures 6, 7 and 8. As we can observe in these figures, the sequential assignment method is almost the same as distributed assignment by user identifiers in terms of the total energy consumption.

Figure 6 shows that the total energy consumption of Hornet Cluster decreases when distributed assignment by number of accesses is used. In Hornet Cluster, there is a high possibility of concurrent execution between different users and some users have a lot more accesses to the system compared to others. Therefore, when distributed assignment by number of accesses is used, the

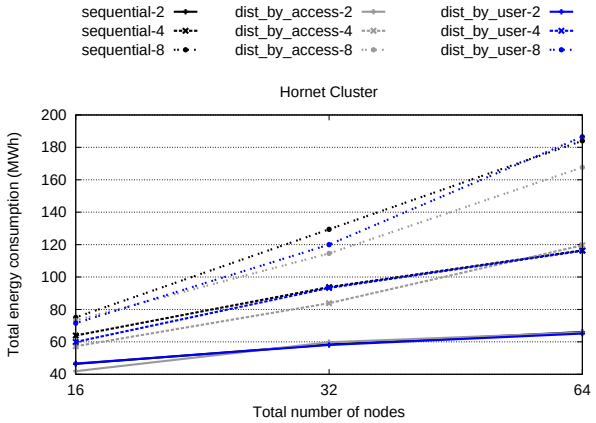

Fig. 6. Total energy consumption for Hornet Cluster

number of users in different groups vary significantly. As a result, there is a high possibility of concurrent execution (overlap) in certain groups and this fact causes the total energy consumption to go down. Without using our approaches, the Hornet Cluster would stay on all the time and it would spend 88 MWh of energy with 16 nodes, 176 MWh of energy with 32 nodes and 352 MWh of energy with 64 nodes. Looking at the numbers in Figure 6, our approaches are able to decrease the energy consumption in the stock case by up to 82%.

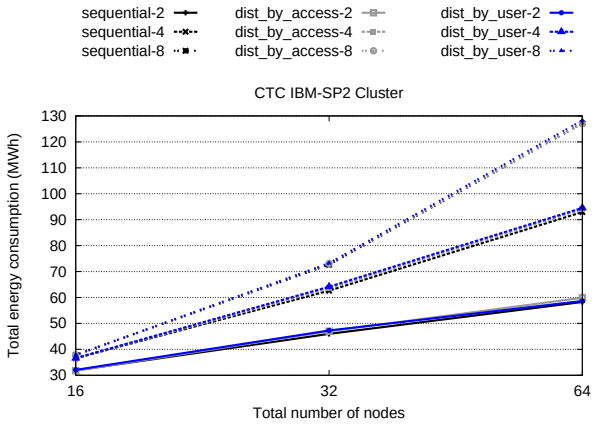

Fig. 7. Total energy consumption for CTC IBM-SP2 Cluster

Figure 7 shows that the total energy consumption for CTC IBM-SP2 parallel workload stays almost the same regardless of the assignment method used. In CTC IBM-SP2 Cluster, there are many (more than 600) users in the system and there is a high possibility of concurrent execution between these users. The total number of accesses is about the same across different users. Since there are too many users in the system, the system resources are

uniformly distributed across all the users even in the sequential case. Therefore, using distributed assignment by number of accesses or user identifiers does not make too much of a difference in terms of the total energy consumption. Without using our approaches, the CTC IBM-SP2 Cluster would stay on all the time and it would spend 39 MWh of energy with 16 nodes, 78 MWh of energy with 32 nodes and 156 MWh of energy with 64 nodes. Looking at the numbers in Figure 7, our approaches are able to decrease the energy consumption in the stock case by up to 63%.

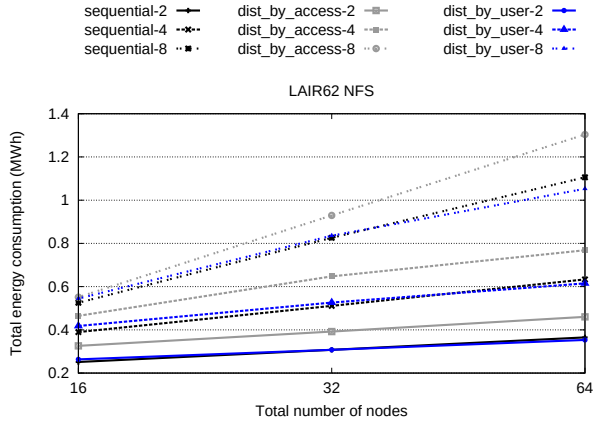

Fig. 8. Total energy consumption for LAIR62 NFS workload

Figure 8 shows that the total energy consumption for LAIR62 NFS workload increases when distributed assignment by number of accesses is used. In LAIR62 NFS workload, there is a very low possibility of concurrent execution between different users and the total number of accesses is about the same across different users. The low possibility of concurrent execution between different users becomes even lower when the users are uniformly distributed across the system by the total number of accesses; because the total number of accesses is about the same for each user. As a result, the total energy consumption goes up when distributed assignment by number of accesses is used, since the possibility of concurrent execution decreases. Without using our approaches, the LAIR62 NFS system would stay on all the time and it would spend 0.58 MWh of energy with 16 nodes, 1.16 MWh of energy with 32 nodes and 2.32 MWh of energy with 64 nodes. Looking at the numbers in Figure 8, our approaches are able to decrease the energy consumption in the stock case by up to 85%.

5.3 Utilization

In this section, we present the system utilization values with different node assignment methods by changing the total number of nodes (16, 32, 64) and the number of nodes assigned to each user (2, 4, 8) in the system. We define *utilization* as the percentage of time a node stays on during the entire simulation time.

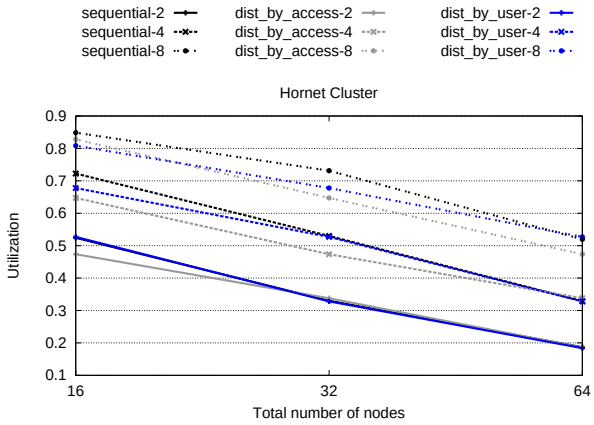

Fig. 9. Utilization for Hornet Cluster

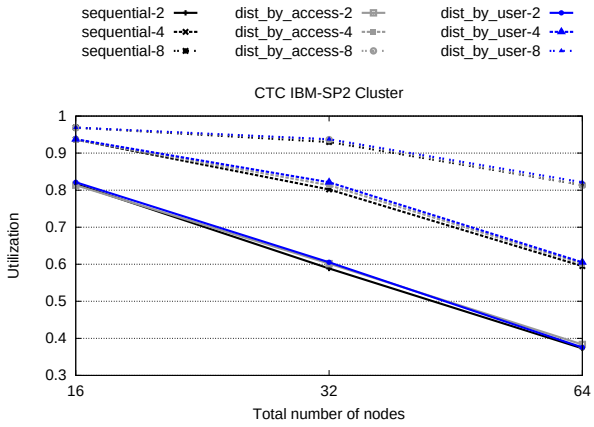

Fig. 10. Utilization for CTC IBM-SP2 Cluster

As it can be seen in Figures 9, 10 and 11, the utilization values are correlated with the total energy consumption values in Figures 6, 7 and 8. If an assignment method ends up saving more energy compared to another assignment method, it also has a lower utilization value compared to the utilization value in the other assignment method. Smaller utilization means that more system resources can be turned off, hence less energy is consumed.

5.4 Effect of Startup Time on Total Energy Consumption

In this test case, we observe how the total energy consumption is affected when the default startup time is varied (10, 60, 120, 240 seconds) for different inactivity threshold values (varied between 50 and 200000 seconds). We use the sequential assignment method with

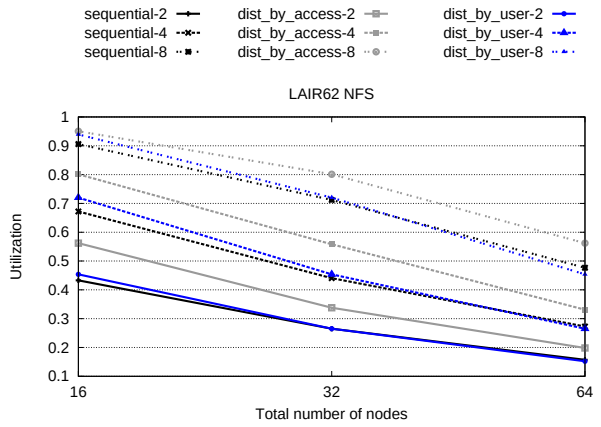

Fig. 11. Utilization for LAIR62 NFS workload

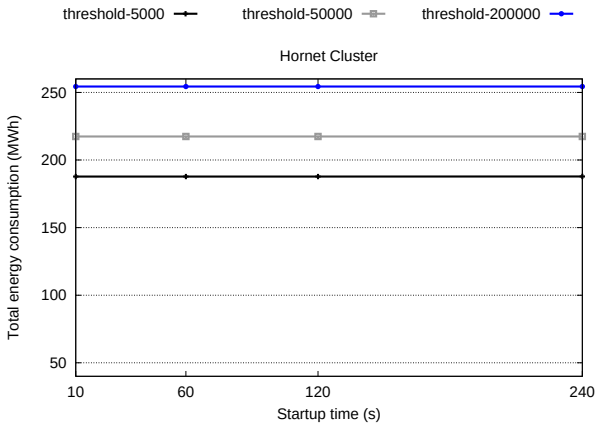

Fig. 12. Startup time vs. total energy consumption for Hornet Cluster

a total of 64 nodes, where 8 nodes are assigned to each user. As shown in Figures 12, 13 and 14, the total energy consumption increases for all the workloads when the inactivity threshold is high, since higher inactivity threshold decreases the possibility of turning off a node due to exceeding the inactivity threshold. When the startup time is increased for a fixed value of the inactivity threshold, the total energy consumption stays almost the same most of the time, except for the case in LAIR62 NFS workload where the inactivity threshold is low. The length of the startup time is usually too small compared to the total time a node stays on or the length of jobs in Hornet and CTC IBM-SP2 clusters. In the LAIR62 NFS workload, most of the transactions are in the range of milliseconds and when the inactivity threshold is low, the possibility of turning off a node due to exceeding the inactivity threshold increases. As a result, we can see the effect of the startup time on the total energy consumption more clearly for lower threshold values in LAIR62 NFS workload.

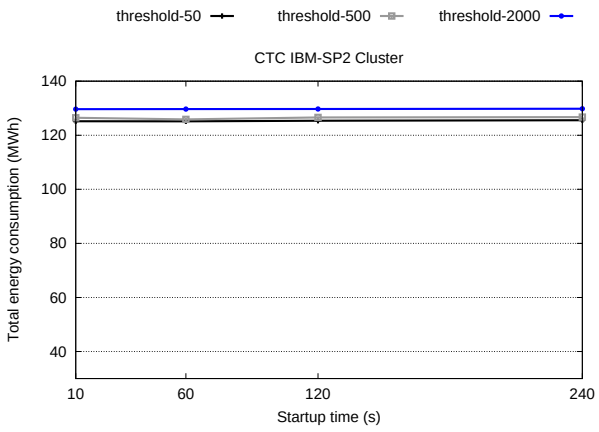

Fig. 13. Startup time vs. total energy consumption for CTC IBM-SP2 Cluster

5.5 Effect of Startup Time on Average Latency

In this test case, we observe how the average latency is affected when the default startup time is varied (10, 60, 120, 240 seconds) for different inactivity threshold values (varied between 50 and 200000 seconds). We use the sequential assignment method with a total of 64 nodes, where 8 nodes are assigned to each user. The definition of *latency* differs between workloads. For the Hornet and CTC IBM-SP2 Clusters, most of the jobs are in the range of tens of minutes. Therefore, the startup time is usually too small compared to the length of these jobs. For these datasets, we consider latency as the difference between the length of a job in stock case (no thresholds or startup times) and the length of a job with startup time and inactivity threshold involved. In other words, the latency is the delay for a scheduled job to get started; because one or more of the required storage nodes is not up. However, for the LAIR62 NFS workload, most of the transactions are in the range of milliseconds. These transactions are individual I/O requests. Therefore, the effect of the startup time would be significant. For the LAIR62 NFS workload, we consider the latency as the total time a transaction takes to complete with startup time and inactivity threshold involved.

As we can see in Figures 15, 16 and 17, average latency goes up when the startup time is increased for different inactivity threshold values. When the startup time is higher, users wait longer for a turned-off node to start up and more users are likely to wait for a node to start up. This, in turn increases the average latency. We can also see that for higher inactivity threshold values, the average latency is lower. Since the possibility of turning off a node due to exceeding the inactivity threshold is lower when the inactivity threshold is high, less users

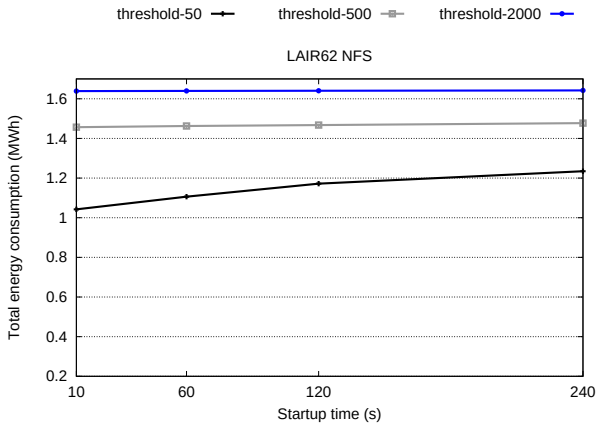

Fig. 14. Startup time vs. total energy consumption for LAIR62 NFS workload

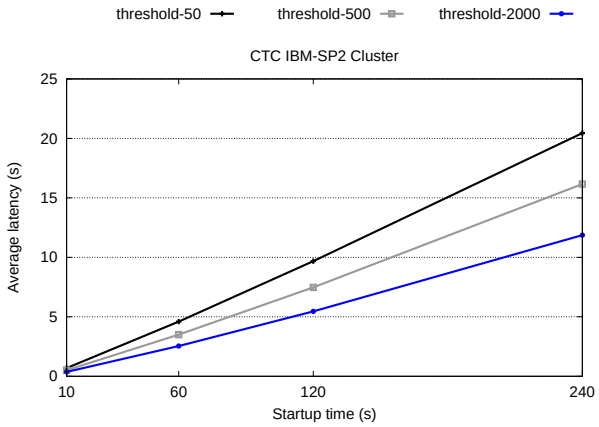

Fig. 16. Startup time vs. average latency for CTC IBM-SP2 Cluster

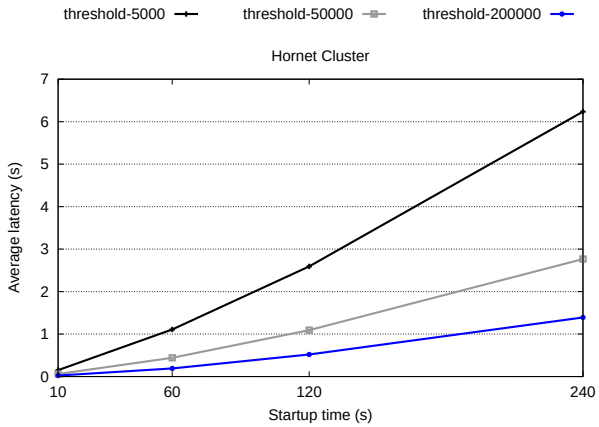

Fig. 15. Startup time vs. average latency for Hornet Cluster

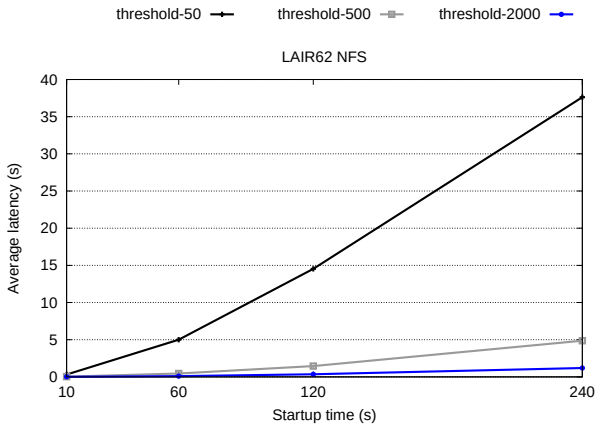

Fig. 17. Startup time vs. average latency for LAIR62 NFS workload

are likely to wait for a node to start up, which in turn decreases the average latency.

In the case of the Hornet and CTC IBM-SP2 clusters, a few seconds delay is minor in the context of hour long jobs. However, for the NFS workload, increasing the average latency by a few seconds can have a significant impact. Very few transactions are actually affected by the startup time, but those that are have a disproportionate impact on the average latency. A better measure is how many transactions are affected. In this test case, we analyze the percentage of LAIR62 NFS transactions exceeding a certain limit (16 milliseconds) by varying the startup time for different inactivity threshold values. We use the sequential assignment method with a total of 64 nodes, where 8 nodes are assigned to each user. We found out that the majority of the LAIR62 NFS transactions take longer than 16 milliseconds to complete and decided to use it as the limit to understand the effect of startup time on transaction length.

Figure 18 shows that the percentage of transactions taking longer than 16 milliseconds increases as the startup time is increased. When we assume the startup time is zero, almost all of the transactions take less than 16 milliseconds. As the startup time is increased, not only more users wait for a node to start up; but also they wait longer. Consequently, the percentage of transactions taking longer than 16 milliseconds increases for all inactivity threshold values. When the inactivity threshold is high, the effect of the startup time on the length of transactions is less, since the possibility of turning off a node due to exceeding the inactivity threshold decreases. In fact, for a threshold of 2000 seconds, less than 2% of the requests are affected.

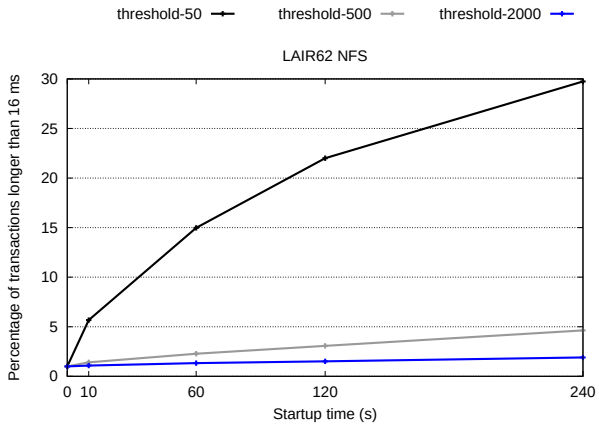

Fig. 18. Startup time vs. percentage of transactions longer than 16 milliseconds for LAIR62 NFS workload

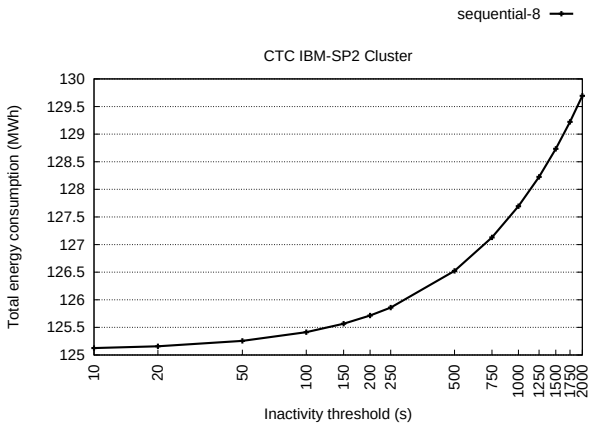

Fig. 20. Inactivity threshold vs. total energy consumption for CTC IBM-SP2 Cluster

5.6 Effect of Inactivity Threshold on Total Energy Consumption

We now analyze the effect of varying the inactivity threshold on the total energy consumption. We use the sequential assignment method with a total of 64 nodes, where 8 nodes are assigned to each user. We define a default threshold value for each workload we are using during the tests. The default inactivity threshold is 2000 seconds for Hornet Cluster, 750 seconds for IBM-SP2 Cluster and 50 seconds for LAIR62 NFS workload. As expected, since the number of transactions in LAIR62 NFS workload is too big, the default inactivity threshold is small; whereas the opposite is true for Hornet Cluster.

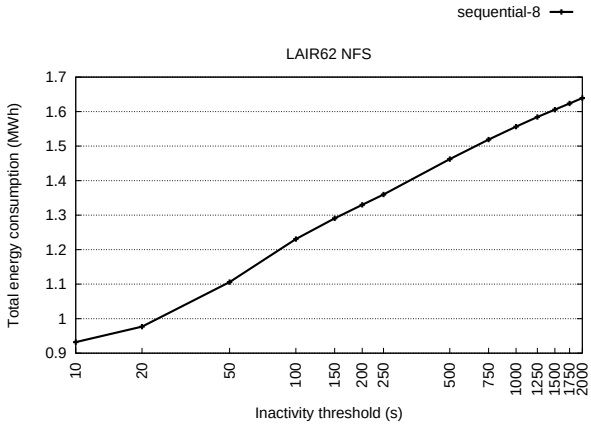

Fig. 21. Inactivity threshold vs. total energy consumption for LAIR62 NFS workload

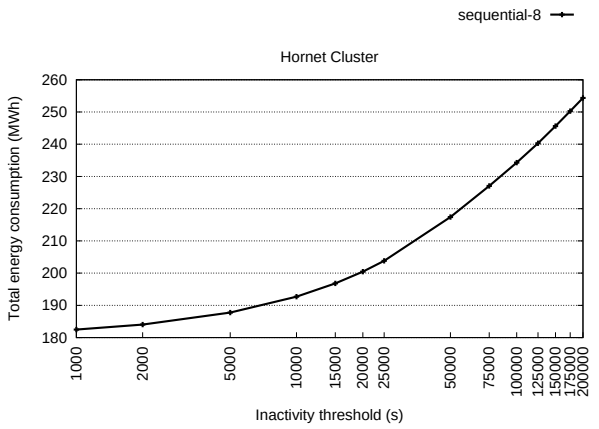

Fig. 19. Inactivity threshold vs. total energy consumption for Hornet Cluster

Figures 19, 20 and 21 show that, the total energy consumption goes up as the inactivity threshold is increased. When the inactivity threshold is high, the possibility of an interarrival period (the period between the end of a transaction or job and the start of the next transaction or job) taking longer than the inactivity threshold is less. Therefore, nodes will be turned off less due to exceeding the inactivity threshold. As a result, the system will stay on longer and the total energy consumption will go up.

5.7 Effect of Inactivity Threshold on Average Latency

In this test case, we observe how the average latency is affected when the default inactivity threshold value is varied. We use the sequential assignment method with a total of 64 nodes, where 8 nodes are assigned to each user. When the inactivity threshold goes up, the possibility of an interarrival period taking longer than the inactivity threshold goes down. Therefore, less users will wait for a node to start up. As expected,

Figures 22, 23 and 24 show that the average latency decreases as the inactivity threshold increases.

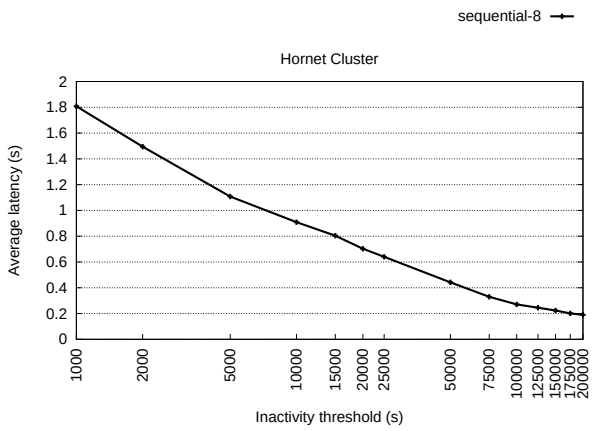

Fig. 22. Inactivity threshold vs. average latency for Hornet Cluster

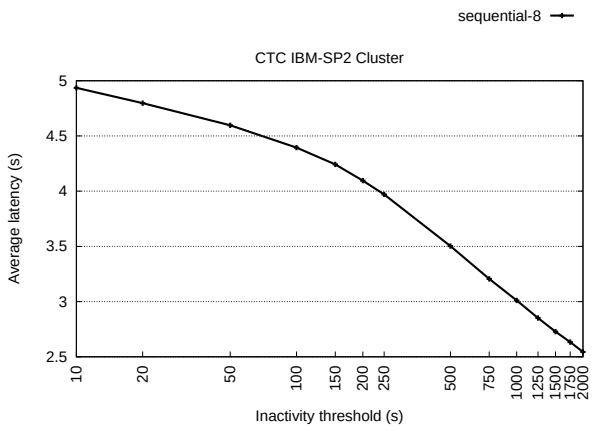

Fig. 23. Inactivity threshold vs. average latency for CTC IBM-SP2 Cluster

5.8 Effect of Inactivity Threshold on Transaction Length

In this test case, we show the effect of changing the inactivity threshold on the percentage of LAIR62 NFS transactions exceeding a certain limit (16 milliseconds). We use the sequential assignment method with a total of 64 nodes, where 8 nodes are assigned to each user. The default startup time is 60 seconds.

Figure 25 shows that the percentage of transactions taking longer than 16 milliseconds decreases as the inactivity threshold is increased. When the inactivity threshold goes up, the possibility of an interarrival period taking longer than the inactivity threshold goes down.

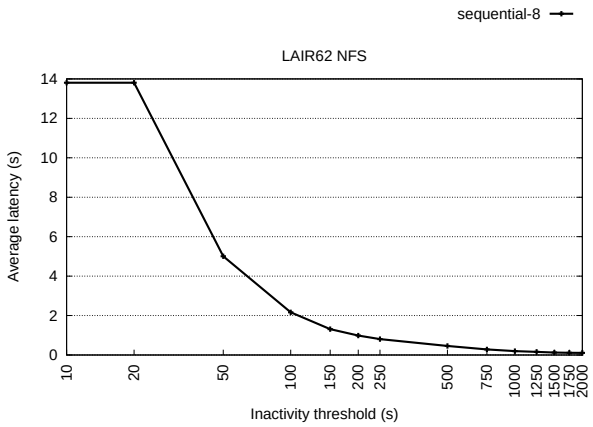

Fig. 24. Inactivity threshold vs. average latency for LAIR62 NFS workload

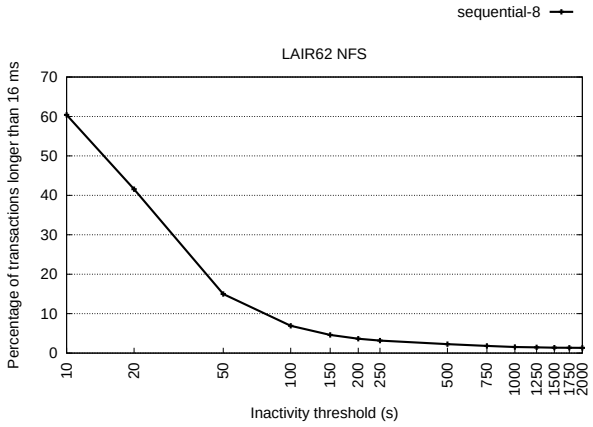

Fig. 25. Inactivity threshold vs. percentage of transactions longer than 16 milliseconds for LAIR62 NFS workload

Therefore, less users will wait for a node to start up, decreasing the percentage of transactions taking longer than 16 milliseconds. Thus, comparing Figures 21 and 25, we see that there is a tradeoff between energy consumption and latency as we change the inactivity threshold. Changing the threshold from 2000 to 10 seconds can reduce the energy consumption by nearly half at the cost of increasing the percentage of long latency requests from near 0 to nearly 60%.

6 Conclusion

We have presented two different I/O node assignment methods in order to reduce energy consumption in parallel storage systems. The proposed methods, sequential and distributed, have been implemented and tested with real-world workloads. The results show that

sequential method performs very close to the distributed assignment by user identifiers. Distributed assignment by number of accesses achieves uniform distribution of system resources and it performs better than the other methods when there are many concurrent users in the system. As a result, the total energy consumption can be decreased by up to 85% using our approaches.

The methods presented in this paper can be improved even more by integrating them into a high-performance parallel file system. Also, in order to compare the existing energy saving techniques with the techniques we are proposing, new low-energy modes (i.e. various disk speeds) can be used and these modes can be implemented physically. Another interesting issue to investigate would be the effectiveness of the approaches when the bandwidth of the storage system is not maxed out.

ACKNOWLEDGMENTS

This work was supported in part by a NSF High End Computing University Research Activity grant (award number CCF-0937879). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF. The authors would also like to thank the valuable assistance of staff at the UConn Booth Engineering Center for Advanced Technologies in providing access to cluster resources which facilitated this research.

REFERENCES

[1] Cengiz Karakoyunlu and John A. Chandy. Techniques for an Energy Aware Parallel File System. In Green Computing Conference (IGCC), 2012 International, pages 1-5, June 2012.

[2] HPC Booth Engineering Center for Advanced Technology. http://becat.uconn.edu/hpc/, February 2014.

[3] Logs of Real Parallel Workloads from Production Systems. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html, February 2014.

[4] Daniel Ellard. Trace-Based Analyses and Optimizations for Network Storage Servers. PhD thesis, Harvard Computer Science Technical Report TR-11-04, May 2004.

[5] APC White Paper #6. Determining Total Cost of Ownership for Data Center and Network Room Infrastructure. Technical report, America Power Conversion, 2005.

[6] Ricardo Bianchini and Ramakrishnan Rajamony. Power and Energy Management for Server Systems. IEEE Computer, 37(11):6874, 2004.

[7] Philip H. Carns, Walter B. Ligon, III, Robert B. Ross, and Rajeev Thakur. PVFS: A Parallel File System for Linux Clusters. In Proceedings of the Annual Linux Showcase and Conference, pages 317- 327, October 2000.

[8] Daniel Wong, Murali Annavaram. KnightShift: Scaling the Energy Proportionality Wall through Server-Level Heterogeneity. Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on, pp.119,130, 1-5 December 2012.

[9] Dennis Colarelli and Dirk Grunwald. Massive Arrays of Idle Disks for Storage Archives. In Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, Supercomputing 02, pages 1-11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[10] Lakshmi Ganesh, Hakim Weatherspoon, Mahesh Balakrishnan, and Ken Birman. Optimizing Power Consumption in Large Scale Storage Systems. In 11th USENIX Workshop on Hot Topics in Operating Systems, May 2007.

[11] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. SIGARCH Comput. Archit. News, 31(2):169181, May 2003.

[12] Ethan L. Miller and Randy H. Katz. An Analysis of File Migration in a Unix Supercomputing Environment. In USENIXWinter 1993, January 1993.

[13] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Srinivasan Sesan. Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems. February 2008.

[14] Eduardo Pinheiro and Ricardo Bianchini. Energy Conservation Techniques for Disk Array-based Servers. In Proceedings of the 18th Annual International Conference on Supercomputing, ICS 04, pages 68-78, New York, NY, USA, 2004. ACM.

[15] Brent Welch, Marc Unangst, Zainul Abbasi, Garth Gibson, Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou. Scalable Performance of the Panasas Parallel File System. In Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST08, pages 2:1-2:17, Berkeley, CA, USA, 2008. USENIX Association.

[16] Lustre: A Scalable, High-Performance File System. Cluster File Systems Inc. white paper, version 1.0, November 2002. http://www.lustre.org/docs/whitepaper.pdf.

[17] Frank Schmuck and Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In In Proceedings of the 2002 Conference on File and Storage Technologies (FAST), pages 231-244, 2002.

[18] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the Energy Efficiency of a Database Server. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10, pages 231-242, New York, NY, USA, 2010. ACM.

Cengiz Karakoyunlu is a Ph.D. candidate in the Deparment of Electrical and Computer Engineering at the University of Connecticut. He received his M.Sc. in Electrical and Computer Engineering from the University of Connecticut in 2013 and his B.Sc in Electrical and Computer Engineering from Worcester Polytechnic Institute in 2010. His research interests include Parallel Computing, Storage Systems, Distributed File Systems and Computer Architecture. His home page is http://engr.uconn.edu/ cek10006/.

John Chandy is an Associate Professor and Associate Head of the Department of Electrical and Computer Engineering at the University of Connecticut. Prior to joining UConn, he had executive and engineering positions as a co-founder in small software companies working particularly in the areas of clustered storage architectures, online delivery of psychotherapy and soft-skills training, and sales force automation. His current research areas are in high-performance storage systems, reconfigurable computing, distributed systems software and architecture, and computer systems security.