Energy Conservation in Disks Utilizing Related Data Concentration

Tejas Hasarali Snigdha Obulam Yevgeniy Ryasnianskiy

Florida State University

May 4, 2018

Abstract

Existing data concentration-based energy conservation techniques have used ad-hoc metrics as the basis for data migration and placement. Current state-of-the-art techniques such as Popular Data Concentration (PDC) and Massive Array of Idle Disks (MAID) are based on simple characteristics such as file use recency or frequency. In this paper, a reinforcement learning based approach is proposed and evaluated for its effectiveness in conserving energy by idling disks in a RAID configuration. This approach is capable of utilizing rich information about the context in which files are accessed and utilized. File accesses are performed by processes on behalf of use, we believe that these usage patterns are critical for effective data concentration. To evaluate our approach, workloads were simulated in DiskSim. Existing traces were parsed using out Related Data Concentration (RDC) software which assigns and migrates data to disks. We determine that this approach is effective in maximizing the frequency and length of disk idle time on real-world workloads. After an initial penalty of 5.7% Normalized Disk Idle Time at improvement of 19.7% Normalized Disk Idle Time was achieved after convergence.

1 Introduction

1.1 Energy Conservation

Energy usage by storage systems accounts for a considerable proportion of energy usage in datacenters. Datacenters are usually composed of a large number of disks in some RAID configuration. These disks spend considerable periods of time idling. [14] These disks can be configured to have variable speed modes or thresholds for powering down. Disks require time and energy to spin up from idle or lower RPMs. Typical disks can take upwards of 10 seconds to spin up to operating speeds. These techniques are most effective when idle periods are concentrated and requests are services for long periods of time. A related challenge is predicting idle periods, especially when request latency is a significant design factor. In the rest this paper we focus on the former problem of maximizing the time spent idling and on ensuring that idle periods are sufficiently long.

1.2 Popular Data Concentration

Popular Data Concentration algorithms [12] maximize disk energy conservation in systems having more than one disk. To achieve this data must be migrated between disks. This strategy is sensitive to the characteristics used to group data and the precise thresholds which determine the frequency and targets of the migration process. Popular Data Concentration works by storing popular data together on the same disks as the other popular data and storing unpopular data together on the same disks as the other unpopular data. The assumption behind this technique is that prior data popularity is a good predictor for future data popularity. The expectation being that the disks containing the popular data will spend very little time idling and the disks with the unpopular data will be idled most of the time.

1.3 Massive Array of Idle Disks

Massive Array of Idle Disks [4] is a system designed to maximize disk idleness in applications where the majority of data is infrequently accessed. This

approach functions by keeping most disks idle with the exception of a small number that store the recently accessed data. Another tradeoff of MAID is that it cannot take advantage of parallel reads and writes in situations where request volume is high. Given the specialized access patterns required for proper application of this approach it is not sufficiently versatile for application to the traces considered in this paper.

1.4 Reinforcement Learning

Reinforcement learning is a machine learning technique where an agent attempts to maximize the reward received for taking actions over time as defined by the Bellman equation. The agent has a policy for taking actions in an environment for which a reward is received. This approach works well in an online setting where the agent guides the production of classified data unlike supervised and unsupervised learning techniques. [15] In the context of the paper: The learner has access to information about incoming IO requests including information about the requesting user and process. The learner tracks information about the current and former locations of previously encountered requests. The learner is required to assign incoming requests to a disk of its choosing. After a certain number of requests is processed, the resulting trace is simulated to determine a reward based on Normalized Disk Idle Time. Recent advances in Reinforcement Learning have enabled efficient searches of large state spaces [16]. In this paper we leverage these advances to develop a novel method of maximizing disk idleness.

1.5 Trace Data

It is important to ensure that simulations are performed on data that reflects real-world performance considerations. Additionally, to apply the proposed Related Data Concentration (RDC) approach, a trace is required which identifies the user and process responsible for a given request. Such a trace is provided by FIU. This trace was obtained by observing the disk usage on FIU's computer science web, email, and development servers.

1.6 Related Data Concentration

We propose Related Data Concentration as an alternative method of concentrating data to take advantage on the energy concentration that can be achieved by idling or slowing arrays of disks. Related Data Concentration is based on the observation that the users and programs usually make frequent concurrent requests to the same subset of files. Additionally, certain users and processes are active at different times resulting in persistent variation in file popularity. There also exists the possibility of second order correlations between users and processes, users and other users, and processes and other processes.

2 Previous Works

2.1 Nomad File System

It has been observed that data access frequency exhibits a Zipf distribution. Filesystems can take advantage of this property in allocating files to devices based on file popularity. It should be noted that too much popular data can overload a disk if not distributed properly. [11]

Nomad FS maximizes disk idleness by sorting hot and cold data on different disks: "Nomad FS is a user-level, event-driven server that works on top of the local file system. The server associates a helper thread with each disk. To conserve energy in disk arrays composed of conventional disks, PDC is used to idle disks, which are then spun down by the server after a fixed period of idleness (the idleness threshold). The server determines that it is time to spin down a disk by keeping the last access time per disk and periodically testing whether any disk has been idle for longer than the idleness threshold. A spun down disk is reactivated on the next access to it."

Nomad FS was tested on real and synthetic data in configurations that utilized both variable speed disks and conventional disks which were spun down completely. Nomad FS was compared to MAID and simple threshold-based techniques. It was determined that both MAID and Nomad FS were only capable of energy conservation utilizing conventional disks in cases where overall load is very low. Where variable speed disks used Nomad FS was capable of

consistently reducing overall system energy consumption without significant latency penalties.

2.2 Storage Device Types

In addition to taking advantage of homogeneous arrays of disks, PDC can be applied to heterogenous arrays of disks. This allows the application to take advantage of the characteristics of each storage type. This concept can be extended to networked storage devices and data caching. [12] PDC has been applied to these heterogenous arrays of disks where data was stored across conventional hard disk drives and a small number of solid state drives. In this approach, files need to be differentiated based on their size and the manner in which they are accessed most often in addition to their popularity. Large files that are accessed sequentially and are predominantly read are placed on traditionally hard disk drives. Small files that are accessed randomly and that are predominantly written are placed on solid state drives. This convention can further be extended by placing the least frequently accessed files on network devices. Caching can be used to buffer writes that are being made to disks which are currently spun down.

2.3 Energy-Efficient VM Consolidation Using Reinforcement Learning

This paper [2] proposes Reinforcement Learning-based on Dynamic Consolidation method (RL-DC) technique to minimize the number of hosts that are active at a given time based on the workload. It uses a popular reinforcement learning algorithm and learns from the past knowledge to decide which host should be active and which should be switched to sleep mode. The RL-DC dynamically learns the pattern of workload and makes the efficient decision about keeping the disk running or switching it to sleep mode. The dynamic consolidation technique refers to turning off the servers which are underutilized. It is a well-established technique that is proven to be effective and used by most of the cloud service providers. The biggest challenge cloud service providers face is providing the service on par with software as a service agreement. The developed model efficiently switches the serve to sleep mode at the same time ensuring SaaS agreement. The developed method has to intelligent decide which hosts to keep active which ones to shut down, the paper proposed a consolidation learning technique with an agent Q. The agent learns the policy to enforce by experience from the environment and prior knowledge.

The reinforcement learning algorithm obtains the agent by dynamic learning without any prior knowledge using State space, action space and reinforcement signal. The state space is responsible for define the initial state and perception of the environment, the action space takes action based the state, and the reinforcement signal is the signal that is received from the environment about the actions performed i.e., this is the feedback obtained due to the action performed based on the observed state. Q learning is one of the most popular techniques of reinforcement learning which starts with a state, performs an action and moves to the next state, receives the signal and updates the Q value. Initially the action is chosen randomly but in the following iteration the actions are chosen based on the feedback. The proposed model works in three phases, first is observing the current state of all the hosts and learning the pattern to make efficient policy. In the second step the hosts with minimal CPU utilization is identified and sleep command is issued. The overloaded hosts are also taken into consideration for performing action. In the third phase all tasks from the hosts which received sleep signal and the overloaded hosts are reallocated intelligently to other hosts. RL-DC also makes the hosts active based on the prediction made to ensure SLA is not violated. By reducing the number hosts that is active the RL-DC reduces the power consumption at the same time ensuring the SLA is not violated. The proposed model efficiently reduces the power consumption using RL-DC technique over the existing dynamic consolidation technique. This technique is analogous to the work we are doing, here RL is applied to hosts but we are applying it to disk arrays.

2.4 Towards Energy-Aware Scheduling in Data Centers Using Machine Learning

The paper [3] using machine learning algorithms trained on previously available data to predict the power consumption levels, workloads and SLA policy requirements. The proposed model is very close to optimal policy and it performs well in uncertain conditions. The two main techniques that has been effective in energy conservation is server

consolidation and shutting down spare servers. But with the advancement of machine learning techniques it opens up a new arena of opportunities to optimize the energy consumption further. The paper proposes a dynamic workload consolidation technique to efficiently consolidate servers to reduce the energy consumption.

The proposed model uses Dynamic backfilling scheduling algorithm for comparison which distribute the workload among different machines. The dynamic back filling tries reduces the number of hosts by filling a host with jobs until it is full. Thus, switching the servers off which are not in use. Whenever the load is high new serves are switched on to compensate the workload requirement. But it has to be kept in mind that switching a server back on takes at least a minute, so it is essential to manage intelligently to respect the SLA agreement. The important factor in machine learning is the dataset used for training the model, the dataset should contain most of the scenarios to make a good prediction. Here the dataset is the previous workloads encountered by the servers and the polices in place to efficiently manage the disk. The outcomes of these policies are used as the labels for predictions. Here outcome refers to result of the used policy which can be good customer satisfaction or bad customer satisfaction. The amount of energy conservation achieved was also collected when the current policy was used. The collected data is used to train the M5P machine learning model which divides the data into trees and it trains the leaf nodes using linear regression model.

One of other important factor was to predict the deadline fulfillment which is essential for SLA satisfaction. For this the author used another regression function using timing values and other jobs in the current machine. The results of the trained models were used to make predictions dynamically to decide what is the best policy of workload distribution to achieve maximum energy conservation while respecting the SaaS License Agreement (SLA). One of the biggest advantage of using this model is, it helps in deciding the best policy during uncertain condition, which other scheduling policies fails to achieve. The proposed model efficiently decides the polices to reduce the energy consumption at the slight cost of performance. The model if extended to use reinforcement learning, it will be able to learn in real time and make better policy decisions. This gives insight on how the machine learning algorithm can be used in our model to extent it to make better decisions during uncertain conditions.

2.5 Massive Arrays of Idle Disks for Storage Archives

The paper [4] proposes a technique of using Massive Arrays of Idle Disk or MAIDS in place of high performance RAID to achieve similar performance with 1/15th reduction in power consumption. Designing a MAID is simple and straight forward compared to RAID design. But maintaining the balance between energy conservation and performance will need many design decisions. The first design decision is whether to use cache or data migration. In case of data caching some disks are used exclusively for caching of files, the advantage of this technique is most of the request will be served by the cached disks, which results in spinning down many disks. But at the same time, it introduces the overhead of mapping these files to the actual location and meta data management.

With the increase in cache size this become harder and harder. Whereas in case of data migration there is no over-head of meta data maintenance or location mapping which makes the model simple. The ideal decision depends on the requirement of the disk arrays. If it is used for data archiving the data is rarely accessed hence it doesn't make any sense to cache but if it is used for streaming service, then data has to be cached for better performance. The second decision is whether to use block level or file level access, in case of file level access the performance will be better due to better caching and in applications where large files are used. But for the study author used block level interface.

The primary focus of the study showed that using MAID we can achieve performance comparable to RAID with very less power consumption. The study also gave some interesting findings, where no cache MAID performed better than cached MAID even for very small files with frequent accesses. This was due to the lack of localization in the workload. The MAID was able reduce the power consumption efficiently

with very less performance degradation. But MAID is mostly suitable for disks that are used infrequently i.e., it is mostly applicable to disks used for data archiving and it is also not optimized for parallel disk volume access.

2.6 Energy-Aware Data Prefetching for Multi-Speed Disks

The paper [5] explores ways to conserve energy using an optimizing compiler to pre-fetch data keeping energy conservation in mind. The paper achieves significant reduction in energy consumption over wide spectrum of data without any impact on performance. With the decrease in disk speed the pre-fetch distance increases and power consumption drastically decreases. The paper employs two techniques to conserve energy one is running the disks at lower speeds another is prefetching more data to reduce disk access. The model first determines the disk speed that results in maximum energy saving, this is carried out by analyzing the program that needs to be executed. For each disk access group suitable disk speed which do not reduce the performance is selected. Then the code is analyzed to determine the amount of localization in data access, this information is used to determine the pre-fetch distance of the data. The developed model was able to reduce the energy consumption more than fifty percent. Interestingly the performance increased more than forty percent using prefetching. This shows that along with migration and duplication of data, perfecting of data can have significant increase in energy conservation. But the model may add delay due to code analysis to identify initial conditions.

3 Methodology

3.1 Simulation Traces

There are factors that are critical in selecting a suitable trace for evaluating the effectiveness of Related Data Concentration. Trace data must be rich, it must contain information that can be used to establish a relationship between the data that is being requested and the users and processes that are requesting the data. For this a user ID and process name is sufficient. Trace data must reflect real-world use patterns. As noted, data concentration-based energy conservation is very sensitive to system use patterns.

To this end trace data obtained by FIU from their Computer Science Department's web, email, and development servers will be used. Figure 3.1.1 provides an example of this trace. The format of these traces are as follows:

1. The number of nanoseconds from the time that the trace was started
2. The user ID of the user that executed the process
3. The name of the process requesting data
4. The offset in bytes of the blocks being requested
5. The number of bytes of data that is read/written
6. The type of I/O operation
7. Miscellaneous data
8. Miscellaneous data
9. A hash of the data being read/written

These traces are pre-processed before being passed to the Related Data Concentration application. Successive operations on the on consecutive data by the same user, process, and I/O type where the subsequent block offset differs from the previous block offset by exactly the number being read/written are consolidated into a single line in the trace. Certain operations which real systems schedule opportunistically are removed from the trace such as flushing of the journal to remove data cached on the disk. Timestamps are normalized with the start of the first trace being treated as second 0. Timestamps are uniformly multiplied by a fractional factor to account

```
1              2     3         4         5 6 7 8 9
691200166966874 15749   pdflush 688832896 8 W 6 0 6cf1c2d64d0c08946825d8c513366b
691202167414505  2522 kjournald 688749800 8 W 6 0 860f0897b2547070de10a2f84275f2
691202167509076  2522 kjournald 688838032 8 W 6 0 ed6632657112438687b7dad4c2e47b
```

Figure 3.1.1: Example FIU trace data

for the difference in the speed and quantity of the disks used by the FIU system and the simulated system.

3.2 DiskSim

The advancement in computation technology has made the storage system determinant of the overall system performance. The storage system advancement has failed to match the speed of the processor and it has become the bottleneck in computer technology. To help the research community in innovating in storage technology DiskSim was developed. It is an efficient, accurate, highly-configurable storage system simulator. [1] DiskSim is written using C programming language, it's a software that can be installed in Linux machines with C compiler. It includes all the components required to simulate a disk like device drivers, buses, controllers, adapters and disk drives. Some of these components are written individually which can be interconnected to simulate different functionality. Each module can be again configured to match the requirement of the researcher. The disk module is given much more importance and it is written to provide maximum functionality. The DiskSim does not maintain any state information, or it will not keep track of the files that are written into the disk, it just simulates a write with all different disk functionalities. The DiskSim has been validated as an individual system as well as system level module running under an operating system. It has been tested against 5 different disk drives provided by 3 different manufacturers. The DiskSim has been already used in many researches and is well established in the research community. Hence, we choose to use DiskSim in our project as disk simulator. The DiskSim was installed in a Linux machine with PROCESSOR.

We can simulate disk loads on DiskSim using three different methods - using the in-built synthetic work load provided by disk, using the trace file generated by the user and using the disk in a larger complete simulation environment. The freedom to change the configuration of the synthetic load provided by the DiskSim is very minimal, so we decided to generate the work load ourselves using trace file. The DiskSim trace file has five fields Request arrival time, Device number, Block number, Request size and Request flag. Here Request arrival time is the time at which the IO request was created, Device number refers to the disk number where the data has to be placed, Block number is the offset location in the disk where data has to be placed, Request size is the size of the data to be written in bytes and Request flag represents if it is read or a write (Read-1 and Write-0).

The trace file can be loaded into the DiskSim in different formats we choose ascii format to create the trace file, the type of the trace file has to be specified along with the parameters defining the file while loading to help DiskSim in reading the file. The disk configurations can be modified using the parameters files provided by the DiskSim. Parameters files contains mainly two sections global and stats, global section is responsible for simulation wide parameters like seed, warm up time, output trace file, disk size and the stats section is responsible for specifying the statistics that needs to be logged during the execution for producing the output file. The parameters file also provides the DiskSim with encoding information of the trace file. The maximum size of each disk was limited to SIZE using the parameters file. The number of disks was limited to NUMBER during the run time using the parameters files.

Since DiskSim do not maintain any state information, the disk block to write the data along with movement of the file was carried out by a separate interface program written in python, which converts the workload file into the format required by the DiskSim, along with maintaining state information. It is also responsible for running the disk, we will discuss more about the interface in the next section. While issuing the command to execute the disk, the cache size, additional synthetic load and scheduling policy has to be specified. The data was not cached during the execution of our experiment, no additional synthetic load was provided to the DiskSim and we used Shortest-Seek-Time-First algorithm to schedule the disk writes. As the name suggests it writes the data with shortest seek delay. Since we are trying the spin down the disk to reduce the energy consumption the effect from the scheduling policy is minimal. Below is an example of command to run the DiskSim:

```
disksim parms.1B stdout ascii t. 0
"disk1 .. disk16" "Segment size 0"
64 "disk*"

"Scheduler:Scheduling policy" 4
```

Here it creates 16 disks and uses ascii.t trace file to create the load with Shortest-Seek-Time-First algorithm scheduling algorithm. The segment size refers to the cache size, since we are not using any caching, it is 0 blocks. The biggest challenge we faced was spinning down the disk using the DiskSim. DiskSim do not support spinning down of disks the work around we came up with was to use the disk idle time as disk spun down time. To make up for the delay in spinning down and spinning up to normal speed, a penalty was added during the calculation of the improvement achieved using reinforcement learning. The disks with idle time more than 20 was used as disk spun down and the penalty was added in both spinning up and spinning down the disk.

After simulating the disk, DiskSim creates an output file with all the statistics of the run, but we are only interested in the IO statistics. The output provided by the disk could not be directly used in our experiment to produce the results. Hence the DiskSim configuration file was modified to provide the output to display the number of disk idle times each disk. The disks with idle time more than 20 seconds with a penalty of 10 seconds was considered as disk spun down and given to the Reinforcement learning algorithm for reward computation. The reinforcement learning uses this to compute the reward for the next run, which we will discuss in detail in the following sections.

3.3 DiskSim Interface

The DiskSim simulates the disk based on the trace file and the parameters supplied. But DiskSim does not maintain any state information, it just simulates the operation specified in the trace file. It is unaware of the blocks which are empty or blocks in which the data is already present, it doesn't care if the user if over writing the data. It just simulates all the operation including disk seek time based on the size and the offset of the data to be written to the disk. In our project it is essential to move the data to reduce the energy consumption, without the knowledge of data location, we cannot move the file. Hence, we wrote a python program which acts as an interface to provide all these functionalities.

DiskSim interface receives the data in text format from the reinforcement learning algorithm, it is responsible for converting the received data into DiskSim readable trace file. The data received from the reinforcement learning algorithm contains ID, request time, disk number, size and read/write. Along with this it also contains the number of disks and size of each disks. The DiskSim interface converts this into Request arrival time, Device number, Block number, Request size and Request flag. The ID is a unique ID given to the data, which helps to keep track the location of the disk. The interface maintains the size and location of all the data encountered using this unique ID. The interface also maintains the current writing block for writing the data to the disk.

Algorithm 1 DiskSim Interface

```
1: procedure INTERFACE(filename)
2:     Input: Work load file generated by Reinforcement Learning algorithm
3:     Output: Trace file for DiskSim
4:     diskBlocks ← {diskNumber : [currentWritingBlock, maxCapacity]}
5:     idToBlock ← {id : [diskNumber, diskBlock, noOfBlocks]}
6:     fOut ← open(output.trace, w)
7:     fIn ← open(filename.txt, r)
8:     stack ← list()
9:     for line in fIn do
10:        while len(stack) and stack[requestTime] < requestTime do
11:            fOut ← Pop stack
12:        end while
13:        if task is read then
14:            if id not in idToBlock then
15:                randomDisk ← Generate the random disk
16:                idToBlock ← Location of the file
17:                currentblock ← Increment
18:                if randomDisk != disk then
19:                    maxCapacity ← Increase
20:                    fOut ← Read the file
21:                    maxCapacity ← Increase
22:                    requestTime ← Add 100 millisecond
23:                    stack ← Push Write the file
24:                    idToBlock ← Update location of the file
25:                    currentblock ← Increment
26:                end if
27:            else
28:                fOut ← Read the file
29:            end if
30:        else
31:            if task is write then
32:                fOut ← Write the file
33:                if id in idToBlock then
34:                    maxCapacity ← Increase
35:                end if
36:                idToBlock ← Location of the file
37:                currentblock ← Increment
38:            end if
39:        end if
40:    end for
41: end procedure
```

To simplify the implementation currently interface do not keep track of the data fragmentation. The actually maximum size of the disk is kept sufficiently larger compared to the required disk size, whenever a file is

moved between the disks the fragmentation is compensated by increasing the size of the required disk. Care is taken to avoid required disk size exceeding the actual provided disk.

The data obtained for our experiment is not from the beginning of the disk usage, this creates a problem of requests for disk reads for the files which are not present. This was handled by randomly allocating the disk to the file that needs to be read. The file is then read from this location and after reading, it is moved to the location specified by the read operation. The disks are assigned randomly to distribute the preexisting files in a manner which do not affect the outcome of reinforcement learning algorithm. In the second scenario if the reinforcement learning algorithm requests for a file that is already present in the disk but in a different location, then the data is moved to the new location and read from the new location. For write operations the data is written to the next available location of the disks, if the data is already present in the disk the old location is deleted and it is updated with the new location. Since the read and write operations are requested continues with very less time gap, we may need to queue moving of the data if the next request arrives in less than 100 milliseconds from the current request. This is done to avoid two or more operation overlapping. Hence moving operation is queued until it finds an interval greater than 100 milliseconds.

The interface continuously writes the read and write operations to trace file and, in the end, it also executes the DiskSim to simulate the disk using the trace file generated. The interface also produces an output file with the statistics on how much space was occupied during the simulation. This was used to observe the current writing block and the capacity of the disk for handling disk fragmentation.

4 Related Data Concentration

4.1 Temporal Difference Learning

Reinforcement learning bases actions on a learned policy. This policy is determined by an algorithm that attempts to maximize future rewards. In this project we will be applying a modified form of Temporal-Difference learning TD(0). For TD(0) policy evaluation can be defined by the equation:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]\,. \end{aligned}$$

Temporal Difference learning relies on an expectation of discounted future rewards. Our reward will be based on Normalized Disk Idle Time. We construct this reward to satisfy the following constraints:

- Idle periods in excess of the disk idle threshold by an amount greater than the disk spin up penalty should be rewarded proportional to the overall excess idle time
- Idle periods in excess of the disk idle threshold by an amount less than the disk spin up penalty should be penalized proportional to the overall resulting excess disk spin up time
- Idle periods not meeting the disk idle threshold should be penalized proportional the overall lengths
- Excess disk spin up time should be penalized more then excess disk idle time is rewarded
- Expectations will be optimistically initialized to 1 in order to satisfy the exploration requirement necessary for convergence
- Rewards will be normalized to some value between 1 and -1 (usually no less than 0.)

Normalized Disk Idle Time (NDIT) is defined as follows:

$$NDIT = \frac{Excess\ Idle\ Time - Excess\ Completion\ Time}{Baseline\ Completion\ Time}$$

Where: Baseline Completion Time is the time taken to complete the trace assuming Related Data Concentration is not used to parse the input. Excess Idle time is the time spent idling above the disk spin up penalty. Excess Completion Time is double the time difference between breakeven idle time (accounting for the spin up penalty) and actual idle time.

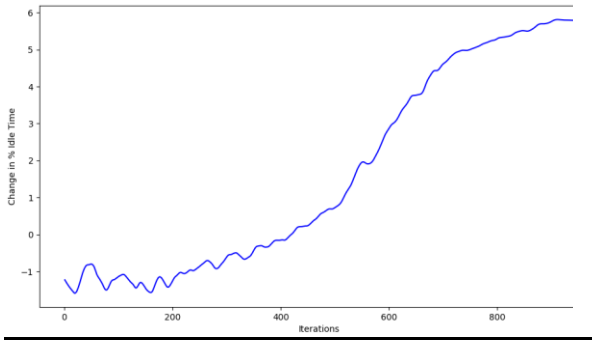

Figure 4.2.1: Change in absolute Normalized Disk Idle Time over ~955 iterations

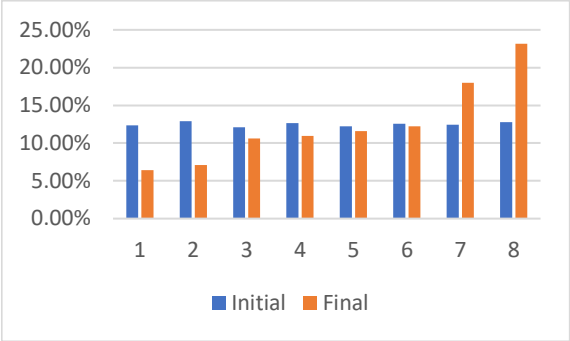

Figure 4.2.2: Distribution of I/O operations across disks in the initial run and after convergence

Using this metric, the following Temporal-Difference learning algorithm can be applied at the conclusion of each iteration to update the current policy:

Input: the policy π to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

The policy consists of seventy-two values for each user and each process. There are eight disks and nine possible actions for each disk. One action to initialize the data coming from a particular user or process to belong to that particular disk. One action to continue to service requests from a particular user or process to the current disk. Seven actions to migrate the user of process from the current disk to one of the other disks. This requires maintaining an expectation of reward for each action. Additionally, the disk which a given user or process most recently preferred must be maintained.

Whenever a data location is encountered the Related Data Concentration application chooses a disk which the request should be serviced from based on the disk which is expected to result in the best possible reward across the given user and process policy space. This choice is recorded such that the expectation can be updated during the at the conclusion of a single iteration once the Normalized Disk Idle Time has been calculated.

This choice is passed on to the DiskSim interface which will determine an action as defied in Section 3. If the passed disk differs from the current location of the data it will be migrated appropriately based on the type of I/O operation.

4.2 Measurement and Results

The Related Data Concentration application parsed and simulated traces iteratively until convergence. Each iteration contained 100,000 I/O operations and concluded in a single policy improvement step. Convergence was defined as an insignificant change in Normalized Disk Idle Time for at least 50 iterations. As summarized Figure 4.2.1: Around 955 iterations were required before convergence. Each operation had an average size of 87 bytes. This represents a total of 8.3 gigabytes of data processed. Improvement in Normalized Disk Idle accelerated around iteration 600. An initial penalty was incurred and around 400 iterations were required to overcome this penalty. This penalty was the result of additional move operations that are not present in the baseline.

As summarized in Figure 4.2.2: Initially I/O operations are distributed evenly across disks. At the time of convergence 2 disks received markedly fewer I/O requests and 2 disks received significantly more I/O requests. The hottest disk received around 4 times the number of requests as the coldest disk. This parallels the expected results of Popular Data Concentration. Unlike PDC there exist disks with relatively similar levels of request activity. This is believed to occur as a result of second order correlations between data, users, and processes where a user or process may access some frequently accessed

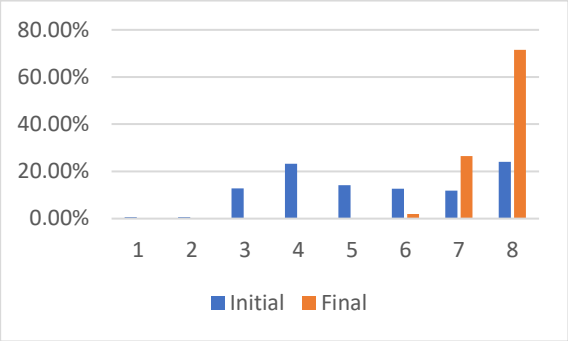

Figure 4.2.3: Distribution of data across disks touched by an average user in the initial run and after convergence

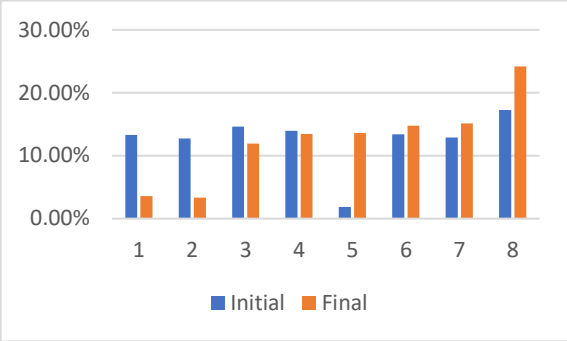

Figure 4.2.4: Distribution of data across disks touched by an average process in the initial run and after convergence

data and other infrequently accessed data. However, anytime one piece of data is accessed, other pieces are significantly more likely to be accessed based on the usage patterns associated with a particular user or process. In this case it may be advantageous to collocate strongly correlated files that have different access frequencies.

As summarized in Figure 4.2.3: Individual users access a small subset of all available data. With only 3 drives being required to service all of the requests made by a particular user within the given timeframe. Over 75% of all requests were serviced by a single drive for this particular user. Comparing these drives against those in Figure 4.2.2 reveals that the drive that serviced the majority of requests for this user received an average number of I/O operations. The drive that serviced almost all of the remaining requests was one of the two hot drives. From this it can be inferred that a quarter of user requests was made to hot files and the rest to files that are likely only accessed by the user them self or by a subset of users.

As summarized in Figure 4.2.4: Process behavior is more difficult to characterize then user behavior. The access patterns of process data are closely correlated with the overall system behavior. The average process is likely to interact with multiple users and access a diverse range of files. Even so, accesses to two disks were infrequent. One of these disks is a cold disk in the overall system, the other an average disk. It is difficult to infer from the available data why this is the case.

5 Conclusion

For a homogeneous disk the model is able to improve the energy efficiency from baseline by up to twenty percent. The project uses real world data collected from FIU traces which contains traces belonging to different user, using different processes. It was used by reinforcement learning algorithm with 72 parameters and reward system to place and migrate files in the disks to improve energy efficiency. The developed reinforcement learning algorithm learns the disk access pattern and migrates hot and cold files to improve energy efficiency through the rewards from each run. The trace files were converted into DiskSim readable format and an interface was developed to emulate file system to overcome the statelessness of DiskSim. DiskSim was simulated using the trace files to produces the idle times of each disk and it is used as an input to reinforcement learning algorithm to generate the reward to learn the disk access pattern.

The reinforcement learning algorithm successfully learns the disk access pattern to migrate files to improve the energy efficiency. The disk access pattern also can be used to improve the performance, reliability and availability of disks, by efficient replication and localization of data. This also can help the companies decide the location and number of disk arrays required, which in turn helps in efficient utilization of disk arrays. Reduction in the number of disk arrays results in increased revenue and decreased energy consumption. Learning disk access pattern can also give insight on the usage pattern of individual user, which can be used in advertisement and recommendation system. Hence outcome of the

reinforcement learning which is disk access pattern can be used in many applications which is beyond the scope of this project.

6 Future Work

The developed system is able to successfully improve the energy efficiency from base line by up to twenty percent. Currently the developed system uses only homogeneous disks, we would like to extend the system to heterogeneous disks. The reinforcement learning algorithm considers all the disks similarly and do not priorities any disks over another with respect to performance, with the addition of disks with different performance and types the algorithm needs to priorities disks with higher speed for hot files over disks with lower speeds. All the files with low usage can be placed in slower disk speeds, this has two implications one is performance improvement and higher disk idle time. One interesting factor that needs to be considered is the difference in energy consumption between the heterogeneous disks. The disk with higher speed can utilize more power and can even, even-out the gain in disk idle time. So, we would like to explore more about heterogonous disks and how the current system can be extended. The current system does not utilize cache memory, but cache size can affect the disk usage. We would like to explore the effect of cache size and levels of cache on energy consumption through the disk. The developed system uses the workload generate from one source. The model may have performed well since the workload is similar. We would like to extend our current model and explore how diversity and complexity in the workload is affecting the energy consumption and how it can be improved.

The current model migrates the files between disks depending on the file access pattern to increase energy efficiency, instead of moving we can also copy all the hot files to disks with higher performance or we can just increase the number of a hot file by replication. This can increase the energy efficiency significantly and can also affect the availability and reliability of disks. We would like to explore more on how file replication and disk performance is affecting the energy efficiency. The current model uses a simple reinforcement learning algorithm with 72 parameters with reward system, we want to experiment more using different parameters and check if we can improve the energy efficiency of the system. In our current model we have just focused on the energy efficiency and we have not observed how it is affecting the performance of the system. In a real-world energy efficiency and performance go hand in hand, increasing the efficiency at the cost of performance is of limited use in a real world. If the built system is reducing the performance, it cannot be used in a real-world scenario unless the improvement in the energy efficiency is compensating for the loss in performance. We want to extend the model to explore how the improvement in energy efficiency is affecting the performance of the system due to additional overhead created by file migration. Machine learning algorithms can be used to increase efficiency of the policy in uncertain conditions. With help of previous knowledge machine learning technique can help in making better decisions during uncertain conditions.

References

[1] The DiskSim Simulation environment (4.0)

[2] Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers using Reinforcement Learning by Fahimeh Farahnakian, Pasi Liljeberg, and Juha Plosila

[3] Towards energy-aware scheduling in data centers using machine learning by Josep L. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà and Jordi Torres

[4] Massive Arrays of Idle Disks for Storage Archives by Dennis Colarelli, Dirk Grunwald

[5] Energy-Aware Data Prefetching for Multi-Speed Disks by Seung Woo Son Mahmut Kandemir

[6] Dynamic Power Management Using Machine Learning by Gaurav Dhiman and Tajana Simunic Rosing

[7] A Dynamic Disk Spin-down Technique for Mobile Computing by David P. Helmbold, Darrell D. E. Long and Bruce Sherrodt

[8] Energy Efficient Prefetching and Caching by Athanasios E. Papathanasiou and Michael L. Scott

[9] Hibernator: Helping Disk Arrays Sleep through the Winter by Qingbo Zhu, Zhifeng

Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton and John Wilkes

[10] Energy Aware Resource Scheduling Algorithm for Data Center using Reinforcement Learning by Jingling Yuan, Xing Jiang, Luo ZhongHui Yu

[11] Pinheiro, E., & Bianchini, R. (2004, June). Energy conservation techniques for disk array-based servers. In Proceedings of the 18th annual international conference on Supercomputing (pp. 68-78). ACM.

[12] Lee, D. K., & Koh, K. (2009, October). PDC-NH: Popular data concentration on NAND flash and hard disk drive. In Grid Computing, 2009 10th IEEE/ACM International Conference on (pp. 196-200). IEEE.

[13] Karakoyunlu, C., & Chandy, J. A. (2012, June). Techniques for an energy aware parallel file system. In Green Computing Conference (IGCC), 2012 International (pp. 1-5). IEEE.

[14] Buzbee, B., Wang, W. E. I. S. U., & Wang, A. A. (2013). Power-saving approaches and trade-offs for storage systems. Florida State University.

[15] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (Vol. 1, No. 2). Cambridge: MIT press

[16] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Lillicrap, T. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv preprint arXiv:1712.01815.