Energy Efficiency for HDFS

Abhishek Das
Dept. of Computer Science and Engineering
PES University
Bangalore, India
abhishek262014@gmail.com

Bhargav SNV
Dept. of Computer Science and Engineering
PES University
Bangalore, India
bhargavsnv100@gmail.com

N Sanketh Reddy
Dept. of Computer Science and Engineering
PES University
Bangalore, India
sankethn1947@gmail.com

Phalachandra H L
Center for cloud computing and Big Data
PES University
Bangalore, India
phalachandra@pes.edu

***Abstract*— The rapidly increasing growth of big data is leading to increased usage of distributed computing and clusters as this can enable tasks to be split and processed in parallel resulting in higher computational efficiency. Environments like Apache Hadoop use Distributed file systems like the HDFS to support large clusters of typically commodity hardware to achieve the same. Commodity hardware present in HDFS consumes significant energy even when not in an active state, and considering our fragile ecosystem, reduction of the carbon footprint and conservation of energy in these environments is the focus of our work.**

We have evolved on a dynamic approach to Hadoop Distributed File Systems (HDFS) that provides an energy efficient paradigm for Hadoop clusters by segregating the servers in the cluster to two "Zones", a "Hot-Zone", and a "Cold-Zone". We use energy efficient storage devices (SSDs), to store data that is frequently used for the hot zone. The Cold-Zone would have lesser energy efficient devices (HDDs) and would be used to store data that is not frequently used. The Hot-Zone would always be in an active state to compute while the cold zone would have the HDDs to be in a low power consumption state. We have also enhanced the block placement policy for HDFS to support dynamic block transfer among the two zones.

Keywords—HDFS, Energy efficiency, SSD and HDD, Cloud computing.

I. INTRODUCTION

The exponential growth of scientific and business data due to the rapidly increasing volume and complexity of data because of growing mobile data traffic, cloud-computing traffic and burgeoning development and adoption of technologies including IoT and AI, has resulted in increasing volume of data to spur revenue growth from ~37 Billion in 2018 to a projected 105 Billion in 2027 [1]. This has led to ~2.5 Exabyte of data being generated and stored on a daily basis.

This explosion in data volume calls for an efficient approach to processing Big Data. Hadoop provides a software framework for distributing and running applications on clusters of servers for Big Data. It consists of hardware and software infrastructures to process the collected data which is spread among various servers. Challenges relating to energy consumption and power usage, however, are faced due to the increase in compute nodes and processing power following the increase in collected data.

The increased demand for more energy to power data centres brings about substantial environmental impacts, thereby increasing the carbon footprint. Server maintenance and cooling also lead to enhanced operational costs. Furthermore, even idle nodes remain powered on to ensure data availability [2]. We focus on reducing energy consumption in HDFS clusters to improve cost efficiency and lower its impact on the environment.

Our work aims to optimize energy consumption in HDFS clusters and propose an energy efficient approach. We look to dynamically cluster nodes based on the data access patterns, integrate the different types of disks with energy characteristics and also dynamically tweak the replication factor based on the access and usage patterns while balancing the risk of data loss and optimization for enhanced energy efficiencies. We have leveraged the HDFS-Replication simulator, enhanced it for our HDFS algorithms to establish the efficacy of our approach.

II. RELATED WORK

A majority of existing techniques to improve energy efficiency of HDFS clusters is to configure the cluster into an active and in-active set of nodes based on different criteria of replication factor, workload and data access pattern [3]. Use of these methods negatively impact the performance, availability and fault-tolerance of the cluster as these cannot be varied later. Our approach reduces energy consumption of a cluster without having such negative side-effects.

There has been work on energy efficient data placement and cluster reconfiguration (balancing) which dynamically scales clusters in accordance to the workload imposed on it in [4], but this methodology requires frequent dynamic reconfiguration of the cluster which is a resource heavy operation. The approach for hybrid multi zone layout [5] is implemented with added policies to shift and balance data among Zones. There has also been considerable research about the use of different types of memory storage devices to improve energy and compute efficiency [6] and power proportional data placement strategies [7].

Our paper is subsequently organised with Section II summarizing the Hadoop Distributed File System, memory storage devices used in servers and the differences between these storage devices. Section III, describes the custom zone layout implemented for our approach. Section IV details the

algorithm used for data transfer between zones. Section V describes details of the simulator used and Section VI comprises the results of the study.

III. BACKGROUND

A. *Hadoop Distributed File System (HDFS)*

Hadoop Distributed File System is an integral part of a majority of cloud distributed Eco-systems. HDFS stores data as a collection of blocks. Data which is being written into the file system is broken down into blocks of fixed size and each block has duplicates corresponding to the replication factor. These blocks are distributed across the cluster such that no two replicas are stored in the same compute node. This will help in ensuring that even in case of failure of associated commodity hardware, the data is not compromised by corruption/ or loss and can be recovered from the duplicates present in the other nodes. This introduces redundancy and leads to fault tolerance of data.

HDFS implements master/slave architecture. A typical HDFS cluster is configured for a single NameNode, which manages the entire file system namespace and regulates access to files and data operations along with the supervision of data nodes (compute nodes which store data).

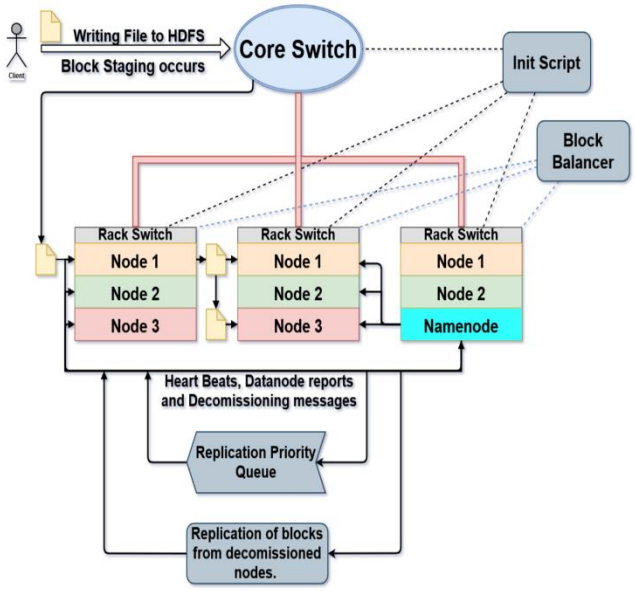

Figure 1. Default Hadoop Architecture

B. *Memory Storage devices in servers*

There are different classes of storage devices with differences in the attributes like the storage sizes, the latencies involved and the cost. We have considered two types of disks for our experiment with well-known characteristics as below.

1) *Hard Disk Drive (HDD)*

Hard Disk Drives are non-volatile magnetic storage devices which are used to store data as being part of the data storage devices in Servers.

HDDs support high capacity, and support different modes of operation like the active, idle, standby and sleep. Each of these modes have different power characteristics and vary in terms of energy consumed. These different modes can be dynamically set on the devices and can be used to manage the total energy consumed.

2) *Solid State Drive (SSD)*

Solid State Drives are non-volatile storage devices that store and continuously retrieve data from solid-state flash memory.

SSDs consume significantly lower power than the HDDs but have significantly faster read and write speeds, smaller form factor but are significantly more expensive than HDDs.

IV. CUSTOM ZONE LAYOUT

We have customized our HDFS cluster layout into two well defined zones, the Hot zone and the Cold zone [5]. The zones are divided on the basis of frequency of data access.

Data is again divided as hot and cold data. Hot data is frequently accessed data, while the rarely or infrequently used data is categorized as cold Data. Hot data will reside in the hot zone while the Cold data will reside in the cold zone.

The clustering into hot and cold zones is to optimize the energy consumption. The frequently accessed hot zone can continue to have its servers and disks in an active state. The servers or the HDDs with the cold data can be taken to a low power state to save energy.

By keeping fewer nodes running at higher levels of computation, we can also reduce the energy consumed due to idle time. More hot nodes are brought up as and when more computation is required.

We also use two separate block balancers, one for each zone. Since blocks stored in the cold zone are not frequently accessed, they are less prone to data loss. Considering the above statement with the fact that cold nodes use HDDs which consume more power and time for read/write operations, we can optionally choose to reduce the replication factor in the cold zone.

The default replication factor of 3 can be used in the hot zone while the cold zone could use a replication factor of 2. Although we reduce the replication factor in the cold zone, we keep it higher than 1 so as to ensure fault tolerance and resilience of data. This way data even if critical but rarely accessed, is still preserved without loss. Thus, the two configured block balancers can implement different replication factors for blocks according to the zone the block belongs to.

A. *Hot Zone*

This zone makes use of SSDs as the storage device. Nodes in this zone store the frequently used Hot Data. Due to the use of SSDs, reads and writes are faster and each of these read/write operations consume lesser power.

This zone is always active, Hot data is concentrated among nodes in this zone. The nodes in the hot zone are built for high computation and lower energy consumption. Given

that the SSDs are more expensive, the Hot Zone is structured (percentage of cluster nodes to be considered as Hot) in such a way so as to be able to handle computation of Hot Data without storing unnecessary data that would waste precious SSD space. Thus, the approach is to keep the smallest percentage of nodes as Hot Nodes to reduce the cost involved.

Statistically, almost 80% of data stored in clusters is almost never used after a certain period of time (approximately 1 month after creation). Data is used mostly close to the time period since it was created. Thus, our implementation of Hot zone will store this data as it is being used frequently and as soon as it goes cold, we move these data to the cold zone. This avoids wastage of space in the hot zone. The cold zone on the other hand is built precisely to store data that is not frequently used.

B. *Cold Zone*

This zone comprises HDDs being used as storage devices. Infrequently accessed data or cold data is stored in this zone as performance is traded off for higher energy conservation in this zone. This zone would have the servers stay in an inactive power-saving state whenever possible.

Power management scheme applied is to transition the cold-zone servers towards a low power consuming and inactive power mode to reduce server active idle times. This zone maximizes use of already powered-on servers by a placement policy which places most of the cold blocks in the initial servers defined by an order.

Blocks of data are moved to the cold-zone(from the hot-zone) as the block activity decreases to save energy. Upon transfer/access of a block, only the target server will wake up from its inactive power-saving state. Data inside the cold zone will have a lower replication factor than the hot-zone.

V. ALGORITHM IMPLEMENTATION

We attempt to reduce power consumption in our cluster by reconfiguring its block placement policy for a more qualitative approach suited to the needs of our modified cluster which operates based on zones. Our block placement policy is derived from the default approach and enhanced to work under conditions we have set up for our energy efficient model. Our policy is divided into two types: frequent and infrequent transfer. Cluster administrators can choose between the above two policies based on the type of workload their cluster deals with.

A. *Heartbeat Information*

Let *H(d)* be the heartbeat information sent out by each data node, where *'d'* corresponds to a unique node ID. *H(d)* for a data node would contain information about the blocks in that data node along with metadata (about the block). This metadata tells us which blocks have turned cold based on the last accessed time. This last accessed time is considered as a parameter *'L'*. Thus the information a heartbeat sends *H(d)* is read by the data node, which in turn determines if blocks have gone cold or not.

Blocks are determined as hot or cold when the time difference between the current time and last accessed time *L* is compared with the value *'LT'* set globally over the cluster and can be overridden for particular servers. *LT* defines the maximum time period a block can stay unused before turning cold. If *(CurrentTime – L) > LT*, then the block is deemed cold. If *(CurrentTime – L) < LT*, the block is still hot.

B. *Transition Script*

This script running in the NameNode would process the heartbeat information *H(d)* from each data node and instruct the NameNode to shift blocks accordingly based on the approach chosen i.e., frequent/infrequent (these approaches are further explained in detail in the following section). Thus heartbeats from all the data nodes are analysed and cold blocks are then transferred to the cold zone via instructions from the NameNode. This could be thought of as a heartbeat checker daemon specific to the parameter *L*. The NameNode would be instructed to transfer cold blocks which have been marked for transfer (by one of the two algorithms) and handle the replication factor in each zone after the transfer.

C. *Infrequent Transfer*

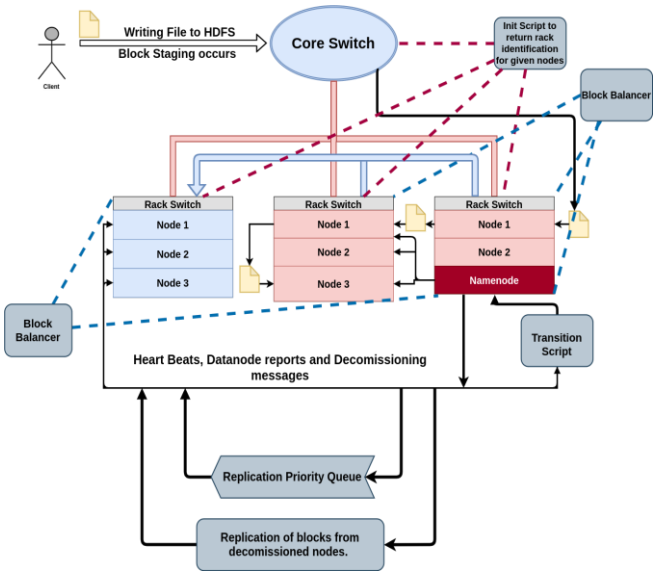

Figure 2. Customized block placement policy - Infrequent Data Transfer.

The above figure represents a cluster set-up. The nodes marked in red comprise the hot zone whereas the nodes marked in blue comprise the cold zone. Each zone has its own block balancer due to the different replication factor in each zone. A transition script runs in the NameNode marked by the color red.

Algorithm

This algorithm is based on the value of cold blocks in a particular data node in the hot zone at a given time. Blocks turn cold based on the condition (CurrentTime – L) > LT and thus cold blocks accumulate on each data node with time. The transition script checks if the percentage of cold blocks which have been accumulated on a particular data node have crossed a threshold value (T).

If this globally set threshold value becomes lesser than the size of blocks accumulated over a certain data node, the cold blocks are then flushed to the cold zone. The cold blocks being transferred lose their replicas in the hot zone and are then replicated in the cold zone as required. Thus when the amount of cold blocks in any data node in the hot zone reaches the threshold value T, the cold blocks are

flushed to the cold zone. This happens continuously to each data node in the hot zone as more and more cold blocks form.

The flushing process wakes up the cold zone servers and the NameNode assigns the cold blocks coming from the hot zone to its respective places in the cold zone servers. The resulting blocks are then balanced in both the zones using block balancers.

We look at this as an infrequent method, since this approach is more effective when the frequency of blocks going cold is low.

D. *Frequent Transfer*

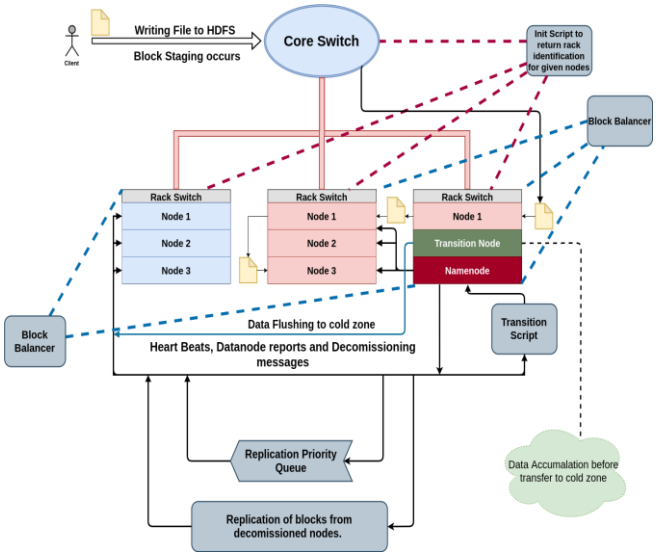

Figure 3. Customized block placement policy - Frequent Data Transfer

This set-up has a similar configuration to the one in Figure 1, but with an extra transition node for cold data accumulation before flushing it to the cold zone. The transition node is represented in green.

Transition Node

The transition node is unique based on its functional aspect. It only stores cold blocks of data and is composed of HDD(s). It is an intermediate node for the flushing of data from the hot to the cold zone. HDDs are used here as it only consists of cold data to improve cost efficiency.

Algorithm

Blocks which turn cold are transferred to the transition node immediately. The transition node accumulates cold blocks until it reaches a threshold value T. Once the threshold value is reached, the node flushes all cold data it contains into the cold zone. It wakes up the required number of servers in the cold zone according to the number of blocks in the transition node for data transfer. Thus, data transfer happens through an intermediary transition node.

This is termed as a frequent method since this approach is more effective when the frequency of blocks going cold is high.

VI. SIMULATOR DETAILS

We have enhanced the simulator built by Pedro Álvarez-Tabío Togores' (HDFS-Replication simulator) [8]. The simulator handled default policies of HDFS and accurately simulated HDFS functioning and failure responses. In addition, modules were added to better fit our modified policies and features like block placement, block transfer and balancing, etc.

The Modules added include:

- A class *power* which contains implementations of power measurements. This helps measure power during reads, writes, boot-up, sleep, etc. for both types of nodes SSDs and HDDs.
- Modified data node class which implements the use of SSDs and HDDs.
- Custom block placement policies.
- Custom block balancers.
- Modified *block* and *blockInfo* which allows us to record and utilize metadata of blocks.
- Transition scripts to handle conversion of hot data to cold data and transfer between zones.

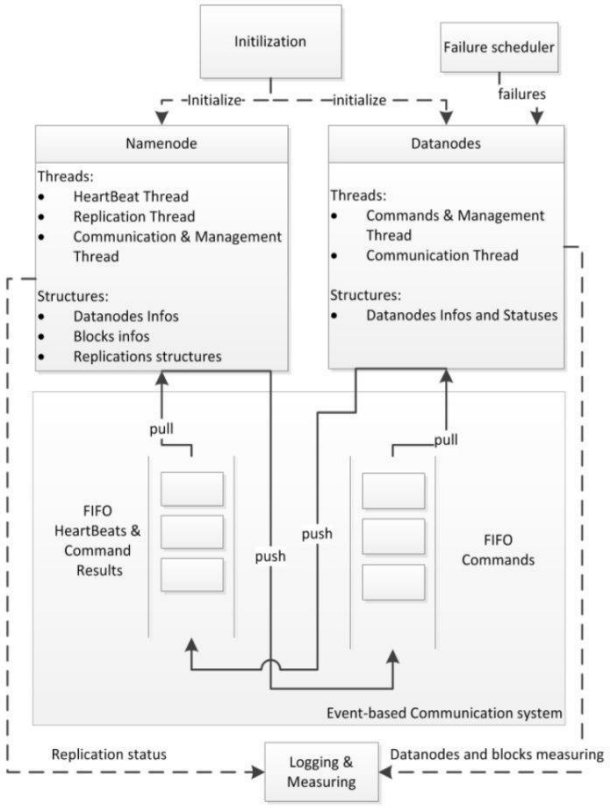

Figure 4. Simulator architecture

By making the above stated modifications to the simulator, we were able to accurately test functionality and measure energy consumption for various workloads and cluster configurations. The simulator would be used to measure energy consumption via the conduction of several test runs on it. Each test run takes in a configuration file as input specifying the number of replicas in the hot zone, number of replicas in the cold zone, block size, heartbeat interval,

number of nodes in the cluster, percentage of the cluster which would comprise the hot zone, simulated percentage of data going cold, bandwidth and time threshold for deeming blocks cold. These are the parameters the simulator was fed and allowed to run. Each run was associated with a specified number of blocks turning cold and being moved into the cold zone based on the parameters mentioned above. Data blocks were accessed at random and power consumption to keep the servers running were noted. Power consumption has been divided into various sections for better analysis of the domains where power is consumed.

VII. RESULTS AND DISCUSSION

We ran our modified simulator which has been evolved to use our block placement strategies and adhere to the multizone approach. Our approach towards establishing our architecture and algorithms is to first show that in the HDFS cluster, the total energy consumption varies with the size or number of nodes in the hot zone and cold zones and should be lower with our multi zone approach. This is reflected in the set of test run results seen as part of section A.
We then look at the energy contributions during each of the various stages of a lifecycle in Section B. This profiling of the energy consumed in each stage allows for the analysis of the energy consumed in detail. In order to study the impact of the volume of data on the total energy consumption, Section C details how energy consumption varies with increase in the number of blocks. We also evaluate how the number of nodes in a cluster impacts the energy consumption in our cluster as part of section D.

A. *Energy consumed vs Hot Zone percentage*

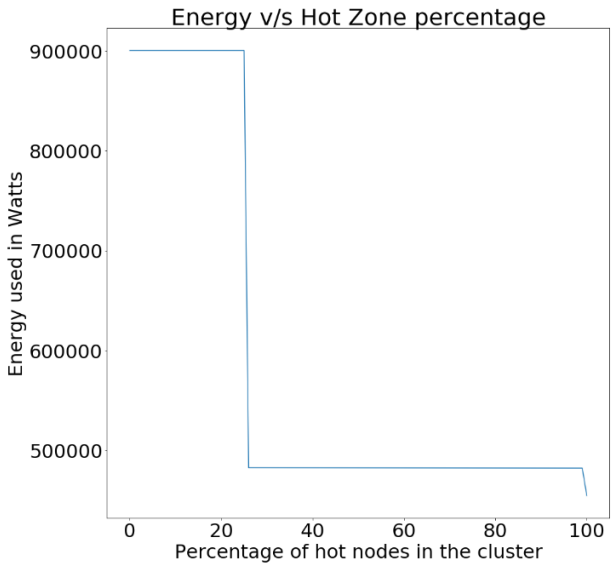

Figure 5. Power consumed as a function of Hot Zone percentage variation

The simulator was run for multiple iterations to get the total power consumption as a function of the percentage of hot nodes in the cluster i.e. nodes consisting of SSDs. The number of hot nodes were varied keeping all the other parameters constant.

The above Figure 5 shows how the total energy consumed varies when we vary the Hot-Zone percentage in the cluster. Here there is a sudden drop in the energy consumption at around Hot Zone % = 30 because a minimum of about 30% of the cluster must belong to the hot zone to handle peak workloads, else it defaults to the default Hadoop configuration (no hot/cold zone). The energy consumed then slowly decreases after this dip. The second dip at HZ% 100 is because there are no HDDs in the cluster, this indicates that all the nodes in the cluster make use of SSDs, thus saving a significant amount of energy.

B. *Energy per cluster life cycle*

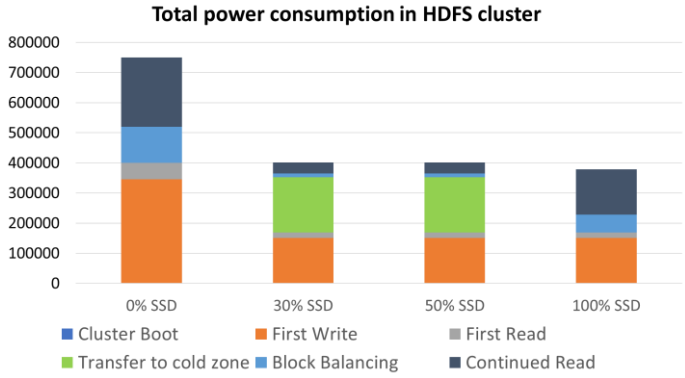

Figure 6. Total power consumption per life cycle

This Figure 6 shows the total energy consumed for the HDFS over one life-cycle. The simulator was run multiple times with varying ratios of hot zone: cold zone. All other parameters were kept constant. The life-cycle of the simulator run has the following stages:

1. *Initial Boot*: This is the energy consumed in bringing up all the nodes in the cluster. This tends to increase with the increase in the number of HDDs as storage devices, as HDDs consume more power during boot than SSDs.
2. *First read and write*: This simulates the blocks being created and placed into the cluster. Majority of data is placed into the cluster and is used within a small period of time after which it goes cold. The first read and write simulates data being read as soon as it is first written into the cluster.
3. *Transfer to cold zone*: After a certain period of time, the blocks that were written are not used as frequently. Their usage becomes minimal and thus they turn cold. This stage simulates energy consumed when transferring blocks from the hot zone to the cold zone. This occurs only in the case where a Hot and Cold zone exists. This transfer does not occur with 0% or 100% SSD as the former lacks a Hot Zone while the latter lacks a Cold Zone.
4. *Block balancing*: This stage refers to balancing of blocks after transfer. In both cases (use of hot/cold zone and without), blocks are corrupted/moved frequently. Thus a block balancer distributes these blocks evenly across a cluster. Energy consumed during these writes are higher when done on HDDs. Movement of these blocks are reduced when they are separated as hot and cold. The new block balancer kicks in and tries to accumulate blocks within a lesser number of SSDs so as to keep a lesser number of nodes active. This maintains

redundancy and data integrity while saving on unnecessary wastage of node active time and space.

5. *Continued read*: This stage refers to the continued read of data after the majority of blocks have gone cold and been transferred to the cold zone. This involves read of hot data along with the occasional minimal read of cold data.

C. *Energy vs Data Size*

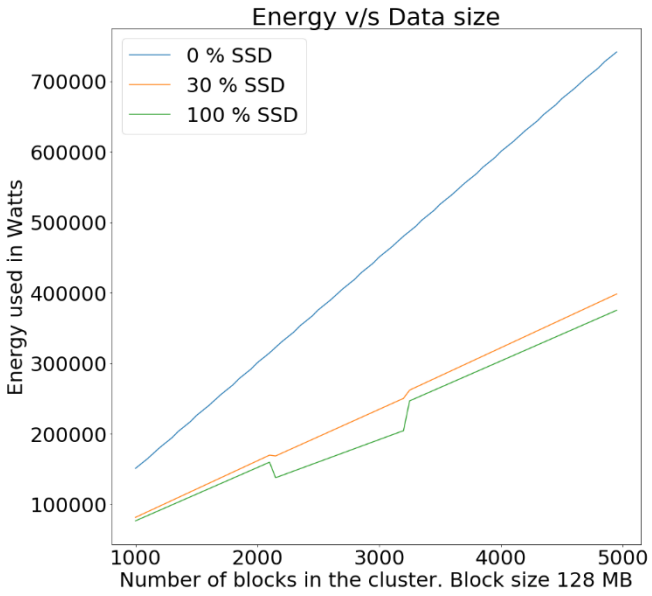

Figure 7. Power consumed as a function of variation of data stored in the cluster

The above Figure 7 shows the variation of energy consumed when the total amount of data (number of blocks) in the cluster is varied. The simulator was run with varying percentages of hot zone to output the energy used in Watts as a function of the number of blocks in the cluster. Again, all other parameters of the simulator were kept constant. As is clearly visible from the graph, the custom layout, policies and transfer algorithm show significant decrease in energy consumption when compared to the default working.

The depressions in the two lines corresponding to 30% Hot Zone and 100% Hot Zone are due to the blocks fitting perfectly into the active nodes. This indicates that the number of blocks that exist are distributed evenly across the active nodes and these nodes are utilized to their full capacity. For example, if there existed 5 hot nodes with each node having the capacity to store a maximum of 3 blocks, and the total number of blocks needed to be stored are 9, these blocks are evenly distributed across 3 nodes and the fourth and fifth node can be put to sleep. But in the case when an extra block exists, i.e., the total number of blocks is 10, we need an additional node just to store one block.

D. *Energy vs Number of Nodes*

The Figure 8 shows the variation of energy consumed by the cluster as a function of the number of nodes in the cluster. A key point to note here is that with increase of nodes, the simulated size of the workload was also increased. For every node added, 200 additional blocks were placed in the cluster. This shows that the custom zone implementation along with custom policies and a transfer algorithm can easily keep up when a cluster is scaled. The dips here too signify the even distribution of blocks in the hot zone.

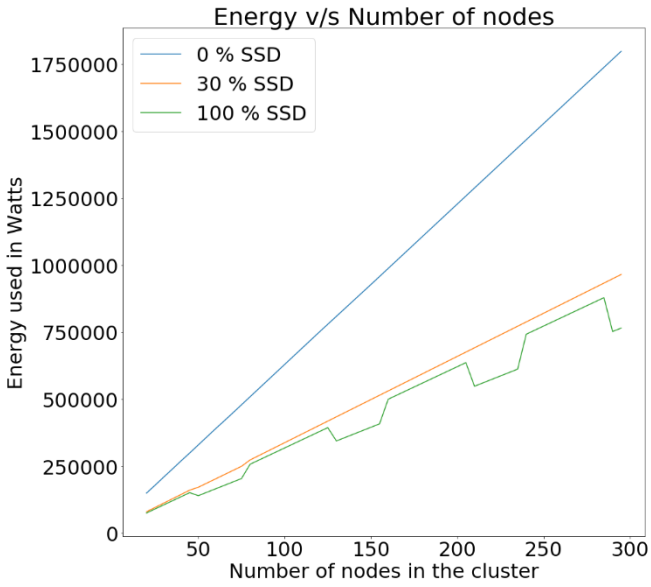

Figure 8. Power consumed as a function of variation of number of nodes in the cluster

VIII. CONCLUSION

Our work attempted to increase the energy efficiency of Hadoop clusters to lower the environmental impacts caused due to high power consumption by the cluster. A hybrid and multi-zoned Hadoop cluster model was discussed.

The results in A, B, C and D shows that using SSDs in the hot zone positively impacts the workload processing and the energy consumed. We also rationalized on the observed results by profiling and analyzing the energy consumed. We then analyzed the impact of the size of the data and the number of nodes in the cluster.

As was seen, optimizing the hot zone to an optimal number of ~30% of nodes and usage of our approach will get the most return in terms of the energy consumption with cost consideration.

Summarily we demonstrate with our limited experimentation, significant reduction in power consumption upon the adaption of our model which results in cost savings for keeping the servers running.

REFERENCES

[1] https://www.globenewswire.com/news-release/2020/03/02/1993369/0/en/Big-Data-Analytics-Industry-Report-2020-Rapidly-Increasing-Volume-Complexity-of-Data-Cloud-Computing-Traffic-and-Adoption-of-IoT-AI-are-Driving-Growth.html (2020).

[2] Luiz André Barroso and Urs Hölzle. The Case forEnergy-Proportional Computing.Computer, 40(12), 2007.

[3] Nidhi Tiwari Prof. Umesh Bellur Prof. Maria Indrawan Dr. Santonu Sarkar. Improving the Energy Efficiency of MapReduce Framework (2014).

[4] Nitesh Maheshwari, Radheshyam Nanduri, Vasudeva Varma. Dynamic Energy Efficient Data Placement and Cluster Reconfiguration Algorithm for MapReduce Framework (2012).

[5] Rini T. Kaushik, Milind Bhandarkar. GreenHDFS: Towards An Energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluster (2010).

[6] Ivanilton Polato, Fabio Kon, Denilson Barbosa and Abram Hindle. Hybrid HDFS: Decreasing Energy Consumption and Speeding up Hadoop using SSDs (2015).

[7] Hieu Hanh LE, Satoshi Hikida, Haruo Yokota. An Evaluation of Power-proportional Data Placement for Hadoop Distributed File Systems (2011).

[8] Corentin Debains, Pedro Alvarez-Tabio Togores, Firat Karakusoglu: Reliability of Data-Intensive Distributed File System: A Simulation Approach (2012).