Energy Efficiency for HDFS

Phalachandra H L
Dept. of Computer Science and Engineering
(Professor)
PES University
Bangalore, India
phalachandra@pes.edu

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

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

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

***Abstract*—With the growth of big data, the need for distributed computing and clusters is now higher than ever. By making use of distributed computing, tasks can be split and processed in parallel resulting in higher computational efficiency. But the energy consumed by these clusters is very high. Majority of the energy is wasted due to servers being in an idle state and servers use of cheap, energy hungry hardware. Our proposal is a smarter and more dynamic HDFS that provides an energy efficient approach for clusters using HDFS.**

The approach to saving energy is to split the cluster into two "Zones", a "Hot-Zone", and a "Cold-Zone". The Hot-Zone would have energy efficient storage devices (SSDs) and would store data that is used frequently. 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 be active 24/7 to compute while the cold zone would be in a low power state.

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

I. INTRODUCTION

The exponential growth of scientific and business data has resulted in the evolution of cloud computing.

Cloud computing has been rising in popularity and it's usage has been quadrupled in the last 4 years. According to recent research [5], there was just over 1 Exabyte or 1024 Petabytes of data stored in the cloud. Now, Google Cloud Storage alone has about 30 Exabytes of data and there exists even bigger giants in the industry like Amazon Web Services(AWS), Microsoft Azure and many more equally efficient cloud storage providers like Alibaba Cloud, IBM Cloud, etc.

These cloud companies currently provide the resources that data-intensive computing needs which include advertising optimizations, user interest predictions, mail anti-spam detection and many such similar data analytics. It's also currently being used for live streaming of data which turned out to be the main reason for the success story of one of the most famous media-services provider, Netflix.

The implementation that the current cloud providers have adopted to structure their data-centres for maximum efficiency of data storage/retrieval is using the HDFS architecture i.e., the Hadoop Distributed File System and parallel processing of data for the effective data processing. However, this implementation faces tremendous energy consumption and associated cost concerns. With energy consumption becoming key issue for the operation and maintenance of cloud data-centres, cloud computing providers are becoming profoundly concerned.

So most of the cloud enterprises today are focusing their attention on energy efficient computing, motivated by high operational costs for their large scale clusters and warehouses. This power related cost includes investment, operating expenses, cooling costs and environmental impacts.

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

A. Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) is a key part of many cloud Eco-systems, as it provides a reliable means for managing pools of big data and supporting related big data analytics based applications. HDFS supports the rapid transfer of data between compute nodes.

When HDFS takes in data, it breaks the information down into separate blocks and distributes them to different nodes in a cluster, thus enabling highly efficient distributed processing. Moreover, the Hadoop Distributed File System is specially designed to be highly fault-tolerant. The file system replicates each block multiple times and distributes them across several nodes, placing at least one copy on a different server rack than the others. As a result, the data on nodes that crash can be found elsewhere within a cluster. This ensures that processing can continue while data is recovered.

HDFS uses master/slave architecture. In its initial incarnation, each Hadoop cluster consisted of a single NameNode that managed file system operations and

supporting Data Nodes that managed data storage on individual compute nodes.

Because HDFS is typically deployed as part of very large-scale implementations, support for low-cost commodity hardware is a particularly useful feature. Such systems, running web search and related applications, for example, can range into the hundreds of petabytes from thousands of nodes. They must be especially resilient, as server failures are common at such a large scale.

B. *Memory Storage devices*

1) Hard Disk Drive (HDD)

An HDD uses magnetism, which allows you to store data on a rotating platter. It has a read/write head that floats above the spinning platter for reading and writing of data. The faster the platter spins, the quicker an HDD can perform.

An HDD also consists of an I/O controller and firmware, which tells the hardware what to do and communicates with the remaining system. The hard disk is made up of a number of spinning magnetic platters that store data and a number of reading/write heads on mechanical arms that move on the surface of the platters.

To read or write data at a certain sector of a platter, the head requires to move to the appropriate position and then need to wait for the sector to pass underneath it when the platter rotates.

2) Solid State Drive (SSD)

Solid State Drives (SSDs) are a non-volatile storage device that stores and retrieves data constantly on solid-state flash memory. However, this data is stored on interconnected flash memory chips instead of platters, which makes them faster than HDDs. It provides better performance compared to HDD [2].

An SSD works differently compared to an HDD. It uses a solid-state medium, typically NAND (often known as flash). Data is written to or read from the NAND by a controller, which is consider the brains of the device. With SSD, there is no variable seek time or rotational latency because all the parts of the SSD can be accessed in the same amount of time.

SSD read and write speeds are uneven, so data reads are very fast, but SSD write speeds are quite slower. That is because SSD storage is made up of individual NAND cells, which helps you to store one bit of data, and groups of cells are organized into pages. Moreover, groups of pages are organized into blocks.

C. *Difference between HDDs and SSDs*

- HDD has a slower speed for reading and writing data and SSD is faster at reading and writing data.
- HDD has higher latency whereas SSD has a lower latency.
- HDD supports fewer I/O operations per second (IOPS) while SSD supports more I/O operations per second(IOPS).
- HDD can produce noise due to mechanical movements on the other hand, SSD does not produce such noise.
- The moving parts of HDDs make them vulnerable to crashes and damages but SSD drives can tolerate vibration up to 2000Hz.
- The Key Difference between HDDs and SSDs that affects the algorithm is the fact that SSDs consume far less energy and are much more efficient than the HDDs but the cost of an SSD is approximately 5 times the cost of am HDD for the same amount of storage.

II. CUSTOM ZONE LAYOUT

In this section we shall describe the zonal layout of our customized HDFS cluster. The cluster has been divided into two well defined zones, the Hot zone and the Cold zone [1]. The zones are divided on the basis of frequency of data access.

Data is again divided as hot and cold data. Hot data resides in the hot zone. This is data that is frequently accessed. Cold data resides in the cold zone. This is data that is rarely accessed.

The purpose of dividing the cluster into hot and cold zones is to optimize the performance and reduce energy consumption by the cluster. Since only hot data is used frequently and cold data is often untouched, we need not keep the servers containing cold data active. They can be put to a low power state to save energy. The hot data is concentrated within nodes of the hot zone. These hot nodes perform all the computation necessary when the cluster is active. Only the required few nodes of Hot Zone are active while the rest of the hot nodes are shut down. These active nodes thus run at higher levels of computation.

By keeping fewer nodes running at higher levels of computation, we avoid wastage of both CPU power and Energy 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. 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 data that is used frequently (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. But this comes at the cost of price. SSDs are more expensive with respect to the alternative of HDDs. Thus 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 least percentage of nodes possible are made into Hot Nodes to avoid excessive cost overruns.

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, it is transferred over 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 of 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.

This zone is mostly inactive and has lower computational capacity than the hot zone. Power management schemes applied here will be gravitating towards transitioning the cold-zone servers towards a low power consuming and inactive power mode. Various volumes of cold data would be stored in the HDDs in this zone. The main idea here is to reduce server idle times and transition most of the servers into the inactive power-saving mode. 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 inactivity 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.

III. 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. Furthermore, 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 have 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 name node would process the heartbeat information *H(d)* from each data node and instruct the name node 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 name node. This could be thought of as a heartbeat checker daemon specific to the parameter *L*. The name node 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*

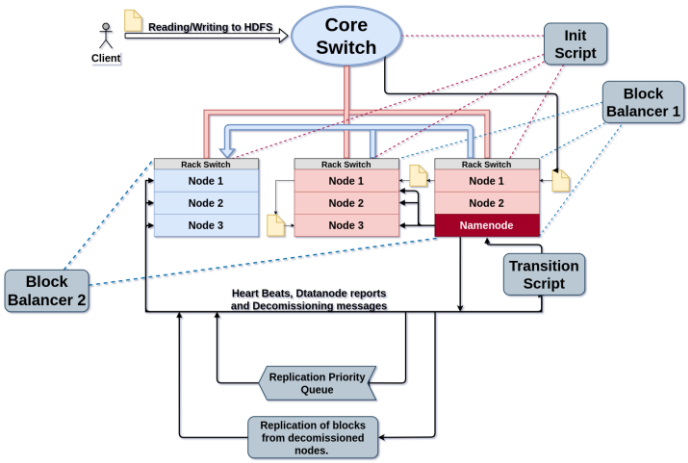

Figure 1. Customized block placement policy - Infrequent Data Transfer

This 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 name node marked by the colour red.

1) Algorithm explanation

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 off 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 name node 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.

This is termed as infrequent method since this approach is more effective when frequency of blocks going cold is low.

D. Frequent Transfer

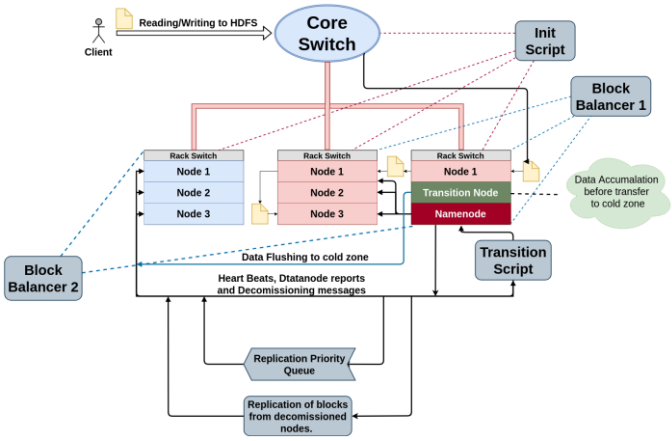

Figure 2. Customized block placement policy - Frequent Data Transfer

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.

1) Transition Node

The transition node is unique based on its functional aspect. It only stores cold blocks of data and is comprised 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.

2) Algorithm explanation

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 frequent method since this approach is more effective when frequency of blocks going cold is high.

IV. Simulator Details

The simulator is built over Pedro Álvarez-Tabío's HDFS-Replication simulator [6]. 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. 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.

By making said modifications to the simulator, we were able to accurately test functionality and measure energy consumption for various workloads and cluster configurations. The results of these evaluations are described in the following section.

V. Evaluation Results

A. Energy consumed vs Hot Zone percentage

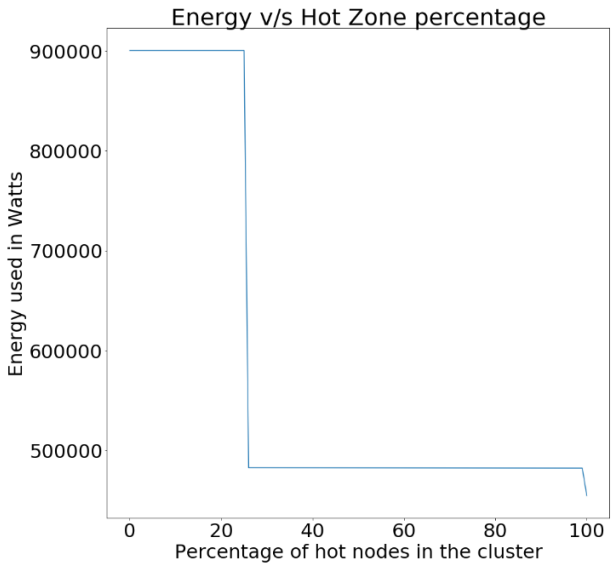

Figure 3. Power consumed as a function of Hot Zone percentage variation

The above graph 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 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

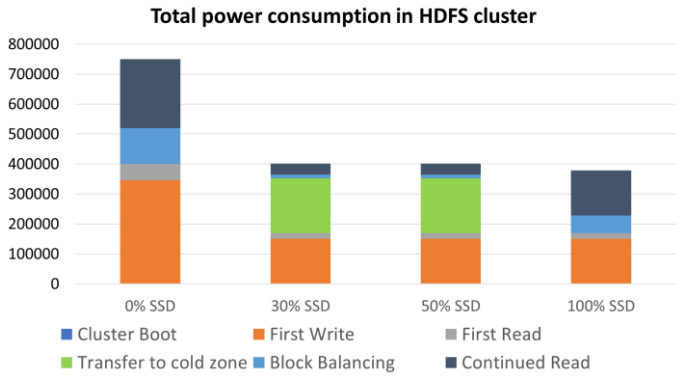

Figure 4. Total power consumption per life cycle

This is the total energy consumed for the HDFS over one life-cycle. This life-cycle has the following stages:

- *Initial Boot*: This is the energy consumed in bringing up all the nodes in the cluster. This tends to increase with increase in number of HDDs as storage devices, as HDDs consume more power during boot than SSDs.
- *First read and write*: This simulates the blocks being created and placed into the cluster. Majority of data is placed into the cluster 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.
- *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 exist. This transfer does not occur with 0% or 100% SSD as the former lacks a Hot Zone while the latter lacks a Cold Zone.
- *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 lesser number of SSDs so as to keep lesser number of nodes active. This maintains redundancy and data integrity while saving on unnecessary wastage of node active time and space.
- *Continued read*: This stage refers to the continued read of data after 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

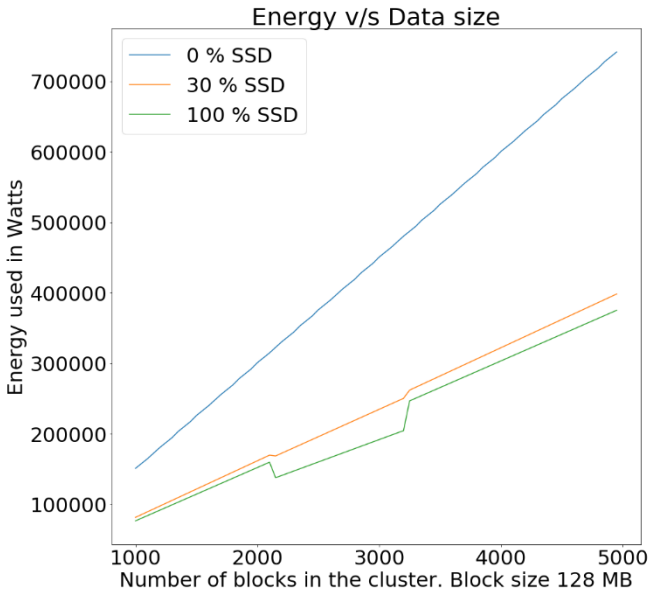

Figure 5. Power consumed as a function of variation of data stored in the cluster

The above graph shows the variation of energy consumed when the total amount of data (number of blocks) in the cluster is varied. As 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. What this means is that the number of blocks that exist are such that they 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., total number of blocks is 10, we need an additional node just to store one block.

D. Energy vs Number of Nodes

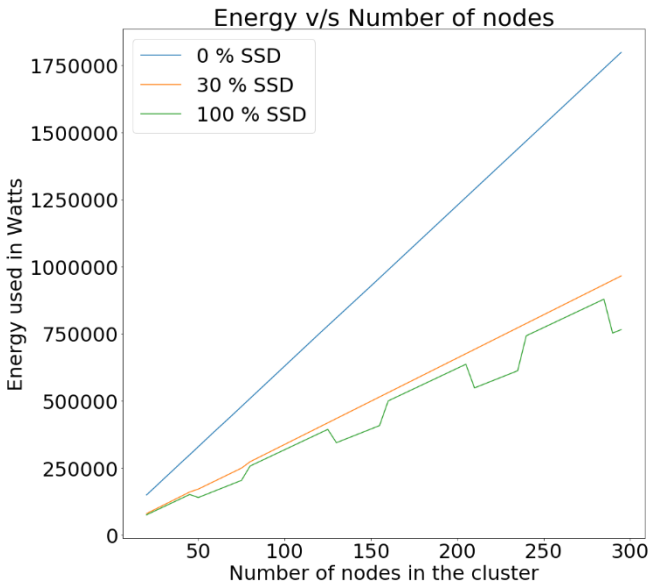

Figure 5. Power consumed as a function of variation of number of nodes in the cluster

The above graph shows the variation of energy consumed by the cluster as a function of 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.

VI. Conclusion and Future Work

The growth of cloud computing has led to an increase in power consumption by cloud compute clusters. To reduce power consumption and increase energy efficiency in HDFS clusters, an energy-conserving, hybrid and multi-zoned Hadoop cluster model was described. Cold data which sits idle for a long period of time is transferred to the cold zone, a zone representing low power state. This zone comprises of hard disks in low power mode and low computational power for significant energy cost savings. The overall performance of the cluster would not be affected since most computations would be done in the hot zone where hot data resides. This is achieved because there usually is low utilization in compute servers[7].

The usage of SSDs in the hot zone positively impacts workload processing and the SSD's high cost is circumvented by the fact that the cold zone entirely consist of HDDs. Furthermore, algorithms for data transfer among the zones were discussed to consolidate our model. Integration of our proposed zonal layout and algorithms were done in a simulator[6] and various tests were run. Simulator results show a significant reduction in power consumption upon the adaption of our model which results in cost savings for keeping the servers running. Thus a hybrid zonal approach for distributing workload in a Hadoop cluster can be used to increase cluster power efficiency.

Acknowledgement

Foremost, we would like to express our sincere gratitude to our mentor Prof. H L Phalachandra for his continuous support whilst working on the research with his patience, motivation and his immense knowledge of the subject and pointing us in the right direction throughout the period. We are also thankful to all the CCBD lab faculty in PES University for providing us a workspace to work in.

We are also thankful to Mr. Pedro Alvarez-Tabio Togores for sharing his HDFS simulator to the opensource community, it was very helpful for us to implement our algorithm on it directly without having to create a whole new simulator. His simulator also helped us is comparing our new algorithm's efficiency along with the old algorithm effectively

.

References

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

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

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

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

[5] B. Ramesh, R. Balu. Cloud Computing : An Analysis of the previous decade.

[6] Corentin Debains, Pedro Alvarez-Tabio Togores, Firat Karakusoglu: Reliability of Data-Intensive Distributed File System: A Simulation Approac

[7] Corentin Debains, Pedro Alvarez-Tabio Togores, Firat Karakusoglu: Reliability of Data-Intensive Distributed File System: A Simulation Approach