12/5/2016

Enhancement Proposal

EEECS4314 – Assignment 4

Reverse Em' All

Table of Contents

Abstract

This report will introduce a new feature to Hadoop. We will first examine the role and design of HDFS (Hadoop Distributed File System), specifically how it replicates data in Data Nodes. We will then examine the Balancer class in HDFS. This class balances DataNode usage in clusters to increase efficiency and productivity when reading or writing data. We will examine the drawbacks of this Balance class, as well as our motivation for implementing a new feature. The feature is an Auto-Balancer that automatically keeps the clusters balanced. We will discuss two ways to implement Auto-Balancer, one being a script that runs Balancer automatically, and the other that entails balancing the cluster automatically during each replication process. We will discuss their advantages and disadvantages and why we chose the replication process. We will discuss in detail the replication process, its impact on the current architecture of HDFS, and the testing methods one can use to test this new feature. We will discuss the limitations and potential risks of this feature, the lessons we have learned through the making of this report, and finally conclude the report.

Introduction

Apache Hadoop is always improving. From Hadoop version 1.0 to Hadoop version 2.0, a myriad of changes was made to improve the system. Take for instance the role of YARN in Hadoop 2.0. YARN is a resource manager that was created in Hadoop 2.0 to overcome the shortcomings in MapReduce 1.0. YARN's philosophy was to separate the resource management and the MapReduce segment. YARN only focuses on resource management, such as supplying Data Nodes with available CPU power to complete a job/task.

Hadoop is constantly improving. As of today, version 3.0 of Apache Hadoop is in its alpha stage, with a slew of changes that are much different from Hadoop 2.7.3. We will ignore Hadoop 3.0 for now and focus on the needs of Hadoop 2.7.3.

This report shall delve into HDFS, a system within Hadoop. HDFS (Hadoop Distributed File System), is a distributed file system for reading and writing Big Data. It allows clients to access and process data stored on the server. HDFS is a simple yet powerful system, but there are areas that can be improved. The area which we will focus on is the Balancer.

Balancer is a class within HDFS that balances data nodes in clusters. We will go into more detail about how it works in the next section. The shortcoming to the current way HDFS balances data is that it does not keep the nodes balanced. It is not automatic process. We will discuss the drawbacks of the balancing setup in Hadoop 2.7.3 and propose a new feature, the Auto-Balancer. The Auto-Balancer ensures that the clusters and Data Nodes within those clusters are balanced as often as possible. A balanced DFS results in increased reading and writing speeds, because it reduces bottleneck in Data Nodes that may contain too much data.

One approach is creating an Automated Script. We shall discuss the advantages and disadvantages of this approach. A second approach is delving into the code and modifying the data replica algorithm. We shall discuss this approach, its advantages and disadvantages, and why we ultimately went with this approach instead of the first one. We will go into detail how we will implement this approach. Figure 1 and Figure 2 are state diagrams that explain the process of choosing a place to store Data Node replicas. Table 1 contains testing methods to test this new feature. Note that our proposed feature has not been officially implemented yet, although there are criteria to check and see if an Automated-Balancer is efficient and effective.

HDFS Block Allocation

Objective of current block allocation strategy

The default block placement strategy is highly fault-tolerant through data redundancy. HDFS uses a simple but highly effective policy to allocate replicas for a block. Its Rack Awareness concept enables it to have better data reliability, availability and network.

By using replication pipeline policy and allowing not more than two replicas within the same rack, the current block strategy reduces inter-rack write traffic, maximizes bandwidth utilization, and minimizes write cost with the given objective achieved. The default strategy not only places replicas efficiently, but also takes the network bandwidth and fault tolerance into consideration. This strategy also achieves reduced inter-rack writing with the given requirement of high availability and bandwidth utilization.

Issue of current block allocation strategy

Over time the distribution of blocks across DataNodes can become unbalanced. An unbalanced cluster can affect locality for MapReduce, and it puts a greater strain on the highly utilized DataNodes, so it’s best avoided. Unfortunately, the block placement strategy has not yet succeeded in placing a fair and even distribution of blocks across the cluster. The HDFS block placement strategy does not take into account DataNode disk space utilization, therefore data might not always be placed fairly and uniformly in the DataNodes. Imbalance also occurs when new DataNodes are added to the cluster.

HDFS Balancer

Balancer is a class in HDFS that distributes data across Data Nodes. It attempts to fix issues of the current block allocation strategy. Its purpose is to balance a cluster. In a typical HDFS cluster, as new data is added and data is replicated, the cluster may become unbalanced. For example, one DataNode may have a high usage. This means that the amount of data in that DataNode may be close to the maximum capacity of data that DataNode can hold. Other DataNodes in that cluster may have low DataNode usage. If there is a Data Node at a high usage, there will be a lot of client access to that Data Node because it contains most of the data the client needs. This can bottleneck the system, leading to low read and write speeds. The Balancer moves data from over-utilized Data Nodes to underutilized Data Nodes. The HDFS clusters are considered balanced when the utilization of every DataNode differs from the utilization of the cluster by no more than a given threshold percentage. For example:

40% Cluster Utilization -> 30% - 50% Data Node utilization

If the current cluster usage is at 40%, i.e., the cluster is 40% full, the balancer rebalances all the DataNodes such that their utilization is plus or minus the cluster usage by no more than the given threshold. In this case with a threshold percentage of 10%, the Data Nodes will be around 30% to 50% usage after balancing.

Drawback

One of the drawbacks of the Balancer is that it must be manually run. Issuing the command,

```
HDSF BALANCER 10
```

In the command line will balance the Data Nodes within a cluster. There is nothing within Hadoop that automatically checks if a cluster needs to be balanced and to rebalances them. There may be moments where HDFS become extremely unbalanced and there are extreme bottlenecks at some Data Nodes.

Another drawback is that it may take a long time to run. The more a cluster is left unbalanced, the worse it becomes. The usage for some DataNodes may continue to rise while other Data Nodes may be left unaffected. The balancer will then take a long time since there are more unbalanced DataNodes.

Although clients are still able to access data and issue jobs in HDFS, balancing is a demanding process. Running the balancer during high client activity will result in an overall slower performance in HDFS.

Motivation

We ultimately want to uniformly place HDFS data across DataNodes and clusters. A reason for non-uniform placement of data is the creation of new DataNodes to an existing cluster. The NameNode is responsible for placing new blocks of data into a DataNode under certain considerations that compete against each other. This results in the non-uniform placement of data. The reason non-uniform/uniform placement of data is important is because when a new DataNode joins the HDFS cluster, it does not hold much data, therefore any map task assigned to the machine most likely will not read local data. Receiving data from another node in the cluster increases the use of network bandwidth to the upload/downloading machines. On the other end of the spectrum when DataNodes become full, the new data blocks are placed on non-full nodes reducing their read parallelism. Both these cases present issues to data availability. The idea is to balance the data blocks on DataNodes both within and external to the cluster allowing for higher data availability. The current Balancer is a command line tool that is done manually at the Administrators discretion, and thus can be neglected and not done as often as it should to increase data availability and access.

Stakeholders

<u>System Users</u>

The system users are the stakeholders who ultimately make use of the system. The enhancement will mainly affect them in two ways:

1. It is expected that the enhancement will lead to a more balanced cluster of data at all times. This will boost performance of MapReduce and other processes by shortening the run time due to the more balanced cluster.

2. The user will experience a longer write time since NameNode needs to find the appropriate DataNode to ensure a balanced cluster. Because there is a higher probability of replica placed at different racks, there is a decrease in bandwidth as they are farther apart.

<u>System Administrator</u>

The system administrators run and maintain the system once it is deployed. One of the tasks of a system administrator is to manually run the balancer, per the team's agreed policy: "the frequency of running the tool and the preferred time".

Approach 1: Automated Script

To solve our balancer issue, one approach is to create a simple script that automatically runs the balancer during a specified time. The script could be stored on the server/administrator side. The idea is that since the balancer must be manually run, and that a cluster becomes unbalanced over time, an automated script will ensure the distributed file system is kept balanced. The time at which the script will run can be determined beforehand. An administrator can look at the time when their HDFS usage is at the lowest, possibly during night time or early morning, and schedule a balance time at that moment. The script can also be designed to run once per week, or once per month, depending on how large the system is and how many clients are using the system.

Advantages

- **No change in the Hadoop architecture**

The Automated script is “outside” the boundary of HDFS, as in there is no changes in the architecture of Hadoop. The script is already using a feature in Hadoop, which is the balancer, all it is doing is running the balancer command line automatically.

- **Simple to make and customize when the Balancer should run**

A script can be customized at any time without having adverse effects on the system. The script does not call methods or functions inside Hadoop, and can exist and operate if there are changes within the system. We can specify the time when the script should run, the threshold to use, and any other options using the script.

Disadvantages

- **Script may run at inopportune times and slow down the system**

Despite determining the best time to run the balancer in the script, there is a possibility that the balancer could run during above normal client usage. In this case, running the balancer will slow down the system and have a negative impact in the client’s experience.

- **May run at times when there is no need to rebalance**

This again plays apart to the automatic nature of the script. Take for example a time when HDFS usage is low for a week or so. In this scenario, there are very few client jobs, and as such there is less need to read, write, or replicate data. Since the script is automated, it will run during this time and expend resources when it does not need to.

Approach 2: Modifying Replica Placement Policy

Another approach of implementing the auto-balancer in HDFS is to manipulate the logic NameNode decides on the DataNode for every replica. That means modifying the Replica Placement Policy. This approach makes use of Hadoop’s Rack Awareness feature.

Using this approach, the balancing is done during the block replication process. The replicated block will be placed on an underutilized DataNode.

This method is different from the first one as it requires modification on the implementation on the method, of the subsystem NameNode, that decides and returns the ID of the DataNode to store a block replica.

Advantages

- **The impact on other procedure is at minimum.**

The balancing is occurring during the placement of the block replica itself, hence it will not interfere with other procedures. This differs from the first approach, where the balancer is running at every scheduled frequency and hence will interfere and competed for bandwidth with concurrent procedures.

- **Retaining original objective of Block Placement Policy**

Though the Block Placement Policy is amended, it is still possible to revise such that the main objective in the original policy is retained, while achieving greater balanced cluster (as shall be shown later).

Disadvantages

- **Requires modifying existing code**

This approach requires amending existing methods for the Replica Placement Policy. The placement of replica is critical to HDFS's reliability and performance, hence great care is required to ensure retaining the stability of the original method while achieving the goal of the revision.

- **Longer runtime in procedures involving choosing DataNode for block replica**

The main disadvantage is that the runtime for NameNode to return the DataNode will be longer, as NameNodes need to scan for underutilized DataNodes. Any procedure that involves placing or transferring block replicas, such as writing a file, will take longer runtime.

Proposed Implementation of approach – Modifying Replica Placement Policy

NameNode delegates the task of deciding where to place each replica to its Replica Placement Policy.

Here is the original default Replica Placement Policy:

1. Place the first replica somewhere – either a random node (if the HDFS client is outside the Hadoop/DataNode cluster) or on the local node (if the HDFS client is running on a node inside the cluster).

2. Place the second replica in a different rack.

3. Place the third replica in the same rack as the second replica

4. If there are more replicas – spread them across the rest of the racks.

To implement the enhanced feature of 'auto balancer', the Replica Placement Policy will be amended as follow (with those strike-out parts omitted):

1. Place the first replica somewhere ~~either a random node or on local node~~
   a. In the case where the HDFS Client is within the Hadoop cluster:
      Search within the rack for node with lowest capacity.
      If the node capacity is not over-utilized, it will be the node hosting the first replica.
      Else look for rack with the lowest capacity node

   b. In the case where the HDFS Client is outside Hadoop cluster:
      Pick the node with the lowest capacity

2. The second replica is written to a different rack from the first replica, ~~chosen at random~~
   Pick a node at a different rack from the first with the lowest capacity

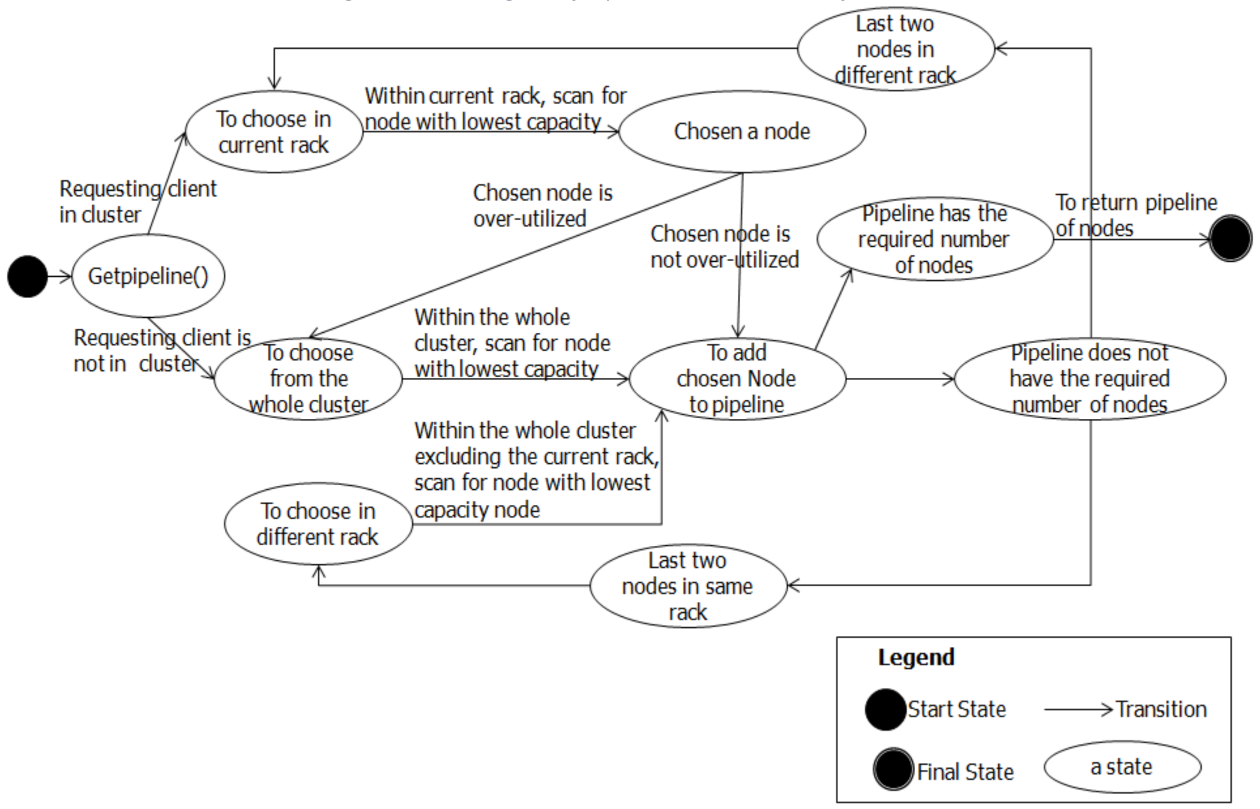
3. The third replica is written to the same rack as the second replica, but on a different node
   AND
   Pick a node, in that same rack, with the lowest capacity

4. If there are more replicas, spread them across the rest of the racks, following the policy of no more than two replicas in the same rack and no more than one replica in the same node, and fulfilling the following additional conditions:
   - When needing to choose a node on a different rack, pick the rack with the node of lowest capacity and place the replica in that node
   - When choosing within the same rack, pick the node with the lowest capacity in that rack

Replica Allocation Decision – Write

The state diagram in Figure 1 explains how Replica Allocation is decided after amending the Replica Allocation Policy when a file is written.

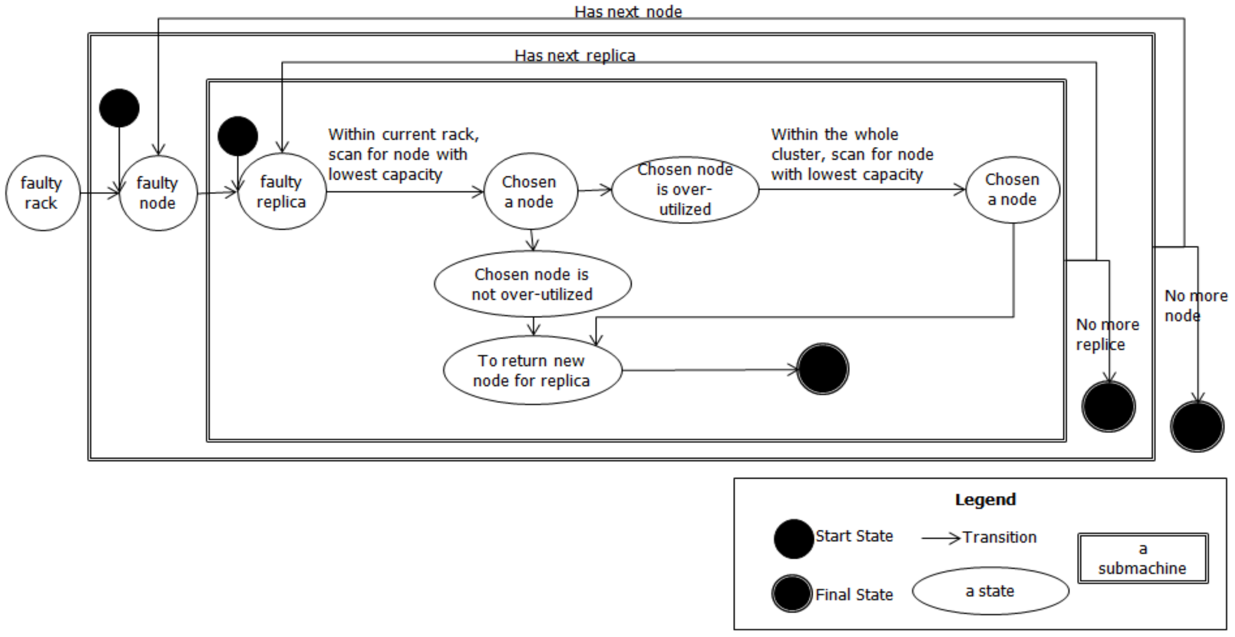
Figure 1: State diagram of replica allocation decision for Write


Last two nodes in different rack
To choose in current rack
Within current rack, scan for node with lowest capacity
Chosen a node
Requesting client in cluster
Chosen node is over-utilized
Chosen node is not over-utilized
Pipeline has the required number of nodes
To return pipeline of nodes
Getpipeline()
Requesting client is not in cluster
To choose from the whole cluster
Within the whole cluster, scan for node with lowest capacity
To add chosen Node to pipeline
Pipeline does not have the required number of nodes
Within the whole cluster excluding the current rack, scan for node with lowest capacity node
To choose in different rack
Last two nodes in same rack
Legend
Start State
Transition
Final State
a state

Replica Allocation Decision – Faulty Replica, Node or Rack

Figure 2 is a state diagram on how Replica Allocation is decided after amending the Replica Allocation Policy when there is a case of faulty replica block, node or rack.

Figure 2: State diagram of replica allocation decision for Fault Replica, Node, or Rack



Has next node
Has next replica
faulty rack
faulty node
faulty replica
Within current rack, scan for node with lowest capacity
Chosen a node
Chosen node is over-utilized
Within the whole cluster, scan for node with lowest capacity
Chosen a node
Chosen node is not over-utilized
To return new node for replica
No more replica
No more node
Legend
Start State
Transition
a submachine
Final State
a state

Architecture – Effects on concurrency & Team Issues:

Hadoop has a very well Software Architecture Design. All the major subsystems are loosely coupled. In fact, most of the coupling is Message coupling, whereby component communication is done via parameters or message passing.

The changes to implement the 'auto-balancer' will not affect the mechanisms of other subsystems, nor will it impact the communication and dependency between subsystems. It will not affect any of the concurrency within Hadoop system as well. There will also be positive impact on MapReduce. We expect the runtime of MapReduce will be shorter since it will be running on a more balanced cluster of Nodes. In fact, most of the procedures run on HDFS will be expected to have shorter run time.

Impact on the Architecture – within HDFS Component:

The main impact within HDFS is the NameNode, which is the Master server that manages and decide on allocating every block replica. The NameNode manages the file system namespace and provides an entry point to HDFS for client applications. NameNode has all the metadata of all the current block replica. It provides access control to files and directories and controls the mapping of blocks to DataNodes. The Replica Allocation Policy resides within NameNode. Hence, the amendment may only affect one mechanism within NameNode, which is the logic on deciding the location of every replica block.

Neither the Client nor any of the DataNodes depend or need to know how the NameNode determines the allocation of every block replica. Both depends on the NameNode to return them for location of every requested location for a block replica. How NameNode determines the allocation will not affect them. However, it is expected the runtime for NameNode to return them the location for a block replica will be longer.

If we look at the architecture of HDFS (Figure 3), every major subsystem interacts with one another through the Protocol subsystem, which mean the system applied façade design pattern, to loosely couple very subsystem, and hence allowing each of them to have greater dependency to extend.

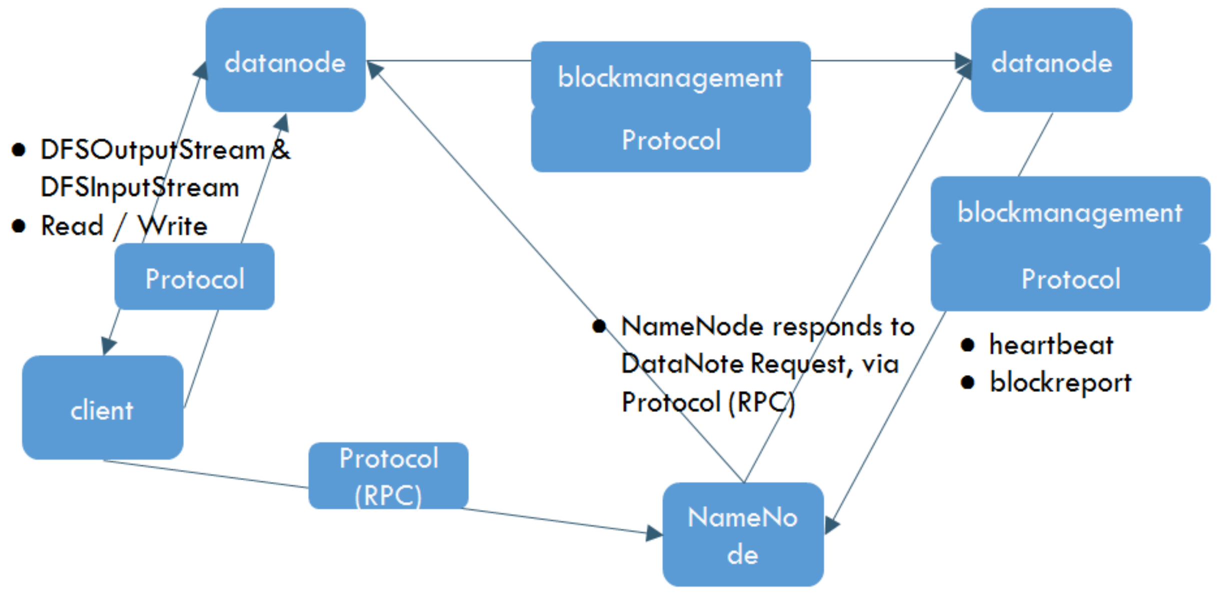


Figure 3: Concrete architecture of HDFS

Testing Plan

An important thing to look at when adding a feature to legacy code is to consider the impact of interaction between the proposed enhancement feature and other current features. Firstly, we would see how the feature impacts the architecture which has been explained previously. We need to consider if and how the feature will embed into the current system (sub-systems), this is mainly impactful for maintainability of code, a good use of current and new design patterns would solve this issue. Another thing is testing the reliability on the feature itself making sure that there are no potential risks of lost data, accessibility of the data whilst the feature is in effect, essentially to create seamless entry into the current system. Considering our proposal there is no change to the way HDFS implements data creation, rather it utilizes the current design patterns and implementation and adds to it, therefore it is expected there would be no detrimental change to the security of the data in question. In the same fashion, maintainability need not change due to the use of current implementation, the only thing to consider would be the maintainability of the new code, making it necessary to maintain

Hadoop's extendible style allowing for ease of maintenance and additions to the process of balancing data. The most important topic to consider for testing the impact on the current system is performance testing. Is our feature enhancement proposal truly better than the current solution? For this situation, we have created metrics to be checked against our feature and the current Balancer. These are:

- Run time
- Fault-tolerance level (if there is any error/fault occurs)
- Average balance-ness (by counting the number of under/over utilized nodes)

The run time is one of the most important metrics as it will determine the seamlessness of the feature as described earlier this is the run time process of writing the data to disk, this affects the users' access to the data and will be a crucial metric to monitor for balancing read/write availability. The fault-tolerance level will determine how accurate the data movement is and if any data is lost, if the delta of correct to faults is too high on either end (our feature/current feature) then the feature is essentially a lost cause as this will destroy the need for balancing due to data being lost. The last combined with the run time will ultimately be the deciding factor in if our feature is indeed better than the current solution. Below you can see a table that will be produced for each feature, showing the processes that are necessary to the feature's success, each process will be checked for the metrics proposed and the two tables would be compared and would be the deciding factor on with feature is better.

Table 1: Testing metrics for Automated-Balancer

Processes	**Run Time**	**Fault Tolerance (no. of error /exception)**	**Balanced Storage (no. of under/over-utilized nodes)**
MapReduce process			
Write a file			
Read a file			
Append a file			
Faulty Node/Replica			
Adding a DataNode			
Decommission a DataNode			
Running balancer			

Limitation & Potential Risk

Looking at the ways we propose testing, without having the metrics we cannot with guarantee say that our features design is the correct solution. In theory, our feature would work better with the only limitations being that the schema to auto-balance occurs when a new node is added into the cluster, this creates some additional writing to disk the tradeoff is that it will be faster to run if it would be balanced and less idle nodes. Another limitation that our enhancement does not consider is the 'latest' or most 'visited' data, the latter if unbalanced could cause higher bandwidth to one node that is full. As discussed before with Hadoop being highly extendible these limitations can all be considered once the feature is working well and goes into a maintaining phase.

Lessons Learned

Hadoop is an efficient system for DFS. However, there are still many areas that can be enhanced for the system.

The system has good system design of being loosely coupled, making the system highly extendible. We are rather surprised when we found the enhancement proposed had little impact on most of the subsystems.

We also learned that many enhancements do come with an overhead price on other subcomponent or processes runtime, such as the enhancement we have proposed. It will lead to shorter runtime on many procedures including MapReduce. However, it comes at the cost of increase write runtime.

Conclusion

We think our auto-balancer will make a great addition to Hadoop. Cluster imbalance is an issue. As data gets added into the system and data is replicated, some Data Nodes that hold these data become over utilized, while other Data Nodes become underutilized. This creates a bottleneck in the system, where most of the read functions in Hadoop is using Data Nodes that contain most of the system. Although there are two ways of implementing our Auto-Balancer, one being an automated script and the other by changing the data replication methodology inside Hadoop. We discussed how Hadoop will store replicas. In summary, we generally chose the Data Node with the lowest capacity within that cluster.

Since introducing this new feature may have an impact on the system, we discussed metrics that can determine how long a process is ran, its fault tolerance, and the number of over-utilized/underutilized Data Nodes. We discussed the limitations and fault risk, and we finally discussed the lessons learned when making this report.