Chapter 5
Proposed Methodology

As we discussed in chapter 3, default block placement policy works well for homogeneous systems. But when data blocks and datanodes are geographically distributed on the heterogeneous environment, default block placement policy in Hadoop, does not prove to be very efficient. In Hadoop, HDFS default block placement policy does not consider node's processing capability while placing data blocks on that node. As a result, nodes with lesser processing capability affect the performance of Hadoop. Therefore, Hadoop default HDFS block placement strategy needs to be improvised, such that node processing capability and heterogeneity of system can be taken into consideration. We propose the Block Rearrangement Algorithm titled "Saksham": Resource Aware Block Rearrangement Algorithm. "Saksham" algorithm only rearranges the blocks according to assigned priority. Actual placing of blocks is still done by default HDFS block placement policy.

5.1 Saksham: A Resource Aware Block Rearrangement Algorithm

Hadoop uses "Rack Awareness" while placing data blocks for fault tolerance and better performance. "Rack Awareness" is a concept Hadoop uses to place read/write request to the same rack or nearby rack (Team, 2018). This concept helps to achieve better data locality as discussed in chapter 3. MapReduce *de facto* standard tries to move the job where data is stored. But, that node may not have sufficient processing capability or job may get skewed due to less processing/memory capability. Hence, we propose "Saksham" A Resource Aware algorithm which rearranges the data blocks according to processing capability of the node or heterogeneity of environment. Since, capability of the node is taken into consideration, we have named the proposed algorithm with the Sanskrit word "***Saksham***", which translates to "***Capable***" in English.

We can apply custom block rearrangement policy by considering two distinct approaches. In the first approach, we consider only heterogeneous nodes having different computational capability. Meaning, we assign the priority based only on the computational capability of the nodes, where each node may be of different configuration from the other. For the nodes with the higher computational capability we can assign higher priority. In the second approach, we create two groups based on homogeneity and heterogeneity of nodes. As per the requirement of an application, we may submit the job only to homogeneous cluster or only to heterogeneous cluster. Depending on where the application needs to be submitted the higher priority may be assigned to one of the groups.

Initially default HDFS block placement policy places data blocks as shown in fig. 5.1. If a file that comprises of 8 blocks, needs to be placed in HDFS. As per the working of HDFS, it will first create the replicas of these blocks of a file. The number of replicas created will depend on the replication factor which can be customized. The default replication factor for the blocks in HDFS is 3. Thus, a file having 8 blocks with each block having 3 replicas i.e. in total 24 blocks need to be placed across different nodes in HDFS. As shown in fig 5.1, the given file's 24 blocks are placed in random manner, such that Node 1 is overloaded with 8 blocks, whereas Node 6 has only 1 block placed on it. So, placing of blocks is skewed. Now, let us assume that there was no processing capability on Nodes 7 to Node 10. If blocks of file are placed on these nodes, and if the task is floated for execution on this node, because it contains data, then it will not be able to go into execution immediately due to its inability to execute the task. Another possibility is that you may need to move the block from these nodes i.e. for E.g. from Nodes 7 to any other Node between Node 2 to Node 6, where the task may go into execution with available resources, but this would involve network transmission and increase in total execution time. As a result the Nodes 7 to Node 10 may inhibit the overall progress of the job and hence create a bottleneck in job completion. To avoid this situation, block rearrangement will be required, which is done in the proposed algorithm.

As per the First Approach of the proposed algorithm "Saksham", we consider processing capability of nodes for forming a group. For this case, we have

considered heterogeneous cluster comprising of nodes with different processing capability. Few nodes may be having lower processing capability and hence, may be comparatively slower. In distributed computing, the speed of execution of the job is proportional to the speed of the slowest node in the cluster. Thus, slower nodes may degrade the overall performance of job processing. To avoid this situation, we've divided nodes into two groups and assigned them the values priority=1 and priority=2. Priority=2 is assigned to nodes which have more processing capability and priority=1 which has lower processing capability. These settings can be defined in Hadoop's config.xml file. Once priority is set our "Saksham" algorithm will rearrange the blocks of specified HDFS file path and store all blocks onto the nodes which has priority=2 and remove all the blocks from nodes which have priority=1.

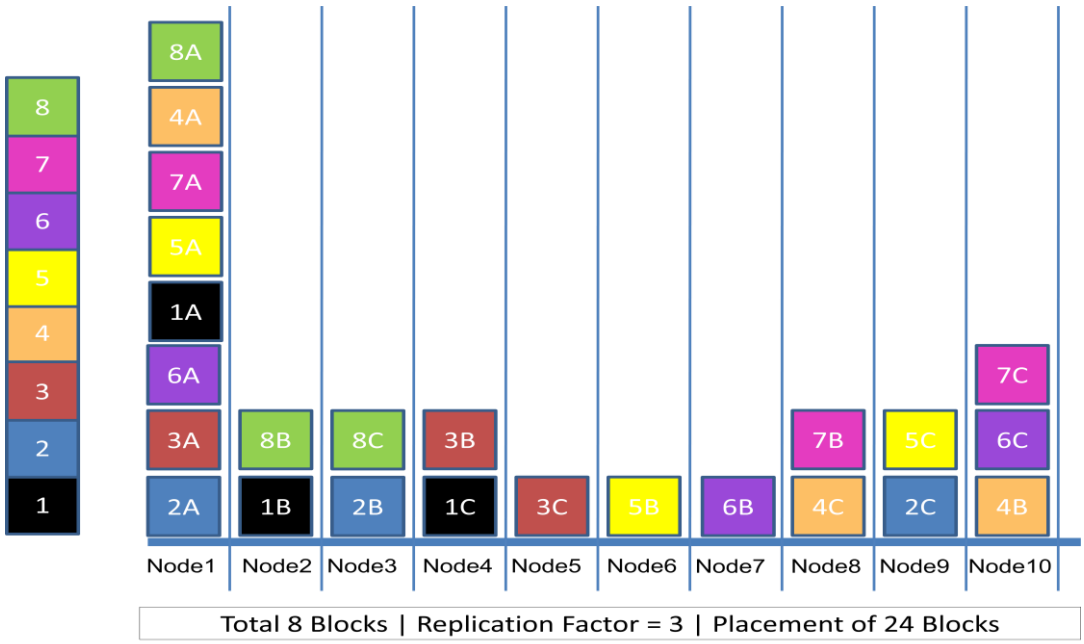

Figure 5.1 HDFS Default Block Placement Example

As per the second approach of the proposed algorithm "Saksham", we divide the nodes into two groups based on the homogeneity or heterogeneity of nodes in the cluster. We can assign priority 1 or 2 to either of the group. Depending on processing need of big data application, we can rearrange all blocks into set of heterogeneous nodes or into set of homogeneous nodes only. Only nodes with priority=2 will store the blocks and with priority=1 will not store any of them. If the requirement of the application be such that it gets executed efficiently on cluster with nodes having similar configuration, then we would like to submit the

application on set of homogeneous nodes and hence would assign the priority = 2 to the set with homogeneous nodes. Thus, keeping the control of block placement in our hands, rather than permitting a default behaviour of Hadoop.

While placing all blocks our proposed algorithm also takes care that blocks of a file will get distributed equally over all the nodes by considering disk usage of each node. Each time, before placing the block of file it checks that node should not contain a replica of the same block. The facts prove that Hadoop works better in homogeneous environment, but that can be fulfilled even on cluster is heterogeneous nodes, using our proposed strategy. Figure 5.2 demonstrates how our proposed "Saksham: Resource Aware Block Rearrangement" policy can rearrange blocks according to a priority assigned.

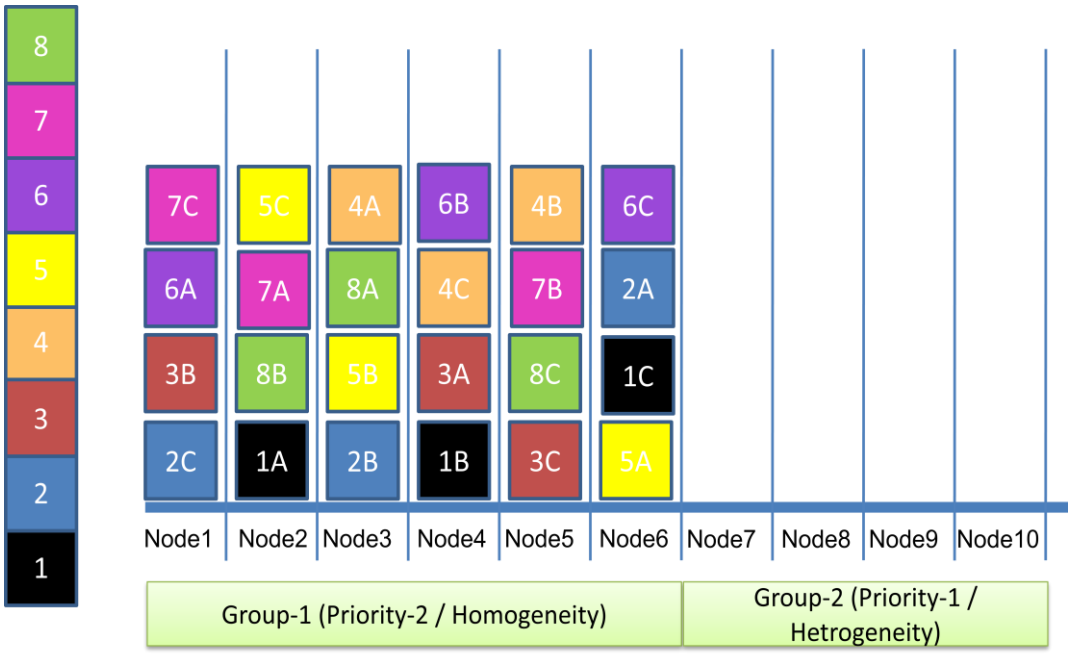

Figure 5.2 "Saksham" Approach: Group-1: Priority=2; Group-2: Priority=1

Before we understand the "Saksham" block rearrangement algorithm, it is important to know how the configurations in the config.xml file refer the parameters for the setup. The parameters for "Saksham" model are also specified in Config.xml file. For "Saksham" model, we specify 4 key parameter settings such as hardware types, hosts, priority and data blocks required to rearrange. First, hardware types segregate all cluster nodes in group-1 and group-2. Second, hosts are categorized in group-1 and group-2 according to the availability of resources. For example, we

assume the group-1 contains the hosts which have more processing capability then t group-2 or vice versa. We can also make an assumption that group-1 is set of homogeneous nodes and group-2 is set of heterogeneous nodes. Third parameter is the priority, where we assign priority as explained above to groups of nodes. Finally, we select the path of HDFS block for the rearrangement and store the blocks information in ArrayList. Reference table 5.1 explains "Saksham" configuration parameters used in config.xml file for further passing to "Saksham" block rearrangement algorithm.

Configuration Parameter	Description
saksham.hardwaretypes	Comma-separated different types of hardware configurations
saksham.<type>.hosts	Comma-separated list of host names or IP addresses of target DataNodes
saksham.<type>.priority	Assign priority=2 for the hardwaretype where we want to rearrange blocks and priority=1 will not store anything
saksham.hdfsblocks	HDFS location of blocks for rearrangement

Table 5.1 Saksham config.xml parameters

Figure 5.3 is the proposed "Saksham: Resource Aware Block Rearrangement" algorithm.

- Firstly, algorithm checks the HDFS path that contains blocks and also collects the list of datanodes from the DatanodeInfo, Hadoop API and stored them in arraylist.

- Secondly, based upon config.xml file it checks the priority assigned to each node and segregates the nodes in two lists i.e. node_list1 (priority-1) and node_list2 (priority-2).

- In last step blocks along with each replica will be rearranged in nodes which are having priority-2. For rearrangement, it checks the disk utilization and processing capability of each node.

Algorithm 1 Saksham: Resource Aware Block Rearrangement Algorithm

Input: HDFS location of input files to be balanced / rearranged
File contains list of data blocks which are placed in HDFS using default policy.
Output: Data blocks will be placed to specific nodes only based on given priority factor.

```
1)  if input HDFS path != null
2)      foreach locatedBlocks block : nameNode.getLocatedBlocks() do
3)          Put blocks in arraylist<block_list>
4)      endfor
5)  endif
6)  foreach DatanodeInfo node : getDatanodeStats(Live) do
7)      if nodes in config.xml != null
8)          Add nodes in arralylist<datanodes>
9)      endif
10) endfor
11) foreach BalancerDataNode node : datanodes do
12)     if nodes.priority = 1.0
13)         Add nodes in arralylist<node_list1>
14)     elseif nodes.priority = 2.0
15)         Add nodes in arralylist<node_list2>
16)     endif
17) endfor
18) Sort node_list2 by disk utilization in ascending order
19) for each block replica do
20)     Initialize <block_list>  queue with all blocks
21)     Initialize <node_list2>  nodes with priority=2
22)     for each block replica in <block_list> do
23)         if (find first node form <node_list2>) doesn't contains(block)
24)             Put block onto selected node
25)             Remove node from <node_list2>
26)             Remove block from <block_list>
27)             if node_list2 is empty
28)                 Initialize with all nodes with priority=2
29)             endif
30)         endif
31)     endfor
32) endfor
```

Figure 5.3 Proposed "Saksham" Algorithm

Above algorithm describes how it rearranges the data blocks. But it is important to note that this algorithm also depends upon so many small algorithms for collecting data blocks and datanodes information from Namenode and datanodestats. The flow diagram of the "Saksham" algorithm is shown in fig 5.4.

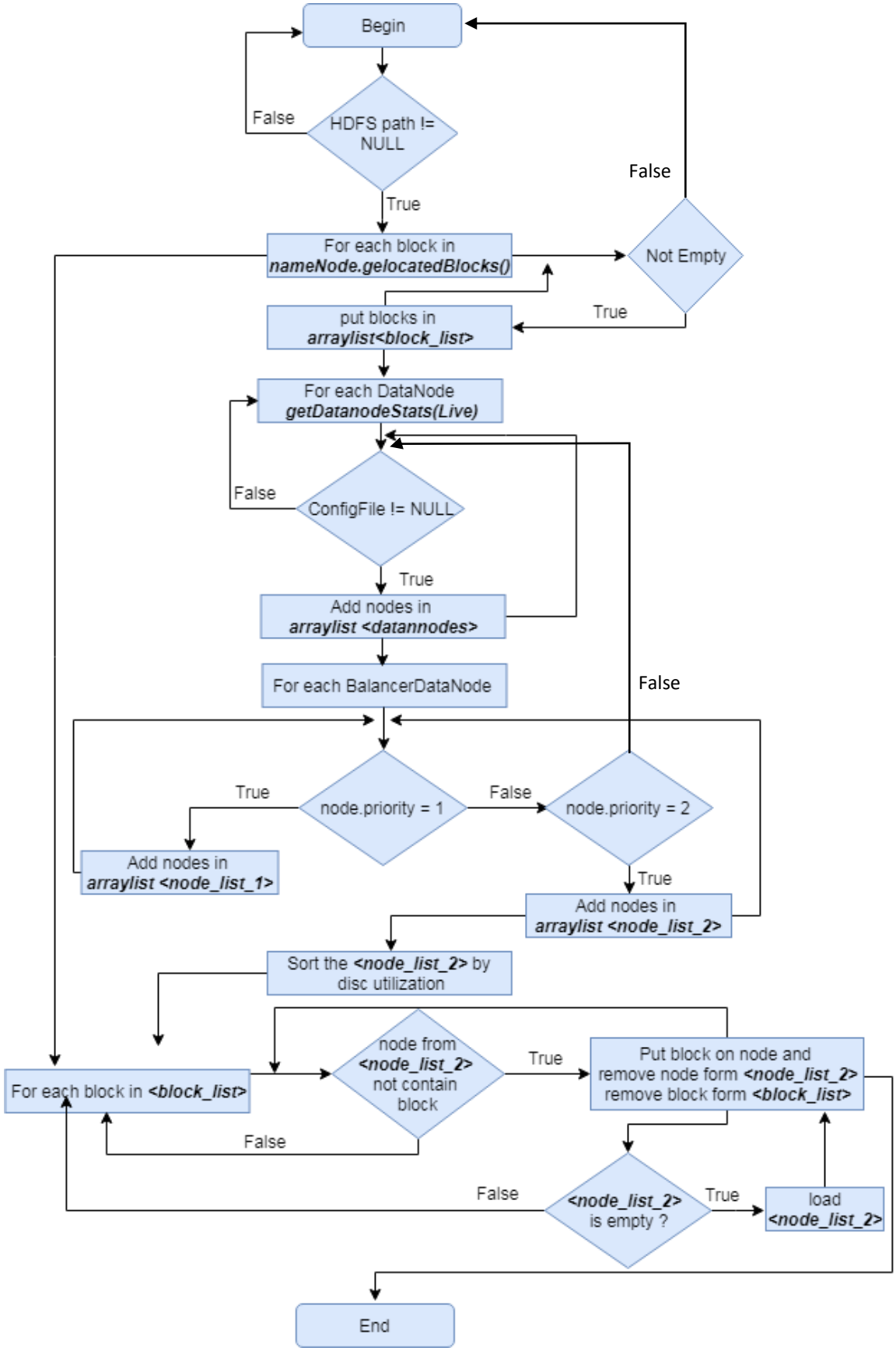

Figure 5.4 Flow Diagram of "Saksham" algorithm

5.2 Saksham Model

To achieve better performance for big data processing, we target upon two important aspects of heterogeneous distributed computing: file system management and process management. Figure 5.5 represents the proposed "Saksham" model.

First, file system management basically controls the block placement and allows us to rearrange the blocks on specified nodes based upon two important approaches of load balancing:

1. Balance the load among heterogeneous and homogeneous nodes of the cluster.
2. Balance the load within the cluster based on the processing capability of each node by giving priority to each node.

Our proposed algorithm allows the user to select the nodes based on its individual data processing capability and rearrange the blocks which are placed using the default policy. Default block placement policy of HDFS fails to achieve optimized performance as it does not check the processing capability of the node while placing blocks on the node. Our proposed algorithm achieves that by considering processing capability of nodes and places blocks on nodes which has higher processing capability. If there are two or more nodes having same processing capability then it checks the utilization of the node, and considers the less utilized node first, for block placement. By doing this, we have control over data to be put on selected nodes, considering processing capacity and utilization of nodes. Saksham does cause an overhead time, for rearrangement, but this rearrangement of blocks that we do using Saksham is not included in the total time of execution. As the movement of blocks is not happening during the job is in execution and hence, job need not wait, or it is not preempted. If default policy is used then, in case, if the resources are not available where the block is placed and the job is submitted, then during the execution of job i.e. after submission, movement of data blocks to where the resources are available, will be done, and that will cause lots of overhead.

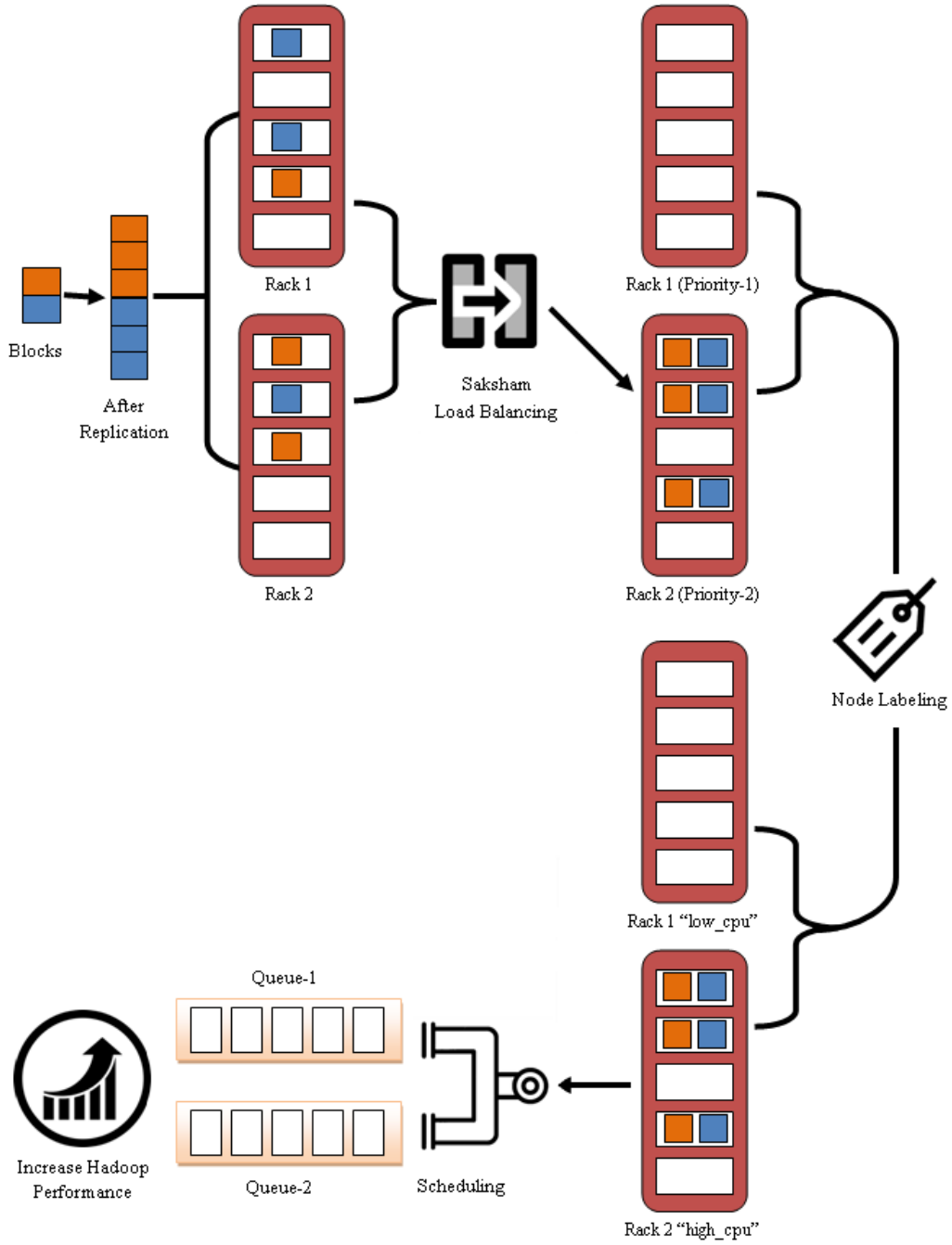

Figure 5.5 Proposed "Saksham" Model for Load Balancing

Second, we use the concept of node labelling to achieve better process management. YARN Node label (Hadoop.apache.org, 2018e) allows partitioning the single cluster among multiple sub-clusters. Using this concept we can mark nodes with meaningful labels i.e. nodes with higher processing capability may be labeled as "high_cpu" and with high memory may be labeled as "high_mem". By combining the proposed algorithm with node label, it can be actually decided where to put jobs. To

achieve better data locality the job can be submitted to the nodes where data is actually rearranged using proposed algorithm. Later, Hadoop scheduling can be used to put jobs in queues for processing. It limits the overhead of internode and inter-rack data transfer since process (containers) and data blocks are on to the same nodes. Thus, along-with node labelling and using Saksham Block Rearrangement algorithm, we have defined two-level architecture, to identify the appropriate nodes, where data blocks should be placed and job execution can thus be done more efficiently.

In this chapter, we explain the concept of our "Saksham" model. We also discussed that "Saksham" block rearrangement algorithm does take rearrangement time overhead, but it will compensate against the total execution time of application. We have integrated "Saksham" policy into the Hadoop-2.7.2 distribution. To validate the importance and applicability of "Saksham" model, we run the various data intensive applications and results are presented in the next chapter.