**Mid-term report**

Thành viên: Cao Nguyên Bình – 1610228

Nguyễn Thành Duy – 1610491

Dương Văn Trường - 1613832

1. **Planning**

With the selected topic, the team made a realistic plan asking for the following:

* Theory of Hadoop
* Theory of Skein Family Hash Function
* Identify Implementation method
* Implementation
* Testing

1. **Completed tasks**

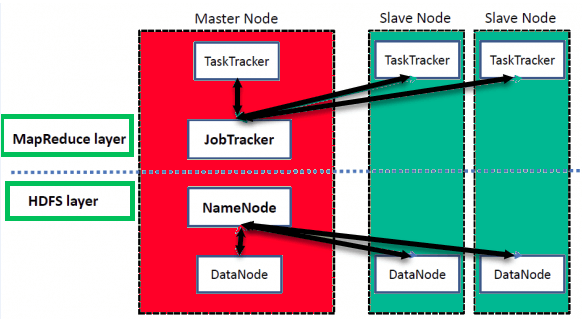
With the content of the plan set out above, so far (2 weeks after selecting the topic), the group has done the following contents:

* Theory of Hadoop
* Theory of Skein Family Hash Function
* Identify Implementation method
* Image dataset

1. ***The theory of Hadoop***

* ***What is Hadoop?***

Hadoop is an open source distributed processing framework that manages data processing and storage for big data applications running in clustered systems. It is at the center of a growing ecosystem of big data technologies that are primarily used to support advanced analytics initiatives, including [predictive analytics](https://searchbusinessanalytics.techtarget.com/definition/predictive-analytics), data mining and [machine learning](https://searchenterpriseai.techtarget.com/definition/machine-learning-ML)applications. Hadoop can handle various forms of structured and unstructured data, giving users more flexibility for collecting, processing and analyzing data than relational databases and data warehouses provide.



* ***Architecture of Hadoop***

There are mainly five building blocks inside this runtime envinroment (from bottom to top):

* Hadoop has a Master-Slave Architecture for data storage and distributed data processing using MapReduce and HDFS methods.
* Namecode represented every files and directory which is used in the namespace.
* DataNode helps you to manage the state of an HDFS node and allows you to interacts with the blocks
* The master node allows you to conduct parallel processing of data using Hadoop MapReduce.
* The slave nodes are the additional machines in the Hadoop cluster which allows you to store data to conduct complex calculations. Moreover, all the slave node comes with Task Tracker and a DataNode. This allows you to synchronize the processes with the NameNode and Job
* ***How Hadoop works***

As we have discussed Apache Hadoop Daemons in detail. Now we will learn how Hadoop works with the help of these daemons.

To process any data, the client first submits data and program. Hadoop store data using HDFSand then process the data using MapReduce.

* Hadoop Data Storage

Let us first learn how Hadoop stores the data?

Hadoop Distributed File System – HDFS is the primary storage system of Hadoop. It stores very large files running on a cluster of commodity hardware.

HDFS stores data reliably even in the case of machine failure. It also provides high throughput access to the application by accessing in parallel. The data is broken into small chunks as blocks. Block is the smallest unit of data that the file system store. Hadoop application distributes data blocks across the multiple nodes. Then, each block is replicated as per the replication factor (by default 3). Once all the blocks of the data are stored on datanode, the user can process the data.

#### Hadoop Data Processing

Let us now learn how Hadoop processes the data?

Hadoop MapReduce is the data processing layer. It is the framework for writing applications that process the vast amount of data stored in the HDFS. MapReduce processes a huge amount of data in parallel by dividing the job into a set of independent tasks (sub-job). In Hadoop, MapReduce works by breaking the processing into phases: Map and Reduce.

* Map – It is the first phase of processing. In which we specify all the complex logic/business rules/costly code. The map takes a set of data and converts it into another set of data. It also breaks individual elements into tuples (key-value pairs).
* Reduce – It is the second phase of processing. In which we specify light-weight processing like aggregation/summation. The output from the map is the input to Reducer. Then, reducer combines tuples (key-value) based on the key. And then, modifies the value of the key accordingly.

1. ***The theory of Skein***

Skein’s novel idea is to build a hash function out of a tweakable block cipher. The use of a tweakable block cipher allows Skein to hash configuration data along with the input text in every block, and make every instance of the compression function unique. This property directly addresses many attacks on hash functions, and greatly improves Skein’s flexibility.

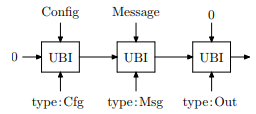
More specifically, Skein is built from these three new components:

• Threefish: Threefish is the tweakable block cipher at the core of Skein, defined with a 256-, 512-, and 1024-bit block size.

• Unique Block Iteration (UBI): UBI is a chaining mode that uses Threefish to build a compression function that maps an arbitrary input size to a fixed output size.

• Optional Argument System: This allows Skein to support a variety of optional features without imposing any overhead on implementations and applications that do not use the features

Skein is built on multiple invocations of UBI. Figure below shows Skein as a straightforward hash function. Starting with a chaining value of 0, there are three UBI invocations: one each for the configuration block, the message (up to 296 − 1 bytes long), and the output transform.



The output transform is required to achieve hashing-appropriate randomness. It also allows Skein to produce any size output up to 264 bits. If a single output block is not enough, run the output transform several times, as shown in Figure 7. The chaining input to all output transforms is the same, and the data field consists of an 8-byte counter. Essentially, this uses Threefish in counter mode. Producing large outputs is often convenient, but—of course—the security of Skein is limited by the internal state size. Notes:

* Reference to material about Skein:

<http://www.skeinhash.info/sites/default/files/skein1.3.pdf>

* Additionally, the group also researches some implement of Skein on Java, such as:

<https://www.schneier.com/academic/skein/>

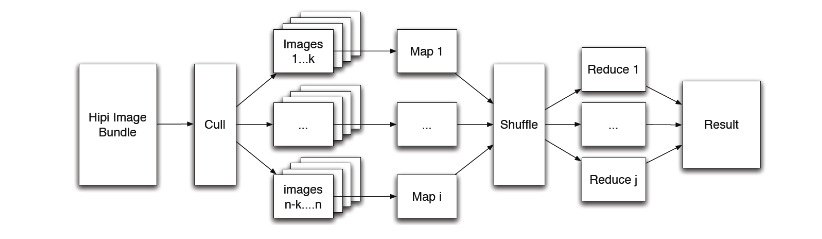
1. ***Implementation method***

* *Overview about HIPI*

HIPI is an image processing library designed to be used with the [Apache Hadoop MapReduce](http://hadoop.apache.org/) parallel programming framework. HIPI facilitates efficient and high-throughput image processing with MapReduce style parallel programs typically executed on a cluster. It provides a solution for how to store a large collection of images on the Hadoop Distributed File System (HDFS) and make them available for efficient distributed processing.

HIPI is developed and maintained by a growing number of developers from around the world. The latest release of HIPI has been tested with Hadoop 2.7.1.

* *How HIPI works*



The primary input object to a HIPI program is a [HipiImageBundle](http://hipi.cs.virginia.edu/javadoc/org/hipi/imagebundle/HipiImageBundle.html) (HIB). A HIB is a collection of images represented as a single file on the HDFS. The HIPI distribution includes [several useful tools](http://hipi.cs.virginia.edu/examples.html) for creating HIBs, including a MapReduce program that builds a HIB from a list of images downloaded from the Internet.

* + The first processing stage of a HIPI program is a culling step that allows filtering the images in a HIB based on a variety of user-defined conditions like spatial resolution or criteria related to the image metadata. This functionality is achieved through the [Culler](http://hipi.cs.virginia.edu/javadoc/org/hipi/mapreduce/Culler.html) class. Images that are culled are never fully decoded, saving processing time.

The images that survive the culling stage are assigned to individual map tasks in a way that attempts to maximize data locality, a cornerstone of the Hadoop MapReduce programming model. This functionality is achieved through the [HibInputFormat](http://hipi.cs.virginia.edu/javadoc/org/hipi/imagebundle/mapreduce/HibInputFormat.html) class. Finally, individual images are presented to the Mapper as objects derived from the [HipiImage](http://hipi.cs.virginia.edu/javadoc/org/hipi/image/HipiImage.html) abstract base class along with an associated [HipiImageHeader](http://hipi.cs.virginia.edu/javadoc/org/hipi/image/HipiImageHeader.html) object.

* + HIPI also includes support for [OpenCV](http://opencv.org/), a popular open-source computer vision library. Specifically, image classes that extend from RasterImage (such as ByteImage and FloatImage, discussed above) may be converted to [OpenCV Java Mat](http://bytedeco.org/javacpp-presets/opencv/apidocs/org/bytedeco/javacpp/opencv_core.Mat.html) objects using routines in the [OpenCVUtils](http://hipi.cs.virginia.edu/javadoc/org/hipi/opencv/OpenCVUtils.html) class. The [OpenCVMatWritable](http://hipi.cs.virginia.edu/javadoc/org/hipi/opencv/OpenCVMatWritable.html) class provides a wrapper around the OpenCV Java Mat class that can be used as a key or value object in MapReduce programs.
  + The records emitted by the Mapper are collected and transmitted to the Reducer according to the built-in MapReduce shuffle algorithm that attemps to minimize network traffic. Finally, the user-defined reduce tasks are executed in parallel and their output is aggregated and written to the HDFS.

1. ***Image dataset:***

The group downloaded the image dataset from coco webpage ([http://cocodataset.org](http://cocodataset.org/#download)). They includes 5000 pictures with all size of approximately 800MB.