MapReduce com Imagens Utilizando a Biblioteca HIPI Para Cálculo de PCA

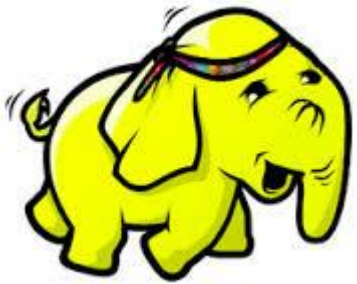
Projeto 1 - Grupo 5

Grupo 5 - Integrantes

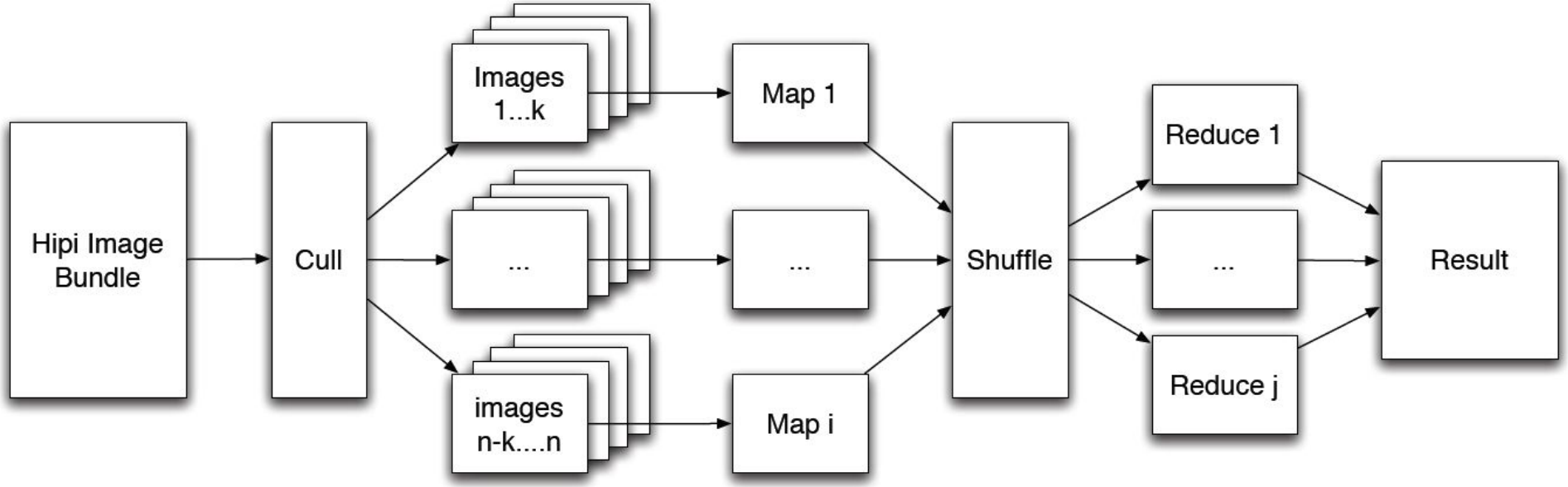
Lucas Alves Racoci	156331
Luiz Fernando Rodrigues da Fonseca	156475
Rafael Zane	157079
Rodrigo Noronha Máximo	157209

HIPI - Hadoop Image Processing Interface

- Biblioteca de processamento de imagens
- Utilizada com o Hadoop
- Armazenar uma grande coleção de imagens no HDFS
- Integração com o OpenCV
- Tipos diferentes de dados na entrada
- HIB - HIPI Image Bundle
- Documentação e instalação através do link: http://hipi.cs.virginia.edu/index.html
- Utilização de Gradle para compilar código Java e o HIPI

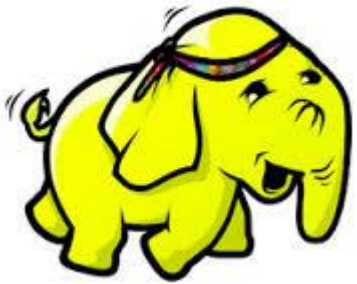


Workflow do HIPI

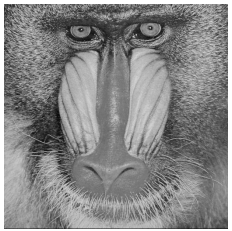


HIPI - Hadoop Image Processing Interface

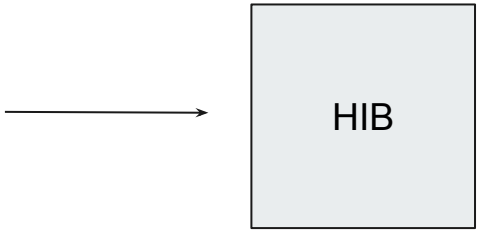
- Ferramentas para transformar um conjunto de imagens em um arquivo do tipo HIPI que serve de entrada para os processos de MapReduce que envolvem o HIPI, e o resultado já é gravado no HDFS:



$ tools/hibImport.sh ~/SampleImages sampleimages.hib

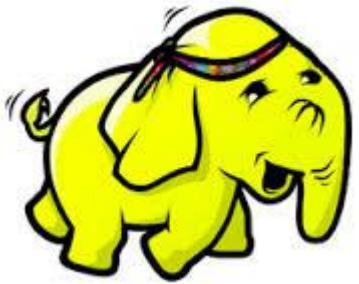


+



HIPI - Hadoop Image Processing Interface

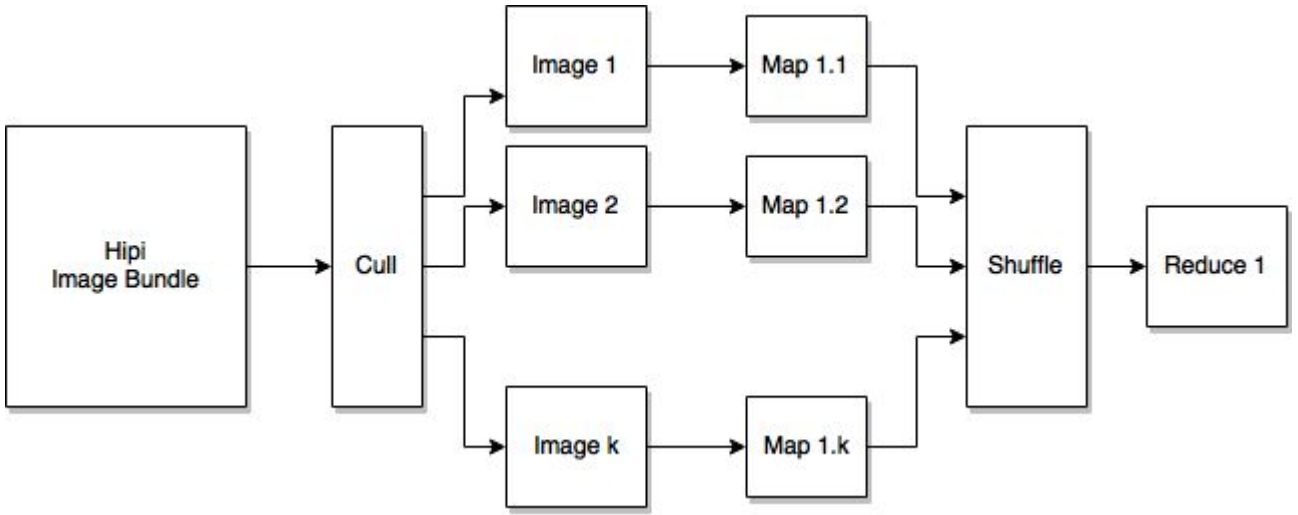
- Também existe uma ferramenta para acessar o HDFS e ver as informações de arquivos HIB:



$ tools/hibInfo.sh sampleimages.hib --show-meta

```
Input HIB: sampleimages.hib
Display meta data: true
Display EXIF data: false
IMAGE INDEX: 0
        512 x 512
        format: 2
        meta: {filename=baboon.png, source=../SampleImages/baboon.png}
IMAGE INDEX: 1
        512 x 512
        format: 2
        meta: {filename=peppers.png, source=../SampleImages/peppers.png}
Found [2] images.
```

Workflow do primeiro teste



Primeiro Teste - Média de Pixels

Configuração:

- Formato de input
- Classe Mapper
- Classe Reducer
- Tipo do par chave, valor
- Caminho para os arquivos de entrada e saída no HDFS
- http://hipi.cs.virginia.edu/gettingstarted.html

```
// Initialize and configure MapReduce job
Job job = Job.getInstance();
// Set input format class which parses the input HIB and spawns map tasks
job.setInputFormatClass(HibInputFormat.class);
// Set the driver, mapper, and reducer classes which express the computation
job.setJarByClass(HelloWorld.class);
job.setMapperClass(HelloWorldMapper.class);
job.setReducerClass(HelloWorldReducer.class);
// Set the types for the key/value pairs passed to/from map and reduce layers
job.setMapOutputKeyClass(IntWritable.class);
job.setMapOutputValueClass(FloatImage.class);
job.setOutputKeyClass(IntWritable.class);
job.setOutputValueClass(Text.class);

// Set the input and output paths on the HDFS
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

// Execute the MapReduce job and block until it complets
boolean success = job.waitForCompletion(true);
```

Mapper

- Cria um vetor com o valor médio de vermelho, verde e azul
- Percorre a matriz da imagem obtendo os valores de cada cor para cada pixel
- Cria uma imagem (vetor com três posições) com a soma dos valores de cada cor e divide pela quantidade total de pixels
- Salva no contexto do hipi a chave 1 com o valor sendo a imagem

```
// Get pointer to image data
float[] valData = value.getData();
// Initialize 3 element array to hold RGB pixel average
float[] avgData = {0,0,0};
// Traverse image pixel data in raster-scan order and update running average
for (int j = 0; j < h; j++) {
  for (int i = 0; i < w; i++) {
    avgData[0] += valData[(j*w+i)*3+0]; // R
    avgData[1] += valData[(j*w+i)*3+1]; // G
    avgData[2] += valData[(j*w+i)*3+2]; // B
  }
}
// Create a FloatImage to store the average value
FloatImage avg = new FloatImage(1, 1, 3, avgData);
// Divide by number of pixels in image
avg.scale(1.0f/(float)(w*h));
// Emit record to reducer
context.write(new IntWritable(1), avg);
```

Reducer

- Cria a imagem que conterá a saída
- Soma os valores recebidos dos mappers
- Obtém uma nova média
- Imprime o resultado e coloca o par chave e valor que será escrito no HDFS

```
// Create FloatImage object to hold final result
FloatImage avg = new FloatImage(1, 1, 3);
// Initialize a counter and iterate over IntWritable/FloatImage records from mapper
int total = 0;
for (FloatImage val : values) {
  avg.add(val);
  total++;
}
if (total > 0) {
  // Normalize sum to obtain average
  avg.scale(1.0f / total);
  // Assemble final output as string
      float[] avgData = avg.getData();
  String result = String.format("Average pixel value: %f %f %f", avgData[0], avgData[1], avgData[2]);
  // Emit output of job which will be written to HDFS
  context.write(key, new Text(result));
```

Resultados

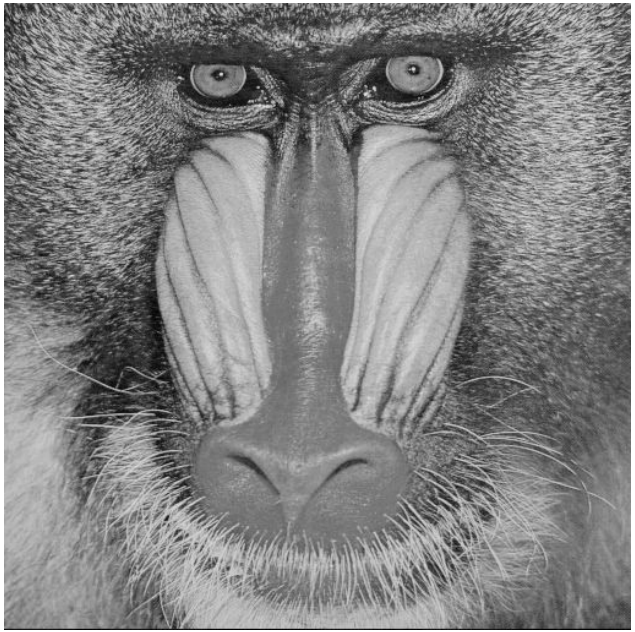
- Para rodar o código e ver o resultado:

$ hadoop jar build/libs/helloWorld.jar sampleimages.hib sampleimages_average

$ hdfs dfs -cat sampleimages_average/part-r-00000

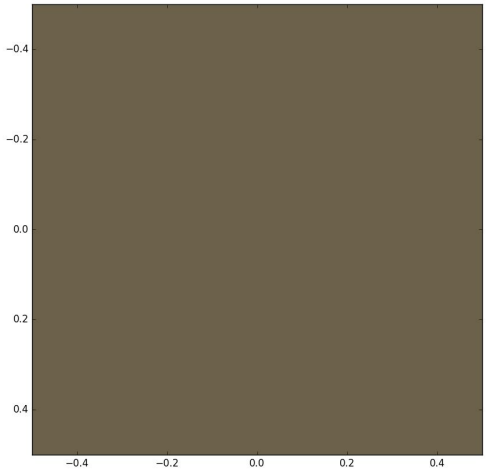
1 Average pixel value: 0,279852 0,260834 0,220869

Resultados



Resultados

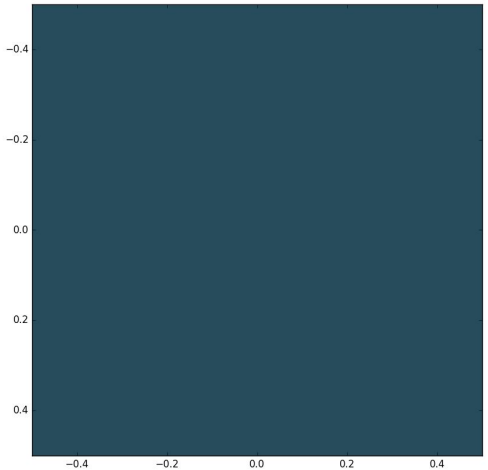
Average pixel value: 0,424189 0,387541 0,295646



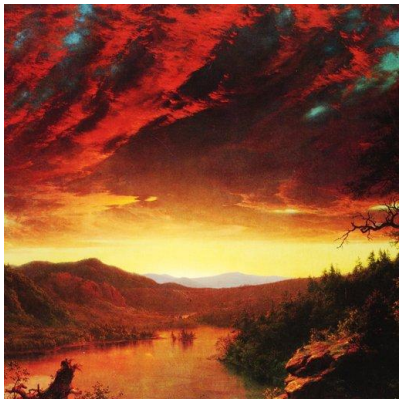
Resultados

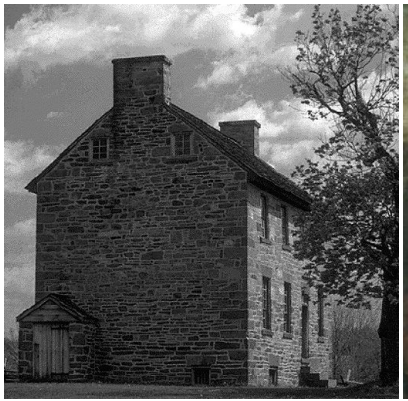
Resultados

Average pixel value: 0,155569 0,302869 0,366745



Resultados

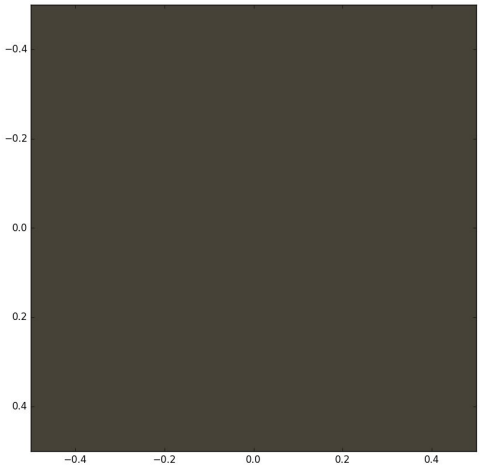





Resultados

Average pixel value: 0,279852 0,260834 0,220869

```
real    0m8.786s
user    0m8.498s
sys     0m4.098s
```



OpenCV

- Open Source
- Biblioteca de visão computacional e machine learning em C++
- Cerca de 2500 algoritmos otimizados
- Interfaces em C++, C, Python, Java e MATLAB

Segundo Teste - Matriz de Covariância (PCA)

$$\Sigma = \mathbf{E}\left[(\mathbf{X} - \mathbf{E}[\mathbf{X}])\,(\mathbf{X} - \mathbf{E}[\mathbf{X}])^{\mathrm{T}}\right]$$

$$= \frac{1}{n}\sum_{i}^{n}(x_i - \bar{x})(x_i - \bar{x})^T = \frac{1}{n}\hat{X}\,\hat{X}^T$$

$$= \begin{pmatrix} | & & | \\ \hat{x_i} & \dots & \hat{x_n} \\ | & & | \end{pmatrix} \begin{pmatrix} - & \hat{x_1} & - \\ & \vdots & \\ - & \hat{x_n} & - \end{pmatrix}$$

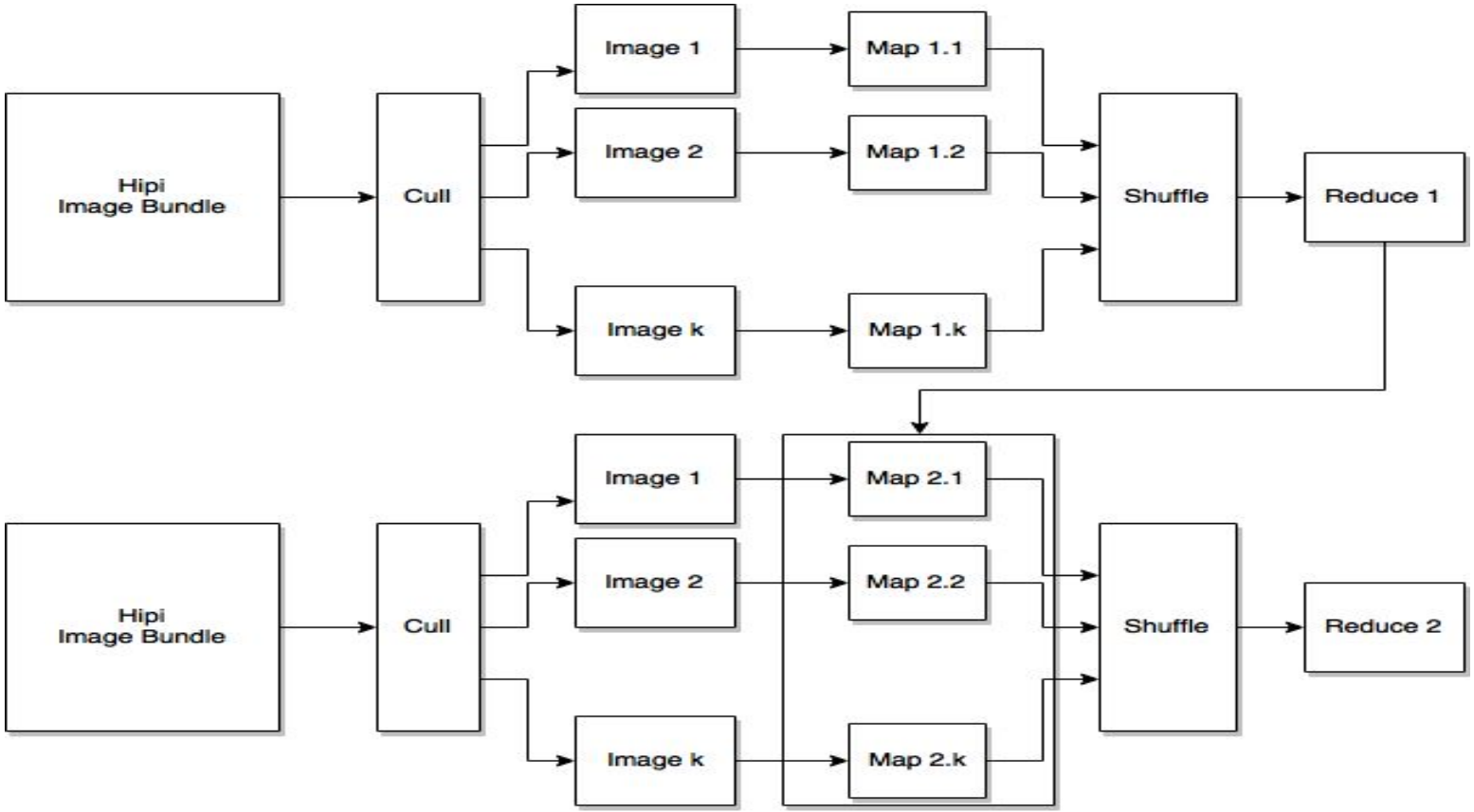
$$= \begin{pmatrix} A & B \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix}$$

$$= AA^T + BB^T$$

- Objetivo: Obter informações estatísticas de segunda ordem entre recortes de 64x64 pixels de imagens naturais.
- Os autovetores de maior autovalor da matriz têm uma utilidade tão grande em outras áreas de Data Analysis que são chamados de Componentes principais, daí o nome PCA (Principal Component Analysis)

Fontes:

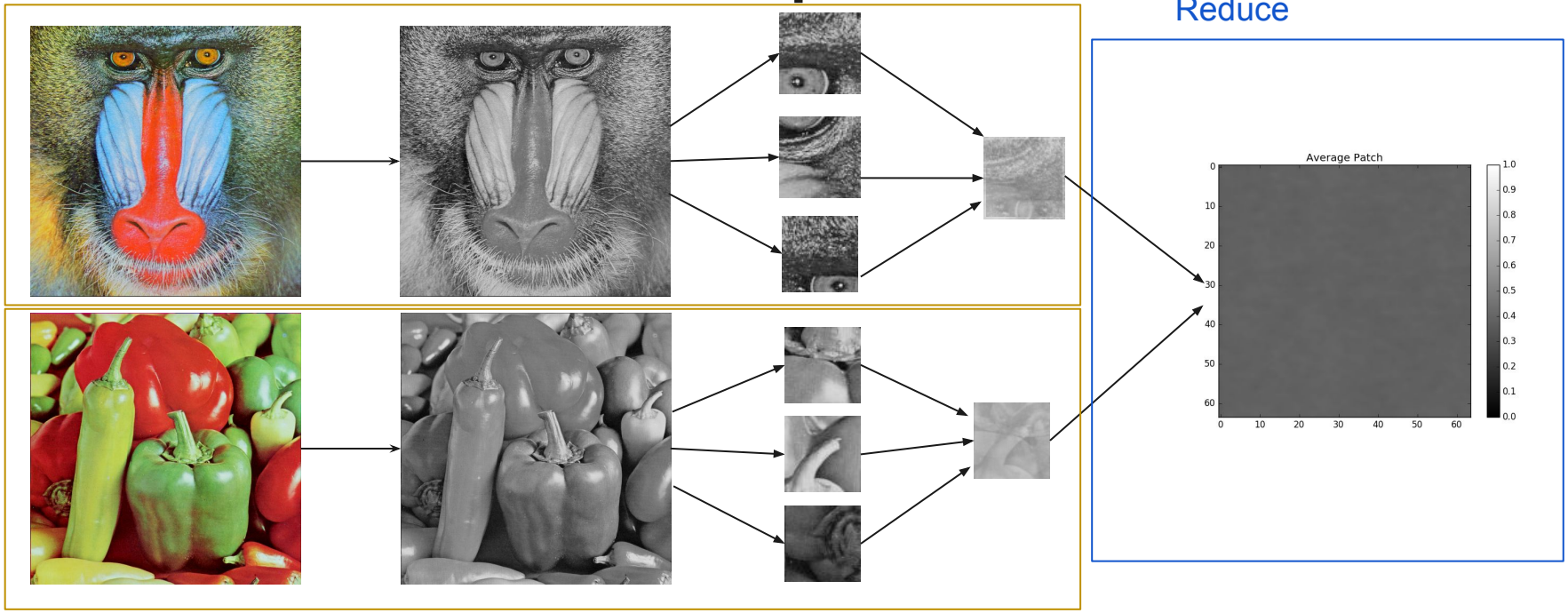
- http://hipi.cs.virginia.edu/examples/pca.html
- The Principal Components of Natural Images by Hancock


Hipi
Image Bundle
Cull
Image 1
Image 2
Image k
Map 1.1
Map 1.2
Map 1.k
Shuffle
Reduce 1
Hipi
Image Bundle
Cull
Image 1
Image 2
Image k
Map 2.1
Map 2.2
Map 2.k
Shuffle
Reduce 2

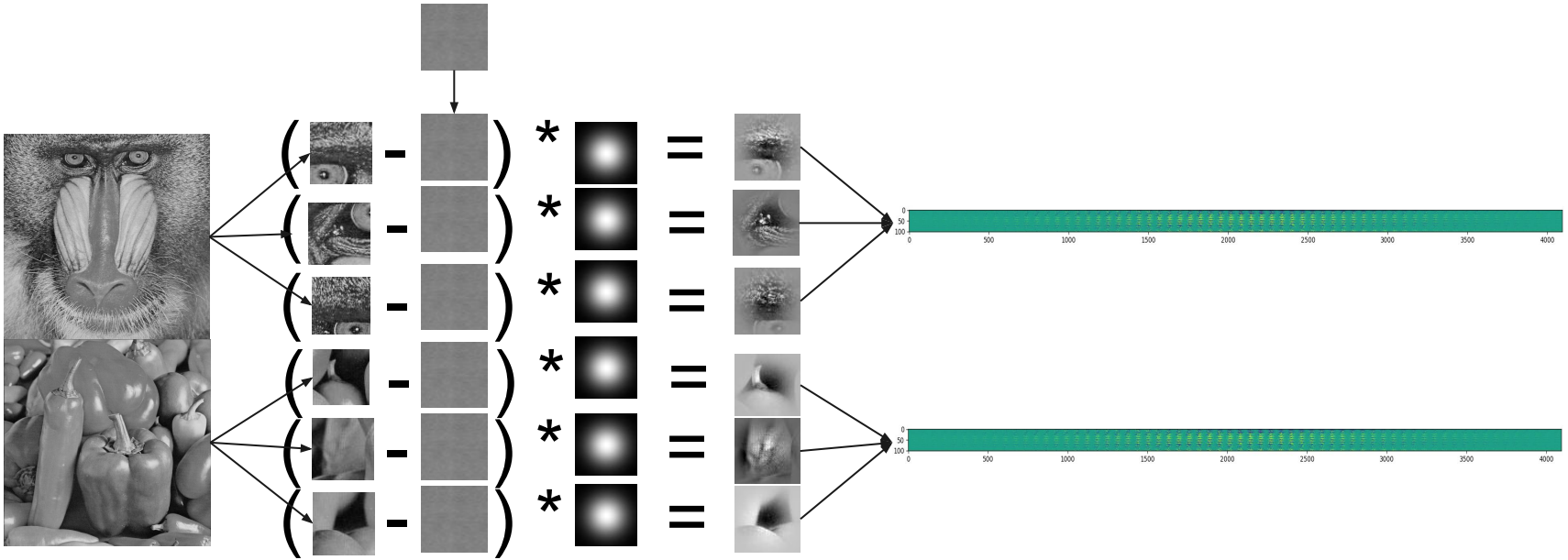
Primeiro MapReduce

Map

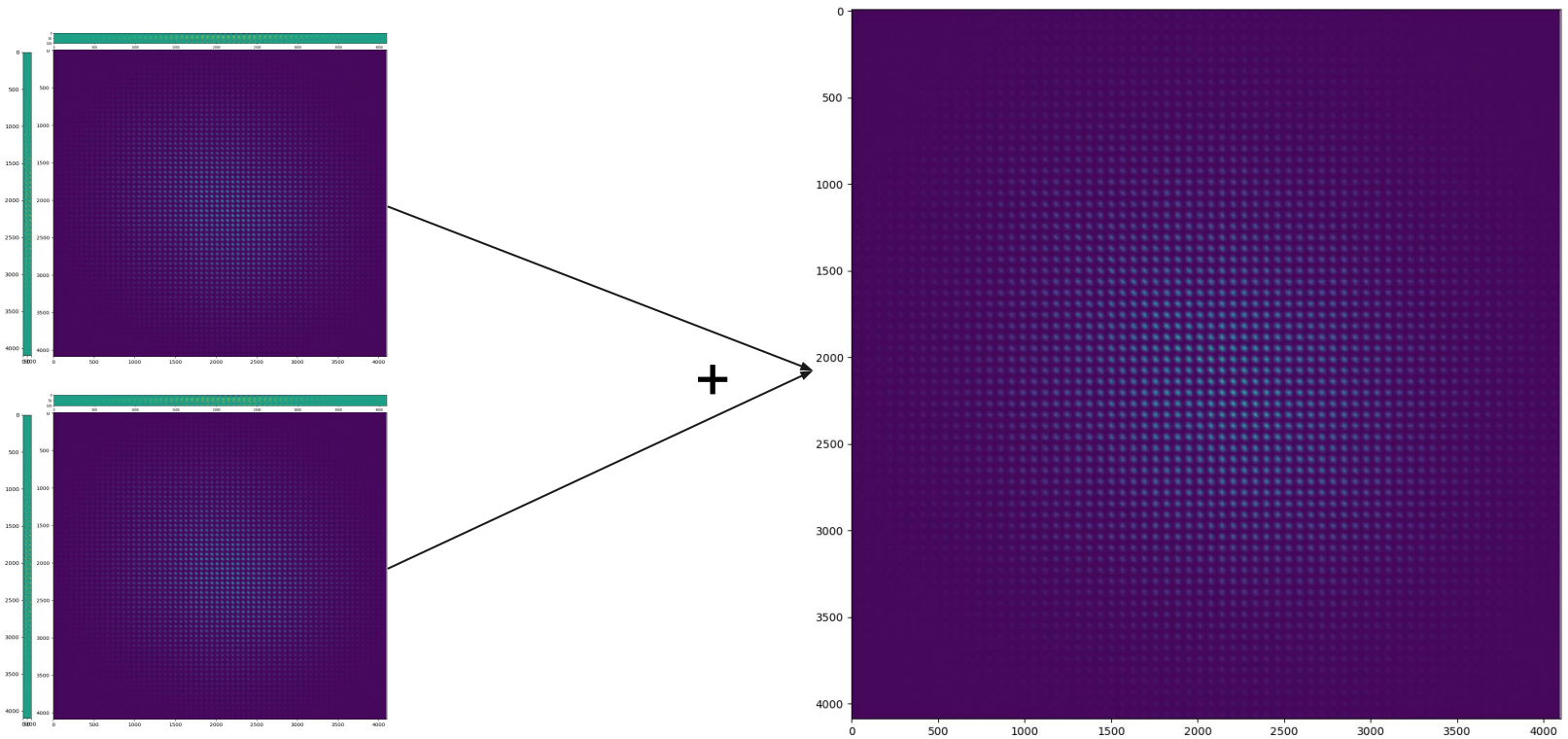
Reduce



Segundo Map


0
50
100
0
500
1000
1500
2000
2500
3000
3500
4000
0
50
100
0
500
1000
1500
2000
2500
3000
3500
4000

Segundo Reduce

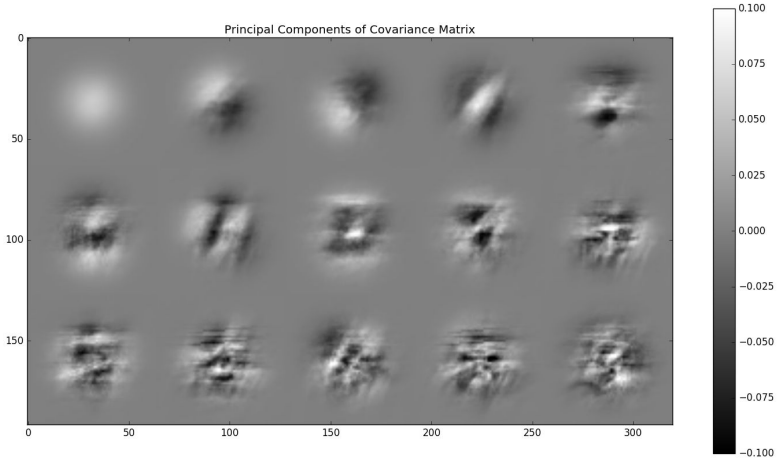


Resultado

A imagem ao lado mostra as 15 primeiras componentes principais (autovetores) da matriz de covariância

Tempo de execução:

real 0m34.810s
user 0m36.481s
sys 0m5.959s



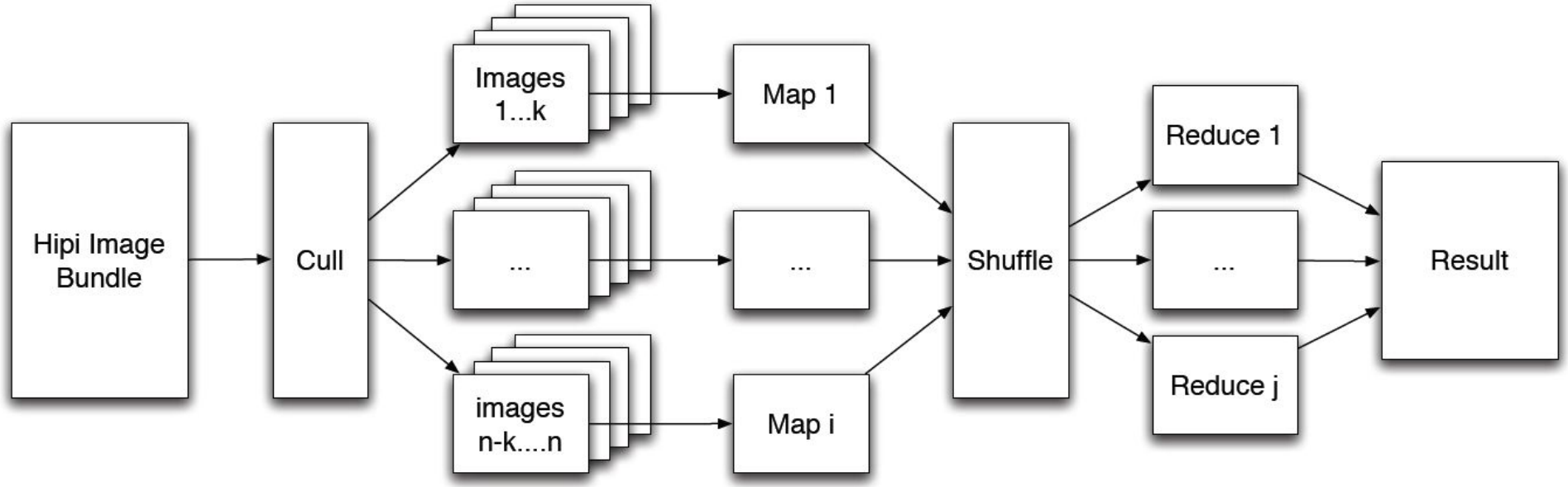
Terceiro Teste - Borramento

Configuração:

- Tipo dos pares chave, valor
- OpenCV
- Saída binária
- hashCode do Java para diferenciar as imagens

```
// Initialize and configure MapReduce job
Job job = Job.getInstance();
// Set input format class which parses the input HIB and spawns map tasks
job.setInputFormatClass(HibInputFormat.class);
// Set the driver, mapper, and reducer classes which express the computation
job.setJarByClass(Gaussiana.class);
job.setMapperClass(GaussianaMapper.class);
job.setReducerClass(GaussianaReducer.class);
// Set the types for the key/value pairs passed to/from map and reduce layers
job.setMapOutputKeyClass(IntWritable.class);
job.setMapOutputValueClass(OpenCVMatWritable.class);
job.setOutputKeyClass(NullWritable.class);
job.setOutputValueClass(OpenCVMatWritable.class);
job.setOutputFormatClass(BinaryOutputFormat.class);

// Set the input and output paths on the HDFS
FileInputFormat.setInputPaths(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

// Execute the MapReduce job and block until it complets
boolean success = job.waitForCompletion(true);
```

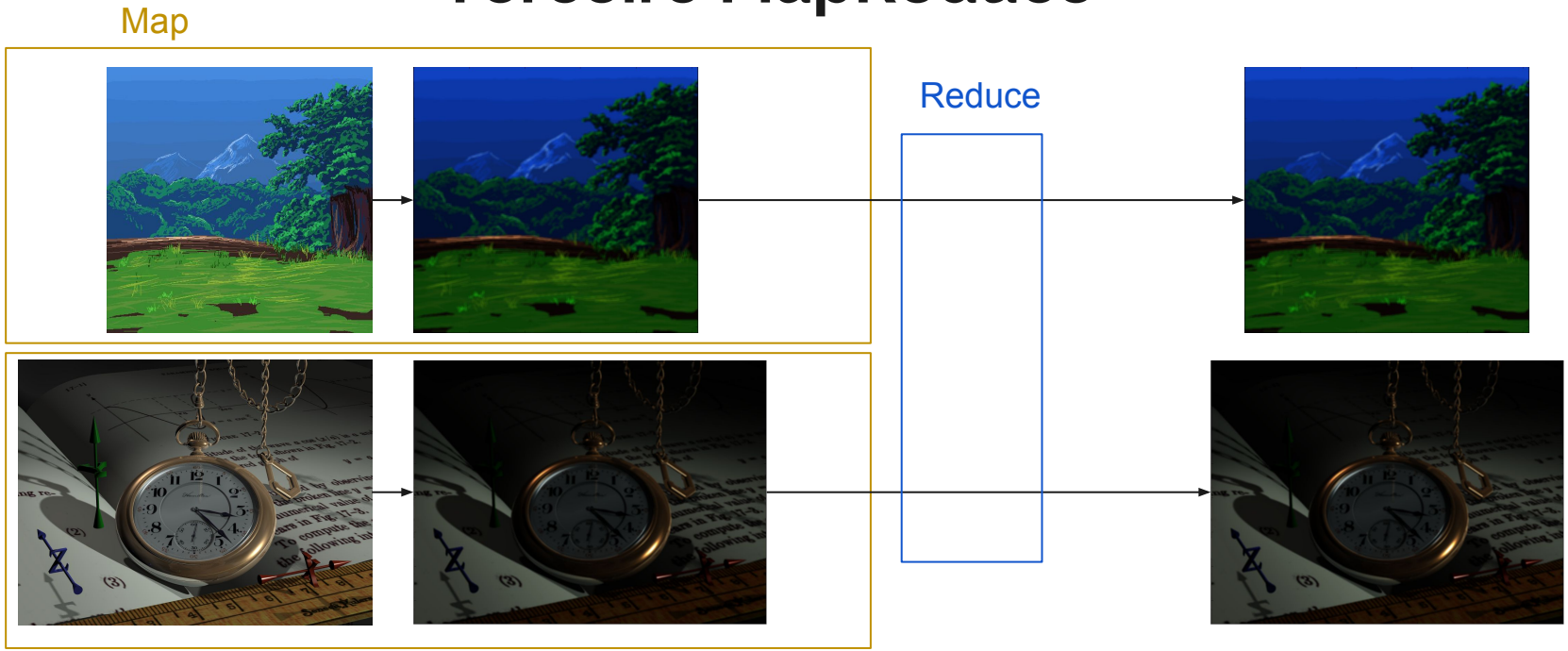
MapReduce


Hipi Image Bundle
Cull
Images 1...k
...
images n-k....n
Map 1
...
Map i
Shuffle
Reduce 1
...
Reduce j
Result

MapReduce

```
public static final int sizeX = 5;
public static final int sizeY = 5;

public static class GaussianaMapper extends Mapper<HipiImageHeader, FloatImage, IntWritable, OpenCVMatWritable> {

  public void map(HipiImageHeader key, FloatImage image, Context context)
      throws IOException, InterruptedException {

    Mat cvImageRGB = OpenCVUtils.convertRasterImageToMat(image);
    Mat cvGaussianRGB = new Mat();
    opencv_imgproc.blur(cvImageRGB, cvGaussianRGB, new Size(sizeX, sizeY));
    context.write(new IntWritable(cvGaussianRGB.hashCode()), new OpenCVMatWritable(cvGaussianRGB));

  } // map()

} // GaussianaMapper

public static class GaussianaReducer extends Reducer<IntWritable, OpenCVMatWritable, NullWritable, OpenCVMatWritable> {

  public void reduce(IntWritable key, Iterable<OpenCVMatWritable> values, Context context)
      throws IOException, InterruptedException {

    for (OpenCVMatWritable value : values) {
      context.write(NullWritable.get(), value);
    }

  } // reduce()

} // GaussianaReducer
```

Terceiro MapReduce



Resultado

Resultado

Conclusão

- HIPI é um framework muito poderoso e robusto para se integrar o processamento de imagens com o sistema do Hadoop
- Isso inclui as ferramentas de conversão de imagens para o formato HIB disponíveis e as bibliotecas integradas, como o OpenCV
- O Hadoop é uma ferramenta que pode ajudar o processamento em paralelo de aplicações, mas ainda possui algumas limitações e funcionalidades extras podem ser necessárias para processar o resultado de saída do sistema (arquivo binário de saída com as imagens concatenadas)