Lab 5 of SP19-BL-ENGR-E599-30563

Lab 5 provides hands-on of single node performance optimization within Harp/Harp-DAAL framework

Goal

- Installation of Harp-DAAL
- Learn API of Harp-DAAL
- Walk through of the K-means example in Harp-DAAL
- Compare performance of K-means between Harp and Harp-DAAL

Deliverables

Submit the experiment results to Canvas.

Run K-means on a single mapper and measure performance gap between Harp and Harp-DAAL implementations.

- Compare Harp and Harp-DAAL K-means static scheduler computation times using the parameters below.
  - Pts=100000, Centroids=10, Dimension=100, Iterations=100, Mappers=1, CollectiveComm=allreduce
  - Thd=4, 8, 16

Evaluation

Lab participation: credit for 1 point based upon a successful completion of the lab tasks

Harp-DAAL

Harp-DAAL improves the thread-level performance further by using hardware acceleration native libraries from Intel DAAL (Data Analytics and Acceleration Library). Therefore, the inter-mapper communication is still handled by Harp runtime while the local computation is offloaded to native binaries provided by Intel DAAL.

Installation of Harp-DAAL

The DAAL Library is already included in the path `harp/third_party/daal-2018/lib`, where

- `lib/daal.jar` is the Java API.
- `lib/intel64_lin/libJavaAPI.so` is the native shared lib of DAAL implementations.

To enable DAAL in Harp, please uncomment *ml* module in `harp/pom.xml'

```
12     <modules>
13         <module>core</module>
14         <module>contrib</module>
15         <module>ml</module>
16         <!-- <module>distribution</module> -->
17         <!--<module>experimental</module>-->
18     </modules>
```

, and recompile the harp source code.

```
cd harp
mvn clean package -Phadoop-2.6.0
```

Harp-DAAL API

The API of Harp-DAAL is located at `harp/core/harp-daal-interface/src/main/java/edu/iu/` , where it has subfolders

- `data_aux` : auxilary functions such as initializer and some configuration constants.
- `data_gen` : generate synthetic datasets in a distributed way.
- `datasource` : load in data block from HDFS and convert them to Intel DAAL data structure.
- `data_comm` : communication wrapper that connects Harp and Intel DAAL data type API

Please read the Harp-DAAL API page at our website for more details.

Initialization

Harp-DAAL provides a class *Initialize* to facilitate the initialization stage of Harp-DAAL application from Hadoop environment.

```
import edu.iu.data_aux.*;
```

To create an *Initialize* instance from the *launcher* class

```
// obtain the hadoop configuration
Configuration conf = this.getConf();
Initialize init = new Initialize(conf, args);
```

To load all the native libraries required by Harp-DAAL

```
init.loadDistributedLibs();
```

To load all the system-wide arguments

```
init.loadSysArgs();
```

To load the application-wide arguments.

```
conf.setInt(HarpDAALConstants.FILE_DIM, Integer.parseInt(args[init.getSysArgNum()]));
conf.setDouble(Constants.MIN_SUPPORT, Double.parseDouble(args[init.getSysArgNum()+1])
);
conf.setDouble(Constants.MIN_CONFIDENCE, Double.parseDouble(args[init.getSysArgNum()+
2]));
```

Data I/O

Harp-DAAL has a dedicated I/O module to load data from HDFS files to memory space accessible by native kernels of DAAL. It currently supports loading dense data (HomogenNumericTable) and CSR data (CSRNumericTable). For dense matrix, we support parallel I/O by using multiple java threads.

To use I/O module:

```
import edu.iu.datasource.*;
```

To create a *HarpDAALDataSource* data loader for dense data:

```
HarpDAALDataSource datasource = new HarpDAALDataSource(harpThreads, conf);
```

To load a dense training dataset stored in CSV files from HDFS

```
NumericTable inputTable = datasource.createDenseNumericTableInput(inputFiles, nFeatur
es, sep,  daal_Context);
```

To load a CSR training dataset from HDFS

```
NumericTable inputTable = datasource.loadCSRNumericTable(inputFiles, sep, context);
```

Inter-mapper Communication

Harp-DAAL provides users a module of interprocess (mappers) communication operations with high-level APIs.

To use the communication module:

```
import edu.iu.data_comm.*;
```

To create a communication module

```
HarpDAALComm comm = new HarpDAALComm(self_id, master_id, num_mappers, context, mapper
);
```

To use broadcast *input* objects from master mapper

```
SerializableBase output = comm.harpdaal_braodcast(input, contextName, operationName,
useSpanTree);
```

Where, *input* and *output* object must be java classes extended from Intel DAAL *SerializableBase* class.

Allgather operation first gathers the data from all mappers to the master mapper, and then it broadcasts the output from the master mapper back to all the worker mappers.

```
SerializableBase[] outputs = comm.harpdaal_allgather(input, contextName, operationNam
e);
```

Code Walk-through of K-means in Harp-DAAL

The Harp-DAAL source codes are at
`harp/ml/daal/src/main/java/edu/iu/daal_kmeans/regroupallgather/`

We will go through the following two files

- `KMeansDaalLauncher.java` specifies the configuration and launch of Harp job
- `KMeansDaalCollectiveMapper.java` implements the K-means mapper function

Configuration and Launch of K-means

In file `KMeansDaalLauncher.java`

```
60 @Override
61 public int run(String[] args) throws Exception {
62
63     /* Put shared libraries into the distributed cache */
64     Configuration conf = this.getConf();
65
66     Initialize init = new Initialize(conf, args);
67
68     /* Put shared libraries into the distributed cache */
69     init.loadDistributedLibs();
70
71     // load args
72     init.loadSysArgs();
73
74     //load app args
75     conf.setInt(HarpDAALConstants.FILE_DIM, Integer.parseInt(args[init.getSysArgNu
m()]));
76     conf.setInt(HarpDAALConstants.FEATURE_DIM, Integer.parseInt(args[init.getSysAr
gNum()+1]));
77     conf.setInt(HarpDAALConstants.NUM_CENTROIDS, Integer.parseInt(args[init.getSys
ArgNum()+2]));
78
79     // config job
80     System.out.println("Starting Job");
81     long perJobSubmitTime = System.currentTimeMillis();
82     System.out.println("Start Job#"  + " "+ new SimpleDateFormat("HH:mm:ss.SSS").f
ormat(Calendar.g
83     Job kmeansJob = init.createJob("kmeansJob", KMeansDaalLauncher.class, KMeansDa
alCollectiveMapp
84     ...
   }
```

Data I/O and Conversion

In file `KMeansDaalCollectiveMapper.java`

```
159   private void runKmeans(Context context) throws IOException
160   {
161
162       long start_execution = System.currentTimeMillis();
163       long compute_time = 0;
164       long comm_time = 0;
165       long start_comp =0;
166       long start_comm = 0;
167
168       // ---------------- load in training data ----------------
169       NumericTable trainingdata_daal = this.datasource.createDenseNumericTable(th
is.inputFiles, th
170
171       // ---------- load in centroids (model) data ----------
172       // create a table to hold centroids data
173       Table<DoubleArray> cenTable = new Table<>(0, new DoubleArrPlus());
174       if (this.isMaster())
175       {
176           createCenTable(cenTable);
177           loadCentroids(cenTable);
178       }
179
180       // Bcast centroids to other mappers
181       bcastCentroids(cenTable, this.getMasterID());
          ...
213   }
```

Local Computation

In file `KMeansDaalCollectiveMapper.java` and function `runKmeans`

```
182  // create a daal kmeans kernel object
183  DistributedStep1Local kmeansLocal = new DistributedStep1Local(daal_Context, Doub
le.class, Me
184  // set up input training data
185  kmeansLocal.input.set(InputId.data, trainingdata_daal);
186  // specify the threads used in DAAL kernel
187  Environment.setNumberOfThreads(numThreads);
188  // create cenTable at daal side
189  NumericTable cenTable_daal = createCenTableDAAL();
190
191  // start the iteration
192  for (int i = 0; i < numIterations; i++) {
193
194      start_comp = System.currentTimeMillis();
195      //Convert Centroids data from Harp to DAAL
196      printTable(cenTable, 10, 10, i);
197      convertCenTableHarpToDAAL(cenTable, cenTable_daal);
198      // specify centroids data to daal kernel
199      kmeansLocal.input.set(InputId.inputCentroids, cenTable_daal);
200      // first step of local computation by using DAAL kernels to get partial resu
lt
201      PartialResult pres = kmeansLocal.compute();
202
203      compute_time += (System.currentTimeMillis() - start_comp);
205      ...
212  }
```

Inter-mapper Communication

In file `KMeansDaalCollectiveMapper.java` and function `runKmeans`

```
206      // comm by regroup-allgather
207      comm_regroup_allgather(cenTable, pres);
208      // comm_allreduce(cenTable, pres);
209      // comm_broadcastreduce(cenTable, pres);
210      // comm_push_pull(cenTable, pres);
```

How to run a Harp-DAAL application ?

The script of harp-daal kmeans is `harp/ml/daal/test_scripts/harp-daal-kmeans.sh`

```
24 ## copy required third_party native libs to HDFS
25 hdfs dfs -mkdir -p /Hadoop
26 hdfs dfs -mkdir -p /Hadoop/Libraries
27 hdfs dfs -rm /Hadoop/Libraries/*
28 hdfs dfs -put ${HARP_ROOT}/third_party/daal-2018/lib/intel64_lin/libJavaAPI.so /Ha
doop/Libraries/
29 hdfs dfs -put ${HARP_ROOT}/third_party/tbb/lib/intel64_lin/gcc4.4/libtbb* /Hadoop/
Libraries/
30
31 export LIBJARS=${HARP_ROOT}/third_party/daal-2018/lib/daal.jar
```

Line 24 to 31 is different from the k-means script of Harp. Harp-DAAL requires that the native daal libraries shall be loaded into HDFS, which would be stored in the distributed cache of hadoop in the runtime.

```
39 ## parameters
40 # num of training data points
41 Pts=100000
42 # num of training data centroids
43 Ced=10
44 # feature vector dimension
45 Dim=100
46 # file per mapper
47 File=5
48 # iteration times
49 ITR=100
50 # memory allocated to each mapper (MB)
51 Mem=110000
52 GenData=true
53 # num of mappers (nodes)
54 Node=1
55 # num of threads on each mapper(node)
56 Thd=24
57 Dataset=kmeans-P$Pts-C$Ced-D$Dim-F$File-N$Node
58
59 logName=Test-daal-kmeans-P$Pts-C$Ced-D$Dim-F$File-ITR$ITR-N$Node-Thd$Thd.log
60 echo "Test-daal-kmeans-P$Pts-C$Ced-D$Dim-F$File-ITR$ITR-N$Node-Thd$Thd Start"
61 hadoop jar harp-daal-0.1.0.jar edu.iu.daal_kmeans.regroupallgather.KMeansDaalLaunc
her -libjars ${LIB}
62 echo "Test-daal-kmeans-P$Pts-C$Ced-D$Dim-F$File-ITR$ITR-N$Node-Thd$Thd End"
```

Line 39 to 62 specify the parameters for k-means program, and `-libjars ${LIB}` links k-means against the `daal.jar`.

```
64 if [ -f ${logDir}/evaluation ];then
65     rm ${logDir}/evaluation
66 fi
67
68 if [ -f ${logDir}/output ];then
69     rm ${logDir}/output
70 fi
71
72 hdfs dfs -get /Hadoop/kmeans-work/evaluation ${logDir}
73 hdfs dfs -get /Hadoop/kmeans-work/centroids/out/output ${logDir}
```

Finally is the output of the program fetched from HDFS.

Experiments

- For the Harp test, make sure that you use static scheduler. (We changed it to dynamic scheduler last week. Don't forget to change it back and build). Or you can use the results from the last submission.

We changed our scheduler to dynamic scheduler last week. So, you need to change it to static scheduler and build it with maven:

```
cd ~/Labsession/harp
mvn clean package -Phadoop-2.6.0
## copy compiled jars to Hadoop directory
cp core/harp-hadoop/target/harp-hadoop-0.1.0.jar $HADOOP_HOME/share/hadoop/mapreduce/
cp core/harp-collective/target/harp-collective-0.1.0.jar $HADOOP_HOME/share/hadoop/ma
preduce/
cp core/harp-daal-interface/target/harp-daal-interface-0.1.0.jar $HADOOP_HOME/share/h
adoop/mapreduce/
cp third_party/*.jar $HADOOP_HOME/share/hadoop/mapreduce/
```

run the `k-means.sh` in the `Labsession/harp/contrib/test_scripts/`

- For the Harp-DAAL experiments, change collective communication method to allreduce (see Inter-mapper Communication section above) and build it again.

Run `~/Labsession/harp/ml/daal/test_scripts/harp-daal-kmeans.sh` for the Harp-DAAL k-means experiments.