Lab 4 of FA18-BL-ENGR-E599-33796

Lab 4 provides hands-on of single node performance optimization within Harp framework

Goal

- Static and Dynamic Thread Scheduling

Deliverables

Submit the experiment results to Canvas.

- The performance comparison results are in the form of a table (e.g., xlsx sheet or plain csv text file) that records the execution time (ms).
- The source code of dynamic scheduler (two Java files)
  - `harp/contrib/src/main/java/edu/iu/kmeans/common/calcCenTask.java`
  - `harp/contrib/src/main/java/edu/iu/kmeans/common/Utils.java`

Static Scheduler vs. Dynamic Scheduler

Run Harp K-means on a single mapper for allreduce and measure the performance

- Thread scaling of static thread scheduler
  - Pts=100000, Centroids=10, Dimension=100, Iterations=100
  - Thd=1, 2, 4, 8, 16

- Add the codes of Dynamic scheduler
- Compare static scheduler with dynamic scheduler
  - Pts=100000, Centroids=10, Dimension=100, Iterations=100
  - Thd=4, 8, 16

Evaluation

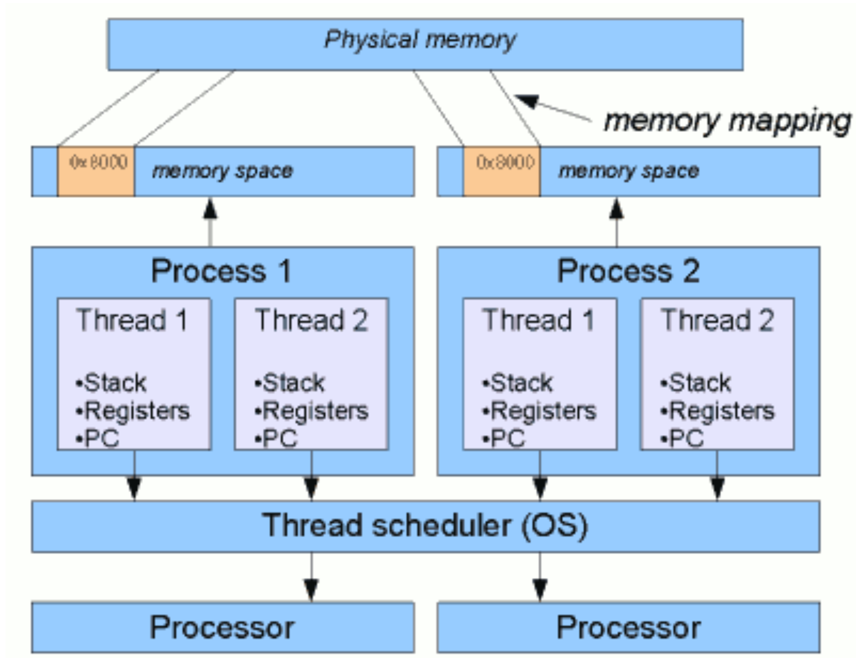
Lab participation: credit for 1 point based upon a successful completion of the lab tasks

Prerequisite

You should have the labsession branch of the Harp on your node. If you don't have, please install it following the Lab 3 instructions.

Hadoop should be already running. You can check it using `jps` command. You should see 6 entries.

Have some basic ideas of the difference between *thread* and *process*



Thread vs. Process Illustration

Thread Scheduler in Harp: Static vs. Dynamic

Harp uses multi-threading programming of Java threads to exploit the intra-node parallelism brought by the multi/many-core processors. It provides both of static and dynamic scheduler for completing parallel tasks.

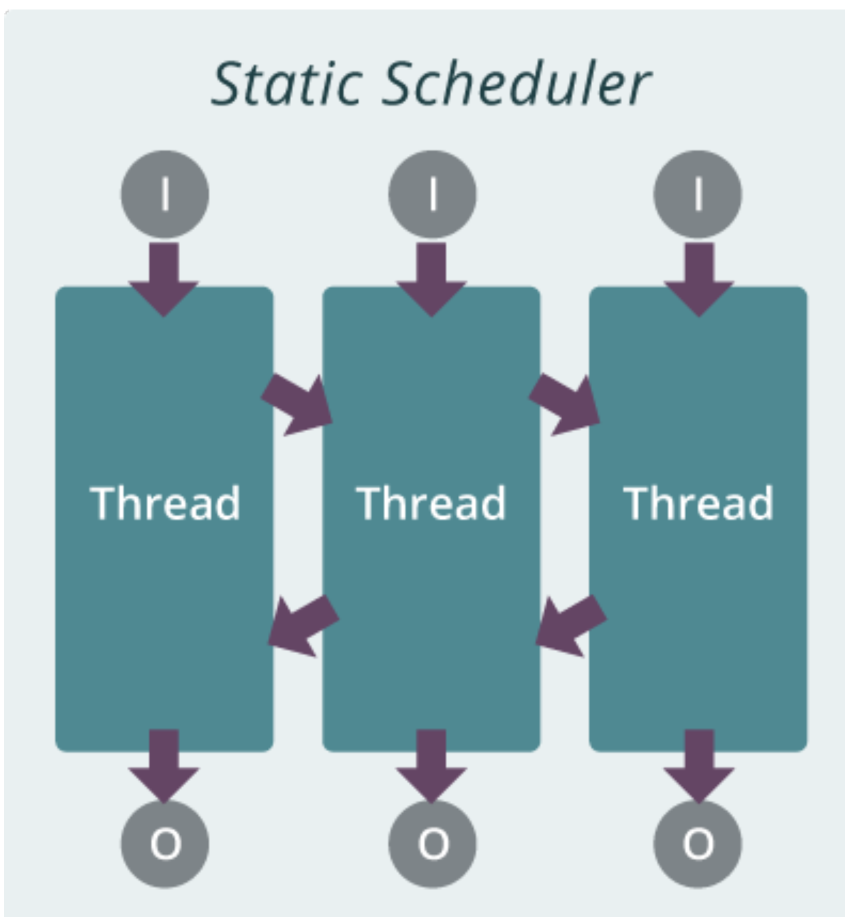
Thread Task

Harp provides an abstract class named *Task* to implement the thread-level computation upon partitioned workload. The users could spawn a certain number of task executors, usually the number of executors is equal to the number of available threads, and store them into a Java LinkedList. The user must override the *run* function, which fetches tasks from a queue either in a static or a dynamic way.

```
//Given three int[] data, find the maximum element in each array.
//First of all, we need to define the `task`.
public class FindMaxTask extends Task<int[], Integer> {

    @Override
    public Integer run(int[] input) throws Exception {
    // TODO Auto-generated method stub
    int max = Integer.MIN_VALUE;
    for(int i=0; i<input.length; i++){
        if(max < input[i]){
        max = input[i];
        }
    }
    return max;
    }
}
```

Also, each task could have its own private data members and other member functions.

Static Scheduler



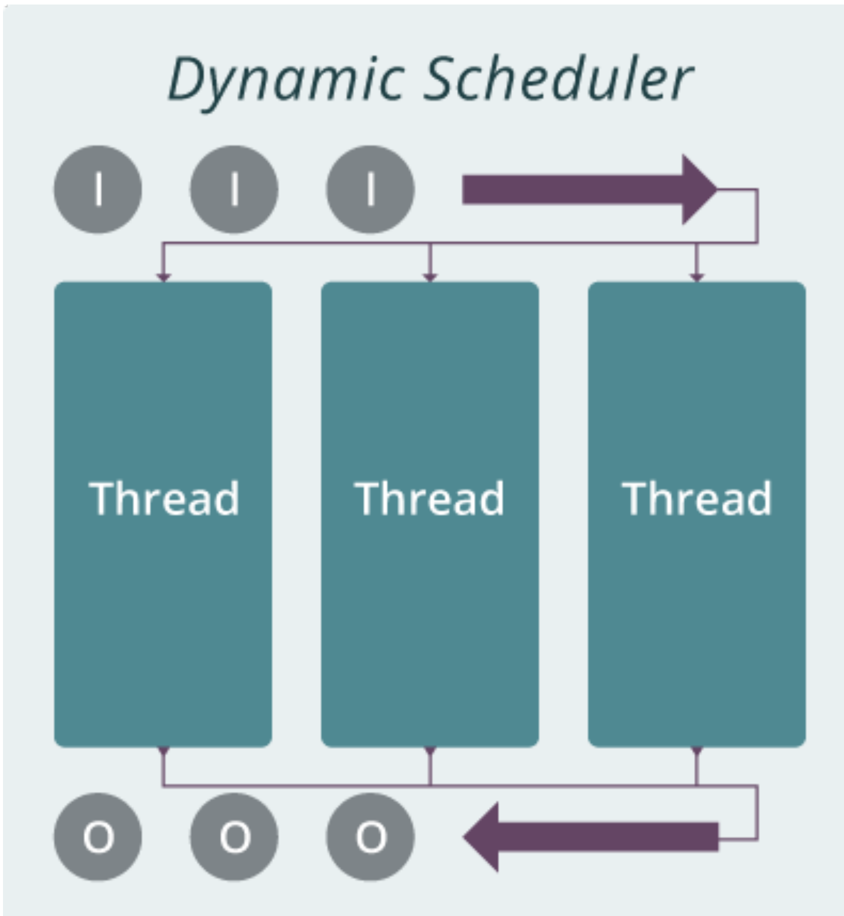
Static Scheduler in Harp

1. All computation models can use this scheduler.
2. Each thread has its own input and output queue.
3. Inputs can be submitted to another thread by each thread.
4. The main thread can retrieve outputs from each task's output queue.

```java
public void findMaxs(){
    int numThreads = 3;
    /*
     * initialize tasks. numThreads is the number of threads. Here the
     * number of tasks we launched equals to numThreads
     */
    List<FindMaxTask> maxTasks = new LinkedList<>();
    for (int i = 0; i < numThreads; i++) {
        maxTasks.add(new FindMaxTask());
    }
    /*
     * initialize the static scheduler; The data type of input is CenPair,
     * the data type of output is Object; The task is MaxTask
     */
    StaticScheduler<int[], Integer, FindMaxTask> maxCompute = new StaticScheduler<>(maxTasks);
    /* Start the Static Scheduler */
    maxCompute.start();

    int[] list1 = new int[] { 1, 2, 3, 4, 5 };
    int[] list2 = new int[] { 14, 5, 6, 7, 8, 1 };
    int[] list3 = new int[] { 53, 43, -1, 43, 63 };

    /* Specify taskID and take inputs */
    maxCompute.submit(1,list1);
    maxCompute.submit(2,list2);
    maxCompute.submit(3,list3);
    /* Get results of task 1*/
    while (maxCompute.hasOutput(1)) {
        Integer out = maxCompute.waitForOutput(1);
        System.out.println(out);
    }

    /* Get results of task 2*/
    while (maxCompute.hasOutput(2)) {
        Integer out = maxCompute.waitForOutput(2);
        System.out.println(out);
```

```
    }

    /* Get results of task 3*/
    while (maxCompute.hasOutput(3)) {
        Integer out = maxCompute.waitForOutput(3);
        System.out.println(out);
    }
}
```

Dynamic Scheduler



Dynamic Scheduler in Harp

1. All computation models can use this scheduler.
2. All the inputs are submitted to one queue.
3. Threads dynamically fetch inputs from the queue.
4. The main thread can retrieve the outputs from the output queue

The `findMaxs` function shows how to use dynamic scheduler to run similar tasks in parallel.

```
public void findMaxs(){
    int numThreads = 3;
    /* initialize tasks. numThreads is the number of threads.
     Here the number of tasks we lanunched equals to numThreads
    */
    List<FindMaxTask> maxTasks = new LinkedList<>();
    for (int i = 0; i < numThreads; i++) {
        maxTasks.add(new FindMaxTask());
    }

    /*initialize the dynamic scheduler;
    The data type of input is CenPair, the data type of output is Object;
    The task is MaxTask*/
    DynamicScheduler<int[], Integer, FindMaxTask> maxCompute
    = new DynamicScheduler<>(maxTasks);
    /*Start the Dynamic Scheduler*/
    maxCompute.start();

    int[] list1 = new int[]{1,2,3,4,5};
    int[] list2 = new int[]{14,5,6,7,8,1};
    int[] list3 = new int[]{53,43,-1,43,63};
    /*Take inputs*/
    maxCompute.submit(list1);
    maxCompute.submit(list2);
    maxCompute.submit(list3);
    /*Get results*/
    while (maxCompute.hasOutput()) {
       Integer out = maxCompute.waitForOutput();
       System.out.println(out);
    }
}
```

Static Scheduler Tests

Since static scheduler already implemented, you don't need to change any code for this. Run the `k-means.sh` inside the `~/labsession/harp/contrib/test_scripts` for the following parameters and collect the results:

- Pts=100000, Centroids=10, Dimension=100, Iterations=100, Mapper=1, MemPerMapper=110000, Thd=1, 2, 4, 8, 16

```
E.g: #16 threads
runtest 100000 10 100 1 16 100 allreduce 110000

Output:
MSE : 28.43249353148915
Compute Time : 24016
Comm Time : 1
Total Threads: 16
```

p.s: Output should be in the `test_km/allreduce/evaluation`

Dynamic Thread Task for k-means

Static thread task is already implemented in the `harp/contrib/src/main/java/edu/iu/kmeans/common/calcCenTaskStatic.java`

Using it as an example, you need to complete Dynamic thread task. (`harp/contrib/src/main/java/edu/iu/kmeans/common/calcCenTaskStatic.java`)

- First, you need to add neccessary private data members.
- Second, you need to add constructor
- Third, you need to add codes to run

```
  implements Task<double[], Object> {

  protected static final Log LOG =
      LogFactory.getLog(calcCenTask.class);

  //TODO: add necessary private data members

  // constructor
  public calcCenTask(Table<DoubleArray> cenTable, int vectorSize)
  {
      //TODO: add constructor

  }

  @Override
  public Object run(double[] aPoint) throws Exception
  {
      //TODO: add codes

      return null;

  }
```

Computing the distance by using dynamic scheduler

In the `harp/contrib/src/main/java/edu/iu/kmeans/common/Utils.java` you need to add codes to the `computationMultiThdDynamic`. You can use `computationMultiThdStatic` method as an example. Besides, please take a look to findMax example to understand what is the difference between static and dynamic scheduler.

```
public static double computationMultiThdDynamic(
  Table<DoubleArray> cenTable,
  Table<DoubleArray> previousCenTable,
  ArrayList<DoubleArray> dataPoints, int threadNum, int vectorSize)
{//{{{

    double err = 0;
    //TODO

    return err;

}//}}}
```

Changing Static Scheduler to Dynamic Scheduler for the allreduce

You will see the following line inside `harp/contrib/src/main/java/edu/iu/kmeans/allreduce/KmeansMapper.java`

```
140 MSE = Utils.computationMultiThdStatic(cenTable, previousCenTable, dataPoints, thi
s.threadNum, this.vectorSize);
```

You need to change this to use Dynamic Scheduler. After changing this, you need to build the code using Maven and copy the jar files.

```
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/
```

Now, you may run the k-means.sh script again and collect the results for the Dynamic scheduler for the Thd=4, 8, 16

Please create a table for the results. Now submit the table and following files on the Canvas.

- `harp/contrib/src/main/java/edu/iu/kmeans/common/calcCenTask.java`
- `harp/contrib/src/main/java/edu/iu/kmeans/common/Utils.java`