Executing Batch Jobs on a High Performance Computing Cluster

Marek Grześ

January 20, 2020

1 High Performance Computing

In this exercise, you will use the Hydra cluster. Hydra is the School of Computing's High Performance Computing cluster. The CPU partition of the Hydra's cluster contains 17 servers each with 2x Intel Xeon E5520 CPUs running at 2.27GHz. Each server has 8 cores (16 threads), and between 12GB and 24GB of RAM. There are also GPUs available on Hydra, but we will not use them in this exercise because they are heavily used by researchers in the School.

Your cluster directory (`/cluster$HOME`) on `raptor` is shared between all the compute nodes; if your software and data are in that folder, the compute nodes can access them when your jobs are executed. The `raptor` server is available via SSH.

In this exercise, you will use the standard cluster software for submitting and managing batch jobs. This process will be briefly introduced during our seminar, but you are expected to read the documentation that is available here:

- `https://www.cs.kent.ac.uk/systems/hydra/`
- `https://slurm.schedmd.com/archive/slurm-17.11.2/quickstart.html`
- `https://slurm.schedmd.com/archive/slurm-17.11.2/sbatch.html` Don't use `srun` to do this assignment.

The default partition on the Hydra cluster is named `test`, and you can use it for testing your small jobs. You should use the `cpu` partition for real experiments. We can see a partition as a queue where the jobs are submitted.

2 Objectives

Imagine that you have program X, and you wish to run it on N different datasets, where N is relatively large and the time to run X on one dataset can be several hours or more. When X is executed on one dataset, the result is a numerical score that is printed by X to the standard output. The numerical score

indicates what was the performance of X on that particular dataset. You would like to compute an average score across all N datasets. If N was small, you could potentially execute X N times on your desktop computer using graphical user interface. The average values of your runs could be computed in Excel. However, when N is large, running such an experiment on a desktop machine becomes tedious if not impossible. For that reason, batch processing on a powerful cluster could be advantageous. In this assignment, you will learn how to run such an experiment on our Hydra cluster. In your experiment, X will be a machine learning programme called Weka, which will be executed on N datasets found in directory `uci-data/*.arff`. After doing this exercise, you will know how to run such experiments, and you will have a chance to see advantages of batch processing on computing clusters.

Methodologically, this experiment (in its current version) does not make much sense from the machine learning point of view. It will be sufficient, however, to demonstrate the benefits of batch processing.

3 SSH Access to Linux Serves

To do this assignment, you will need to use your UNIX password. If you've forgotten your Unix password, visit this page (`https://www.cs.kent.ac.uk/systems/newuser/`) to change it. Having your UNIX password, you can try to `ssh` to `raptor.kent.ac.ak`. If raptor lets you in, it means that your login name and password are correct. If you work in Windows, you can use `putty.exe` to make ssh connections. You don't need VPN to ssh to `raptor`.

4 Assignment—Executing Batch Jobs

In this assignment, you will simultaneously run multiple jobs on the Hydra cluster. Each job is a sequential program, i.e., it does not use concurrency; however, the fact that you will run dozens of such jobs will show the benefit of batch processing on a (multi-core/ multiprocessor) computing cluster.

4.1 Task Description

Your objective is to run Weka on N datasets located in directory `uci-data/*.arff`. This means that Weka will be executed N times using `sbatch` on the cluster. Every execution will produce a text file with Weka's standard output, which will contain the result of a particular run. Your goal is to extract numerical results from the individual output files and to put them in one text file. This file will contain two columns of data. The first column will be the name of the dataset, and the second column will be score of the algorithm on that dataset. The second goal is to write another bash script that will compute the average score and save it to a text file. Note that the average should be computed using the scripts written in `bash` or any other UNIX/LINUX shell that is available on `raptor`. This means that languages like Python or Java should not be used in this assignment.

All jobs should be submitted to the `cpu` partition and executed with a time limit of 45 minutes and 1GB of RAM. These restrictions can be specified as parameters to the `sbatch` command. You will need to read the manual to see how to do that.

4.2 Detailed Instructions

If not explicitly stated otherwise, all the commands that are shown below should be executed in your cluster directory on `raptor`; the directory is `/cluster$HOME`.

1. **Log in** to `raptor`
2. `cd /cluster$HOME`
   Note that `ls /cluster$HOME` will not show any files before you do `cd /cluster$HOME` because this is a network drive.
3. **Install weka**
   (a) You can download weka from my website using:
   `wget http://www.cs.kent.ac.uk/people/staff/mg483/documents/teaching/CO890/assignment1/weka-3-6-13.tar.bz2`
   (b) Then bunzip2 the bz2 file. This will create a new directory called weka-3-6-13. You can do that using:
   `tar xjf weka-3-6-13.tar.bz2`
   (c) If you'd like to check if Weka was installed correctly, you can type:
   `java -Xmx1000M -cp weka-3-6-13/weka.jar weka.classifiers.trees.J48 -h`
   This command will print help for classifier `weka.classifiers.trees.J48` which is one of the classification algorithms available in Weka.
   (d) At this point, Weka is installed in your cluster directory in `/cluster$HOME/weka-3-6-13`
4. **Download data**
   (a) Datasets can be obtained from my web-site using:
   `wget http://www.cs.kent.ac.uk/people/staff/mg483/documents/teaching/CO890/assignment1/uci-data.tar.bz2`
   (b) Then bunzip2 the bz2 file. This will create a new directory called uci-data. You can do that using:
   `tar xjf uci-data.tar.bz2`
   This directory should contain a number of files with the `*.arff` extension.
5. **Local test**
   Before you continue, you can run weka from your terminal to see if it works fine on a small dataset. You can do that using:
   `java -Xmx1000M -cp weka-3-6-13/weka.jar weka.classifiers.trees.RandomForest -t uci-data/iris.arff -i`
   This is a tiny example, and its output should be as follows:

```
Random forest of 100 trees, each constructed while considering 3 random features.
Out of bag error: 0.06



Time taken to build model: 0.21 seconds
Time taken to test model on training data: 0.04 seconds

=== Error on training data ===

Correctly Classified Instances         150               100      %
Incorrectly Classified Instances         0                 0      %
Kappa statistic                          1
Mean absolute error                      0.0142
Root mean squared error                  0.0598
Relative absolute error                  3.19   %
Root relative squared error             12.6925 %
Total Number of Instances              150


=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0          1         1          1          1        Iris-setosa
                 1         0          1         1          1          1        Iris-versicolor
                 1         0          1         1          1          1        Iris-virginica
Weighted Avg.    1         0          1         1          1          1


=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
```

```
  0 50  0 |  b = Iris-versicolor
  0  0 50 |  c = Iris-virginica



=== Stratified cross-validation ===

Correctly Classified Instances         143               95.3333 %
Incorrectly Classified Instances         7                4.6667 %
Kappa statistic                          0.93
Mean absolute error                      0.0393
Root mean squared error                  0.1556
Relative absolute error                  8.85   %
Root relative squared error             33.0021 %
Total Number of Instances              150


=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0          1         1         1          1        Iris-setosa
                 0.94      0.04       0.922     0.94      0.931      0.991    Iris-versicolor
                 0.92      0.03       0.939     0.92      0.929      0.991    Iris-virginica
Weighted Avg.    0.953     0.023      0.953     0.953     0.953      0.994


=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```

This example uses one of the Weka's algorithms that is called 'random forest' and is implemented in `weka.classifiers.trees.RandomForest`. For the purpose of this exercise, you do not need to worry about these details; however, you should know how to find the final numerical score of this execution. Note that in the middle of the above output, you can see this line:

```
=== Stratified cross-validation ===
```

Below that line, we have the results that we are interested in. Specifically, we are interested in the percentage of `Correctly Classified Instances`, i.e., we are interested in numerical values that are at the end of the line `Correctly Classified Instances 143 95.3333 %`. In our experiment, you will be extracting those lines from every (successful) execution of Weka. If Weka fails on a particular dataset, we will ignore such a result. Note that Weka may be terminated by the cluster software if a particular run exceeds your time or memory limit. So, we expect that some jobs will fail.

6. **Cluster Test**

Now, you can run the previous test using `sbatch`. For that, you will need my bash script `runweka-sbatch.sh`:

```
#!/bin/bash

#SBATCH --nodes=1 # number of cluster nodes, abbreviated by -N
#SBATCH -o slurm-%j-%N.out # name of the stdout, using the job number (%j) and the first node (%N)
#SBATCH -e slurm-%j-%N.err # name of the stderr, using job and first node values
#SBATCH --ntasks=1 # number of SLURM tasks, abbreviated by -n

if [ $# -ne 1 ]
then
  (>&2 echo "Usage: `basename $0` {1 arg expected}")
  (>&2 echo "       a path to your data file is required as a command line parameter")
  (>&2 echo "       error message generated by runweka-sbatch.sh")
  exit 1
fi

export DATASETNAME=`basename $1 .arff`

# the first line in the output is the name of the data file
echo "$DATASETNAME"

java -Xmx1000M -cp weka-3-6-13/weka.jar weka.classifiers.trees.RandomForest -t $1 -i -I 10000 -K 100
```

You can download this script from my web site using:
`wget http://www.cs.kent.ac.uk/people/staff/mg483/documents/teaching/CO890/assignment1/runweka-sbatch.sh`
This example bash script can be used to submit jobs to the cluster's queuing system using:
`sbatch runweka-sbatch.sh uci-data/iris.arff`
The script takes a path to your data file as a parameter. Note that by default the `test` partition is used. After you have submitted this job, i.e., after you have executed `sbatch`, you should be able to see this job in the queue using `squeue`. The output of `squeue` will be similar to:

```
mg483@raptor [/cluster/home/cur/mg483/CO890/a1-execution] $ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             42048       cpu ex_30.sh   nobody  R   21:09:44      1 cloud02
             42060       cpu   run.sh   nobody  R   21:07:28      1 legacy14
             42061       gpu run_clt.   nobody  R   21:06:56      1 pascal02
             42062       gpu resnet_a   nobody  R   21:00:52      1 pascal01
             42168      test runweka-    mg483  R       0:01      1 cloud01
```

The last job in this example is the job that we have just started. After your job has been finished, you will see two files in your cluster directory: `slurm-*-*.out` and `slurm-*-*.err`. The first one contains the standard output whereas the second one the standard error of the process (Weka and the bash script in this case) that was submitted as your job. The content of the file that contains standard output should be the same as what you had on your screen when you executed the local test above, except for the first line that contains the name of the data file printed by the bash script.

Before you proceed to the next step, you should take care to notice that the `slurm-*-*.out` file produced above contains the name of the dataset in the first line, and that after the following line `=== Stratified cross-validation ===` there is a line with the result that we are interested in, i.e., `Correctly Classified Instances 143 95.3333 %`.
Note that the expression `Correctly Classified Instances` appears twice in the output. You will need to address this fact explicitly in your bash scripts below.

7. **Main Task (1)—Submitting Jobs**
   Your task is to use the `runweka-sbatch.sh` file in conjunction with **sbatch** to submit one job for every data file that can be found in the `uci-data` directory. This process should be automated, and you should implement it in bash. Your bash script should submit all the jobs in one execution (one job corresponds to executing Weka on one dataset). Note that you will need to make sure that your jobs satisfy the constraints specified in Sec. 4.1.

8. **Main Task (2)—Extracting Individual Results**
   After all the jobs started by your script that you will write for the previous step have been computed (i.e. when you don't see any of your jobs waiting in the queue), you will need to parse the output files and compute the results. The output files are `slurm*.out` and `slurm*.err` in the directory in which you submitted the jobs to the queue. You are interested in the standard output, i.e., in the `slurm*.out` files.

Your goal is to write a bash script that will be executed, for example, as follows: `bash resultsweka.sh slurm*.out`
where the argument to the script specifies the output files that should be processed by the script. Note that this corresponds to the names of files that contain standard output of your jobs. Your script has to process all files `slurm*.out`, and for every `*.out` file, it has to print one line that will contain two values:

(a) The name of the dataset that is in the first line of the `slurm*.out` file

(b) The last numerical number in the line that contains
`Correctly Classified Instances`, but for the instance of that line that appears after the following line
`=== Stratified cross-validation ===`

Specifically, for the example output for the iris dataset computed in our test above, the line should be:
`iris 95.3333`.
Your script should produce a result like this line for every file that matches this pattern `slurm*.out`.

Let's assume that when this script is executed before all the datasets have been processed (i.e., when some jobs are still being executed on the cluster), it will print the lines for those datasets for which the experiment has been completed. The lines for the datasets for which the results are not available should not be printed.

9. **Main Task (3)—Average Performance**

Having results from the previous step, your task is to write a tiny bash script that will compute the average result, and it will save the average value to a separate file. Specifically, your script will read the file produced in the previous step, and it will compute the average value for the second column that contains numerical values. The result of this task (i.e. one number that represents the average) should be saved in a text file.

Note that the file with individual results may contain empty lines, for example, the following file contains three lines where the second line is empty.

```
iris 95.3333
               <- empty line
diabetes 68.5
```

Or here the third line can be empty:

```
iris 95.3333
diabetes 68.5
               <- empty line
```

In these cases, the average should be `81.91` and not `54.61`, which would be returned when the empty line was counted incorrectly. Your script should return a correct value for such input.

10. **Main Task (4)—Second Run**
    In the final task, you won't need to do any coding. The goal of this task is to run the entire experiment again with different settings of the Weka's machine learning algorithm. This is basically for you to appreciate how easy it is to run such experiments after you have set up your environment, i.e., after you have written required scripts.

    For the second run, you will just need to change Weka's parameters in `runweka-sbatch.sh`. For that, you need to replace the last line of that script with the following line:
    `java -Xmx1000M -cp weka-3-6-13/weka.jar weka.classifiers.trees.J48 -t $1 -i`
    and then you simply run the entire experiment again, i.e., you use your updated script on all datasets, extract the results, and compute the average value.

    Hopefully, at this point, you will see the benefit of the entire exercise where you can run the experiment with new parameters again, and the process of running the experiment as well as processing the results are fully automated. Imagine that you need to repeat this experiment 100 times (each time with different parameters), on all N datasets, and report the average performance on each of those 100 parameter settings. Would the software we created in this exercise make your life easier?

4.3 Deliverables

The following files (as one archive gz, zip, bz2, rar or similar) should be uploaded to Turnitin:

1. All files that contain any bash code that you had to write to do this exercise.
2. All output files generated by all your jobs (both the standard output as well as the standard error).
3. A text file that contains the names of datasets and the numeric values extracted from individual jobs (two columns: name of dataset and a numerical value).
4. A text file that contains the average value across all datasets.
5. A text file (max 150 words) with your critical reflection on this task (your conclusion, your challenges, what you've learned etc.)

Note that items 2–4 in the list above should be sent for the second run as well, i.e., you will have two sets of files described in items 2–4.

4.4 Additional Information

1. If you don't know how to copy files between Linux servers and your desktop computer, you can try `winscp` or `pscp.exe` on Windows, `scp` on Mac and Linux, or `konqueror` on Linux.

2. Note that jobs of all students from your seminar group are likely to be submitted to the same queue. Therefore, submit your jobs early because they may not finish by the deadline if all the students submit on the last day.

3. The use of abstractions (e.g. functions and variables whenever appropriate) and succinctness of the bash code will be taken into account during marking. Try to make your code clear and use system tools such as `awk`, `sed`, `grep` etc. The quality of comments that you will put in your bash files will influence your mark as well. For example, all complex, non-standard steps in the bash code should be explained in comments.

4. Useful Slurm commands: `sinfo`, `squeue`, `scontrol show partition`.