The Amarel Cluster: An Extended Example

Yonah Biers-Ariel

May 20, 2019

1 Introduction

The Amarel cluster is a powerful tool, particularly when you need to run the same code on many different inputs[1]. In this example, we will use it to answer the following question: of the 24 permutation classes consisting of permutations avoiding a single length 4 pattern, how many have finite schemes which can be found using Doron Zeilberger's package `VATTER`?

The purpose of this tutorial is to walk the reader through a non-trivial example using Amarel. The purpose of this tutorial is **not** to describe how Amarel works or how any of the tools which we will use work. The best way to learn about those topics is to attend an Introduction to the Amarel Cluster Workshop (see the schedule). It also may be useful to read the guides and examples written by the Office of Advanced Research Computing (the people who run Amarel) which are available here.

This tutorial assumes no knowledge of command line operations.

Disclaimer: Neither I nor this tutorial is affiliated with or endorsed by the Office of Advanced Research Computing. This tutorial is not a substitute for attending the OARC's official workshops which I highly recommend doing.

2 Getting An Account & Logging In

Before using Amarel, you need to sign up for an account. You can do so by filling out the form here. It takes a day or two for them to create an account for you. The form asks for a bunch of information that is only relevant if you're buying into the system (to get priority access). Don't worry about those parts.

Once you have an account you can login via ssh. If you aren't familiar with ssh, you first need to pull up a command prompt (if you're on running MacOS or Linux open the Terminal application - if you're running Windows then google how to get a terminal window). At the command prompt, type `ssh`

[1] Amarel is also useful if you are running code only once if your code has built-in parallelization. Since I have not tried doing that, this example does not cover it

`netid`[2]`@shell.math.rutgers.edu` You will then be prompted to enter your netid password, so enter it. Next, type `ssh amarel.rutgers.edu` and again enter your password when prompted. You're now on the cluster!

Notice that we first logged into the math server, and, from there logged into Amarel. This method of logging in will work as long as you have any internet connection. If you are on a Rutgers wireless network, you can take a shortcut and login directly to Amarel with `ssh netid@amarel.rutgers.edu`. For maximum generality, this tutorial assumes that you are not on a Rutgers network (everything will still work if you are on a Rutgers network, you might just miss out on some shortcuts).

On Amarel, you have two personal directories. Your home directory is for your code and outputs that you want to save indefinitely. Your scratch directory is for output that you don't need permanently. While you're here, you can make folders to store the output that you're going to generate. Navigate to the scratch directory by typing `cd /scratch/netid`. Then make a new folder to hold the output of our test by typing `mkdir VATTER_eg`.

If you want to logout of Amarel, you can do so by typing `exit` at the command prompt.

3 Writing Your Maple Code

To get Maple code onto the cluster, you have two options: write the file on your computer like you would any other code and then transfer it to the cluster or write the code directly on the cluster. Later, we'll see how to transfer files from your computer to the cluster, so here we'll cover creating a new file while on the cluster. If you're still reading this, I'll assume that you're not familiar with command-line text editors; if you are you can skip the next paragraph.

There are a number of text editors which can be accessed from the command line on Amarel (or just about any other computer). The one I use is Vim. To create a new document called, say, `sample.txt` type `vim sample.txt` at the command line. You now enter this document in what's called normal mode (Vim has several different modes, right now we will only discuss two of them). To write text in the file, you need to enter insert mode by pressing `i`. Now that you're in insert mode, you can type whatever you want into your document. When you're done go back to normal mode by pressing the `Esc` key, save your work by typing `:w` and pressing the return key, and quit by typing `:q` and pressing the return key. If you're interested, Vim has many useful tricks and shortcuts which are well worth learning if you use it a lot.

You should now be back at the command line. To see that your file was successfully created, type `ls` to list all the files in your home directory. There should be one called `sample.txt` now.

The actual Maple code that you should write now is pretty similar to whatever Maple code you would run on your own computer[3]. The biggest change is

[2] here, and anywhere else you see netid, type your netid

[3] Unless you're trying to use Maple's built-in parallelization in which case I can't help you

that you need to worry about where you're going to print your output to since you can't just print it all to your screen. The following script (which we'll just call `vscript.txt`) will run the `VATTER` procedure SchemeFast on the permutation $[1, 2, 3, 4]$ and print the success or failure to an output file. Note that ‘ is a tick mark while ’ is a single quotation mark.

```
read ‘VATTER.txt‘:
s:=SchemeFast({[1,2,3,4]},6,2):
f:=fopen(cat("/scratch/netid/VATTER_eg/", 1, ".txt"), ’WRITE’, ’TEXT’):
if s[1] = FAIL then
        fprintf(f, "Failed. Could not find a scheme for {[1,2,3,4]}"):
else
        fprintf(f, "Succeeded! The scheme for {[1,2,3,4]} is: %a", s):
fi:
fclose(f):
```

Before we can run this on the cluster, though, we need to transfer the file `VATTER.txt`. Logout of Amarel by typing `exit`, and then logout of the math server by typing `exit` again. Download `VATTER.txt` from Zeilberger's website, and save it to your computer.

Back in your terminal window, navigate to the directory where you saved `VATTER`. Mine is saved in a subdirectory of my home directory called Research, so I type `cd ~/Research`. Once you're in the right directory, type `sftp netid@shell.math.rutgers.edu` and enter your password when prompted (this is the other place where you can take a shortcut if you're on a Rutgers network). Then, type `put VATTER.txt` to transfer the file to your home directory on the math server. Log out of the math server, and ssh back in. Then sftp into Amarel (`sftp amarel.rutgers.edu`) and put the file into your home directory in Amarel (`put VATTER.txt`). Exit Amarel, and then ssh back into Amarel. Type `ls` to list the contents of your home directory. You should see two files: `vscript.txt` which you made and `VATTER.txt` which you transferred from your computer (if you made `sample.txt` you should see that too).

4 Running Your Maple Code

Now we try to run this on the cluster. Initially, Amarel does not know the command `maple`, so we have to teach it by typing `module load maple/2018`. This is essentially the same as typing `with(somepackage):` in Maple, it just gives you access to new commands.

Now we come to the most important rule (really the only important rule) of the cluster. DO NOT RUN MAPLE CODE ON THE LOGIN NODE. This is basically the only way to accidentally do something bad. So, while we'd like to just type `maple vscript.txt`, we can't (or, rather, we can, but we'd get an email from OARC telling us to stop). Instead, we use SLURM, which is the cluster's workload manager. The purpose of SLURM is to make sure that everyone gets the computing resources that they need, and, more importantly,

that the number of people using any compute node at once is no more than what that node can handle. The first SLURM command we'll use is `srun`. To use it, type `srun maple vscript.txt` (and press enter). This tells Amarel to allocate resources to us, and then to run maple using those resources.

To check to see if our code worked, we navigate to the directory where we printed output (`cd /scratch/netid/VATTER_eg`) and open the file `1.txt` (`vim 1.txt`). If everything worked correctly you should see `Succeeded!` followed by the scheme.

So far, we haven't done anything that we couldn't have done on a personal computer or on the compute server. The benefit of the cluster is that it allows us to run many jobs at once, so if we're not doing that, there's not much point in using it. What we want to do is to run `vscript` on every length-4 pattern at once, not just [1,2,3,4]. To that end, we modify `vscript.txt` so that it will input an integer p from 1 to 24 and use that p to choose which pattern to avoid. Our new `vscript.txt` looks as follows:

```
read `VATTER.txt`:

with(combinat):
perms:=permute(4):
pat:=perms[p]:
s:=SchemeFast({pat},6,2):
f:=fopen(cat("/scratch/netid/VATTER_eg/", p, ".txt"), 'WRITE', 'TEXT'):

if s[1] = FAIL then
        fprintf(f, "Failed. Could not find a scheme for {%a}" pat):
else
        fprintf(f, "Succeeded! The scheme is for {%a} is: %a", pat, s):
fi:
fclose(f):
```

We're ready to call our code again, but now it's not enough to just run `vscript.txt`; we also need to give a value for p on the command line. Our new command will be `srun --time=2:00:00 maple -qc p:=2: vscript.txt`. Here, we've added two new options to our Maple command and one new option to srun. The `q` option makes Maple quiet. When we ran Maple last time, it printed all the code we were running to our screen along with periodic updates on how much memory it had used and how long it had run for. I think that's annoying (especially if we are going to run many instanced of Maple at once) and so I use `-q` to suppress it. The `c` option tells Maple to execute whatever line of code immediately follows it when Maple first starts up. So, when Maple first starts, before it reads `vscript.txt`, it sets p equal to 2. The `time` option to srun tells it to let our program keep running for up to two hours (by default our program will be terminated after running for two minutes). Try running this command, and then look back in `/scratch/netid/VATTER_eg`. Now you should see a file `2.txt` which is much the same as `1.txt`.

At this point we could just repeat this same command for all the numbers from 3 to 24 (i.e. call `srun --time=2:00:00 maple -qc p:=3: vscript.txt`, `srun --time=2:00:00 maple -qc p:=4: vscript.txt`, and so on). If we wanted to test more than 24 different patterns, though, that might not be very much fun, so we'll write a loop to run the command for us 24 times.

The loop will be written in Bash, which is the language we've been using every time we've run code at the command line (`cd`, `ls`, `vim`, etc. are all Bash commands). To write the Bash script, use Vim to open a file called `run.sh`. In the file, write:

```
for i in {1..24}
do
        srun --time=2:00:00 maple -qc p:=$i: vscript.txt &
done
```

Now we only have to run this script once to start 24 instances of Maple, each of which runs `vscript` on a different pattern. To run it, at the command line type `./run.sh`. You should get the message "Permission denied", which tells you that you're not allowed to execute the script. To let yourself execute the script type `chmod 744 run.sh`. This changes the permissions on the file `./run.sh` so that everyone can read it and you can also edit and execute it. Try running it again.

Now you should see 24 jobs being allocated resources. It takes about 90 seconds for them all to run, so while they're running we will learn two new SLURM commands: `squeue` and `scancel`. Enter `squeue -u netid` at the command line. You will see a list of all your active jobs that are running on Amarel. In the leftmost column are all of their jobids. Choose one (call it id) and enter `scancel id`. Whatever job that was, you've cancelled it, and so it will now stop running. If you ever need to cancel all your jobs, you can enter `scancel -u netid`.

You'll know that your jobs have all finished when you enter `squeue -u netid` and don't see any jobs listed. Navigate to `/scratch/netid/VATTER_eg` and list its contents. You should see 24 files. You can open them one by one (using Vim) to see which patterns have schemes, but what if you had hundreds of files? Since the files representing successes all have the word "Succeeded" in them, we can search for that word. To do so enter `grep Succeeded ./*`. You should see four lines telling you that $\{[1,2,3,4]\}, \{[4,3,1,2]\}, \{[4,3,2,1]\}$, and $\{[1,2,4,3]\}$ all have schemes.

5 Conclusion

We now know which patterns have schemes (or, rather, which have schemes within the limited space we searched). This tutorial only scratches the surface of all the topics that it mentions, but hopefully you now know enough keywords that you can learn as much as you care to from Google.