PIV Remote Computing Guide

Travis Burrows

February 21, 2020

Contents

1 Abstract

This is a guide to the process of remotely computing PIV on the Georgia Tech PACE cluster, which includes generating the DaVis processing script on your local machine, installing DaVis on your PACE account, transferring data to the PACE server, submitting a job for PIV processing, and finally downloading the results.

2 Log into PACE via SSH

SSH, or Secure Shell, is the method used to log into your PACE account via command line. This will be required to submit PIV jobs to the PACE computing nodes.

2.1 Software

If you are using Windows, you will have to install software to be able to SSH into PACE. KiTTY and PuTTY are both good options. Download KiTTY from `https://www.fosshub.com/KiTTY.html`. Both portable and installer versions are available.

2.2 Connection Settings

Once you have launched one of these programs, you will need to enter in a url. The proper url for SSH is `login-s.pace.gatech.edu`. Click to save to save it for the future, and then Start or Open to initiate the connection. You will be prompted for your username and password. In addition, if you get asked about an unknown host key, press yes to accept the key.
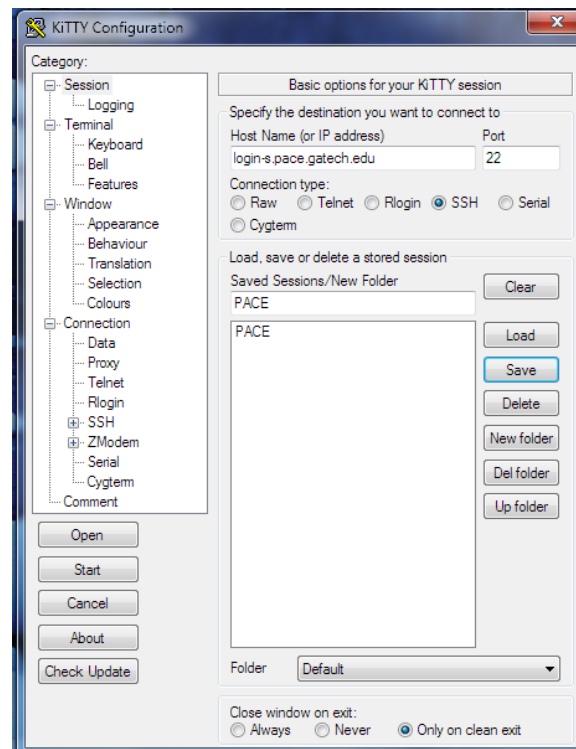
Figure 1: KiTTY Interface

2.3 SSH Commands

Congratulations! You've SSH'd into PACE! Now the possibilities are limitless- as long as you know the commands (you are in a linux system). To change directories, use the command `cd`. To list the files in a folder, use `ls`. To exit, you can simply type `exit`, or exit out of the window like any other.

3 Transferring files to and from PACE

This section will cover how to transfer files to and from PACE. There are a couple of options available, which are using SFTP (SSH File Transfer Protocol) or using the Globus software. Once it is properly set up, Globus will be the faster and more

direct way to get files to PACE. Since it is not set up yet, the procedure is not yet documented.

3.1 SFTP

3.1.1 Software

If you don't already have a favorite file transfer program, I would recommend FileZilla. Download and install the Filezilla Client software from `https://filezilla-project.org`. This works like any typical FTP file transfer program. If you are using Windows and want to mount PACE as if it is another drive on your computer, see Section 3.1.4.

3.1.2 Connection Settings

PACE has special servers for data transfer, which are different from the servers used for SSH. For file transfer, use `iw-dm-4. pace.gatech.edu`. Your GT username and password will work to authenticate and get you connected. If you have not yet been given access to PACE, you will find out when you fail to connect. In FileZilla, go to File → Site Manager → New Site to create this new connection. Select Protocal as SFTP, enter the above url into Host, and enter your GT username and password into the appropriate locations, if you would like auto-login. Save the site entry and click connect.
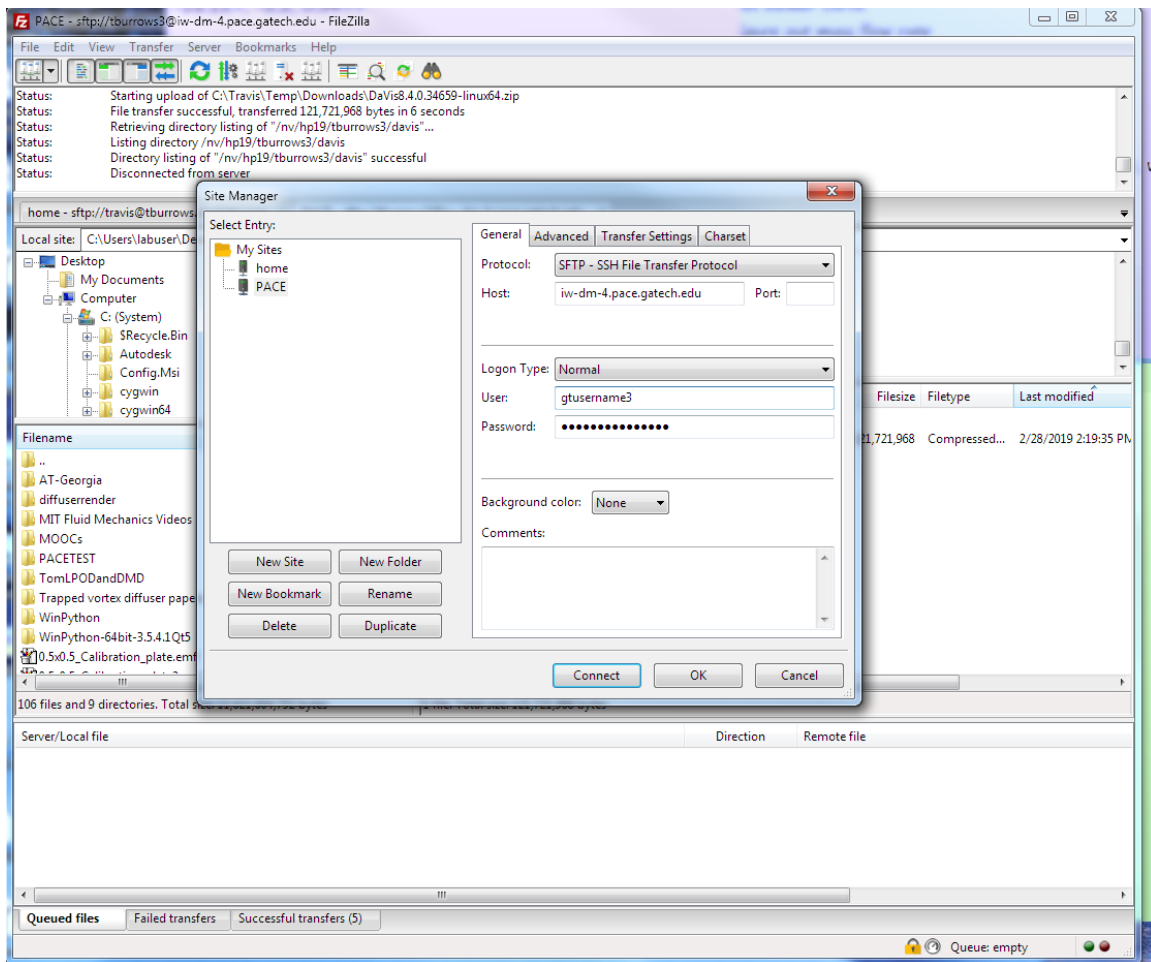
Figure 2: FileZilla Interface

3.1.3 Transferring Files

After connecting, you should see a list of folders populate on the right half of the window. This is the PACE folder structure, while on the left is the local machine. To upload files or folders from the local machine to PACE, you can drag-and-drop, or right-click and select upload. This will upload into currently selected folder on the remote computer. You can download files from the remote to local computer by selecting download or dragging files in the opposite direction.

3.1.4 WebDrive

WebDrive is software that can mount an SFTP connection as a Windows drive letter. This will not take advantage of a 10G connection, but does make accessing the files convenient. Get the software from `http://software.oit.gatech.edu`. Log in, and find WebDrive under FTP/VPN software. Install it using the provided license and enter the same SFTP connection details as described in the Section 3.1.1. Note that this is a connection to only your personal PACE folder and could cause confusion if it is a shared machine.

3.2 Globus

Globus is the preferred method to transfer files to and from PACE. Before your local machine can be used for transfers, Globus software must be installed, so your computer is registered as an Endpoint.

3.2.1 Installing Globus on a Local Machine

1. Check if your compueter has Globus Connect Personal installed in your programs. If so, make sure it is currently running, and note what the Endpoint Name is, by hovering over the icon in the taskbar.

2. Navigate to `https://www.globus.org` and log in with the Glezer shared account

3. Click Endpoints → Administered by You. If your PC is shown here and the status is Ready, then you are ready to transfer files. If your computer does not have Globus Connect Personal installed, or it is installed but does not show up here, you will have to uninstall and install again.

4. To install Globus Connect Personal, click Endpoints, and Create a personal Endpoint, which is in the top right corner of the webpage. Choose a name for the PC that the lab will understand, click Generate Setup Key, and dowload the installer.

5. Run the installer on your local PC, and enter the setup key when asked.

6. When you are asked about editing shared locations on your PC, go in and edit them. Add all locations on the PC that you might want to transfer files to and from. Only these locations will be available for transfer.

3.2.2 Transferring files with Globus

Once your local PC has Globus Connect Personal installed, follow these steps to make transfers.

1. Select your local PC for a transfer by clicking File Manager → Collection Search → Your Collections → Your PC

2. Click Transfer or Sync to... → Search → PACE Internal

3. Log in to PACE Internal with your regular GT credentials (not Glezer shared)

4. On the left side (local PC), navigate to the folder or files you would like to send or receive

5. On the right side (PACE Internal), navigate to the folders or files you would like to send or receive

6. Select all files/folders you would like to transfer

7. If you want to only transfer new files, or update only files that have been changed, you can find advanced settings under Transfer & Sync Options.

8. If you would like to transfer Right → Left, click the start button on the left that points right. If you want to transfer Right ← Left, click the start button on the right side that points left.

9. Find the status of your transfers in the pane called Activity. You can also cancel transfers here.

3.3 PACE Storage Folders

When you connect to PACE you will by default be in your PACE home directory. The path of this folder is something like `/nv/hp19/yourusername`. This folder, the home folder, has a limit of 5 GB of storage, and is backed up daily. Therefore, it should not be used for PIV processing. The subfolder called `scratch` is designed to be a temporary data storage location, and has a limit of 7 TB. Files older than 60 days are automatically deleted, and are not backed up. The folder called `data` has a limit of 2 million files/folders, is daily backed up, and can be used for longer-term storage. Therefore, the safest place to conduct PIV processing is in the `data` folder. Note that `data` and `scratch` are actually shortcuts to other drives. They can be reached by the path `~/scratch`, or the actual path, which can be found in FileZilla upon navigating into one of these folder.

4 Installing DaVis on PACE

1. Obtain the file `DaVis8-linux64.zip` from `P:/Shared/PACE PIV Tutorial` or the latest from the LaVision website.

2. Using a file transfer program, make a folder in your home directory called `davis`. Then, transfer the zip file into that new folder.

3. SSH into your PACE account. By default you should be in your home directory. Change directories into the folder that contains the davis zip file. (`cd davis`).

4. Type the command `unzip DaVis8-linux64.zip` to unzip the contents into the current folder.

5. The linux DaVis will only work with a slight modification - follow these steps to rename one of the folders in the DaVis installation. Rename the `linux64` folder to `linux64.old` (`mv linux64 linux64.old`). Now, rename the folder `linux64-centos69` to `linux64` (`mv linux64-centos69 linux64`).

6. Now, setup DaVis in cluster mode by running `./davis-setup.sh cluster` . This initializes the installation into cluster mode. You should receive this message:
   `Starting DaVis to setup cluster configuration...`
   `Done.  The script generated by the DaVis master can now be processed.`

7. Another test to make sure that DaVis is correctly configured is to run `./davis-start.sh`. This should give a fairly long output, listing the node's specifications, as well as DaVis available packages. It should be clear that the program shut down on its own and didn't throw any errors.

8. Finally, set your davis installation to be read only. This helps prevent processors from changing it while others are using it. Do this by running the following command: `chmod -R a-w ∼/davis` . Your DaVis install is now ready for processing! Proceed to creating a processing script from your local DaVis.

5 DaVis Processing Script

This step is the process of generating a script on your local machine, which tells the remote DaVis installation where the data is that will be processed, and what processing settings will be used. This script is required for PACE remote PIV processing.

5.1 Requirements

The feature in DaVis that we are using is called Cluster Script Processing. This feature must be enabled in your DaVis license, which you can check by going to DaVis → Help → About. If it is not enabled, note your dongle number from the About screen, and contact Callum Gray at LaVision for a replacement license file that contains this feature.

5.2 Script Generation

1. The button for script generation is found in the Processing dialog. Pick a data set (image set) that you want to process, and open the Processing dialog for it.

2. Before clicking the button, choose what operations you would like the remote computers to perform. It is recommended to first run Test Processing on the settings you want to use, to make sure that the results look good. A caveat is that only certain operations are supported by Cluster Script Processing. This is because of how the operations are implemented by LaVision - only certain ones can be parallelized on multiple machines. PIV and vector postprocessing are supported, but many are not, like time subtraction and vector averaging. These do not take much processing power anyway. To use PACE, you will have to process on your own machine the other operations manually, which means doing any time-subtraction or others beforehand, and averaging after the instantaneous vector fields are downloaded from PACE.[1]

3. Once your processing operations are chosen, click "Create Processing Script". This will warn you if you have operations selected that are not distributable. A new window pops up, with a lot of options. The top half of the dialog is auto-filled, and cannot be modified. The bottom half needs to be changed. "Script Folder" is the folder on the local machine that you want to save this script. Pick or make a folder dedicated for these scripts. "Linux local script path" is the path on the PACE machine that you will place this script. "Linux local program path" is the path on the PACE machine where you have installed the linux version of DaVis. "Linux local project path" is the path on the PACE machine where you

---

[1]Another option is to assign different nodes whole data sets, in which case the image pairs are not distributed to different machines. This function has not been explored or documented as of yet.

have copied your project folder from the the local machine where the data was recorded. "Generate script" is the name of the script itself. "File list" will be filled by DaVis, and "Distribution system" should say OpenMPI.

If you have not yet installed DaVis on your PACE account, or chosen a PACE folder for your PIV processing, come back here after you have done so so that you know what paths need to be entered.
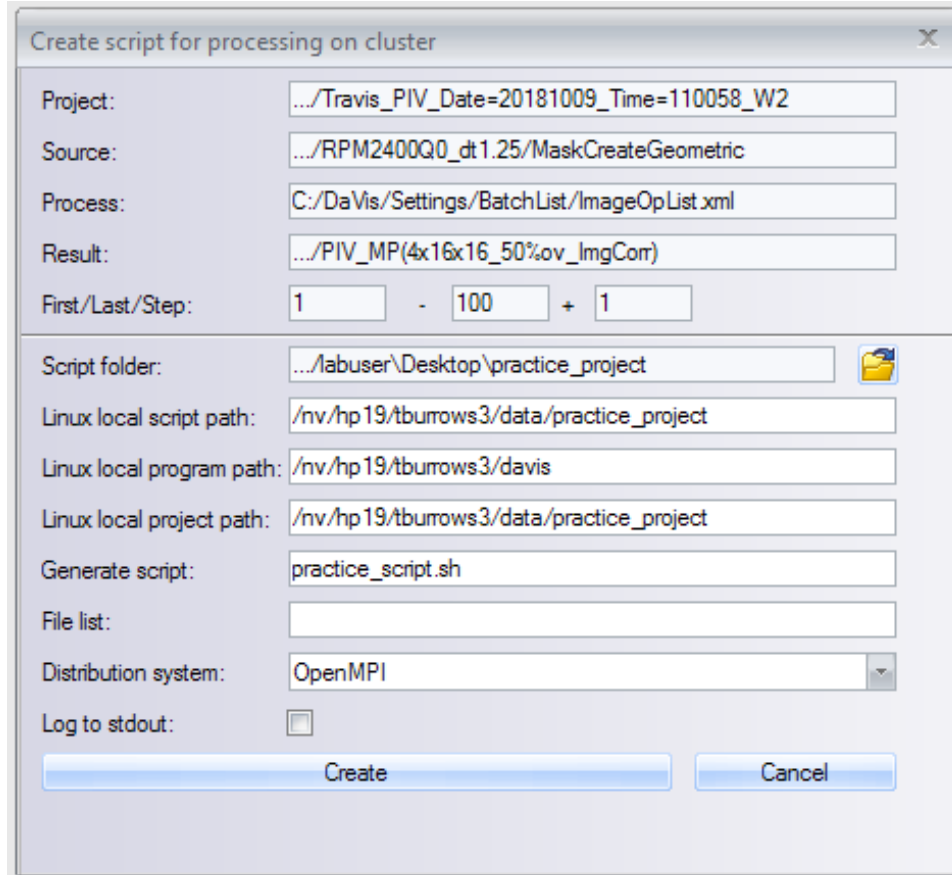

Figure 3: Create Processing Script popup dialog

6 Submitting the job to the PACE Queue

6.1 Prerequisite: Transfer PIV Files to PACE

Before submitting your PIV job, all the files need to be in the correct places on PACE, and these paths need to match those in the DaVis cluster processing script. The project path specified in the script must contain the same files that your local project folder has, including the xml file that corresponds to the project folder. It does not need to contain other data sets that you are not currently processing. See the example zip file and the practice exercise in Section 7 for an example of where the files should be placed.

6.2 Using PACE computing cluster

This section explains how to get PACE to run the DaVis script that has been generated. It is assumed that at this point, the script and PIV data files have been placed on PACE, at the correct location, and that you are SSH'd into PACE. There are two ways to compute on PACE - interactive mode, and batch mode. Batch mode is more convenient for bulk processing PIV.

6.2.1 Interactive mode

In interactive mode, the user requests a certain amount of computing resources, for a certain amount of time. This request is placed via a command. Once this command is ran, it submits the request and waits until it is available. When it is available, access to the computing node is returned, and you will find yourself logged into it, automatically. Once you are here, you can run whatever you would like, until the time is over, when you will be kicked off. The following is the command:

```
qsub -I -q prometheus -l nodes=1:ppn=8,walltime=10:00:00,pmem=4gb
```

This requests for computing resources on the cluster called Prometheus, for one node that has 8 cores per node, for 10 hours, and 4 GB per core (32 GB total) . The choice of cluster and resource specifications will be discussed in Section 6.2.3. This way of interacting with PACE can be useful for debugging or other scenarios, but not really for PIV processing. Once you have figured out how to properly process, using batch mode is more convenient.

6.2.2 Batch mode

Instead of returning the computing cluster access directly to the user, this method takes an input text file that explains what the cluster should do, and submits this in the queue as a job. Once it reaches the front of the queue, the job runs in the background, and can email the user when the job is complete. This text file is called a PBS script, and an example of how it is structured is shown below. A pound sign # followed by a space is considered a comment, while #PBS is considered a command for the queuing system.

```
# PBS Script to submit Davis PIV job

#PBS -N DavisPIV              # Name of job
#PBS -l nodes=2:ppn=2         # Number of nodes and processors per node
#PBS -l pmem=4gb              # Memory per processor
#PBS -l walltime=5:00:00      # Max time, hh:mm:ss
#PBS -q prometforce-6         # Cluster where the job is to be submitted
#PBS -j oe                    # Specifies output and error files to be saved
#PBS -o joboutput.txt         # Output file location
#PBS -m abe                   # Send emails when job is aborted (a), begins (b), or ends (e)
#PBS -M gtusername3@gatech.edu  # Specifies email address to send notifications

# *** Start of instructions to be executed by each processor ***

# Loads modules required for parallelism across nodes
module load java/1.8.0_25
module load openmpi/2.1.1

# Runs DaVis script in parallel over 4 processors.  This must be changed for different nodes or ppn.
# The time command returns the computation timeto the output file specified above.
time mpirun -np 4 /nv/hp19/tburrows3/data/practice_data_set/practice_script.sh
```

The first section of the text file describes to the job scheduler what resources to use, and settings for output and email notification. The second section describes what the cluster should do, which in this case is to load the required modules for MPI parallelism, and then run in parallel the DaVis processing script. The command to submit the job to the queue is simple: `qsub pivjob.txt` .

6.2.3 Choosing job resource parameters

1. Cluster - As a student in ME, you have access to the ME cluster "prometheus". In addition, there are shared queues which contain other clusters as well. An illustration of this concept is shown below. In prometheus, you have the highest priority compared to other users, while in other shared queues that contain prometheus and other clusters, your queue priority goes down. However, in the shared queues, there are more computers available. To check what queues you have access to, you can run `pace-whoami` when logged into PACE. It will list the queues, corresponding priorities, and maximum wall times you are allowed to use on those queues. Shared queues you have access to probably include prometforce-6 and iw-shared-6,. In addition, you can check how many nodes each of these queues has by running `pace-check-queue queuename`.

2. Number of nodes - The more nodes you choose, the faster DaVis will process the PIV, but it may take longer for your requested job to start running, depending on how many nodes are currently in use. In addition, it is not clear exactly how much faster the PIV will process with more nodes. Conduct a study! Note that communication between cores on a single node is much faster than between nodes, so increasing the number of cores per node would have a greater effect on performance.

```
[tburrows3@login-s1 practice_data_set]$ pace-check-queue prometheus

        ** NEW FEATURE : add '-s' to pace-check-queue to list
        ** scheduler features for each node

=== prometheus Queue Summary: ====
        Last Update                      : 03/06/2019 16:00:02
```
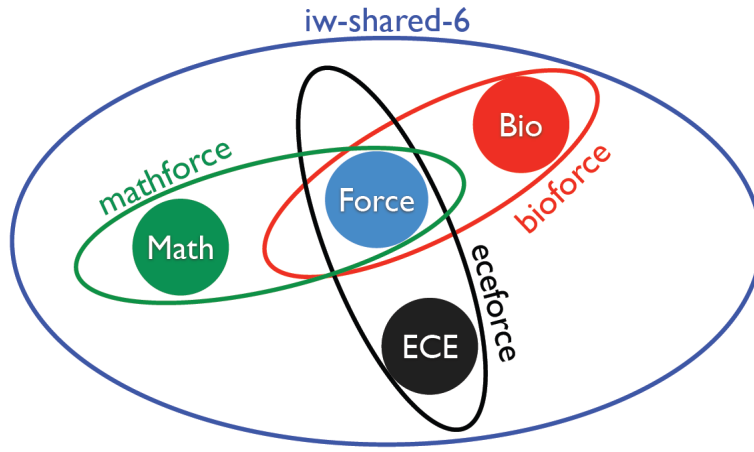
Figure 4: Concept of shared queues

```
         Number of Nodes (Accepting Jobs/Total) : 6/14 (42.86%)
         Number of Cores (Used/Total)           : 269/364 (73.90%)
         Amount of Memory (Used/Total) (MB)     : 100564/1770321 ( 5.68%)
=========================================================================
  Hostname       tasks/np Cpu%  loadav%  used/totmem(MB)  Mem%  Accepting Jobs?
=========================================================================
iw-p33-31-l         0/24   0.0     1.4      2743/66491     4.1   No  (node down or offline)
iw-p33-31-r         0/24   0.0     2.5      2496/66491     3.8   No  (node down or offline)
iw-p33-32-l         0/24   0.0     1.5      2713/66358     4.1   No  (node down or offline)
rich133-c32-18-l   24/28  85.7    85.8      5896/260406    2.3   Yes (free)
rich133-c38-21-l   24/28  85.7    86.0      5156/130989    3.9   Yes (free)
rich133-c38-21-r   28/28 100.0   102.4     16291/130989   12.4   No  (all cores in use)
rich133-c38-22-l   25/28  89.3    89.3     19656/131126   15.0   Yes (free)
rich133-c38-22-r   24/28  85.7    85.8      5166/130989    3.9   Yes (free)
rich133-c38-23-l   24/28  85.7    85.9      5092/130989    3.9   Yes (free)
rich133-c38-23-r   24/28  85.7    86.0      5236/130989    4.0   Yes (free)
rich133-g24-26-l   24/24 100.0   101.6      9672/131126    7.4   No  (all cores in use)
rich133-g24-26-r   24/24 100.0   100.2      5154/131126    3.9   No  (all cores in use)
rich133-g24-27-l   24/24 100.0   100.1      5221/131126    4.0   No  (all cores in use)
rich133-g24-27-r   24/24 100.0   101.6     10072/131126    7.7   No  (all cores in use)
```

3. Processors per node - This requests a certain number of cores on each node to be used. However, different computing queues only have certain types of computers available, which only have so many cores per node. You can browse the computers on a queue by running `pace-check-queue queuename`, as I have on prometheus, shown above. This shows that prometheus only has 24 and 28 core nodes (in the tasks/np column). Therefore, if you request 24 or less nodes, any single node could run the job, when it has that many nodes available. If more than 24 are requested, only the nodes with 28 nodes can complete the job. But if more than 28 cores are requested, the job will never be completed. Experiment to see what results in the best computation performance.

4. Memory per core - This governs how much memory is allocated for your job, per core used. For planar and stereo PIV, 4 GB should be fine. If you're curious, run a study and see if more makes a difference.

5. Wall time - This is the amount of time you would like to reserve the resources. Be conservative but not extremely so, because the longer the wall time, the longer it will take for your job to get started.

7 Practice Exercise

The original practice exercise has been modified in order to circumvent some Davis 8.4 bugs that prevent that technique from working. Many of the steps in this are exactly the same (1-12)

1. Install FileZilla and KiTTY as shown in Section 3.1.1.

2. SSH into PACE and run `pace-whoami` to see what clusters you have access to, and your corresponding priority.

3. Run `pace-check-queue queuename` for each queue you have access to. Check out the number of cores each node has.

4. Run `pace-quota` to check data storage restrictions on your home, data, and scratch drives.

5. Run `pwd` to find out what the absolute path of your home directory is.

6. Install Davis as described in Section 4, and note the absolute path of your Davis installation that should look something like `/nv/hp19/tburrows3/davis`

7. Now, find a PIV computer with the cluster script processing enabled, and unzip the "practice_project.zip" included archive onto the desktop.

8. Open DaVis and set the Project directory to `Desktop/practice_project`. Navigate to `Travis_PIV_Date=20181009_Time=110058_W2 / RPM2400Q_dt1.25 / MaskCreateGeometric`. Now, open the processing dialog and load the processing xml file included in the practice project folder. This has PIV and PIV postprocessing loaded. **NOTE: Check and make sure the first item on the processing list is not disabled. If it is, there will or could be an error in the processing. This is a bug in DaVis.**

9. Press the Create Processing Script button and enter the same values as the included screenshot, except change the absolute path to yours. Your home directory might not be in `/nv/hp19/`. Make sure to set the script directory to the practice project folder as the screenshot does. You can tell this step has completed by finding the script in the practice folder, along with a new directory with the same name as the script. In addition, placeholder folders have been made for the PIV and postprocessing results, within the PIV project folder.

10. Using FileZilla, transfer this whole folder `practice_project` to your data folder on PACE. This folder should contain the cluster script, the new cluster script folder, the PIV folder with corresponding xml and exp files, and the python function `batch-submit.py`.

11. Before submitting the job, you need to modify permissions of the practice script .sh file to make it executable. To back to your SSH terminal and run `chmod +x practice_script.sh`. Your working directory must be the folder which contains this script.

12. Test the script `practice_script.sh` to see if it works properly before submitting the job. To know if it works, first open up the script (`nano practice_script.sh`) and delete the flag on the last line of the file that disables output, which is `-stdoff`. With this removed, DaVis will tell you what is happening. Now, save the file by pressing ctrl+x, y, then enter. Now, type `./practice_script.sh`. You should start seeing messages from DaVis, eventually indicating that it is processing. Note that you are currently processing on the PACE head note, not a compute node. So once you have proved to yourself that it works, cancel it by pressing ctrl+c. Insert the `-stdoff` flag back into the file. An example of what this script looks like is shown below:

```
#!/bin/bash

PROCESSING_XML="/nv/hp19/tburrows3/data/practice_project/practice_script.xml"
PROCESSINGMODE=0
PROJECT_PATH="/nv/hp19/tburrows3/data/practice_project/Travis_PIV_Date=20181009_Time=110058_W2"
SOURCE_PATH="/nv/hp19/tburrows3/data/practice_project/Travis_PIV_Date=20181009_Time=110058_W2/RPM2400Q0_dt1.25/MaskCreateGeometric
RESULT_PATH="/nv/hp19/tburrows3/data/practice_project/Travis_PIV_Date=20181009_Time=110058_W2/RPM2400Q0_dt1.25/MaskCreateGeometric,
FIRSTFILE=1
LASTFILE=100
STEPFILES=1
HASHVALUE=1315337431

if test "$OMPI_COMM_WORLD_RANK" == "" ; then
        OMPI_COMM_WORLD_RANK=0
        OMPI_COMM_WORLD_SIZE=1
fi

/nv/hp19/tburrows3/davis/davis-start.sh -stdoff  -csp "$PROCESSING_XML" $OMPI_COMM_WORLD_RANK $OMPI_COMM_WORLD_SIZE $PROCESSINGMODE
```

13. Instead of directly submitting this script, it will be submitted through the python program `batch-submit.py`. This method submits a single job, which requests for all of the computing resources desired to process the set of images. It is the same as the original method in Section 9.1, but the python function makes it more convenient.

14. Before you can run the python file, you will have to load Python 3, by running `module load python/3.6`

15. The python file acts as a function which requires arguments, including script name, number of processors, and walltime in hours. There are other optional parameters, including queue, memory and email notifications. The default queue is iw-shared-6 and the default memory is 4GB, so it is not strictly necessary to provide this information. Just run `python batch-submit.py` for a detailed description of how to use the function. **For this exercise, try 20 processors and 1 hour of walltime. This will split up the image set into 20 jobs of 5 images each with an hour to finish, which is way more than needed.**

16. Run the function in your terminal on `practice_script.sh` with 20 processors for an hour by typing `python batch-submit.py practice_script.sh 20 1`

17. After running the function, you should see printed output showing that the jobs have been submitted to the queue.

18. Check the status of your job by running `qsub -u username`. If you want to delete the submitted job, type `qdel ********`.

19. To tell if your jobs have completed, you can run `qsub -u username`, which will give your job the status of C (Complete) when the whole array of jobs is finished.

20. When the jobs have completed, go back into FileZilla/Globus and download the vector files that were computed. You are done! Load them in your local DaVis to visualize them and do additional processing.

8 Tips and Tricks

- To check how many vector files have been created so far, in terminal change directories (`cd`) to the folder that will contain the vc7 files. Then type `ls | wc -l` to check the current number of files. To continuously monitor the number of files, type `watch "ls | wc -l"`

- You might find that all your images didn't process. Unfortunately, Davis is not free of bugs. I have found that sometimes when I submit a job that contains PIV and Vector Postprocessing functions, it fails. But when I only submit PIV computing functions, it works. In this situation, you would have to do the postprocessing locally.

8.1 Custom Linux shortcuts

Here is the procedure to create your own custom commands, to make things more convenient.

1. Create a directory in your PACE home directory (∼) called scripts

2. Edit a hidden file in your home directory called `.bashrc` with the command `nano .bashrc`

3. Add the following text to the end of the file: `PATH=∼/scripts:$PATH`

4. Save the `.bashrc` file and run the command `source .bashrc`

5. Go to ∼/scripts and make a file called `qme` with the following contents, where you enter your username instead of gtusername

```
#!/bin/bash
qstat -u gtusername
```

6. Move the `batch-submit.py` file to ∼**/scripts**

7. Make a file called `submit` with the following contents:

```
#!/bin/bash
python ~/scripts/batch_submit.py "$@"
```

8. Now, you can be in any directory and run the python batch submit file with the command `submit`. Also, you can just type `qme` to check your queue rather than the longer command `qstat -u gtusername`

9 Appendix

9.1 Old Practice Exercise

This is the first practice exercise written for using Davis with PACE. It turns out there are some bugs in Davis, preventing this method from always working. Instead, see Section 7.

1. Install FileZilla and KiTTY as shown above.

2. SSH into PACE and run `pace-whoami` to see what clusters you have access to, and your corresponding priority.

3. Run `pace-check-queue queuename` for each queue you have access to. Check out the number of cores each node has.

4. Run `pace-quota` to check data storage restrictions on your home, data, and scratch drives.

5. Run `pwd` to find out what the absolute path of your home directory is.

6. Now, find a PIV computer with the cluster script processing enabled, and unzip the "practice_project.zip" included archive onto the desktop.

7. Open DaVis and set the Project directory to `Desktop/practice_project`. Navigate to `Travis_PIV_Date=20181009 _Time=110058_W2` / `RPM2400Q_dt1.25` / `MaskCreateGeometric`. Now, open the processing dialog and load the processing xml file included in the practice project folder. This has PIV and PIV postprocessing loaded. **NOTE: Check and make sure the first item on the processing list is not disabled. If it is, there will or could be an error in the processing. This is a bug in DaVis.**

8. Press the Create Processing Script button and enter the same values as the included screenshot, except change the username to yours. If other parts of your home directory path are different too, change that (`/nv/hp19/`). Make sure to set the script directory to the practice project folder as the screenshot does. You can tell this step has completed by finding the script in the practice folder, along with a new directory with the same name as the script. In addition, placeholder folders have been made for the PIV and postprocessing results, within the PIV project folder.

9. Exit out of DaVis, and open the pivjob.txt file. This is the PBS script you will submit to PACE. Replace the username placeholders with your username. The rest of the paths should be valid if you have followed the steps correctly thus far.

10. Using FileZilla, transfer this whole folder `practice_project` to your data folder on PACE.

11. Before submitting the job, a couple modifications to the files need to be made. First, you need to modify permissions of the practice script .sh file to make it executable. Run `chmod +x practice_script.sh`. In addition, the PBS script file is in a windows text file format, but needs to be in a unix format. Run `dos2unix pivjob.txt`. This should tell you it successfully converted the file.

12. If you would like, you can test the script `practice_script.sh` to see if it works properly before submitting the job. To know if it works, first open up the script (`nano practice_script.sh`) and delete the flag on the last line of the file that disables output, which is `-stdoff`. With this removed, DaVis will tell you what is happening. Now, save the file by pressing ctrl+x, y, then enter. Now, type `./practice_script.sh`. You should start seeing messages from DaVis, eventually indicating that it is processing. Note that you are currently processing on the PACE head note, not a compute node. So once you have proved to yourself that it works, cancel it by pressing ctrl+c. Insert that flag back into the file, or else your output text file from the job will contain a ton of output you do not need to see. An example of what this script looks like is shown below:

```bash
#!/bin/bash

PROCESSING_XML="/nv/hp19/tburrows3/data/practice_project/practice_script.xml"
PROCESSINGMODE=0
PROJECT_PATH="/nv/hp19/tburrows3/data/practice_project/Travis_PIV_Date=20181009_Time=110058_W2"
SOURCE_PATH="/nv/hp19/tburrows3/data/practice_project/Travis_PIV_Date=20181009_Time=110058_W2/RPM2400Q0_dt1.25/MaskCreateGeometric
RESULT_PATH="/nv/hp19/tburrows3/data/practice_project/Travis_PIV_Date=20181009_Time=110058_W2/RPM2400Q0_dt1.25/MaskCreateGeometric,
FIRSTFILE=1
LASTFILE=100
STEPFILES=1
HASHVALUE=1315337431

if test "$OMPI_COMM_WORLD_RANK" == "" ; then
        OMPI_COMM_WORLD_RANK=0
        OMPI_COMM_WORLD_SIZE=1
fi

/nv/hp19/tburrows3/davis/davis-start.sh -stdoff  -csp "$PROCESSING_XML" $OMPI_COMM_WORLD_RANK $OMPI_COMM_WORLD_SIZE $PROCESSINGMOD
```

13. Now it is time to submit the job, by typing `qsub pivjob.txt`. If there are no syntax errors, this will just return a job number in the queue.

14. Check the status of your job by running `qstat -u yourusername`. This will list all recent jobs, and their status under the "S" column, where Q is queued, R is running, and C is complete. You can also continuously monitor this screen, by typing the command `watch qstat -u yourusername`. When the status turns to C, the job has completed. Note that you can cancel a job by typing `qdel JOBID`, where JOBID is the string of numbers outputted when you submit the job, and the same numbers shown in the qstat screen.

15. When this has completed, go back into FileZilla and download the vector files that were computed. You are done! Load them in your local DaVis to visualize them and do additional processing.

16. The following results are from my own testing of the practice data (100 planar PIV images).

Computer	Nodes	PPN	Time
PACE	1	4	70 min
PACE	2	4	9 min
PACE	1	8	18 min
PACE	1	16	7 min
PACE	1	24	6 min
PACE	2	16	5 min
PACE	4	8	5.5 min
PACE	8	4	24 min
8-core T3500	1	8	27 min
Fastest FMRL PIV PC	1	48	11 min

9.2 batch-submit.py Source Code

```python
"""
batch-submit.py
This is a batch submit function for running Davis 8.4 on PACE.
@author: Travis Burrows
02/13/2020
"""

from subprocess import run, PIPE
```

```python
 9    from os.path import dirname, join, isfile, abspath, basename, splitext, isdir
10    from argparse import ArgumentParser, ArgumentTypeError, ArgumentDefaultsHelpFormatter
11    from math import floor
12    from sys import stderr, exit
13
14    def hoursToString(hours):
15     wholeDays = floor(hours / 24.0)
16     hours -= wholeDays * 24
17     wholeHours = floor(hours)
18     fractionHours = hours - wholeHours
19     wholeMinutes = floor(fractionHours * 60)
20     fractionMinutes = fractionHours*60 - wholeMinutes
21     roundedSeconds = round(fractionMinutes * 60)
22     walltime = '%02d:%02d:%02d' % (wholeHours, wholeMinutes, roundedSeconds)
23     if wholeDays > 0:
24      walltime = ('%02d:' % wholeDays) + walltime
25     return walltime
26
27    def positive_float(x):
28     try:
29      x = float(x)
30     except ValueError:
31      raise ArgumentTypeError("%r not a floating-point literal" % x)
32     if x < 0.0:
33      raise ArgumentTypeError("%r has a negative value" % x)
34     return x
35
36    def runcmd(command):
37     output = run(command, stdout=PIPE, stderr=PIPE)
38     output.stdout = output.stdout.decode('ascii')
39     output.stderr = output.stderr.decode('ascii')
40     if output.returncode == 1 or bool(output.stderr):
41      raise ValueError('Error running "%s".\nSTDOUT:%s\nSTDERR:\n%s' % (command, output.stdout, output.stderr)
          )
42     return output.stdout
43
44    class MyParser(ArgumentParser):
45     def error(self, message):
46      stderr.write('error: %s\n' % message)
47      print()
48      self.print_help()
49      exit(2)
50
51    # Parse arguments
52    parser = MyParser(description='Submit davis cluster script to PACE via qsub',  formatter_class=
          ArgumentDefaultsHelpFormatter)
53    parser.add_argument('script', type=str, help='Path of cluster script to submit')
54    parser.add_argument('procs', type=int, help='Number of processors to use')
55    parser.add_argument('hours',  type=positive_float, help='Time in hours to allow for job to complete (can
          be a decimal)')
56    parser.add_argument('-q','--queue', type=str, dest='queue', choices=['iw-shared-6', 'prometforce-6', '
          prometheus'], default='iw-shared-6', help='Queue to which the job is submitted')
57    parser.add_argument('-m','--memory', type=int, dest='memory', default=4, help='Gigabytes of memory to
          allocate per processor')
58    parser.add_argument('-e','--email', dest='email', help='Email address to send notifications')
59    parser.add_argument('--debug', action='store_true', help='Store debug log files')
60    parser.add_argument('--do-not-submit', dest='donotsubmit', action='store_true', help='Generate PBS file
          but do not submit to PACE')
61
62    args = parser.parse_args()
63    scriptPath = abspath(args.script)
64    queue = args.queue
65    nProcs = args.procs
66    walltime = hoursToString(args.hours)
67    pmem = args.memory
68    email = args.email
69    debug = bool(args.debug)
70
71    if not isfile(scriptPath):
72     raise ValueError('The script path is incorrect.  There is not a file at %s' % scriptPath)
73
74    scriptName = basename(scriptPath)
75    scriptWithoutExtension = splitext(scriptName)[0]
76    outputFile = scriptWithoutExtension + '.txt'
77
78    # Generate path for PBS file
```

```python
79    scriptFolder = dirname(scriptPath)
80    pbsName = scriptWithoutExtension + '.pbs'
81    pbsFilename = join(scriptFolder, pbsName)
82
83    with open(scriptPath) as f:
84      content = f.readlines()
85
86    numLines = len(content)
87
88    # Test to confirm files/folders exist that are referenced in script file
89    keywords = {'PROCESSING_XML':1, 'PROJECT_PATH':0, 'SOURCE_PATH':0, 'RESULT_PATH':0}
90    for label, IsFile in keywords.items():
91      labelIndex = [ indx for indx, strng in zip(list(range(0, numLines)), content) if strng.startswith(label)]
92      if (not len(labelIndex) == 1):
93        raise ValueError('Error finding %s keyword' % (label))
94      else:
95        labelIndex = labelIndex[0]
96      labelFilepath = content[labelIndex][(len(label)+2):-2]
97      if (IsFile and (not isfile(labelFilepath))) or ((not IsFile) and (not isdir(labelFilepath))):
98        raise ValueError('%s has invalid path:%s' % (label, labelFilepath))
99
100   # Test to confirm davis-start.sh exists
101   davisKeyword = 'davis-start'
102   davisIndex = [ indx for indx, strng in zip(list(range(0, numLines)), content) if davisKeyword in strng]
103   if (not len(davisIndex) == 1):
104     raise ValueError('Error finding %s keyword' % (davisKeyword))
105   else:
106     davisIndex = davisIndex[0]
107   keywordIndex = content[davisIndex].index(davisKeyword)
108   davisPath = content[davisIndex][:keywordIndex-1]
109   davisStartPath = join(davisPath, 'davis-start.sh')
110   if not isfile(davisStartPath):
111     raise ValueError('Davis path is invalid: %s' % (davisStartPath))
112
113   # If debug, remove -stdoff command from cluster script:
114   if debug:
115     content[davisIndex] = content[davisIndex].replace('-stdoff','')
116     with open(scriptPath,'w') as f:
117       for line in content:
118         f.write(line)
119
120   outputFilePath = join(scriptFolder, outputFile)
121
122   # Generate contents of temporary PBS script
123   pbsFile = []
124   pbsFile.append('#PBS -N %s' % scriptName)
125   pbsFile.append('#PBS -l nodes=%d:ppn=1' % nProcs)
126   pbsFile.append('#PBS -l pmem=%dgb' % pmem)
127   pbsFile.append('#PBS -l walltime=%s' % walltime)
128   pbsFile.append('#PBS -q %s' % queue)
129   pbsFile.append('#PBS -j oe')
130
131   if debug:
132     pbsFile.append('#PBS -o %s' % outputFile)
133   else:
134     pbsFile.append('#PBS -o /dev/null')
135
136   if bool(email):
137     pbsFile.append('#PBS -m abe')
138     pbsFile.append('#PBS -M %s' % email)
139
140   # Load mpi modules
141   pbsFile.append('module load java/1.8.0_25')
142   pbsFile.append('module load openmpi/2.1.1')
143
144   pbsFile.append('chmod +x %s' % scriptPath)
145
146   if debug:
147     pbsFile.append('mpirun -v -np %d -machinefile $PBS_NODEFILE -output-filename %s %s' % (nProcs,
          outputFilePath, scriptPath))
148   else:
149     pbsFile.append('mpirun -v -np %d -machinefile $PBS_NODEFILE %s' % (nProcs, scriptPath))
150
151   # Write PBS script
152   with open(pbsFilename,'w') as f:
153     for line in pbsFile:
```

```python
154      f.write(line + '\n')

155
156  # Submit script to PACE
157  if not args.donotsubmit:
158    output = runcmd(["qsub", pbsFilename])
159    jobNumber = output[:output.index('.')]
160    print('%s submitted to %s with %d processors' % (scriptName, queue, nProcs))
161    print('Job number: %s' % jobNumber)
162  else:
163    print('Successfully created %s and did not submit.' % pbsName)
```