**Guide on deploying the LUTO model**

**on a High-performance computing (HPC) node**

Why deploy LUTO on an HPC?

Running LUTO (Land-Use Trade-Offs) requires 50G to 200G, depending on the spatial gratuity, of memory, and 10 to 30 CPU cores to parallelize the parameter searching and output writing. To answer the many “what-if” questions, we often want to test LUTO with different scenario settings such as climate change, demand trajectory, and cost and revenue. Intensive computation and multiple scenarios exploiting demand for an efficient way to run LUTO and deploy it on HPC is a proper choice.

Knowledge requirement.

Although we have tried our best to wrap the deploying process into simple scripts, it is still suggested that users have a basic level of understanding of Python and Linux commands.

* For the Python part, users are assumed to know what packages are, understand how to create a Python environment with [Conda](https://conda.io/projects/conda/en/latest/user-guide/install/index.html), and be able to type commands using IPython.
* For the Linux part, users are assumed to understand the concept of relative and absolute paths, know how to change directories, and use bash to execute a script.
* For the code management part, users are assumed to have the knowledge of cloning (i.e., downloading) the LUTO model and updating the code from [GitHub](https://github.com/land-use-trade-offs/luto-2.0).

The minimal deploying case.

|  |  |  |
| --- | --- | --- |
| No. | Operation | One-time operation? |
| 1 | Download the Gurobi TM optimizer | Yes |
| 2 | Login to the HPC and install Gurobi | No/Yes |
| 3 | Clone LUTO from GitHub | Yes |
| 4 | Download raw data | Yes |
| 5 | Create the Python environment | Yes |
| 6 | Convert raw data and create a scenario template | No |
| 7 | Submit tasks to HPC | No |

1. Download the Gurobi TM optimizer.

The HPC is mostly a Linux platform so we chose Gurobi’s Linux version for demonstration. Gurobi is a commercial software but offers a free trial for academic users. Below are the steps to obtain an academic trial account, make sure you have an education email before applying.

* [Register for a free Gurobi account](https://portal.gurobi.com/iam/register/) as an academic.
* [Login](https://portal.gurobi.com/iam/login/) an click the A blue background with white text and a file

  Description automatically generated button, then A blue circle with a white cross in it

  Description automatically generated for “Named-User Academic”.
* Go back to A blue background with white text and a file

  Description automatically generated and observe the  row, click the right-most 
* In the newly opened page, find the license block and hit .
* Now a will be downloaded to your computer.
* Go the [the Gurobi download](https://www.gurobi.com/downloads/gurobi-software/) page and download its Linux version.

1. Login to the HPC and install Gurobi.

Now you can use Gurobi optimizer, but the software is not yet installed. Unlike simple double-click to install apps in a Windows system, installing apps in a Linux platform can be challenging because it requires extra steps to config the paths. Below is a walk-through to install Gurobi on a remote Linux server.

* Install an SSH tool. Here use [MobaXterm](https://mobaxterm.mobatek.net/download.html) as an example.
* After installing the MobaXterm, open it and hit the  to create a new connection.
* Click the A yellow key in a square with black text

  Description automatically generated to specify the connection type as SSH.
* Fill in the ， , and . These are the settings you can obtain from the HPC manager.
* Hit OK, and now you are connected to the HPC.
* By default, you should see a white panel on the left and a black screen on the right.
* The left white panel is your “home” directory, just think of it as a regular file folder is fine.
* The right black screen is the “terminal”, which is the place we give instructions to HPC.

We assume that the users are working on HPC with [Slurm Workload Manager](https://slurm.schedmd.com/documentation.html). It's very important that we switch to the node we want to run LUTO. By default, when we log in to HCP, we are in the login node. We can conceptually think of HPC as a hotel, and the login node is like the reception desk. We need to tell the reception (login node) which room we want to get into (the working node). Below is the step to switch to the working node.

* Excute `sinfo` to show available nodes. In my case, I want to use the “mem” node.
* Excute `srun -p mem --pty /bin/bash` to get into the “mem” node. Note that if we successfully get into the working node, the machine name will change from  to .

It is always important to check that we are on a working note before making any changes. Now that we are on the “mem” note, we can process installing the Gurobi.

* Drag the downloaded Gurobi file to the left white panel.
* After a while, you should see the file appear in the left white panel.
* Now type in `ls` in the right black screen and hit enter. You should see a line showing the file like .
* Now type in `tar -xzf gurobi11.0.1\_linux64.tar.gz` in the terminal. After a few seconds, there should be nothing happened in the terminal.
* Now go to the left pannel, hit the “refresh folder” button , you will see a new folder appears .

Now Gurobi is technically functional, but the problem is how to use it. Linux needs us to tell it the existence of Gurobi, which, in formal terms, is to configure the system path to Gurobi. Below are the steps to do that.

* Let's try to enter a Gurobi command `gurobi.sh` in the terminal, and you will get an error message as follows . This is because Linux does not know the location to execute this command.
* Hit the ‘Show hidden files’ button 
* Now we can see many transparent icons and the  is the file that Linux hears from us to answer “which app has been installed on where”.
* Double click the , a text editor will pop out and possibly show many alien-like text.
* Scroll down to the last line and we need to type in something to inform Linux where Gurobi has been installed.
* Go back to the file explorer panel (left white panel), double click the , and make sure you can see some directories like , , and .
* If there is only a directory, then you need to double-click it to make sure you can see many directories like in the previous step.
* Now we can see the many directories, we are confident that all Gurobi files are stored here, and we need to send this message to Linux.
* Copy the path in the navigation column  (your path should be different to this), then go to the opened  text editor, and type in the following information. Your path should be different from the example, make sure you copied the path from the navigation line.
  + export GUROBI\_HOME=/home/jinzhu/gurobi1101/linux64/
  + PATH=$PATH:/home/jinzhu/gurobi1101/linux64/bin
  + LD\_LIBRARY\_PATH=/home/jinzhu/gurobi1101/linux64/lib
* Now let’s run the command `source .bashrc`, which can be conceptually understood as “refresh” in a Windows system.
* At last, let’s move the to our home directory. To get the path of “home directory”, we can first enter `cd` in the terminal and then enter ‘pwd’. The returning path is the “home directory”.
* Change the file explorer panel to the “home directory”, and then drag the to it.
* Now if we run `gurobi.sh`, we will get some information saying Gurobi is setting up arguments and so on. Enter `exit()` to leave this secession. Congratulations, you are now able to use Grurobi optimizer, and we probably will not do anything with Grurobi in the future.

1. Clone LUTO from GitHub.

LUTO is publicly accessible on its GitHub page, and we can easily clone (download) it to a remote server (e.g., HPC).

* Go to [LUTO’s GitHub page](https://github.com/land-use-trade-offs/luto-2.0).
* Click the  button, choose “HTTPS”, and then copy the appeared link.
* Go back to the MobaXterm interface, we need to create a folder to store the LUTO code.

It’s always been a good habit to be organized in storing files. Before cloning LUTO to the remote server, we need to think about where to place the code and where to store data. LUTO is designed to treat codes and data separately, so we will create a “LUTO” folder in the home directory, and then two separate sub-folders to store codes and data individually. If the data folder has a different name compared to the below steps, the code will not be able to find it. So, let’s keep the naming conventions to make everything work.

* Enter `cd` in the terminal to go to home directory.
* Enter `mkdir LUTO` to create a directory with the name of “LUTO”.
* Enter `cd LUTO` get get insider the LUTO directory, and then `mkdir raw\_data` to create the data folder.
* Run the command of ‘git clone <link copied from previous step>’ to download the LUTO model.
* Now run the `ls` command, and we should see two folder names .

1. Download raw data.

The raw data is about 4G in size, and it’s not recommended to just drag these files using MobaXterm because of low transfer speed and unstable connection. We will use a professional data transferring tool named FileZilla to transfer the raw data from our computer to the remote server.

* Make sure you are inside a working node.
* [Download](https://filezilla-project.org/download.php) and install the FileZilla.
* Open FileZilla and hit the server icon A screen shot of a computer

  Description automatically generated to start establishing a connection.
* In the pop-out panel, specify the protocol as “SFTP …”, and enter “Host”, “Port”, “User”, and “Password” obtained from the HPC manager.
* Hit “Connect”, and we can see two split panels, the left is the “Local” and the right is the “Remote”.
* Now [download raw data from “Zenode”,](https://zenodo.org/records/8328560) and then drag all downloaded files from “Local” to the ~/LUTO/raw\_data directory in the “Remote”.

1. Create the Python environment.

LUTO is built with Python language, so we need to install Python first. Besides Python itself, LUTO also required support for processing geospatial data, creating graphs, loading/saving Excel files, etc. We can install packages individually to meet all requirements, but that would be trivial and error-prone. Luckily, some apps are specifically designed to manage installing packages and keeping good compatibility among install packages. We choose [Miniforge](https://github.com/conda-forge/miniforge) for creating the Python environment because of its lightweight and fast installation speed.

* Make sure you are inside a working node.
* Change the terminal directory to the bash script location with the below command.

“cd ~/LUTO2/luto-2.0/luto/tools/create\_task\_runs/bash\_scripts”

* List the directory and make sure you can see files as shown below.



* Execute `bash create\_env.sh` and the miniforge3 will be installed automatically.
* After a few seconds, you should see some instructions like “close and re-open your current shell.”
* Now we need to refresh the terminal by executing `source ~/.bashrc`.
* Note there will be a “(conda)” append to the start of terminal .
* Execute ` bash install\_pkg.sh` and it will install all required packages for LUTO.
* The installation could take a few minutes, and we will at last see some progress bar have finished successfully.
* Now let's try the command `conda env list`, and we should see a “base” and a “luto”, where the “luto” is the Python environment we need.
* Execute the command `conda activate luto`, and you will see the start of the terminal is now becoming “luto”, which means we can run the LUTO model now!

1. Convert raw data and create the scenario template.

We are finally able to run the LUTO model, but before starting the model immediately, we need to convert the raw data into a format that is acceptable to LUTO.

* Make sure you are inside a working node.
* Change the terminal directory to `~/LUTO2/luto-2.0/luto/tools/create\_task\_runs/bash\_scripts/`
* Execute the command `bash conver\_raw\_data.sh`
* This process could take ~ 20 minutes.

After successfully converting the raw data, we can now create the scenario that reflects our understanding of how the future world looks like to LUTO. For example, we can specify different climate change, food demand, and greenhouse gas emissions targets to LUTO. We will create a scenario template first and then manually add our interested settings for LUTO to run with.

* Change the terminal directory to `~/LUTO2/luto-2.0/`
* Activate the LUTO environment with `conda activate luto`
* Now type `ipython` into the terminal and hit enter, we will now be inside a Python session.
* Execute the below 3 lines of command.

`from luto.tools.create\_task\_runs.helpers import create\_settings\_template`

`create\_settings\_template()`

`exit()`

* Back to the left file explorer panel, navigate to `~/LUTO2/`, and a new  shows up.
* Inside the , we can see a file, double click to open it.
* Now we can see a scenario template sheet pops out. The first column is the name of all parameters, and the second column (Default\_run) is the default settings for running the LUTO model. The “Default\_run” is just a reference and will not used to create tasks, so we need to create our setting columns.
* Copy the “Default\_run” past it to the rightward column. Change the column head to a sensible name, and pay attention to some of the runing parameters in the last few rows:

**NODE**: must be a name that comes from the command `sinfo`

**MEM**: the memory allocated for each job. Default to ‘auto’ but can be modified with the format “50G”.

**CPU\_PER\_TASK**: the CPU cores allocated to each job. Default to ‘auto’ but can be modified using an integer like “16”.

**TIME**: the maximum running time for each job. Default to ‘auto’ but can be modified with the format “dd-HH:MM:SS”.

**JOB\_NAME**: the name for each job. Default to ‘auto’ that copied the column name, can be changed to a string.

* It’s OK to set multiple columns in the template. Note the more columns set, the less CPU will be allocated to each task.

1. Convert raw data and create the scenario template.

The last step of deploying LUTO on the HPC cluster is to submit the tasks. The task submission is easy but we still need to monitor the running process.

* Make sure you are inside a working node.
* Change the terminal directory to `~/LUTO2/luto-2.0/`
* Now type `ipython` into the terminal and hit enter, we will now be inside a Python session.
* Execute the below 3 lines of command.

`from luto.tools.create\_task\_runs.helpers import create\_task\_runs`

`create\_task\_runs()`

`exit()`

* Now we should see some report saying “task was submitted to \*\*\*”

There are two ways to monitor the running process. One is to look at the log, and the other is to check the resource usage of the node.

* Navigate to the `~/LUTO2/Custom\_runs/` using the left file explorer panel.
* You can new directories created, and their names are the columns of the .
* Enter in any of the new directories, the structure is what we can see in the `luto2.0`.
* Further enter the `output` directory, we can see some logs with timestamps. Double-click one log, and we can see the running process with timestamps.

To check the running tasks with resource usage, we can use the `top` and `squeue` commands.

* Make sure you are inside a working node.
* Execute the `top` command, we can see the running process within the working node and their CPU, memory usage, etc.
* Execute the `squeue` command, we can see how many tasks is now running on the node, and their status such as running time, if it was waiting for enough resources.