NeuroManager User and Programmer's Guide – 0.980 Addendum

David B. Stockton

May 24, 2016

1 Introduction

This addendum is a supplement to the NeuroManager 0.961 User and Programmer's Guide. Since the publication of that guide, we have substantially revised the way NeuroManager is configured, in ways that should be quite helpful to the user. We have also added support for cloud servers, both permanent and ephemeral. We list all changes in summary form, then present details in subsequent sections. The new version is numbered 0.980.

2 Current Deficiencies

Certain features of the 0.961 version have not yet been brought into this version. Here is a list:

1. LINUX host: The software does not yet have LINUX host support. It is coming soon.
2. GPU examples: We have not yet brought the GPU examples up to date.
3. XML examples: The XML examples are in the works.
4. Stampede config files are not yet stable.

3 Summary of changes and additions

1. User data — we have moved all user data (such as location of external programs, dual key file information, notifications phone number and email) to an external ini file. Section 4.
2. Greater location control — we have added the ability to specify the location of the SimSpec file and the location where the simulation results should be placed. Section 5.
3. Revised machine class hierarchy — We have simplified the inheritance tree and yet made it more powerful. Section 6.
4. Improved cluster queue handling. Rather than make each queue a subclass, we have placed queue definition into the cluster's JSON-format configuration file. Section 7.
5. Added cloud servers. We have added the ability to use existing cloud servers in addition to standalone physical servers. We have also added the ability to create and destroy ephemeral servers, which we call 'wisps', as part of an overall script. This minimizes the cost associated with using cloud servers. The cloud management facilities are defined with a class hierarchy which allows the user to add new cloud types and extend existing ones. We use the API directly to avoid the need to add client cloud programs. Section 8.
6. Machine configuration data — we have moved this from the 'create...' m-files to a JSON–based format which improves usability without sacrificing much in the way of flexibility. Section 9.

7. Consistent configuration. We have extended the cloud concepts of ‘flavor’ and ‘image’ to the other machine types so that they are all treated the same. So a standalone physical feature has an ‘image’ which is its software/hardware configuration, and a ‘flavor’ which is its number of cores, amount of RAM, and so on. Section 10.

8. Machine – Simulator compatibility checking. We have added some limited degree of ensuring the compatibility of the SimCore / hardware facilities required by a Simulator and the setup of a given Machine. Section 11.

9. Improved scheduling. We have made major improvements to NeuroManager’s scheduling of Simulations on Simulators, including the ability to cancel a job that is stuck in a cluster wait queue and reschedule it on a free Simulator with better performance. In addition, we have isolated the scheduler to allow for easier future improvement. Section 12.

10. Improved Simulator naming. We have revised the automatic naming of Simulators so that they are shorter, easier to locate, and have unique names. Section 13.

11. Better logging — we have moved the machine set configuration inside the Neuromanager class so that it has easy access to the log, notifications, etc.. Section 14.

4 User data

We wanted to separate code from data when it comes to user information. So we adopted the ini file format and created a new m–file to read it. The new format has sections for local machine directory location, cURL program location, authentication information such as key directory and key names, and notification information such as phone number, carrier, and email address. Simply place your data into an ini file in the same format as seen in the example file, ‘userStaticData.ini’, located in the NeuroManager installation directory.

5 Greater location control

We have added more directory specifications to reduce the need to copy/paste key files. We have added the ability to specify the location of the SimSpec file and the ability to place the simulation results in a specific place.

6 Revised machine class hierarchy

The addition of cloud servers and the possibility of supporting cloud clusters demanded a substantial revision and simplification of the machine class hierarchy. The current version of the hierarchy looks like that seen in Figure 1. We have provided updated versions of all the examples seen in the User Guide proper.

7 Improved cluster queue handling

The new JSON file configuration allows us to specify specific flavors for each queue, and (in the future) to specify other queue–specific characteristics like GPU support. The Information file for each cluster has a list of its queues and all its specific data, such as name and job file string. In concert, the machine classes have been simplified.

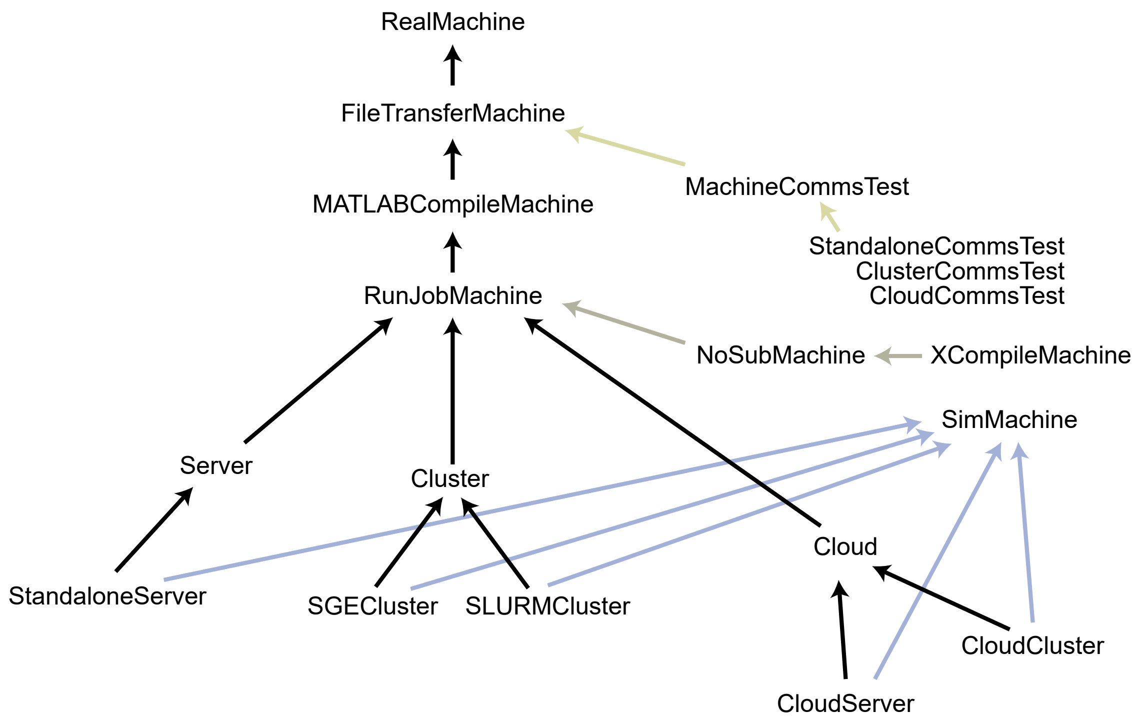

Figure 1: Updated machine class hierarchy. The NEURON details are handled completely differently. The major types of Server, Cluster, and Cloud improve the clarity of the classes, while JSON–format config files handle the remaining details.

8 Added support for cloud servers

Once instantiated and configured correctly, cloud servers are just like standalone physical servers in terms of NeuroManager functionality, so we have added the ability to use existing cloud servers in addition to standalone physical servers. The ability to create and destroy temporary servers automatically is more involved, because of the need to specify image, flavor, and other details. We use the name 'wisps' for these temporary, or ephemeral, servers and have provided cloud management classes that are hidden from daily use by the machine set config class. As an example, the 'addWispSet' command gives the user the ability to add a number of temporary servers, each with a specified number of Simulators. The user chooses the image from which the servers will be cloned, and the JSON–format file which describes the cloud lists the images and their supported SimCores.

The cloud management facilities are defined with a class hierarchy which allows the user to add new cloud types and extend existing ones. We have worked with two OpenStack clouds – Chameleon Cloud and Rackspace. The management classes use the OpenStack API directly to avoid the need to use client cloud programs, but the ability to subclass allows for differences between cloud administration and setup (Figure 2. NeuroManager uses a very small subset of the API in the form of an object–oriented SDK that is just enough to perform the tasks needed by NeuroManager.

The MachineSetConfig methods addStandaloneServer, addCluster, addCloudServer, addWisp, and addWispSet are all given a presence at the NeuroManager class level by mirroring methods. Note that the 'wispness' of a cloud server is only a flag; all cloud servers use the CloudServer class. The treatment of wisps is a bit different though, since a) there is no config file specifically for each wisp, and b) some wisp characteristics, especially IPAddress, are not known until after the wisp instance has been created, whereas each nontemporary cloud server, like a physical server, has its own configuration file.

8.1 The SinSimSet_SS10 example

We have added a new example, 'SineSimSet_SS10.m', which adds cloud servers and wisps. This section describes how to make this example work (in the style of the descriptions of the other examples).

The first thing to do is to create the image from which the servers will be cloned, using your cloud's dashboard. Your cloud documentation should have most of the information you need[1]. We have provided two scripts that worked for us to configure the instance properly. It's a time–consuming process but then the image is available for as many clones as you like:

1. Launch an instance using desired flavor and OS–image

2. Connect to the instance using your SSH key.

[1]See, for example, https://support.rackspace.com/how-to/create-a-cloud-server and https://support.rackspace.com/how-to/rackspace-cloud-essentials-checking-a-server-s-ssh-host-fingerprint-with-the-web-console

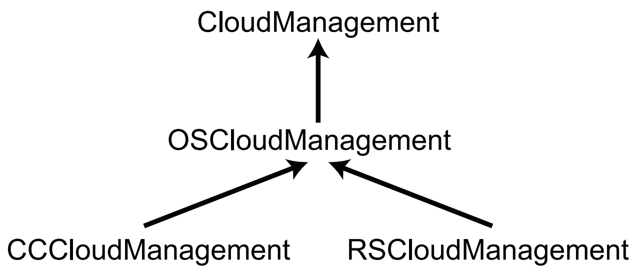

Figure 2: Cloud management class hierarchy. Chameleon Cloud (CC) and Rackspace (RS) are both OpenStack (OS) clouds but have their differences.

3. Install all OS utilities and applications required to support the SimCore (for this example, the SimCore is MATLAB of the version matching your compiler).
4. Install the MATLAB MCR that matches your MATLAB compiler version.
5. Create a work directory for NeuroManager to use.
6. Log into the server using dual key
7. Record the hostKeyFingerprint using ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
8. Save the instance as an image
9. Add the image information to your CloudInfo.json file, including the hostKeyFingerprint

Now that we have an image and it is recorded in the CloudInfo file, we can create the 'pre–existing cloud server':

1. Create the pre–existing server, selecting the image you created above
2. Create the info file for the server, using the example file supplied with NeuroManager in the LocalMachine directory.

In contrast with pre–existing cloud servers, wisps don't have a resource info file. Instead, they have a small 'recipe' file that just tells NeuroManager cloud management what ingredients to use when creating the wisp or wisp set, including the image, flavor, and network. All wisps configure as cloud servers, but they set the 'isWisp' switch that the **removeWisps()** method uses to decide which cloud servers in the config to remove.

8.2 Cloud management type

Figure 2 shows the localization of the cloud management classes. We have left the original files with the installation since there is no personal information in them; this should make the files more accessible to the user. The file *CloudManagementType.m* makes the connection between the leaf classes (e.g., CCCloudManagement) and the cloudManagementType field of the Cloud Info file (see Section 15 below).

9 Machine configuration data

To achieve clearer and more flexible machine configuration, we wanted to move much of the machine configuration details to configuration files that supplement the now–simplified class hierarchy. The ini file format is not complex enough and the XML format is very difficult to read. As a result, we chose the JSON format which, although it does not permit comments, is remarkably concise and clear, while permitting hierarchical data formats. It also works well with cloud API facilities.

The 'addXXX...' methods act as an interface to the machine config classes that load in the JSON files and construct the machineSetConfig which is used to construct the actual SimMachines on the remote resources.

10 Consistent configuration

One of NeuroManager's goals is to make daily use of any supported computational resource type the same as any other. The cloud concepts of 'image' and 'flavor' were perfect vehicles for describing the thing a Simulator would be facing during operation, so we adopted it for all resources. The goal is not to capture the entire machine, the way a general–purpose solution might need to do, but just the points that are salient to the user's research goals.

The concept of ‘image’ encompasses identification of the software stack, to the least amount that is useful to NeuroManager and the user’s intention. Standalone servers and permanent cloud servers will have just one image, which will contain information such as image name, user name, ip address, the location and details of the MATLAB MCR installation, and a list of the SimCores available on that machine together with their installation details. Clusters will have a single image as well, based on the assumption that the cluster has a parallel filesystem [get proper terminology for this]. Ephemeral cloud servers make use of an image file for their creation, which is a snapshot of a previous cloud server that was configured to provide the SimCores that the user is interested in. In these cases the images are identified in the cloud information file, which lists all the images on that cloud which are relevant to the user. Each image in the list has its reference on the cloud together with the characteristics that the image represents — once again, the image name, user name, MCR location, and the like. The NeuroManager cloud management software will make use of this image information to construct the cloud server. Note that the ephemeral cloud server’s id and ip address are not available until the wisp has been created. This is the reason the wisps are created separately from the MachineSet constructor call and before the testCommunications() call.

The concept of ‘flavor’ encompasses the physical nature of the machine in question, including number of processors, cores per processor, and amount of RAM. In NeuroManager the flavor is used both to construct cloud servers and to ensure a machine meets the requirements of the Simulator type in play. Standalone servers have a fixed hardware configuration that we capture in their single flavor. The queues of a cluster are typically separated, at least in part, by hardware differences, so each cluster information file has a list of queues, each of which has its own flavor. A nonephemeral cloud server has been built with a specific flavor, so their information files contain that information. Ephemeral cloud server creation requires the identification of a cloud–specific flavor, so the cloud information file contains all the flavors, and their details, that the user thinks is relevant. Note that, although the cloud API can be used to list flavors automatically, not all the flavor information can be obtained through the API. We were required to search the browser interface to fill in some of the missing information.

The user is free to extend the concepts of flavor and image in order to capture specific details of the software stack or hardware/virtual hardware configuration that are relevant to the simulation research at hand.

11 Machine–Simulator compatibility checking

We have added a limited degree of compatibility testing to ensure that each machine in the user’s MachineSet has the ability to host the Simulator of choice.

First, the machine must have installed on it the SimCore upon which the Simulator is based. In order to accomplish this, we have associated with each Simulator definition the list of SimCores with which it is compatible. This is compared with the machine’s image description, which contains a detailed list of the SimCores that that image supports.

Second, the machine must have the proper hardware to support the Simulator; for example, a Simulator that requires eight cores must be hosted on a machines that have eight cores. This is a question of ‘flavor’, so NeuroManager will ensure that the machine’s flavor has at least the flavor required by the Simulator requirements.

12 Improved scheduling

NeuroManager has been with what we are calling ‘Scheduler2.0’, which is a major improvement over the original scheduler.

The original scheduler simply placed the next Simulation on the first index–order Simulator that was available. In addition, once a Simulation was placed on a Simulator, it was stuck there no matter what happened. This was a difficulty when the wait queue time for a cluster–hosted Simulator was long; the Simulation on that Simulator would delay the entire set; often for many hours.

Scheduler2.0 is much more sophisticated. It is built using a modified min–min strategy with the reasonable assumption that all the simulations in the set will have identical runtimes on a given machine, so that all runtime differences are due to machine differences. NeuroManager gathers four timing statistics for each machine in play and at the top of each scheduling cycle sorts the machines by estimated time of completion. NeuroManager then uses that table to place the next batch of simulations on the contiguous set of open Simulators that have the lowest estimated time of completion. To combat the stuck–in–wait problem, Scheduler2.0 waits until the set of simulations is nearly complete. If there are simulations that have been submitted but have not yet run, and there are open Simulators that have an actual runtime statistic that is shorter than the time the stuck simulation has been waiting, Scheduler2.0 will cancel the job, retire the stuck Simulator, and place the cancelled simulation on the better–performing Simulator.

13 Simulator naming

In the original NeuroManager, Simulators were named automatically using the Simulator name, which lead to confusion and names that were unwieldy and not unique. The automatic naming has been changed to increment the machine name, which is shorter and unique.

14 MachineSet configuration move

The MachineSet configuration activity has been moved inside the NeuroManager class. This new location gives the classes easy access to NeuroManager facilities such as the log, and notifications, which improves the reporting aspect of NeuroManager. This is especially important in the light of cloud server creation and deletion, which should be logged like any other significant operation.

15 JSON file list and description

- Resource information file (example: *Server01Info.json, Cluster01Info.json, PersistentCloud-Server01.json*). Gathers all relevant information about an existing resource into one place for the purpose of Simulator placement (wisps are not represented by files, though their cloud is — see the next entry). *resourceName* is a unique identifier for the resource. *resourceType* is one of (STANDALONESERVER, SGECLUSTER, SLURMCLUSTER, or CLOUDSERVER), though the user can add types via MachineType.m. Existing resources have only one *image*. The image entry specifies user login data, the MATLAB information for that image, the SimCores supported by that image, and the specifics for each SimCore. The *flavor* entry lists the effective hardware characteristics of the resource. For clusters, the *queues* section describes the cloud's queues, including the strings that must be included in the job file that NeuroManager constructs for each simulation submission. Since each queue has its own hardware, each queue entry contains its individual flavor. (Pre–existing) CLOUDSERVER instances are essentially the same as STANDALONESERVER, except that the file includes 'instanceName' which is the cloud's name for the instance. 'resourceName' is the cloud name. In addition, the 'cloudInfoFile' gives the name of the information file for the cloud on which the instance is located.

- Cloud information file (example: *Cloud01Info.json*). Gathers all relevant information about a cloud into one place for the purpose of wisp management. *resourceName* is a unique identifier for the cloud. *resourceType* is CLOUD. *cloudManagementType* is one of (ChameleonOS_KVM, RACKSPACE), though the user can add types via CloudManagementType.m. The 'OS_...' entries and network, powerStatePhrase, and extAddressRoot entries are user–specific to the OS cloud in question; other cloud types would have different entries. Since representing the full complexity of a cloud is not desirable in this application, some of these entries are memorized in this file rather than determined from other pieces of information. The *networks* entries are those user networks that must be specified (if any) for an instance to be formed on that cloud. It is not necessary for all networks to be entered here; only those which the user intends to select for NeuroManager use. The *flavors* entries are those flavors from the cloud that the user wishes to be able to select. The *images* entries are those images of which the user wishes NeuroManager to make use for constructing wisps. Each image entry specifies the MATLAB information for that image and the SimCores supported by that image and the specifics for each SimCore. Note that clouds have image–flavor compatibilities which NeuroManager does not handle; it is up to the user to select a correct combination (this is easy with a small amount of experience with the cloud). Images in this file also must contain the 'hostKeyFingerprint' which is obtained from any instance constructed from the image in question. The hostKeyFingerprint is used to allow SSH to connect to the instance despite having never connected to it before. All wisps formed from an image have the same hostKeyFingerprint.

- Wisp definition file (example: *Cloud01SineSimWispInfo.json.* Determines how an ephemeral cloud instance will be built. It selects a cloud information file, an image, flavor, and network from that file, and states that it is a CLOUDSERVER type (future types may include CLOUDCLUSTER, etc).

- SimCore type definition file (example: *SimCoreType.json*). Defines the acceptable SimCore names and their properties. These properties are added dynamically to the machine configuration, and are checked against the resource information provided by the user. Note that each resource description file's SimCore entry has to specify what type the SimCore is; that type comes from this file.

16 Troubleshooting tips