Automatic Configuration of CESM/CCSM4 on Amazon EC2 Cloud GT-CS-10-15

By Sameer Ansari and Rocky Dunlap, September 7, 2010

Abstract

The goal is to develop a prototype configurator service that automatically creates Amazon EC2 machine instances running on the EC2 Cloud containing ready-to-run configurations of the Community Earth System Model (CESM). The CESM Climate Configurator (C3) service takes in high-level experimental configuration tasks as an XML specification. In the future this will be encapsulated by a web-based graphical user interface. From the high-level configuration input C3 determines the closest match from a list of preconfigured Amazon Machine Images (AMI), and starts an instance (a running AMI on the EC2 cloud). C3 then creates and sends a shell script which finishes the configuration of the instance to the requested experiment, and returns a link to the running instance with the specified simulation ready to run.

Motivation

Configuration of CESM simulations is a highly complex task due to the need for CESM to support a large number of scientific scenarios. Secondly, complexity of configuration and heterogeneous hardware and software environments leads to an inability of scientists to easily share and reproduce previous experimental configurations. This impedes scientific progress. The cloud computing paradigm offers a potential answer to these problems by encapsulating complete configurations of CESM inside virtual machines. This includes the entire software stack: the operating system, compilers, software dependencies (such as MPI and NetCDF), the CESM source code itself, and configuration files and scripts.

This prototype system shows the viability of configuring CESM to execute in a cloud computing environment, and furthermore, that the configuration itself can be automated removing scientists from the details of the underlying computational environment.

Background

Community Climate System Model (CCSM)

The Community Climate System Model (CCSM) was created by NCAR in 1983 as a freely available global atmosphere model for use by the wider climate research community. It was initially called the Community Climate Model (CCM) until 1996. The formulation of the CCSM has steadily improved over the past two decades. Computers powerful enough to run the model have become relatively inexpensive and widely available, and usage of the model has become widespread in the university community, and at some national laboratories [8].

Computational Climate Modeling

The current process for climate modeling begins with Climate scientists who must take a high-level hypothesis and determine an experiment which can be validated or refuted by a simulation. Easterbrook found that Climate scientists build large, complex simulations, typically with little or no formal software engineering training, and that they rely on self-organization of teams, extensive use of informal communication channels, and developers who are also users and domain experts [5]. In the Met Office – the UK's national weather service – as the range of expertise needed to develop climate models has grown; it has become increasingly hard to provide all the necessary expertise in-house [5]. The software development process for the Met Office involves a layered approach often seen in open source projects [5]. At the core, about 12 people from Met R&D and CR control the acceptance of changes into the trunk of the Unified Model (UM), the numerical modeling system developed and used at the Met Office [7]. Most have PhDs in numerical computing. Next, about 20 or more senior scientists act as code owners, each responsible for specific sections of the UM (e.g. atmosphere, ocean, etc). Code owners are domain experts who keep up to date with the relevant science and maintain oversight of developments to their sections of the model. In the outer layers are the scientists who run the models as part of their research. A "configuration manager" is appointed for each climate model, usually a more junior scientist. They become the local experts for knowledge about how to configure the model for particular runs, and keep track of the experiments performed with the model. Finally a broader group of scientists both within and outside the Met Office make occasional use of the models [5].

Traditional Grid and Cluster resources

When two or more computers are used together to solve a problem, it is called a computer cluster. Grid computing is similar to cluster computing; however the big difference is that a cluster is homogenous while grids are heterogeneous. A grid makes use of the spare computing power of each computer in the group, while all the machines in a cluster are dedicated to work as a single unit and nothing else [6]. Grid computing is decentralized with distributed job management; a cluster computer acts as one large computer, or supercomputer.

Typically, climate modeling scientists who wish to run numerical experimentsnon-virtualized cluster computing resources to run simulations. The National Center for Atmospheric Research's (NCAR) Computational & Information Systems Laboratory (CISL) provides supercomputing and data management services to its user community [3]. CISL operates and oversees computing facilities that are both virtual and physical, using major supercomputers such as Bluefire, Frost and Lynx [3]. A disadvantage of such local resources is that it costs large amounts to obtain and maintain the hardware. The Met Office currently uses a £33 million supercomputing system for weather predications, with a theoretical speed of 1 petaflop [2]. Not only does it occupy two halls each the size of a football pitch, it also uses 1.2 megawatts of power, enough for a small town [2].

Cloud Computing

Cloud computing is a technology where an online repository of resources, software, and information are provided to computers and other devices on demand, like the electricity grid. In the case of modeling, cloud computing creates the possibility of renting processing power and storage. Using cloud resources instead of local hardware resources brings the main advantage of flexibility. Instead

of spending large amounts to buy permanent physical assets, a user can rent as much processing power and space as needed and change it on the fly. Cloud computing also gives scientists the ability to do experiments outside of the limits of local resources and time, giving equal ability to small groups or large institutions to run large-scale simulations at practical costs for brief and/or uncertain lengths of time. Some major cloud computing providers include [9] the Google Apps Engine[1], Amazon Elastic Cloud 2[2], Microsoft Windows Live[3], Symphony VPDC[4], and Terremark Worldwide The Enterprise Cloud[5]. In this paper Amazon's Elastic Cloud 2 (EC2) is chosen for its straightforward API and convenient web interface for managing cloud resources.

Amazon EC2

One such cloud resource is Amazon's Elastic Cloud. The Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud [1]. In addition to the cloud computing resources provided, it also provides a web service interface that allows one to modify and transfer cloud resources with minimal user knowledge. The amount of time required to obtain and boot new server instances is at most minutes, allowing one to quickly scale capacity, both up and down, as computing requirements change. Amazon EC2 also provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios. The terms used in EC2 include the following [1]:

- Amazon Machine Image (AMI)
    - An AMI is a computer file system which is packaged and can be deployed on a virtual machine.
- Instance
    - An instance is a running server, it is a virtual computer in the sense that it is located in the cloud, and can be accessed and utilized like any normal server. An instance is created by running an AMI, and it is connected to an EBS volume for data storage.
- Elastic Block Storage (EBS) Volume
    - An EBS volume starts at 10GB, and expands as the data needing to be stored increases; it connects to an instance as if it were a physical storage device.
- Amazon Web Services (AWS)
    - A convenient, straightforward web-interface for managing cloud resources.

As of July 29, 2010, the pricing of Amazons EC2 services are 34 cents an hour for a large instance [1], which has the following properties:

- Linux/UNIX Usage
- 7.5 GB memory
- 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)
- 850 GB instance storage (2×420 GB plus 10 GB root partition)
- 64-bit platform

[1] http://code.google.com/appengine/
[2] http://aws.amazon.com/ec2/
[3] http://www.microsoft.com/cloud/
[4] http://www.savvisknowscloud.com/
[5] http://www.terremark.com/services/cloudcomputing.aspx

One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor [1].

Amazon EC2 Cluster Cloud Computing

On July 13, 2010, Amazon introduced the High Performance Computing (HPC) instance type[1] which is used for cluster cloud computing, combating the major arguments against utilizing virtual computers, namely bandwidth bottlenecks, by designing for parallel processing between multiple cloud computer instances. As stated directly by Amazon:

> "Cluster Compute Instances provide similar functionality to other Amazon EC2 instances but have been specifically engineered to provide high-performance compute and network capability. Cluster Compute Instances provide more CPU than any other Amazon EC2 instance. Customers can also group Cluster Compute Instances into clusters allowing applications to get the low-latency network performance required for tightly coupled, node-to-node communication (typical of many HPC applications). Cluster Compute Instances also provide significantly increased network throughput making them well suited for customer applications that need to perform network-intensive operations. Depending on usage patterns, applications can see up to 10 times the network throughput of the largest current Amazon EC2 instance types."

The advantages of Cluster Compute Instances can be applied directly to global climate model simulations which are typically run in parallel.

Software expertise and configuration complexity

Without cloud resources, climate modeling scientists must have working knowledge of the Community Earth System Model (CESM) modeling software to be able to install and utilize it on computing resources, keep abreast of changes in the software, and have local knowledge of all the models' configuration options and what they do. Complexity in model configuration impedes scientific progress.

This research is focused on creating an interface between scientists and CESM simulations on cloud computers. C3's technical details and source code are provided in the appendix. The system creates virtual cloud computer instances with a configured copy of the CESM software based on user specifications. This removes the burden of learning how to not only install and use the CESM software, but to learn and build Amazon EC2 cloud instances and then install experimental setups on them. The input is currently through and XML file containing simulation setup specifications. This can easily be overlaid with a visual user interface which asks only for simulation specifications, sending the data to the interface which determines the type of computing resources needed, obtains it from the cloud resources, sets up the experiment, and returns a link to a computer image which can be run at any time, or shared easily.

Distribution of experiments

A major disadvantage of utilizing traditional computing resources is that experiment configurations are unique to the user, and rarely well-documented enough to be transferrable to interested outside parties. Scientists wishing to compare experiments and notes must first communicate the complex process of setting up the same environment to repeat or modify the experiments.

With AMIs of experiments stored on the EC2 cloud, archiving of important runs is possible. A new possibility arises with the storage of the experiments themselves rather than just the output data. A few advantages of storing the model rather than the output include being able to modify the input, sharing modifiable models with colleagues, and simplifying the recreation of the experiment when referred to by future sources. The usage of the C3 automatically creates a standardized type of computer, based on the cloud paradigm (currently Amazon EC2), thereby entirely avoiding the difficulty of recreating unique experimental setups. Re-creation of an experiment becomes as simple as passing an AMI id string and running an instance of that AMI. The distribution of AMIs becomes a matter of choice then, as it can be private to the scientist, spread to colleagues and other institutions securely via privacy controls, or publicly available to anyone.

Earth System Curator

> "The Earth System Curator is a National Science Foundation sponsored project developing a metadata formalism for describing the digital resources used in climate simulations. The primary motivating observation of the project is that a simulation/model's source code plus the configuration parameters required for a model run are a compact representation of the dataset generated when the model is executed. The end goal of the project is a convergence of models and data where both resources are accessed uniformly from a single registry" [4].

One direct possibility that arises from experiments as AMIs is storage of AMI ids inside the Earth System Curator database. If experiments are saved as AMIs on the Amazon EC2 cloud as persistent copies, it becomes possible to link such experiments directly to a single registry simply as string AMI ids, which can be accessed publicly or privately according to the scientists' will.

CESM/CCSM4 Configuration

The normal process for configuring and running a CESM simulation requires a Linux-based computer with the following installed: Intel Fortran, Intel C, MPI, and NetCDF. The CESM/CCSM4 software must also be installed. The following process is based on the CESM/CCSM documentation [9].

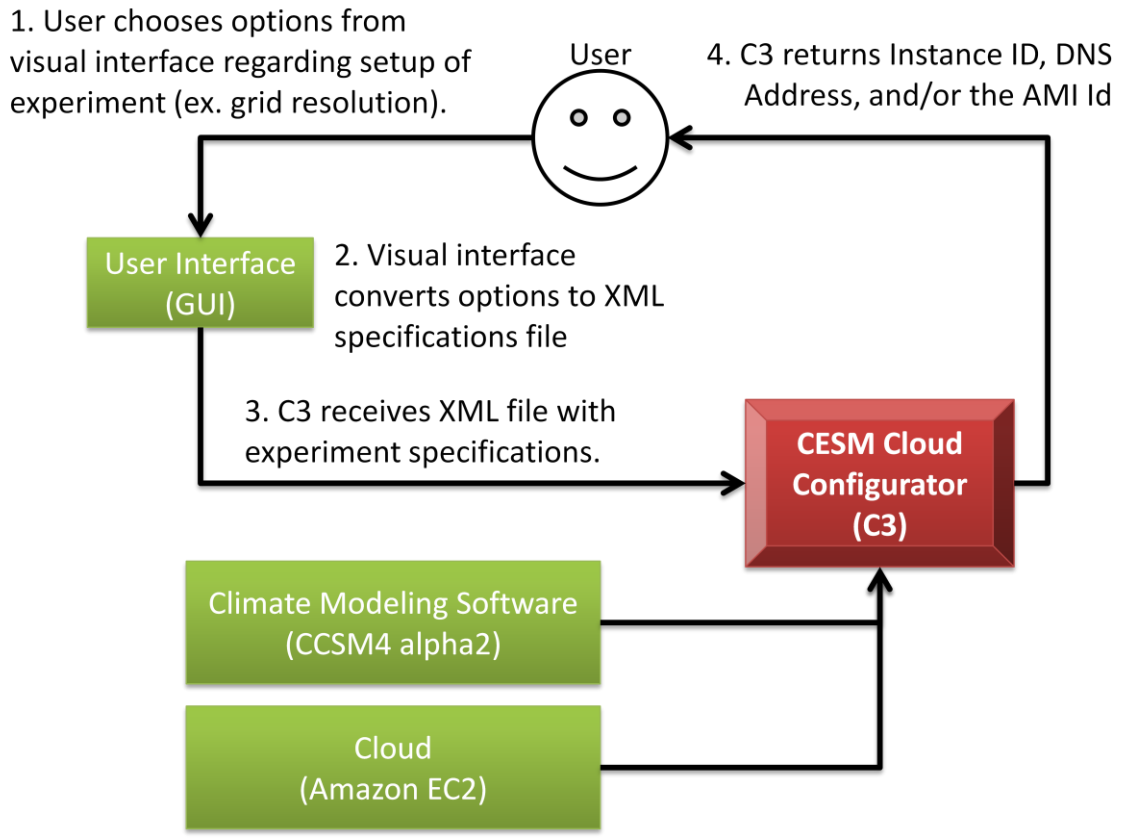
In CESM, each experiment is run as a 'case', which details the specifics of a simulation and the type of machine(s) it will be run on. To create a CESM case, using terminal in the `scripts` directory of the CESM/CCSM software a new case is created with the command `create_newcase;` a non-generic machine description stores environment variables for a specific machine. The case is configured using the command `configure`. It is then built and run using the commands `<case_name>.build` and `<case_name>.run` [9].

CESM Climate Configurator (C3)

A user will interact with the CESM Cloud Configurator (C3) system via a web interface, choosing experiment options such as grid resolution, model components (atmosphere, land-ice, etc.), number of virtual cores, memory size, etc. C3 receives from the interface an XML file with the specifications. It then determines the closest 'base' AMI (preconfigured AMI) to use, instantiates the number of cloud instances required, configuring them to the experiment, and depending on the option, either begins the simulation, returning to the user the instance id's for overview, or returns an AMI id for future usage.

As of August, 2010, the C3 system takes in an XML specification list containing the same CESM `create_newcase` parameters in addition to Amazon Web Services (AWS) security information. It creates an instance of a preconfigured Ubuntu Linux installation with CESM/CCSM4 alpha02 software installed on it. It then automatically creates a new case, configures, builds and runs the case using the same scripts as a manual configuration.

High Level System Architecture



C3 automates the creation of cloud instances configured with CESM simulations. Through C3, experiments can be repeated and modified easily by others, and accessed through the web via Amazon Web Services. The experiments are saved as AMI's with a CESM model ready to be run, or ready for further refinement.

Use Case

Clarissa's research has begun to show the possibility of mitigating the effects of global warming, but she needs to run a few minor simulations to test her hypothesis. Unfortunately, the simulations are too large to run in a reasonable amount of time on anything she has access to personally.

Clarissa goes online to the C3 website, where she inputs her simulation parameters and a timeframe of within 24 hours. C3 determines that the simulation will require at least 10,000 instances, the web interface tells her roughly how much it will cost her overall, creates the instances on EC2, configures and runs the experiment, returning a link for her to check any of the instances remotely. She is emailed by C3 when the simulation finishes, with the data accessible online. She decides to correspond with Jane about the data results, passing the link from which Jane is able to not only access the raw output data, but also the simulation instances itself. If Jane wanted to counter-examine Clarissa's research with the same experiment or with modified parameters, she would be able to configure the experiment from Clarissa's exact setup via the link.

Architecture

The architecture of C3 consists of two primary components, the launcher component which creates an instance (running virtual machine on EC2 Cloud), and the configurator component which sets up the climate modeling experiment on the instance.

Launcher

The launcher takes input in the form of XML specifications containing input climate modeling parameters such as year, grid size, and simulation type, and determining the 'base AMI' (pre-built Amazon Machine Image (AMI) with CESM software installed on it) that most closely matches the user's specifications. It then starts an instance of the chosen base AMI on the Amazon EC2 service.

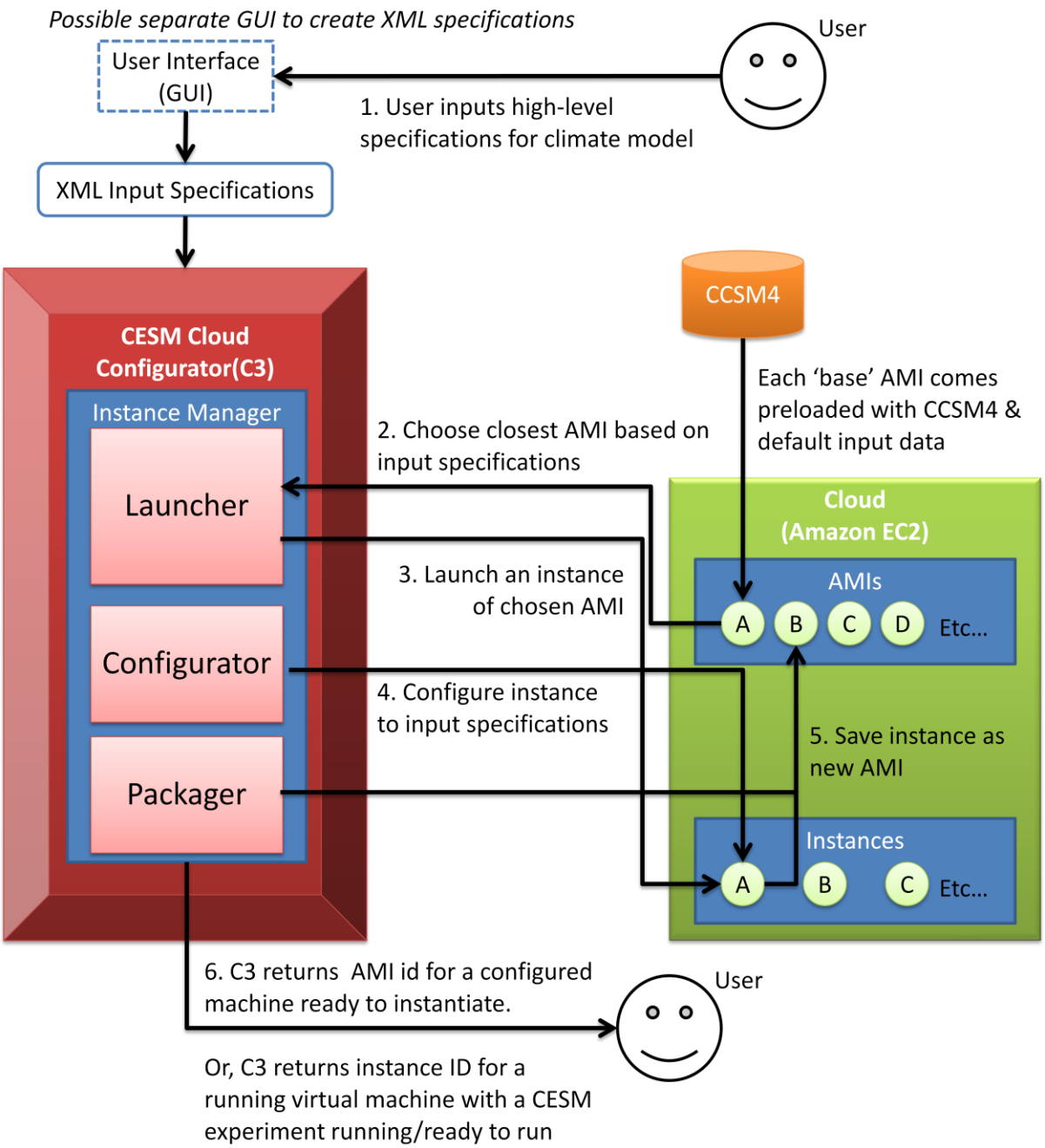
Configurator

The configurator accesses the launched instance of the base AMI and submits shell scripts that complete the configuration determined by user-specified experimental parameters.

Packager

The packager communicates with the EC2 cloud to take the configured instance and convert it into a new AMI. It then returns the AMI id, where the AMI is a ready-to-instantiate image of a virtual Linux machine with a CESM model configured and ready to run.

C3 Interaction with Cloud



Current Results

As of August 31, 2010, the C3 prototype is able to run an instance of an AMI, configure the instance based on input xml specifications, and return an instance id for the running instance. This involves the use of a limited launcher which chooses the closest AMI from a set of preconfigured base AMIs, which is currently just a single base AMI. This base AMI is currently an Ubuntu machine with CCSM4alpha2, it is able to create, configure, and build a case; however, it does not successfully finish a run yet, this is possibly due to differences in software prerequisite versions, and is found to be

unrelated to the cloud via testing on a local machine. The C3 system is written using the Ruby language as a set of procedural scripts, which read in from an XML specification file which contains locations of key pair information as well as new case configuration information. The system then creates a script to run on the instance that creates a new case, configures it accordingly, and finally builds and runs it, with all output being piped back to the host.

Conclusions and Future Work

The current C3 prototype system shows the viability of running climate model simulations virtually and sharing virtual instances containing ready-to-run simulations. Using the Amazon EC2 API C3 is able to automatically refine a preconfigured AMI, create a new CESM case on the cloud instance and configure the case based on an XML specification, build the case, and run the case.

The C3 system is still a prototype and can be expanded upon in myriad ways. Some major future goals include rebuilding the system with separate classes defining launcher, configurator, and packager, designing a separate web-based user interface, running C3 completely on the cloud, utilizing the newly introduced EC2 Cluster Compute Instances, and abstracting the configuration knowledge to an outside explicit knowledge base.

Appendix

Technologies Used

Amazon EC2 - The Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud[6]. EC2 is used to create virtual computers set up with working experiments.

Ruby – A dynamic, open-source programming language[7]. This is the language used to write the CESM Cloud Configurator (C3) system. It utilizes an XML Parser library to read in an XML instruction set. The scripts also run shell commands to access the Amazon API for cloud access.

CESM/CCSM4 – The Community Earth System Model (CESM) is a fully-coupled, global climate model that provides state-of-the-art computer simulations of the Earth's past, present, and future climate states [8]. The CESM/CCSM4 system is installed on cloud instances to run climate models.

Process

The first step is to create the base AMIs that will be used by the C3 system, this only needs to be done once. After that, the set of base pre-configured AMIs can be shared to be used by any running C3 system. Second, a computer running Linux is set up that will run the C3 system. This host computer will take in XML instructions from the user. Once a new cloud computer instance is created by the C3 system, and the link is returned, the user can access the system via SSH from their local machine.

[6] http://aws.amazon.com/ec2/
[7] http://www.ruby-lang.org/en/

Creation of base AMIs

Setting up an Amazon EC2 Cloud Instance (Linux 64bit)

To use Amazon EC2, an Amazon Web Services (AWS) account is required. New accounts can be created by going to `http://aws.amazon.com` . Once an account is setup the user signs into the AWS Management Console and creates a base instance from the community AMIs. The AMI used in the prototype system is Ubuntu 9.04 Jaunty 64 bit running on an `m1.large` type system, with an AMI ID `ami-49c72920`.

Security – Private Key, Certificate, Public Key

Before creating the instance, the user creates a new RSA keypair and saves the public key to their local machine. To get the X.509 Certificate and private key go to the AWS 'Account' and then to 'Security'. Down the page there is a tab with X.509 Certificates. Inside that box, click on create a new X.509 Certificate and download both the private key and X.509 Certificate. To create a public key go to the 'Key Pairs' tab in the AWS EC2 Console, and creates and downloads a new key-pair from there. The name of the keypair is used whenever creating new instances from the terminal.

Setting permissions

All the keys and certificates should be set using `chmod 700` so that most Secure Shell (SSH) clients will willingly use them.

Connecting to instance

The terminology for shell/terminal commands is `$>` signifies the command prompt. To connect to the instance using Linux, SSH is used with the following terminal command:

```
$> ssh -i <key.pem> root@<Public DNS>
```

Where key.pem is the public key saved on the computer, and Public DNS is the address of the instance obtainable from the AWS Console. On a Windows machine, the instance can be connected using an SSH client (such as SecureCRT) using the public DNS as the hostname, 22 for the port, 'root' for the username, and the downloaded public keypair file.

Install Prerequisites on an empty Community Cloud Instance

A Base AMI is created by first taking an empty Community cloud instance and installing the pre-requisite software needed to run a CESM/CCSM4 scientific experiment. A community Ubuntu AMI is selected using the Amazon Web Service (AWS). Then an instance of it is created and connected to from the user's local machine. First the pre-requisite software is installed and then the CESM/CCSM4 software is installed. After that, the scripts required to communicate with the C3 system are added. Finally, the instance is saved as a new AMI; this is one of the preconfigured 'base' AMIs that the C3 system will use for future cloud instance creation.

Starting from fresh Ubuntu machine instance

Using terminal type:

```
$> apt-get update
$> apt-get install csh
```

CCSM uses gmake, which has been replaced by make in newer Ubuntu installations, to create a link between make and gmake, type:

```
$> ln -s /usr/bin/make /usr/bin/gmake
```

Installing libstdc++5 library

```
$> wget http://fr.archive.ubuntu.com/ubuntu/pool/universe/g/gcc-3.3/libstdc++5_3.3.6-17ubuntu1_amd64.deb
$> dpkg -i libstdc++5_3.3.6-17ubuntu1_amd64.deb
```

Installing Intel C Compiler, Intel 64 bit

Download file, unpack, go to folder and install

```
$> tar xvzf l_cproc_p_11.1.072_intel64
$> cd l_cproc_p_11.1.072_intel64
$> ./install.sh
```

Then follow typical install, providing serial as needed, there should be no missing prerequisites.

Installing Intel Fortran Compiler, Intel 64 bit

Download file, unpack, go to folder and install

```
$> tar xvzf l_cprof_p_11.1.072_intel64
$> cd l_cprof_p_11.1.072_intel64 % go to it's folder
$> ./install.sh
```

Then follow typical install, providing serial as needed, there should be no missing prerequisites.

Install NetCDF 4.0.1

Download from http://www.unidata.ucar.edu/downloads/netcdf/netcdf-4_0_1/index.jsp
Download the source code: `netcdf-4.0.1.tar.gz - Latest stable release of netCDF` as a gzipped tar file. Unzip to folder, go to folder in terminal and type the following:

```
$>
export CC=/opt/intel/Compiler/11.1/072/bin/intel64/icc
export CXX=/opt/intel/Compiler/11.1/072/bin/intel64/icpc
export F77=/opt/intel/Compiler/11.1/072/bin/intel64/ifort
export FC=/opt/intel/Compiler/11.1/072/bin/intel64/ifort
export CPPFLAGS="-fPIC -DpgiFortran"
export CFLAGS="-i-static"
export FFLAGS="-i-static"
```

Then configure with the following options

```
$> ./configure --prefix=/opt/netcdf4.1-intel --disable-netcdf-4
```

Then make

```
$> make
```

Check the make, it should pass with no errors

```
$> make check
```

Finally install

```
$> make install
```

Install OpenMPI 1.4.2

Download from http://www.open-mpi.org/software/ompi/v1.4/
Download the source code: openmpi-1.4.2.tar.bz2
Unzip, go to folder in terminal:

```
$>
tar xvjf openmpi-1.4.2.tar.bz2
cd openmpi-1.4.2.tar.bz2
```

Type the following:

```
$> export LD_LIBRARY_PATH=/opt/intel/Compiler/11.1/072/lib/intel64
$> ./configure --prefix=/opt/openmpi-1.4.2-intel \
        CC=/opt/intel/Compiler/11.1/072/bin/intel64/icc \
        CXX=/opt/intel/Compiler/11.1/072/bin/intel64/icpc \
```

```
F77=/opt/intel/Compiler/11.1/072/bin/intel64/ifort \
FC=/opt/intel/Compiler/11.1/072/bin/intel64/ifort
```

Then make

```
$> make
```

Check the make, it should pass with no errors

```
$> make check
```

Finally install

```
$> make install
```

Installing CESM/CCSM4 alpha 2 on Cloud Instance

- We will be following the instructions on

http://www.cesm.ucar.edu/models/ccsm4.0/ccsm_doc/x367.html [9]

- First, to be able to use subversion, type:

```
$> apt-get install subversion
```

- Then, to get CCSM model version, type:

```
$> svn list https://svn-ccsm-release.cgd.ucar.edu/model_versions
```

- type 'p' to accept certificate permanently
- When asking for password for your username, enter to skip
- type 'guestuser' for username
- type 'friendly' for password
- You will see a list of types of CCSM, in this case we will use `ccsm4_0_a02/`
- Type to move files to `~/ccsm4` (in your home directory):

```
$> svn co https://svn-ccsm-
release.cgd.ucar.edu/model_versions/ccsm4_0_a02 ccsm4
```

- CCSM is now installed in your home directory as `~/ccsm4`

Making a Custom CESM/CCSM Machine type

A custom CESM/CCSM machine is the machine specifications detailing things ranging from which compilers are used to the type of processor, defining what type of hardware will be interacting with the CESM/CCSM installation. A new custom machine should be made for the newly created base AMI and all its software, to simplify the running of experiments. The adding of a new custom machine to the CCSM4 system is as follows:

- Following instructions on

http://www.ccsm.ucar.edu/models/ccsm4.0/ccsm_doc/c2151.html#port_adding_mach [9]

- Go to `~/ccsm4` in terminal and type

```
$> mkdir inputdata
```

- Go to `~/ccsm4/scripts`, and type

```
./create_newcase -case test1 \
                 -res f19_g16 \
                 -compset X \
                 -mach generic_linux_intel \
                 -scratchroot /ptmp/ccsm4usr \
                 -din_loc_root_csmdata ~/ccsm4/inputdata \
                 -max_tasks_per_node 8
```

- Go to `~/ccsm4/scripts/test1` in terminal and type:

```
        $> cp env_mach_specific
../ccsm_utils/Machines/env_machopts.samubuntu
```

Replace `'samubuntu'` with any name of choice for specific machine, keeping consistent with that name in future commands and scripts.

- Then

```
$> cp Macros.generic_linux_intel
../ccsm_utils/Machines/Macros.samubuntu
```

- Then edit the `~/ccsm4/scripts/ccsm_utils/Machines/Macros.samubuntu` file and delete (from the start forward) everything up to the lines:

```
$
$
$#============================================================
==
$                # The following always need to be set
$
```

- That first section of the Macros file is added automatically when a case is configured so should not be included in the machine specific setting.
- Then continue editing the Macros.samubuntu file:
- Changing from:

```
FC             := mpif90
CC             := mpicc
NETCDF_PATH    := /usr/local/netcdf-3.6.3-intel-3.2.02
MPICH_PATH     := /usr/local/mpich-1.2.7p1-intel-3.2.02
MPI_LIB_NAME   := mpich
```

- To:

```
FC              := /opt/openmpi-1.4.2-intel/bin/mpif90
CC              := /opt/openmpi-1.4.2-intel/bin/mpicc
NETCDF_PATH    := /opt/netcdf4.1-intel
MPICH_PATH     := /opt/openmpi-1.4.2-intel
MPI_LIB_NAME   := mpi_f90
```

- Create a mkbatch.samubuntu file in `ccsm4/scripts/ccsm_utils/Machines`.

```
$> cd ~/ccsm4/scripts/ccsm_utils/Machines
$> cp mkbatch.generic_linux_intel mkbatch.samubuntu
```

- Then edit the file `mkbatch.samubuntu` (terminal in `~/ccsm4/scripts/ccsm_utils/Machines`) on the 2nd line:
- Changing from:

```
set mach = generic_linux_intel
```

- To:

```
set mach = samubuntu
```

- Add '`/opt/openmpi-1.4.2-intel/bin/mpirun -np ${maxtasks} ./ccsm.exe >&! ccsm.log.\$LID`' below these lines:

```
#mpiexec -n ${maxtasks} ./ccsm.exe >&! ccsm.log.\$LID
#mpirun -np ${maxtasks} ./ccsm.exe >&! ccsm.log.\$LID
```

- , resulting in:

```
#mpiexec -n ${maxtasks} ./ccsm.exe >&! ccsm.log.\$LID
#mpirun -np ${maxtasks} ./ccsm.exe >&! ccsm.log.\$LID
/opt/openmpi-1.4.2-intel/bin/mpirun -np ${maxtasks} ./ccsm.exe >&!
ccsm.log.\$LID
```

- Then edit the `env_machopts.samubuntu` file:
- Changing from:

```
#--- set paths
```

```
#setenv INTEL_PATH /usr/local/intel-cluster-3.2.02
#setenv MPICH_PATH /usr/local/mpich-1.2.7p1-intel-3.2.02
#setenv PATH
${INTEL_PATH}/fc/11.0.074/bin/intel64:${INTEL_PATH}/cc/11.0.074/bi
n/intel64:${MPICH_PATH}/bin:${PATH}
#setenv LD_LIBRARY_PATH
${INTEL_PATH}/cc/11.0.074/lib/intel64:${INTEL_PATH}/fc/11.0.074/li
b/intel64:${LD_LIBRARY_PATH}

#--- set env variables for Macros if needed
#setenv NETCDF_PATH something
```

- To:

```
#--- set paths
setenv INTEL_PATH /opt/intel/Compiler/11.1/072
setenv MPICH_PATH /opt/openmpi-1.4.2-intel
setenv PATH
${INTEL_PATH}/bin/intel64/ifort:${INTEL_PATH}/bin/intel64/icc:${MP
ICH_PATH}/bin:${PATH}
setenv LD_LIBRARY_PATH ${INTEL_PATH}/lib/intel64:${MPICH_PATH}/lib

#--- set env variables for Macros if needed
setenv NETCDF_PATH /opt/netcdf4.1-intel
```

- Then edit the `config_machines.xml` file, placing between the following two lines:

```
         <config_machines>

         <machine MACH="bluefire"
```

- put the following below '`<config_machines>`' and above '`<machine MACH="bluefire"`'

```
<machine MACH="samubuntu"
         DESC="Sam's Ubuntu Setup for Cloud Instance"
         EXEROOT="~/ccsm4/tmp/$CASE"
         OBJROOT="$EXEROOT"
         LIBROOT="$EXEROOT/lib"
         INCROOT="$EXEROOT/lib/include"
         DIN_LOC_ROOT_CSMDATA="~/ccsm4/inputdata"
         DOUT_S_ROOT="~/ccsm4/tmp/archive/$CASE"
         CCSM_BASELINE="~/ccsm4/ccsm_baselines"
         GMAKE_J="2"
                 MAX_TASKS_PER_NODE="8"
         PES_PER_NODE="1"
         OS="Linux" />
```

Setting up a Host Computer (Linux)

The host Linux computer which will host the C3 system (in this case the scripts that interact with EC2) needs access to the Internet to be able to connect to the cloud, and the following Amazon API tools installed:

```
$> apt-get install ec2-api-tools
$> apt-get install ec2-ami-tools
```

The API tools package is used for launching instances, and the AMI tools package is used for packaging instances into AMIs. The host computer does not need the CESM/CCSM4 software installed, as that will already be installed in the base AMIs.

Saving Instance to AMI

On the Amazon AWS web interface, a running instance can be turned into an AMI by right clicking and choosing "`Turn into an Image (EBS AMI)`." Then, once an AMI is made, the AMI ID can be used in the future for creating new instances.

Automated instance saving to AMI

To automatically turn an instance into an AMI, such as when a new instance is made and the setup for running a test is done and ready to go, the command used is `ec2-create-image`. Documentation can be found at http://docs.amazonwebservices.com/AWSEC2/latest/CommandLineReference/ .

To create an AMI from an instance, the following command is used:

```
$> ec2-create-image <instance id>
```

where `<instance id>` is something of the form `i-######`. This AMI can then be used to create as many instances as needed.

XML input syntax

The C3 system will take input via an XML configuration file, separate from the CESM/CCSM software. Eventually the XML file will be populated via a GUI rather than manually. The XML configuration file used by the C3 system is setup with the following hierarchy:

<ccsm4>
- <create_newcase>
- <keypairs>

The first section is the `create_newcase` directory, which holds the configuration setup for an experimental run, including things like the computer setup, the machine used, the case name, and the resolution acronym. The options that can be added here include all that are allowed the by CCSM4 create_newcase script. The second major directory is the `keypairs` directory, which holds both the keypair name created on the Amazon Web Services EC2 Cloud, as well as the directory locations of the private key, X509 Certificate, public key, and the AMI id of the pre-configured base AMI to be launched as an instance.

Example XML configuration file ccsm4_samubuntu.xml

```
<?xml version="1.0"?>
<ccsm4>
 <create_newcase>
  <case>test2</case>
  <res>f19_g16</res>
  <compset>X</compset>
  <mach>samubuntu</mach>
 </create_newcase>

 <keypairs>
  <!-- keypair_name is the name of the keypair as known on the AWS
keypair tab, not a location -->
  <keypair_name>key2</keypair_name>
  <public_key>~/Documents/Keys/key2.pem</public_key>
  <private_key>~/Documents/Keys/pk-
T7NUKMNPGWFDQDYJ2QYXNYTAX6ERL6WJ.pem</private_key>
  <certificate>~/Documents/Keys/cert-
T7NUKMNPGWFDQDYJ2QYXNYTAX6ERL6WJ.pem</certificate>
  <!-- This ami_id corresponds to the base 64bit Ubuntu private
AMI, with CCSM4 already made -->
  <ami_id>ami-52e2093b</ami_id>
 </keypairs>
</ccsm4>
```

Automatically creating new instance from AMI

The script 'ReadXML_CreateNewInstance.rb' reads in a set of XML instructions detailing a scientific simulation (i.e., a particular CESM case) as well as the keypair and base AMI id information, and uses

that to create a new cloud instance from the base AMI on the Amazon EC2 cloud, returning an instance id link which can be used to connect to the instance. This file is a procedural script written using the Ruby language, which interacts with Amazon EC2 via the Amazon API through the Linux terminal.

```
#!/usr/bin/env ruby

# This file needs to be located in same folder as key to be used

require 'net/http'
require 'rexml/document'
require 'time'
include REXML

security = "SSHonly" # Premade security on AWS for new instances,
allows only SSH ports

# Input XML Filename
xml_filename = "ccsm4_samubuntu.xml"
xml_filename = ARGV[0] if ARGV.length == 1
#####################################################################
####
puts "-----------------------------------------------"
puts "-- READING XML : #{xml_filename}"
puts "-----------------------------------------------"

ami_id = nil
doc = Document.new(File.new(xml_filename))
XPath.each( doc, "//ami_id") { |element| ami_id = element.text }


if ami_id == nil
  raise "No ami_id given, need <ami_id>something</ami_id> in xml
inside <keypairs>"
end
#####################################################################
####

#####################
# Start up an instance

# If private key tag or certificate is given, append it to the
ec2run script
keypair_name = nil
private_key = nil
certificate = nil
public_key = nil
XPath.each( doc, "//keypair_name") {|element|  keypair_name =
element.text}
XPath.each( doc, "//public_key") {|element|  public_key =
element.text}
XPath.each( doc, "//private_key") {|element| private_key =
element.text if element.text }
XPath.each( doc, "//certificate") {|element| certificate =
element.text if element.text }
raise "No Keypair Name provided" unless keypair_name
raise "No Public Key location provided" unless public_key
raise "No Private Key location provided." if private_key == nil
raise "No Certificate location provided." if certificate == nil
#####################################################################
####
#~ puts "-----------------------------------------------"
#~ puts "-- EXPORTING KEY TO ENVIRONMENT "
#~ puts "-----------------------------------------------"

#~ # Update environment variables
```

```
#~ system "echo BEGINNING EXPORTS..."
#~ system "export EC2_PRIVATE_KEY=#{private_key}"
#~ system "export EC2_CERT=#{certificate}"
#~ system "echo EXPORTS DONE"
################################################################
####

puts "-----------------------------------------------"
puts "-- CREATING NEW CLOUD INSTANCE "
puts "-----------------------------------------------"

run_cmd = "ec2run #{ami_id} --instance-type m1.large --group
\"#{security}\""
run_cmd << " --key #{keypair_name}"
run_cmd << " --private-key #{private_key} --cert #{certificate}"

puts "Creating new instance and running with following command:"
puts "$> #{run_cmd}"
run_output = `#{run_cmd}`
puts run_output

re = /i-\w{8}/
instance_id = re.match(run_output)
raise "Unable to start instance" unless instance_id

puts "INSTANCE SUCCESSFULLY MADE"

puts
puts "Instance ID : #{instance_id}"
puts
################################################################

puts "-----------------------------------------------"
puts "-- CONNECTING TO INSTANCE"
puts "-----------------------------------------------"
# Get ssh info
desc_instance_command = "ec2din #{instance_id}"
puts "Getting DNS info using command:\n#{desc_instance_command}"
pends = /pending/
pending = true
public_dns = nil

start = Time.now
i=0
while pending
  i+=1
  puts
  puts "Attempt #{i}: #{Time.now-start}s"
  desc_instance = `#{desc_instance_command}`
  puts "Output from command: "
  puts desc_instance
  pending = pends.match(desc_instance)
  raise "Instance taking too long to get running" if (Time.now -
start) > 60 # If it takes more than a minute give up
end

re = /ec2-\S+/
public_dns = re.match(desc_instance)
puts "INSTANCE RUNNING..."
puts "Public DNS: #{public_dns}"
puts

ssh_prefix= "ssh -i #{public_key} root@#{public_dns} "
ssh_cmd = ssh_prefix + "ls -l"

puts "Testing connection via 'ls -l' command..."
puts "Connecting via SSH using following:"
puts ssh_cmd
```

```
output = `#{ssh_cmd}`

puts "BEGIN OUTPUT"
puts output
puts "END OUTPUT"
puts
```

Converting input XML into configuration instructions

The script 'ReadXML_MakeConfig.rb' is used to read in a set of XML instructions detailing a scientific simulation (i.e., a particular CESM case), and convert that into a shell script that creates and configures a CCSM4 case. The shell script is saved as a file to be transferred to a respective cloud instance with the script 'ConnectToInstance.rb'. This Ruby file also interacts with EC2.

```
#!/usr/bin/env ruby

# This file needs to be located in same folder as key to be used

require 'net/http'
require 'rexml/document'
require 'time'
include REXML

# Input XML Filename
xml_filename = "ccsm4_samubuntu.xml"
xml_filename = ARGV[0] if ARGV.length == 1
#####################################################################
####
puts "-----------------------------------------------"
puts "-- READING XML : #{xml_filename}"
puts "-----------------------------------------------"

# Output script name
out_script = "ccsm4_make_simulation_script.txt"

commands = "#!/bin/bash\n\n"
commands << "cd ~/ccsm4/scripts/\n\n"
commands <<  "echo \"Creating new case...\"\n"
create_newcase_cmd = "./create_newcase "
case_name = nil
mach_name = nil
doc = Document.new(File.new(xml_filename))

doc.root.each_element("create_newcase/*") { |element|
  create_newcase_cmd << "-#{element.name} #{element.text} "
}

XPath.each( doc, "//case") { |element| case_name = element.text }
XPath.each( doc, "//mach") { |element| mach_name = element.text
}raise "No case name given, need <case>something</case> in xml
inside <create_newcase>" unless case_name
raise "No machine name given, need <mach>something</mach> in xml
inside <create_newcase>" unless mach_name

commands << create_newcase_cmd + "\n"
commands << "cd ~/ccsm4/scripts\/#{case_name}/\n\n"
commands <<  "\necho \"Configuring case...\"\n"
commands << "./configure -case\n\n"


# BUILD CASE
commands <<  "\necho \"Building case...\"\n"
commands << "./#{case_name}.#{mach_name}.clean_build\n"
commands << "./#{case_name}.#{mach_name}.build\n"
# RUN CASE
```

```
commands << "\necho \"Running simulation...\"\n"
commands << "./#{case_name}.#{mach_name}.run\n"
commands << "\necho \"Run complete.\""

puts "Outputting following commands to file '#{out_script}'...\n"
puts "############################################################"
puts commands
File.open(out_script,'w'){|f| f.write(commands)}
puts "############################################################"
puts "Finished writing to '#{out_script}'"
```

Sending configuration instructions to new instance

The script ConnectToInstance.rb connects to a running cloud instance created using the script 'ReadXML_CreateNewInstance.rb'. To connect it reads in the same initial XML C3 system configuration file for the key and certificate information. It then transfers the shell script created by 'ReadXML_MakeConfig.rb' to the cloud instance, and runs the script, which causes the machine to create and configure a climate model case, and run the model, piping all output back to the local machine. These files are procedural scripts written using the Ruby language, which interact with Amazon EC2 via the Amazon API through the Linux terminal.

```
#!/usr/bin/env ruby

# This file connects to a previously known instance, sends the
config file to it, and runs the config file
# Which in essence creates and runs a climate model on the
instance

require 'net/http'
require 'rexml/document'
require 'time'
include REXML


instance_id = "i-c102a7ab"
instance_id = ARGV[0] if ARGV.length == 1

security = "SSHonly" # Premade security on AWS for new instances,
allows only SSH ports

# Input XML Filename
xml_filename = "ccsm4_samubuntu.xml"
##################################################################
####
# READ XML
puts "-------------------------------------------------"
puts "-- READING XML : #{xml_filename}"
puts "-------------------------------------------------"
doc = Document.new(File.new(xml_filename))

keypair_name = nil
private_key = nil
certificate = nil
public_key = nil
XPath.each( doc, "//keypair_name") {|element|  keypair_name =
element.text}
XPath.each( doc, "//public_key") {|element|  public_key =
element.text}
XPath.each( doc, "//private_key") {|element| private_key =
element.text if element.text }
XPath.each( doc, "//certificate") {|element| certificate =
element.text if element.text }
raise "No Keypair Name provided" unless keypair_name
raise "No Public Key location provided" unless public_key
```

```
raise "No Private Key location provided." if private_key == nil
raise "No Certificate location provided." if certificate == nil
################################################################
####
# INSTANCE INFO
puts "Instance ID : #{instance_id}"
puts
################################################################
# CONNET TO INSTANCE
puts "-------------------------------------------------"
puts "-- CONNECTING TO INSTANCE"
puts "-------------------------------------------------"
# Get ssh info
desc_instance_command = "ec2din #{instance_id}"
#desc_instance_command << " --key #{keypair_name}"
desc_instance_command << " --private-key #{private_key} --cert
#{certificate}"
puts "Getting DNS info using command:\n#{desc_instance_command}"
pends = /pending/
pending = true
public_dns = nil

start = Time.now
i=0
while pending
  i+=1
  puts
  puts "Attempt #{i}: #{Time.now-start}s"
  desc_instance = `#{desc_instance_command}`
  puts "Output from command: "
  puts desc_instance
  pending = pends.match(desc_instance)
  raise "Instance taking too long to get running" if (Time.now -
start) > 60 # If it takes more than a minute give up
end

re = /ec2-\S+/
public_dns = re.match(desc_instance)
puts
puts "FOUND INSTANCE"
puts
puts "Public DNS: #{public_dns}"
puts
raise "Not pending, but no Public DNS either, can't connect "
unless public_dns

ssh_prefix= "ssh -i #{public_key} root@#{public_dns} "
ssh_cmd = ssh_prefix + "ls -l"

puts "Testing connection via 'ls -l' command..."
puts "Connecting via SSH using following:"
puts ssh_cmd

output = `#{ssh_cmd}`

puts "BEGIN OUTPUT"
puts output
puts "END OUTPUT"
puts

################################################
# Send config file to instance
puts "-------------------------------------------------"
puts "-- SENDING CONFIG FILE TO INSTANCE"
puts "-------------------------------------------------"
config_filename = "ccsm4_make_simulation_script.txt"
# scp -i key from_loc server:to_loc
puts "Sending #{config_filename} via scp using command: "
```

```
scp_cmd = "scp -i #{public_key} #{config_filename}
root@#{public_dns}:~/"
puts scp_cmd
output = `#{scp_cmd}`
puts output
#####################################################
puts "-------------------------------------------------"
puts "-- RUNNING CONFIG FILE ON INSTANCE"
puts "-------------------------------------------------"

#`echo -ne '\015'`
cmd1 = "chmod 700 #{config_filename}; "
cmd2 = "./#{config_filename}"

puts "Changing permissions of config file and running config file"
ssh_cmd = ssh_prefix + "-t \""+cmd1+cmd2+ "\""
puts "Running following command on instance: #{ssh_cmd}"
system ssh_cmd
#puts output
```

Running model on new instance

Create New Case

- Following instructions on http://www.ccsm.ucar.edu/models/ccsm4.0/ccsm_doc/x444.html and http://www.ccsm.ucar.edu/models/ccsm4.0/ccsm_doc/c2151.html [9]
- If folder doesn't already exist, go to ~/ccsm4 in terminal and type

```
$> mkdir inputdata
```

- Go to ~/ccsm4/scripts in terminal and type the following (replace samubuntu with machine name chosen):

```
$>
./create_newcase -case test1 -res f19_g16 -compset X -mach
samubuntu
```

- A folder /test1 is made in current directory, going there in terminal and typing

```
$> ./configure -case
```

Build Case

- Then, run:

```
$> ./test1.generic_linux_intel.build
```

which on a first run will download missing input data over an extended period of time (Also possibly requiring reentry of CCSM login information).

- If it is not the first run, type:

```
$> ./test1.generic_linux_intel.clean_build
```

The output rm : no match is normal

Run

- Then, run:

```
$> ./test1.generic_linux_intel.run
```

References

[1] Amazon. *Amazon Elastic Compute Cloud*. Retrieved June 2010, from Amazon EC2. <http://aws.amazon.com/ec2/>

[2] S. de Bruxelles. *Met Office says new super-computer will give more accurate forecasts*. The Times. May 22, 2009. <http://www.timesonline.co.uk/tol/news/weather/article6338014.ece>

[3] Computational & Information Systems Laboratory (CISL). *CISL: a computing laboratory*. Retrieved September, 2010, from CISL. <http://www2.cisl.ucar.edu/cisl-computing>

[4] R. Dunlap, Mark, L., Rugabear, S., Balaji, V., Chastang, J., Cinquini, L., et al. *Earth system curator: metadata infrastructure for climate modeling*. Earth Science Informatics, Vol 1 (3-4), pp131-149. March 2008. < http://www.springerlink.com/content/t781174802510364/>

[5] S. M. Easterbrook and T. Johns, *Engineering the Software for Understanding Climate Change*. IEEE Computing in Science and Engineering, Vol 11 (6), pp65-74. November 2009. <http://www.cs.toronto.edu/~sme/papers/2008/Easterbrook-Johns-2008.pdf>

[6] Journal of Theoretical and Applied Information Technology (JATIT). *Comparison of Grid Computing vs. Cluster Computing*. Retrieved September, 2010, from JATIT. <http://www.jatit.org/research/introduction_grid_computing.htm>

[7] Met Office. *Met Office Unified Model*. Retrieved September, 2010, from Met Office Beta. <http://www.metoffice.gov.uk/research/modelling-systems/unified-model>

[8] National Center for Atmospheric Research (NCAR). *Community Earth System Model*. Retrieved May 2010, from Community Earth System Model. <http://ccsm.ucar.edu/>

[9] M. Verstenstein, T. Craig, A. Middleton, D. Feddema, and C. Fischer. *CCSM4.0 User's Guide*. Community Earth System Model (CESM). Retrieved on July, 2010. <http://www.cesm.ucar.edu/models/ccsm4.0/ccsm_doc/book1.html>

[10] WikiInvest. *Cloud Computing*. Retrieved September, 2010. <http://www.wikinvest.com/concept/Cloud_Computing>