Geoscientific
Model Development

Enabling BOINC in infrastructure as a service cloud system

Diego Montes[1], Juan A. Añel[1,2], Tomás F. Pena[3], Peter Uhe[4,5], and David C. H. Wallom[5]

[1]EPhysLab, Universidade de Vigo, Ourense, Spain
[2]Smith School of Enterprise and the Environment, University of Oxford, Oxford, UK
[3]Centro de Investigación en Tecnoloxías da Información (CITIUS), University of Santiago de Compostela,
Santiago de Compostela, Spain
[4]School of Geography and the Environment, University of Oxford, Oxford, UK
[5]Oxford e-Research Centre, University of Oxford, Oxford, UK

Correspondence to: Diego Montes (kabute@uvigo.es)

Abstract. Volunteer or crowd computing is becoming increasingly popular for solving complex research problems from an increasingly diverse range of areas. The majority of these have been built using the Berkeley Open Infrastructure for Network Computing (BOINC) platform, which provides a range of different services to manage all computation aspects of a project. The BOINC system is ideal in those cases where not only does the research community involved need low-cost access to massive computing resources but also where there is a significant public interest in the research being done.

We discuss the way in which cloud services can help BOINC-based projects to deliver results in a fast, on demand manner. This is difficult to achieve using volunteers, and at the same time, using scalable cloud resources for short on demand projects can optimize the use of the available resources. We show how this design can be used as an efficient distributed computing platform within the cloud, and outline new approaches that could open up new possibilities in this field, using Climateprediction.net (http://www.climateprediction.net/) as a case study.

1 Introduction

Traditionally, climate models have been run using supercomputers because of their vast computational complexity and high cost. Since its early development, climate modelling has been an undertaking that has tested the limits of high-performance computing (HPC). This application of models

to answer different types of questions has led to them being used in manners not originally foreseen. This is because, for some types of simulations, it can take several months to finish a modelling experiment given the scale of resources involved. One reason for including climate modelling as a high-throughput computing (HTC) problem, as opposed to an HPC problem is due to the application design model, where there is a number (not usually greater than 20) of uncoupled, long-running tasks, each corresponding to a single climate simulation and its results.

The aim of increasing the total number of members in an ensemble of climate simulations, together with the need to achieve increased computational power to better represent the physical and chemical processes being modelled, has been well understood for some decades in meteorological and climate research. Climate models make use of ensemble means to improve the accuracy of the results and quantify uncertainty, but the number of members in each ensemble tends to be small due to computational constraints. The overwhelming majority of research projects use ensembles that generally contain only a very small number of simulations, which has an obvious impact in terms of the statistical uncertainty of the results.

The Climateprediction.net project (CPDN) was created in 1999 (Allen, 1999; CPDN, 2015) as a distributed computing initiative to address the uncertainties described above. Its aim is to run thousands of different climate modelling simulations in order to research the uncertainties associated with some of the parameters. This is essential for understanding how small changes or variations in initial conditions can affect

both the models themselves and the results of climate simulations. The project is currently run by the University of Oxford using volunteer computing via the BOINC (Berkeley Open Infrastructure for Network Computing) framework (BOINC, 2014; Anderson, 2004). In its early use of distributed computing, CPDN became a precursor of the many-task computing (MTC) paradigm (Raicu et al., 2008).

CPDN has been running for more than 10 years and faces a number of evolving challenges, such as

– an increasing and variable need for new computational and storage resources;

– the processing power and memory of current volunteers' computers that restricts the use of more complex models and higher resolution; and

– the need to manage costs and budgeting (this is of particular interest in researching on-demand projects requested by external research collaborators and stakeholders).

To address these issues, we have explored the combination of MTC/volunteer and cloud computing as a possible improvement of, or extension to, a real existing project. This kind of solution has previously been proposed for scientific purposes by Iosup et al. (2011) and is supported by initiatives such as Microsoft Azure for Research (Microsoft, 2014).

2 Background

It is not the aim of this paper to describe the internals of BOINC, and for better comprehension of the problem that we are trying to solve, it is recommended to review previous works about this knowledge, such as Ries et al. (2011).

2.1 Problem description

Here, we describe some of the problems that we intend to address, as well as proposed implementations of possible solutions.

– To run more complex and computationally more expensive versions of the model, resources greater than those that can be provided by volunteer computers may be needed. One solution is a re-engineering and deployment of the client side from a volunteer computing architecture to an infrastructure as a service (IaaS) based on cloud computing (e.g. Amazon Web Services, AWS).

– There is a growing need for an on-demand and more predictable return of simulation results. A good example of this is urgent simulations for critical events in real time (e.g. floods) where it is not possible to rely on volunteers; instead, a widely available and massive scaling system is preferable (like the one described here). The current architecture and infrastructure based on BOINC

does not provide a solution that can be scaled up for this purpose. This is because the models are running over a heterogeneous and decentralized environment (on a number of variable and different volunteers' computers with varying configurations), where their behaviour cannot be clearly anticipated or measured, and any control over the available resources is severely limited.

– A rationalization of the costs is required (and establishing useful metrics), not just for internal control but also to provide monetary quotations to project partners and funding bodies; this led us to the need of the development of a control plane together with a front end to display the statistics information and metrics.

– Free software can be used in order to promote scientific reproducibility (Añel, 2011).

– Complete documentation of the process will allow knowledge to be transferred or migrated easily to other systems (Montes, 2014). Additional explanations can be found in the appendices (Appendices A, B, and C).

Furthermore, in this work, we wish to prove the feasibility of running complex applications in this environment. We use weather@home (Massey et al., 2015), a high-resolution regional climate model nested in a global climate model as an example. The remainder of this paper is organized as follows. We firstly present benchmarks of the weather@home application run in AWS in Sect. 3.1, then describe the migration of the CPDN infrastructure to AWS in Sect. 3.2. We also describe our control plane in Sect. 3.3 to conduct the simulations and manage the cloud resources. Lastly, the results are discussed in the conclusion.

3 BOINC deployment to the cloud

3.1 Application benchmarks in Amazon Web Services (AWS)

The example presented here is running CPDN in AWS. AWS is the largest infrastructure as a service (IaaS) provider, it is very well documented, and is the most suitable solution for the problem at present (and with fewer limitations than other providers).

The first step was to benchmark different AWS EC2 instance types[1] to determine their performance running CPDN simulations. These tests were done with a range of instance types, but only choosing instance types that have hardware virtual machine (HVM) virtualization available. Elastic block store (EBS) gp2 storage[2] was used for all instances for ease of comparison. These tests were carried out running multiple copies of a single work unit, in parallel with

---

[1]https://aws.amazon.com/ec2/instance-types/
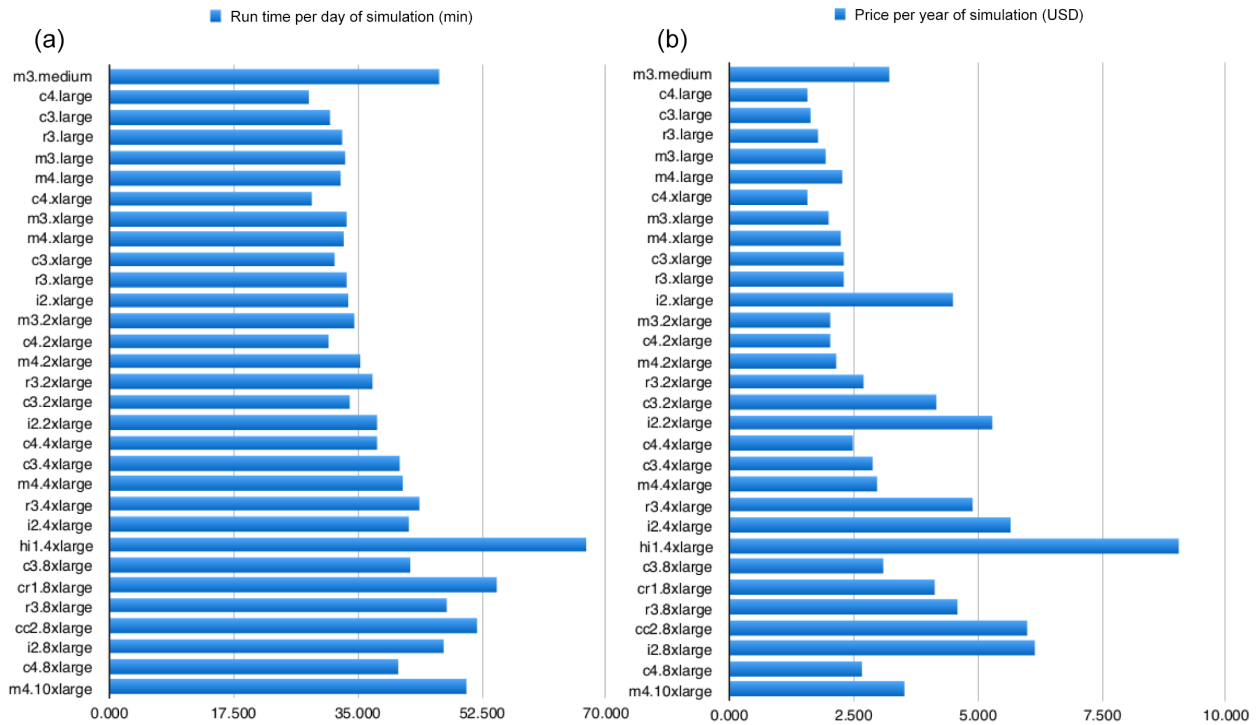[2]http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/

Figure 1. (a) Work-unit run time and (b) cost per simulation year.

the number of simulations matching the number of vCPUs (hyperthreads) available to each instance type. For each instance's type, at least four tests were run.

For benchmarking purposes, short 1-day climate simulations were run. The model used here is weather@home2 which consists of an atmosphere-only model (HadAM3P; Gordon et al., 2000) driving the regional version of the same model (HadRM3P; Pope et al., 2000). This version of the model uses the MOSES 2 land surface scheme. The region chosen is at $0.22°$ (≈ 25 km) resolution over Europe.

Figure 1a shows the average time to run all of the simulations on a particular instance, by instance type. We see a general trend of smaller instances performing better than larger instances. This is likely due to the hardware these instances are on being at a lower load. Running only a single simulation per instance resulted in similar run times for instances of the same category (e.g. c4), and they are not shown in the figure. However, we have verified that it is more cost effective to run the maximum number of simulations per instance than to run instances at a lower load.

Figure 1b shows the estimated cost of running a 1-year simulation on each instance type. The pricing here is based on the spot price[3] in the cheapest availability zone in the us-east-1 region (based on AWS regions) as of June 2016. This shows that the current-generation compute-optimized instances (c4) had three out of the four most cost-effective choices, but other small instance types are amongst the

cheapest. We emphasize that these results are very variable in time and between regions. In the us-west-1 and us-west-2 regions in AWS, the cheapest instance types were m4.large and m4.xlarge, respectively, due to the lower spot price for those particular instances in those regions.
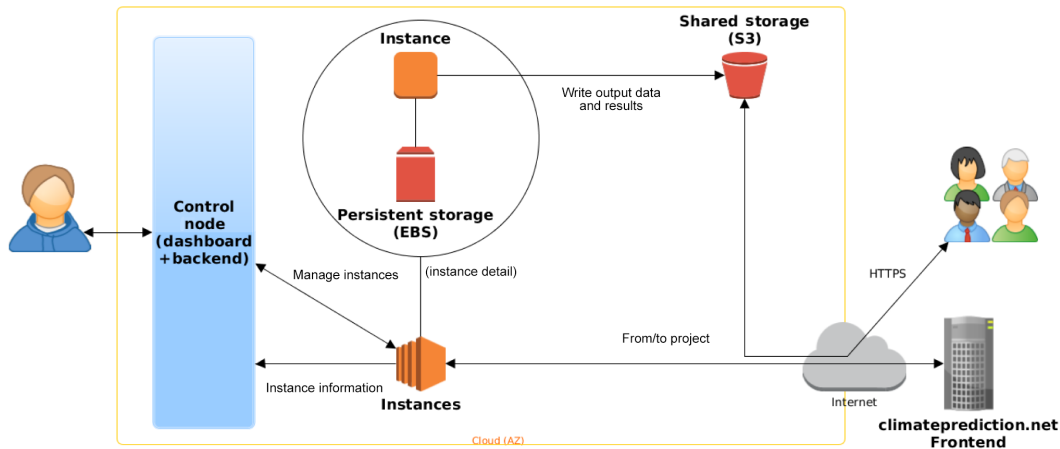
3.2 CPDN infrastructure in AWS

Based on the previous tests, new infrastructure was designed on the cloud (Fig. 2). Several steps were required for its implementation, as described below.

3.2.1 Computing infrastructure

1. First of all, a template was created to allow automated instance creation including

   – instance selection, based on the benchmarks presented in Sect. 3.1;

   – base operating system installation (Amazon Linux (64 bit) was used for this work);

   – storage definition (16 GB, persistent, standard EBS for this case);

   – firewall configuration (inbound – only SSH(22) accepted; outbound – everything accepted); and

   – installation and configuration of BOINC client inside the template image, including its dependencies, such as 32 bit libraries (it is recommended to use the latest version from git).

[3]https://aws.amazon.com/ec2/spot/pricing/

Figure 2. Proposed cloud infrastructure.

2. This was followed by instance post-installation config-
   uration (contextualization); for example, in AWS this is
   achieved by creating a machine image (AMI) and ad-
   justing it by selecting the appropriate options such as
   the kernel image (AKI).

3. Finally, an (optional) installation and configuration of
   AWS EC2 command-line interface is performed. This
   can be useful to debug or troubleshoot issues with the
   infrastructure.

3.2.2 Storage infrastructure

Another problem that needs to be solved is the need for a de-
centralized, low-latency and world-wide-accessible storage
for the output data (each simulation (36 000 work units) gen-
erates ~ 656 GB of results). A solution for this could be a dis-
tributed (accessed within different and synchronized world-
wide endpoints) and scalable massive storage (Fig. 3). Here,
we tested an architecture in which the clients send the results
(tasks) to an Amazon Simple Storage Service (S3) bucket
(storage endpoint). At the same time, CPDN can access these
data over the internet to run postprocessing (e.g. a custom as-
similator); this can be achieved using the AWS API.

Given these values, every work unit returns a result of
~ 0.018 GB with a price of $\sim$ USD 0.005414 each (AWS,
2016a), and $\sim$ USD 194.904 for the full simulation (both
storage and data transfer).

3.3 Project control plane

Having set up the computing and storage infrastructure, we
still lack a control plane to provide a layer for abstraction
and automation, and provide more consistency to the project.
The aim of developing the central control system (Fig. 4) is
to provide a cloud-agnostic, easy-to-use front end (Fig. 5) to
manage the experiments with minimal knowledge of the un-
derlying architecture and obtain a real-time overview of the

current status (including the resources used and run comple-
tion data). Moreover, the central control system lends more
consistency to the view of the project as an IaaS by providing
a simple interface (both back end and front end).

The control plane is still in its early developmental stages
(e.g. although it is cloud agnostic, so far only AWS has a
connector and is supported), and further work will describe
its improvements over time.

It consists of two main components:

– the back end provides the user with a RESTful API with
  basic functionalities related to simulation information
  and management, with the intention of providing (even
  more) agnostic access to the cloud; and

– the front end makes it easier to communicate with the
  API as intuitively and simplistically as possible.

The core component, the RESTful back end (using
JavaScript object notation – JSON), provides simple access
and wraps common actions: start simulation with n nodes,
stop simulation, modify simulation parameters (n nodes), get
simulation status, and get simulation metrics.

4 Conclusions

Several experiments (using all the defined infrastructure)
were done by using standard work units developed by the
climateprediction.net/weather@home project. We processed
work units from two main experiments: the weather@home
UK floods (Schaller et al., 2016) and the weather@home
Australia/New Zealand project (Black et al., 2016), both with
an horizontal resolution of 50 km.

It has been successfully demonstrated that it is possible to
run simulations of a climatic model using infrastructure in
the cloud; while this might not seem complex, to the best
of our knowledge, it has never previously been tested. This
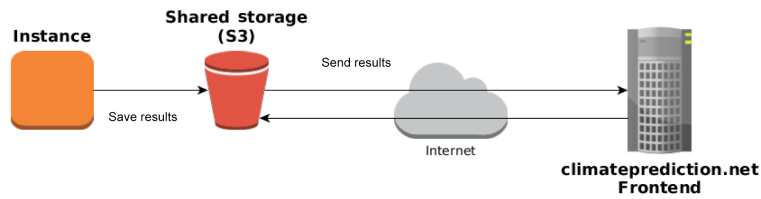efficient use of MTC resources for scientific computing has

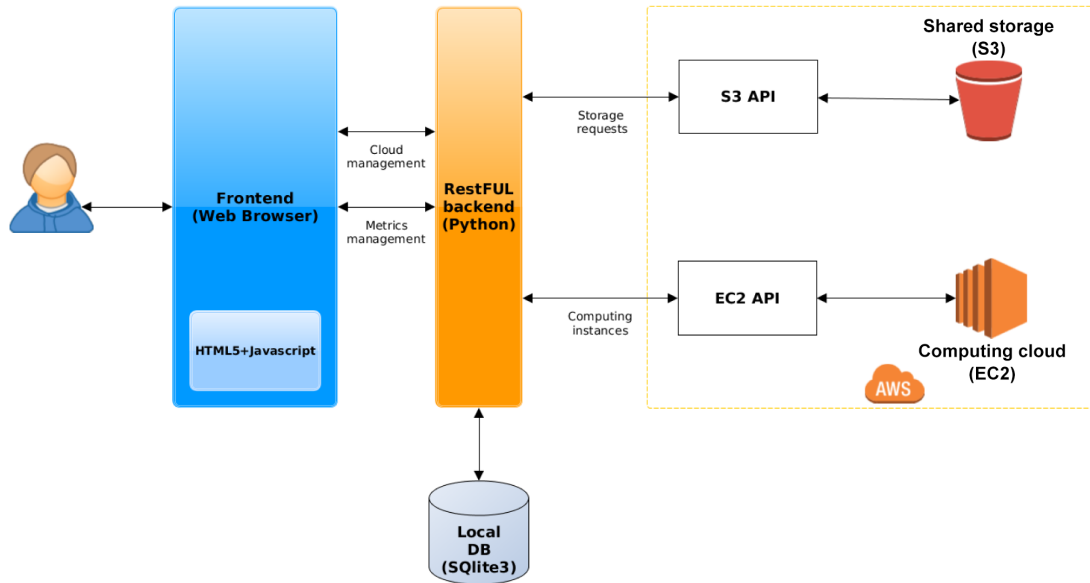**Figure 3.** Shared storage architecture.

Figure 4. Dashboard and metrics application architecture.

previously been used to facilitate real research in other areas (Añel et al., 2014; Schaller et al., 2014).

We have benchmarked a number of Amazon EC2 instance types running CPDN work units. Prices for spot instances vary significantly over time and between instance, but we estimate a price as low as USD 1.50 to run a 1-year simulation based on the c4.large instance in the us-west-1 region in June 2016 (see Fig. 1). To optimize the costs of running simulations in this environment, it will be important to automatically re-evaluate the spot prices to choose the cheapest instance type at the time simulations are submitted. The better performance with smaller instance is due to the fact that vCPUs are hyperthreads, and in smaller instance types there is greater chance the CPU is running at a lower utilization and our instances can scavenge extra CPU cycles (Uhe et al., 2016).

It is interesting to note that cloud services enable us to achieve a given number of tasks completed in some cases 5 times faster than using the regular volunteer computing infrastructure. However, the financial implications can only be justified for critical cases where stakeholders are able to justify through a specific cost–benefit analysis. Anyway, aca-

demic institutions and different type of organizations can benefit from waivers to reduce the fees (AWS, 2016b).

Regarding our usage and solution for storage, S3 was a good fit for this work (it comes out of the box with AWS and the pricing is convenient). However, we would not suggest it as suitable long-term archival of this output but instead suggest to make use of community repositories where such data are curated (it should be noted though that CPDN produces output in community standard NetCDF). Also, we understand that even though the infrastructure described here covers a good number of use cases for different projects and experiments, other alternatives could be analysed:

- AWS Glacier is an interesting option to study in case that long-term storage for data is needed for non-immediate access and with lower cost (AWS, 2016c). In our case, a full simulation (36 000 work units) would have cost us USD 2.624 per month of storage.

- S3 file size is limited to 5 TB and this could be a problem for bigger projects so, options like a CephFS cluster on EC2 could be interesting (Zhao et al., 2015).

This research has also served as a basis for obtaining new research funding as part of climateprediction.net for state-
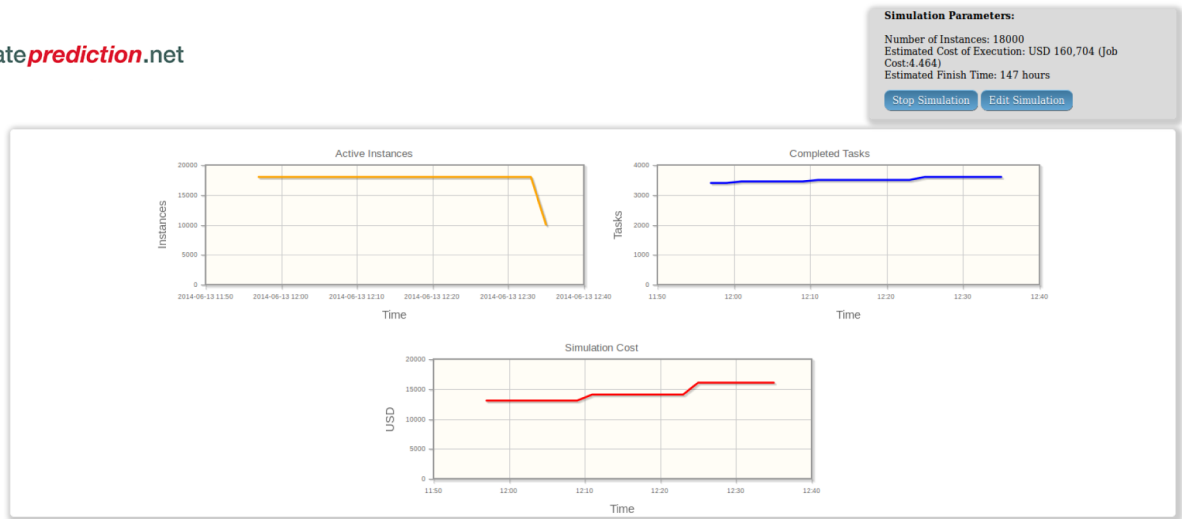
Figure 5. Dashboard.

of-the-art studies using cloud computing technologies. This project is based on demonstrated successes in the application of technologies and solutions of the type described here.

In summary, the achieved high-level objectives were to ensure that

– the client side was successfully migrated to the cloud (EC2);

– the upload server capability was configured to be redirected to AWS S3 buckets;

– different simulations were successfully run over the new infrastructure;

– a control plane (including a dashboard: front end and back end) was developed, deployed, and tested; and

– a comprehensive costing of the project and the simulation were obtained, together with metrics.

Future improvements should focus on providing more logic to the interaction with client status (such as through remote procedure calls – RPCs), allowing more metrics to be pulled from them, and creating new software as a service (a SaaS layer). From the infrastructure point of view, two main improvements are possible: first, a probe/dummy-automated execution will be needed to adjust the price to a real one before each simulation; second, full migration of the server side into the cloud, allowing the costs of data transfer and latency to be dramatically reduced.

Appendix A: Computing infrastructure design and implementation

The new computing infrastructure was built over virtualized instances (AWS EC2). Amazon provides also autoscaling groups that allow the user to define policies to dynamically add or remove instances triggered by a defined metric or alarm. As the purposes of this work are to use the rationalization of the resources and to have full control over them (via the central system), as well as any type of load balancing or failover, this feature will not be used in the cloud side but in the control system node that serves as back end for the dashboard.

After tasks have been setup in the server side and are ready to be sent to the clients (this can be currently checked in the public URL http://climateapps2.oerc.ox.ac.uk/cpdnboinc/server_status.html), the new workflow for a project/model execution is as follows:

1. The (project) administrator user configures and launches a new simulation via the dashboard.

2. The required number of instances are created based on a given template that contains a parametrized image of GNU/Linux with a configured BOINC client.

3. Every instance connects to the server and fetches two tasks (one per CPU, as the used instances have two CPUs).

4. When a task is processed, the data will be returned to the server, and also stored in a shared storage so they will be accessible for a given set of authorized users.

5. Once there are no more tasks available, the control node will shut down the instances.

Table A1. Parameters for the template instances.

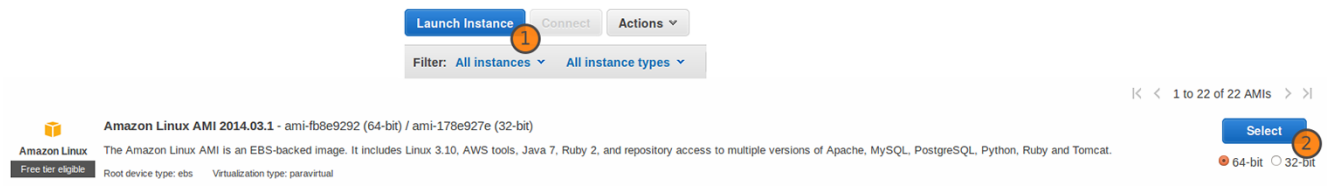|	
OS image (AMI)	Amazon Linux AMI 2014.03.1 (64 bit)
Instance type	m1.large
Firewall (security groups)	Inbound: only SSH (22) accepted Outbound: everything accepted
Persistent storage	Root 16 GB (volume type: standard)

It should be noted that, at any point, the administrator will be able to have real-time data about the execution (metrics, costs, etc.) as well as be able to change the running parameters and apply them over the infrastructure.
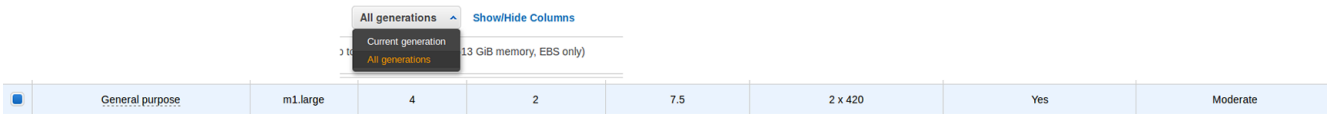
A1 Template instance creation

In order to be able to create a homogeneous infrastructure, the first step is to create an (EC2) instance that can be used as template for the other instances.

The high-level steps to follow to get a template instance (with the parameters defined in Table A1) are provided on the next page.

1. On the AWS dashboard, click "Launch Instance", then select the given OS image (AMI) type.

2. Select the image type.

3. Revise and set the parameters.

4. Launch the template instance.

Note that one should remember to create a new keypair (public–private key used for password-less SSH access to the instances) and save it (it will be used for the central system), or use another one that already exists and is currently accessible. Because of the limited space in this article, the line length (new line) has been truncated with \; please consider this when running any command described in here.

A1.1 Installing and testing AWS and EC2 command-line interface

Prerequisites include wget, unzip, and Python 2.7.x.

This step is optional, but it is highly recommendable because this will be the advanced control of the infrastructure through the shell. The following description applies and has been tested on Ubuntu 14.04 Canonical Ltd. (2014), but can be reproduced into any GNU/Linux system.

First, create an "Access Key" (and secret and/or password), via the AWS web interface in the "Security Credentials" section. With these data, the "AWS_ACCESS_KEY" and "AWS_SECRET_KEY" variables should be exported/updated; please have in mind that this mechanism will be also used for the dashboard/metrics application.

```
$ echo "export AWS_ACCESS_KEY = \
<your-aws-access-key-id>" \
>> $HOME/.bashrc
$ echo "export AWS_SECRET_KEY = \
<your-aws-secret-key>" \
>> $HOME/.bashrc

$ source $HOME/.bashrc

#Download and install the AWS CLI
$ wget https://s3.amazonaws.com/\
aws-cli/awscli-bundle.zip &&\
unzip awscli-bundle.zip &&\
sudo ./awscli-bundle/install -i \
/usr/local/aws -b /usr/local/bin/aws

#Download EC2 API tools
wget http://s3.amazonaws.com/\
ec2-downloads/ec2-api-tools.zip &&\

sudo mkdir /usr/local/ec2 &&\
sudo unzip ec2-api-tools.zip -d \
/usr/local/ec2

#Remember to update the
#PATH=$PATH:/usr/local/ec2/\
#ec2-api-tools-<API_VERSION>/bin
$ echo "export JAVA_HOME=/usr/lib/jvm\
/java-6-openjdk-amd64/" >>
$HOME/.bashrc

$ source $HOME/.bashrc
```

A1.2 Installing BOINC and its dependencies

The project executes both 32 and 64 bit binaries for the simulation, so once the template instance is running, the needed packages and dependencies need to be installed via

```
$ sudo yum install expat.i686 flac.i686 \
fontconfig.i686 freetype.i686 gamin.i686 \
glib2.i686 glibc.i686gnutls.i686 \
gtk2.i686 libX11.i686 libXau.i686 \
libXext.i686 libXfixes.i686 libXft.i686 \
libXi.i686 libcom_err.i686 libgcc.i686 \
libgcrypt.i686 libgpg-error.i686 \
libstdc++.i686 libxcb.i686 xcb-util.i686 \
zlib.i686 libcurl.i686 openssl097a.i686
```

The version of BOINC used will be the latest from git (Torvalds, 2014). To download and compile it, the following must be done:

```
#Install needed development tools
$ sudo yum groupinstall
'Development Tools'

$ sudo yum install git libcurl \
libcurl-devel openssl097a.i686 \
openssl-devel

#Fetch BOINC source code, compile and
#install the client
$ git clone \
git://boinc.berkeley.edu/boinc-v2.git \
boinc && cd boinc && ./_autosetup \
&& ./configure --disable-server \
--disable-manager --enable-optimize \
&& make \
&& sudo make install

#Create user and set permissions and
#ownership
#(in case that we want a boinc user)
$ sudo adduser boinc
$ sudo chown boinc /var/lib/boinc
```

Once the BOINC client is installed, it must be configured so it will automatically run on every instance with the same parameters.

1. Create a new account in the project.

```
$ boinccmd -create_account \
climateprediction.net <EMAIL> \
<PASSWORD> <NAME>
```

2. With the account created (or if already done), the client needs to be associated to the project by creating a configuration file with the user token.

```
$ boinccmd -lookup_account \
climateprediction.net <EMAIL> \
<PASSWORD>| grep "account key" \
| sed 's/\(.*\): \(.*\)/\2/g' \
| xargs boinccmd -project_attach \
climateprediction.net

#Status check
$ boinccmd -get_state
```

3. Make BOINC to start with the system (ec2 user will be used because of permissions).

```
$sudo echo 'su - ec2-user -c \

"cd /home/ec2-user/boinc-client/\
bin/ && ./boinc -daemon"'\
>> /etc/rc.local
```

A1.3 Simulation terminator

An essential piece of software, developed for this work, is the "simulation terminator", which decides if a node should shut down itself in the event that work units were not processed for a given amount of time (by default 6 h, via Cron), or there were no jobs waiting on the server.

This application will be provided upon request to the authors.

To install it (by default into /opt/climateprediction/), the following must be done:

```
  $ sudo ./installClient
  >> Simulation Client succesfully installed!
```

When an instance is powered off, it will be terminated (destroyed) by the Reaper service that runs in the central control system.

### A2  Contextualization

Now that the template instance is ready, this means that all the parameters have been configured and the BOINC client is ready to start processing tasks; the next stage is to contextualize it. This means that an OS image will be created from it, which will give our infrastructure the capacity of being scalable by creating new instances from this new image. Unfortunately, this part is strongly related to the cloud type, and although it can be replicated into another system, by now it will only explicitly work in this way for AWS.

The steps to follow:

1. On the instances list (AWS dashboard), select the template instance, right click and select "Create Image",
with the name of the image "CLIMATE_PREDICTION_TEMPLATE". This will create a disk image that can be used for
a full instance template (AMI):
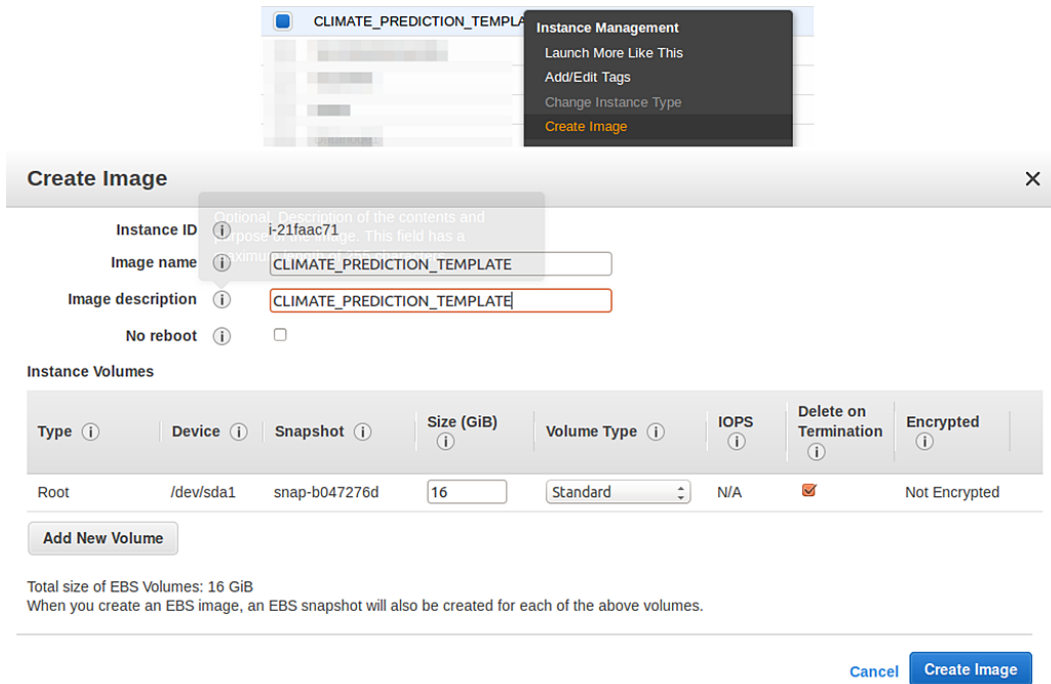


2. To finally create the instance image (in the AWS dashboard), go to Images→AMIs and right click on
''CLIMATE_PREDICTION_TEMPLATE" and fill the parameters, use the name "CLIMATE_PREDICTION_TEMPLATE"
(the same as image, for better identification) and match the kernel image (AKI) with the original template instance
(currently: aki-919dcaf8). This step is very important; otherwise, the new instances created for the project simulations
will not boot correctly.
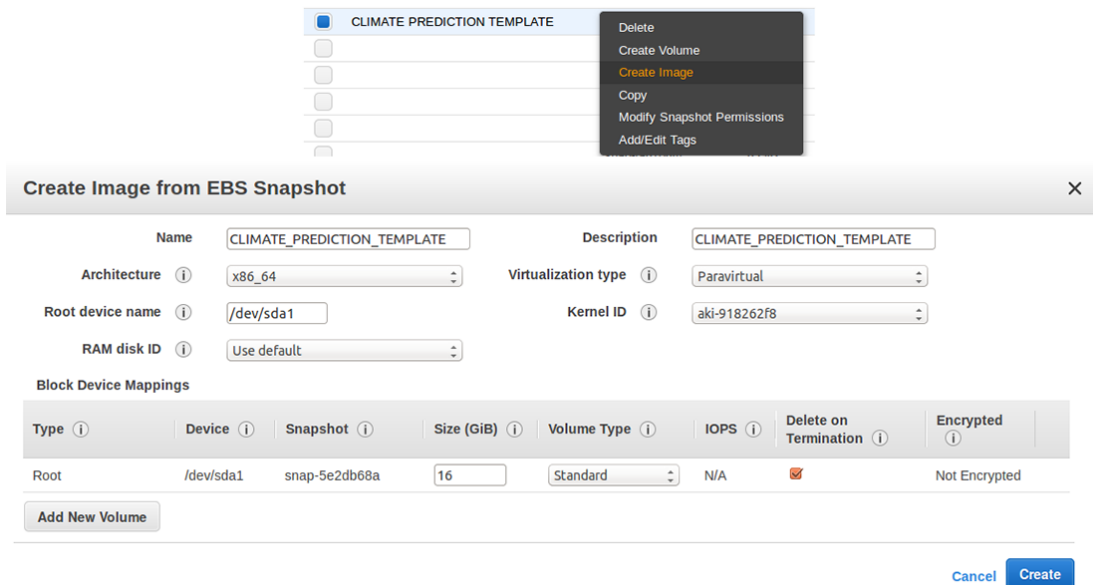


At this point, the computing infrastructure is ready to be deployed and scaled; this will be done through the dashboard.

## Appendix B: Upload server

Once a client has processed a work unit, the task (result) is created and sent to the defined upload server, that for the CPDN is http://upload2.cpdn.org/cgi-bin/file_upload_handler. This needs to be done in a transparent way for the clients and without modifying the server because we do not want to affect the actual running experiments (but in the future the servers should distribute a configuration that directly points to the S3 bucket). To do this, the data should be intercepted, and this can be done in two steps/components:

– **The name resolution** should be faked by changing the CNAME http://upload2.cpdn.org/cgi-bin/file_upload_handler point to the created S3 bucket endpoint. Bind documentation can be reviewed for this in howtoforge.com (2010).

– **A web server must serve as the endpoint**, with HTTP and HTTPs support, configured to resolve the URL http://$UPLOAD_SERVER/cpdn_cgi/ (where "$UPLOAD_SERVER" is the address of the S3 endpoint, because the jobs are created to target this URL). To simplify this stage, the storage provided by AWS (S3) will be used because it has a simple HTTP(s) server that supports all the required HTTP methods (GET and POST). The (expected) content of the "file_upload_handler" must be

```
<data_server_reply>
    <status>1</status>
</data_server_reply>
```

1. Access to the S3 service from the AWS dashboard.

S3
Scalable Storage in the Cloud

2. Click on "Create" bucket; the name should be "CLIMATE_PREDICTION" and must be in the same region as the instances.

3. Activate (in the options) the HTTP/HTTPs server.
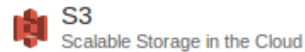
4. To secure the bucket, remember to modify the policy so only allowed IP ranges can access it (in this case, only IP ranges from instances and from CPDN servers).

```
{
 "Version": "2012-10-17",
 "Id": "S3PolicyId1",
  "Statement": [
   {
   "Sid": "IPAllow",
   "Effect": "Allow",
   "Principal": "*",
   "Action": "s3:*",
   "Resource":
    "arn:aws:s3:::CLIMATE_PREDICTION/*",

   "Condition" : {
    "IpAddress" : {
     "aws:SourceIp": "ALLOWED_IP_RANGES"
                    }
                   }
   } ]
}
```

Appendix C: Central control system and dashboard

C1 Back end and API

The back end of the central system consists of

- a RESTful (representational state transfer) API over Flask (a Python web microframework; Grinberg, 2013) that controls the infrastructure (with Boto, a Python interface to Amazon Web Services; Garnaat, 2010);

- a "simple scheduler" that will be in the background and will take care that the simulation is running with the given parameters (e.g. all the required instances are up); and

- the "Reaper", a subsystem of the simple scheduler that is some sort of garbage collector and will terminate powered-off instances in order to release resources.

C1.1 API reference

The back end can be reused and integrated into another system in order to give the full abstraction over the project. The available requests (HTTP) are

- **Get simulation status:**
  Call: *status*
  Request Type *GET*
  Returns: *JSON* object with current simulation status:

```
{
 requestedInstances:
  <int:REQUESTED_INSTANCES>

 executionCost:
  <float:EXECUTION_COST_IN_USD>,

 workunitCost:
  <float:WORKUNIT_COST_IN_USD>,
}
```

- **Get metric:**
  Call: *metric/METRICNAME*
  Request Type *GET*
  Returns: *JSON* object with metric (time series):

```
{
 <METRIC_VARIABLE>:
  [{<int|float:METRIC_DATA>,
    <timestamp:TIMESTAMP>}],
}
```

- **Set/modify simulation:**
  Call: *simulation*
  Request Type *POST*
  Input: *JSON* object with simulation parameters, if already running will modify it:
  Returns: *JSON* object with result (0=fail, 1=successful):

```
{
 instances: <int:REQUESTED_INSTANCES>

}
```

```
{
 result: <0|1>
}
```

- **Stop simulation:**
  Call: *simulation/stop*
  Request Type *GET*
  Returns: *JSON* object with result (0=fail, 1=successful):

```
{
 result: <0|1>
}
```

A simplistic (but functional) GUI (graphical user interface) has been designed to make the execution of the simulation on the cloud more understandable.

Two control actions are available:

- "Start/edit simulation" sets the parameters (cloud type, number of instances, etc.) of the simulation and runs it.

– "Stop simulation" forces all the instances to terminate.

There are three default metrics (default time lapse: 6 h):

– Active instances are the number of active instances.

– Completed tasks are the number of work units successfully completed.

– Simulation cost is the accumulated cost for the simulation.

C2　Installation and configuration

The applications are intended to run at any GNU/Linux. The only requirements are (apart from Python 2.7) Flask and Boto, that can be easily installed into any GNU/Linux:

```
$ pip install flask virtualenv \
  boto pysqlite daemonize
```

C2.1　First configuration and run

For this step, the file `controlSystem.tar.gz`, which contains all the software and configurations for the central system, needs to be uncompressed into /opt/climateprediction/; then just

```
$ cd /opt/climateprediction
$ ./firstRun.py
>> Connector Type? aws
>> AWS Key: \
    <TYPE_YOUR_AWS_KEY>
>> AWS Secret: \
    <TYPE_YOUR_AWS_KEY_SECRET>

>> Dashboard Username: \
    <TYPE_DASHBOARD_USERNAME>
>> Dashboard Password: \
    <TYPE_DASHBOARD_PASSWORD>

[OK] Central System Ready to Run!
    Type ./run.py to start.

#Start the service...
$ ./run.py

#... and Central System starts
#resolving backend and frontend requests
```

Optionally, the configuration can be set manually by editing the file "Config.cfg" (parameters in < >):

```
#Main Configuration
[main]
connector=<CLOUD_CONNECTOR>
pollingTime=<REFRESH_TIME_IN_SECONDS>

[HTTPAuth]
user=<DASHBOARD_USERNAME>
```

```
password=<DASHBOARD_PASSWORD>

#AWS Credentials Configuration
[AWSCredentials]
KEY=<AWS_KEY>
PASSWORD=<AWS_SECRET>

#AWS Connector Configuration
[AWS]
AMI=<AMI>
instanceType=<INSTANCE_TYPE>
securityGroup=CLIMATE_PREDICTION
keyPair=<CLIMATE_PREDICTION_KEYPAIR>
```

C3　Use and project deployment

Now that the central system has been installed and configured, it will be listening and accepting connections into any network interface (0.0.0.0) on port 5000, protocol HTTP, so it can be accessed via web browser. Firefox or Chromium are recommended because of Javascript compatibility.

C3.1　Launch a new simulation

When starting a simulation, the number of instances will be 0. This can be changed by clicking "Edit Simulation", setting the number into the input box, and clicking on "Apply Changes". Within some minutes (defined in the configuration file, in the "pollingTime" variable), the system will start to deploy instances (workers).
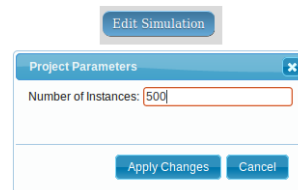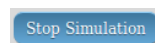
C3.2　Modify a simulation

If the number of instances needs to be adjusted when a simulation is running, the procedure is the same as launching a new simulation ("Edit Simulation"). Please be aware that if the number of instances is reduced, unfinished work units will be lost (the scheduler will stop and terminate them using a FIFO).

C3.3　End current simulation

When a simulation wants to be stopped, click "Stop Simulation". This will reduce the number of instances to 0, copy the database as "SIMULATION-TIMESTAMP" for further analysis, and reset all the parameters and metrics.

References

Allen, M.: Do-it-yourself climate prediction, Nature, 401, 642, doi:10.1038/44266, 1999.

Añel, J. A.: The importance of reviewing the code, Commun. ACM, 54, 40–41, doi:10.1145/1941487.1941502, 2011.

Añel, J. A., López-Moreno, J. I., Otto, F. E. L., Vicente-Serrano, S., Schaller, N., Massey, N., Buisán, S., and Allen, M. R.: The extreme snow accumulation in the western Spanish Pyrenees during winter and spring 2013, B. Am. Meterol. Soc., 95, S73–S76, 2014.

Anderson, D. P.: Boinc: A system for public-resource computing and storage, in: 5th IEEE/ACM International Workshop on Grid Computing, GRID 2004, Pittsburgh, USA, 8 November 2004, IEEE Computer Society Washington, DC, USA, 4–10, doi:10.1109/GRID.2004.14, 2004.

AWS: S3 Princing, available at: https://aws.amazon.com/s3/pricing/ (last access: 22 December 2016), 2016a.

AWS: AWS Offers Data Egress Discount to Researchers, available at: https://aws.amazon.com/blogs/publicsector/aws-offers-data-egress-discount-to-researchers/ (last access: 22 December 2016), 2016b.

AWS: Glacier, available at: https://aws.amazon.com/glacier/(last access: 22 December 2016), 2016c.

Black, M. T., Karoly, D. J., Rosier, S. M., Dean, S. M., King, A. D., Massey, N. R., Sparrow, S. N., Bowery, A., Wallom, D., Jones, R. G., Otto, F. E. L., and Allen, M. R.: The weather@home regional climate modelling project for Australia and New Zealand, Geosci. Model Dev., 9, 3161–3176, doi:10.5194/gmd-9-3161-2016, 2016.

BOINC: Berkeley Open Infrastructure for Network Computing, available at: http://boinc.berkeley.edu/ (last access: 19 June 2014), 2014.

Canonical Ltd.: Ubuntu, available at: http://www.ubuntu.com (last access: 19 June 2014), 2014.

CPDN: ClimatePrediction.net, http://www.climateprediction.net (last access: 3 November 2015), 2015.

Garnaat, M.: boto: A Python interface to Amazon Web Services, available at: http://boto.readthedocs.org/en/latest/, last access: 3 November 2015, 2010.

Gordon, C., Cooper, C., Senior, C. A., Banks, H., Gregory, J. M., Johns, T. C., Mitchell, J. F. B., and Wood, R. A.: The simulation of SST, sea ice extents and ocean heat transports in a version of the Hadley Centre coupled model without flux adjustments, Clim. Dynam., 16, 147–168, doi:10.1007/s003820050010, 2000.

Grinberg, M.: Designing a RESTful API with Python and Flask, available at: http://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask, last access: 3 November 2015, 2013.

howtoforge.com: BIND Installation On CentOS, available at: http://www.howtoforge.com/bind-installation-on-centos, last access: 3 November 2015, 2010.

Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. H.: Performance analysis of cloud computing services for many-tasks scientific computing, IEEE T. Parall. Distr., 22, 931–945, 2011.

Massey, N., Jones, R., Otto, F. E. L., Aina, T., Wilson, S., Murphy, J. M., Hassell, D., Yamazaki, Y. H., and Allen, M. R.: weather@home–development and validation of a very large ensemble modelling system for probabilistic event attribution, Q. J. Roy. Meteor. Soc., 141, 1528–1545, doi:10.1002/qj.2455, 2015.

Microsoft: Azure Research Awards, available at: https://blogs.msdn.microsoft.com/azure (last access: 19 June 2014), 2014.

Montes, D.: climateprediction.net: A Cloudy Approach, Master thesis, High Performance Computing Masters, University of Santiago de Compostela, Spain, 2014.

Pope, D. V., Gallani, M. L., Rowntree, P. R., and Stratton, R. A.: The impact of new physical parametrizations in the Hadley Centre climate model: HadAM3, Clim. Dynam., 16, 123–146, 2000.

Raicu, I., Foster, I. T., and Zhao, Y.: Many-task computing for grids and supercomputers, in: 2008 Workshop on Many-Task Computing on Grids and Supercomputers, MTAGS 2008, Austin, TX, 17–17 November 2008, IEEE, 11 pp., doi:10.1109/MTAGS.2008.4777912, 2008.

Ries, C. B., Schröder, C., and Grout, V.: Approach of a UML profile for Berkeley Open Infrastructure for network computing (BOINC), 2011 IEEE International Conference on Computer Applications and Industrial Electronics (ICCAEI), Penang, 4–7 December 2011, IEEE, 483–488, doi:10.1109/ICCAIE.2011.6162183, 2011.

Schaller, N., Otto, F. E. L., van Oldenborgh, G. J., Massey, N. R., Sparrow, S., and Allen, M. R.: The heavy precipitation event of May-June 2013 in the upper Danube and Elbe basins, B. Am. Meteorol. Soc., 95, S69–S72, 2014.

Schaller, N., Kay, A. L., Lamb, R., Massey, N. R., van Oldenborgh, G. J., Otto, F. E., Sparrow, S. N., Vautard, R., Yiou, P., Ashpole, I., Bowery, A., Crooks, S. M., Haustein, K., Huntingford, C., Ingram, W. J., Jones, R. G., Legg, T., Miller, J., Skeggs, J., Wallom, D., Weisheimer, A., Wilson, S., Stott, P. A., and Allen, M. R.: Human influence on climate in the 2014 southern England winter floods and their impacts, Nature Climate Change, 6, 627–634, doi:10.1038/nclimate2927, 2016.

Torvalds, L.: Git: free and open source distributed version control system, http://www.git-scm.com, last access: 19 June 2014, 2015,

Uhe, P., Otto, F. E. L., Rashid, M. M., and Wallom, D. C. H.: Utilising Amazon Web Services to provide an on demand urgent computing facility for climateprediction.net, in: Proceedings of the 2016 IEEE 12th International Conference on e-Science, IEEE, 1–7, 2016.

Zhao, D., Yang, X., Sadooghi, I., Garzoglio, G., Timm, S., and Raicu, I.: High-Performance Storage Support for Scientific Applications on the Cloud, in: ScienceCloud '15 Proceedings of the 6th Workshop on Scientific Cloud Computing, ACM, 33–36, 2015.