

UPPSALA UNIVERSITY

PROJECT IN COMPUTATIONAL SCIENCE

1TD316

Towards Moving Scientific Applications in the Cloud

Authors:

Oscar Möller

oscar.moller.7838@student.uu.se

Saim Mehmood

saim.mehmood.7275@student.uu.se

Supervisors:

Maya Neytcheva

maya.neytcheva@it.uu.se

Salman Toor

salman.toor@it.uu.se

Ali Dorostkar

ali.dorostkar@it.uu.se

Behrang Mahjani

behrang.mahjani@it.uu.se

February 16, 2016



UPPSALA
UNIVERSITET

Contents

1	Introduction and Project Aims	2
1.1	Traditional High Performance Computing (HPC)	2
1.2	Cloud Computing	2
1.2.1	Advantages	3
1.2.2	Disadvantages	4
1.3	Aims	4
2	Description of the Cloud Framework	4
2.1	Technologies Used	4
2.2	System Architecture	5
2.3	Cloud as a Service for QTL Analysis	7
2.4	Service Automation	7
2.5	Code Snippets	8
3	Cloud for Numerical Simulations - Performance Analysis	9
3.1	SNIC Science Cloud	10
3.2	Test problem	11
4	Results	12
4.1	Scale Up	12
4.2	Scale Out (shared virtual vs bare-metal)	13
4.3	Scale Out (dedicated virtual vs bare-metal)	14
4.4	Inter vs. intra communication	14
4.5	Parallel Efficiency	17
5	Conclusion and Future Work	18
6	References	18

Abstract

This report discusses a framework, designed to run scientific applications in cloud environment. And also to quantify performance overhead when solving a numerical experiment using cloud. Cloud computing provides usability, scalability and on demand availability of computational and storage resources, remotely. These are the characteristics required by scientific applications and that's why we are using it. This project has two dimensions. First one addresses the benefits of cloud infrastructure for end users. In the second portion, we are trying to do performance analysis. We have designed an architecture in cloud that help scientists run their applications elastically. To evaluate performance, we ran already existing MPI code on cloud.

The computations were performed on resources provided by SNIC through Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) under Project c2015007.

1 Introduction and Project Aims

Scientific applications require the availability of a massive number of computers for performing large scale experiments. Traditionally, these needs have been addressed by using high-performance computing solutions and installed facilities such as clusters and super computers, which are difficult to setup, maintain and operate. Cloud computing provides a completely new model of utilizing the computing infrastructure, namely compute resources, storage resources and as well as software services.

In the current project, we deal with a specific scientific application i.e. quantitative traits loci (QTL) analysis and provided it as application as a service. Our project has two dimensions:

1. Application as a service (QTL as a service)
2. Performance Analysis of scientific applications

For application as a service, we have designed a framework inside cloud and for performance analysis we have solved a numerical experiment using Deal.ii [7].

1.1 Traditional High Performance Computing (HPC)

The term HPC refers to any computational activity requiring more than a single computer to execute a task. When we talk about performance, we are referring to a desired amount of computational throughput, given a set of time and resources. It's an art of HPC to squeeze as many FLOPS (CPU performance) and IOPS (disk performance) out of a HPC system as possible. As a result, the performance of the aggregate system is high (high bandwidth interconnects, high disk throughput), but it generally costs a lot of money. When you want to get something done, you ask for a set amount of resource for a set amount of time. Same service for everyone. Same CPUs, same disk. One item on the menu. And an upper limit that will take another 3 or 4 years to change. Whereas, cloud computing offers scalability.

1.2 Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable resources e.g. networks, servers, storage, applications, and services that

can be rapidly provisioned and released with minimal management effort or service provider interaction. [7]

Cloud computing is a type of computing that relies on sharing computing resources rather than having local servers or personal devices to handle applications. Where cloud truly differentiates itself from the HPC is that cloud is an Infrastructure as a Service (IaaS). That means a user can pick and choose what 'unit' of compute he wants, what performance he wants and how many of them he want.

1.2.1 Advantages

Cloud computing offer lots of benefits over traditional high performance computing capabilities made available using clusters and grid computing. Following are the benefits of using cloud:

1. On-demand availability

Cloud computing provides the facility of on-demand availability of resources i.e. resources are provided on *as-needed* and *when-needed* basis. This makes it possible for small organizations or individuals to purchase cloud resources of storage, computing or applications on the go and pay according to the actual usage, instead of purchasing the expensive hardware resources themselves.

2. Scalability

The ability to scale on demand is one of the biggest advantages of cloud computing. Often, when considering the range of benefits of cloud, it is difficult to conceptualize the power of scaling on-demand, but organizations enjoy tremendous benefits when they implement auto scaling. Scalability makes it easier for even small organizations to deal with situations such as high traffic flow towards their services. Again, one pays only for the resources that were being consumed.

3. User Convenience

Cloud Computing has become a convenient and cost efficient way for companies to store data while using remote, shared servers located in the cloud. In case of the cloud infrastructure we have designed, it offer scientists the convenience of performing extensive computations from their own local machines without even worrying about computation power it required to run their applications.

4. Reliability

With a managed service platform, cloud computing is much more reliable and consistent than in-house IT infrastructure. Most providers offer a Service Level Agreement which guarantees 24/7/365 and 99.9% availability. An organization can benefit from a massive pool of redundant IT resources, as well as quick failover mechanism - if a server fails, hosted applications and services can easily be transited to any of the available servers.

1.2.2 Disadvantages

The ability of cloud computing to provide scalability, on-demand availability and reliability comes with a cost. There are some pitfalls of using cloud as well:

1. **Security**

Although cloud service providers implement the best security standards, but using cloud-powered technologies means you need to give your service provider with access to important data. Meanwhile, cloud being a public service opens it up to security challenges on a routine basis. The ease in procuring and accessing cloud services can also give nefarious users the ability to scan, identify and exploit loopholes and vulnerabilities. For instance, in a multi-tenant architecture where multiple users are hosted on the same server, a hacker might try to break into the data of other users hosted and stored on the same server. However, such exploits and loopholes are not likely to surface, and the likelihood of a compromise is not great.

2. **Vendor Lock-in**

Although cloud providers promise that the cloud will be flexible to use and integrate. Switching cloud services is something that hasn't yet completely evolved. Organizations may find it difficult to migrate their services from one vendor to another. Hosting and integrating current cloud applications on another platform, may throw up interoperability and support issues.

3. **Limited Control**

Since the cloud infrastructure is entirely owned, managed and monitored by the service provider, it transfers minimal control over to the customer. The customer can only control and manage the applications, data and services operated on top of that, not the backend infrastructure itself. Key administrative tasks such as server shell access, updating and firmware management may not be passed to customer or end user.

1.3 Aims

To illustrate two aspects of cloud computing:

1. Enabling a cloud based application as a service to an end user.
2. Study of the suitability of cloud computing for running large scale computationally demanding numerical simulations.

2 Description of the Cloud Framework

2.1 Technologies Used

1. **Cloud Platform:**

Cloud computing services can be offered in three categories:

- (a) **SaaS:** stands for Software as a Service. It refers to applications that are designed for end-users and delivered over the web.

- (b) **PaaS:** stands for Platform as a Service. It is the set of tools and services designed to make coding and deploying those applications quick and efficient.
- (c) **IaaS:** stands for Infrastructure as a Service. Denotes the hardware and software that powers it all - servers, storage, networks, operating systems.

- i. **OpenStack:**

The SNIC cloud [8] we have used for this project is based on OpenStack cloud suite. OpenStack is a free and open-source software platform for cloud computing, mostly deployed as an infrastructure-as-a-service (IaaS). The software platform consists of inter-related components that control hardware pools of processing, storage, and networking resources throughout a data center. Users either manage it through web-based dashboard, through command-line tools, or through a RESTful API (application programming interface) [1].

2. R language:

We have used R statistical software to interact with the user. R is familiar environment to many geneticists. The power of R compared to more specialized environments is the ability for the end user to freely adapt and extend the methods and work flows [2].

3. Apache Spark:

Apache Spark is an open source cluster computing framework. In contrast to Hadoops two-stage disk-based MapReduce paradigm, Sparks multi-stage in-memory primitives provides performance up to 100 times faster for certain applications. By allowing user programs to load data into a cluster memory and query it repeatedly. Spark is well suited to machine learning algorithms [3].

4. Jupyter Notebook:

The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Uses include: data cleaning and transformation, numerical simulations, statistical modeling and machine learning. The best thing about Jupyter notebook is that, it supports up to 40 languages [4].

2.2 System Architecture

Before digging into the details of cloud framework we have designed, we briefly describe the types of cloud models available in the market i.e. Public, Private, Hybrid and Community clouds.

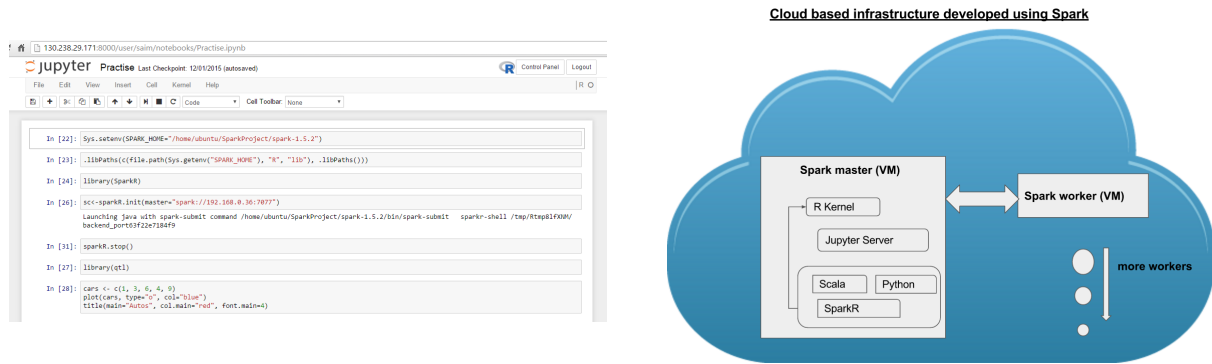
1. Cloud Models

- (a) **Public Cloud** In a public cloud, the computing infrastructure is hosted by the cloud vendor at the vendor premises. The customer has no visibility and control over where the computing infrastructure is hosted. The computing infrastructure is shared between all users (organizations, private individuals etc.)
- (b) **Private Cloud** This computing infrastructure is dedicated to a particular organization and not shared with other organizations. Private clouds are of two types:
 - i. **On-premise private clouds:** cloud facilities located in the premises of a particular organization.

- ii. **Externally hosted private clouds:** hosted by a third party specialized in providing cloud infrastructure.
- (c) **Hybrid Cloud** Organizations may host critical applications on private clouds and applications with relatively less security concerns on public cloud. The combined private and public clouds together is called a hybrid cloud.
- (d) **Community Cloud** Community cloud involves sharing of computing infrastructure in between organizations of the same community. The SNIC Science Cloud used for this project is also a community cloud for Swedish universities. And is based on OpenStack cloud suite.

2. Framework

Using already existing technologies , i.e. , R language, Apache Spark, SparkR, Jupyter notebook and OpenStack cloud infrastructure, we have designed a framework that help biologists run their QTL code on cloud. The ability of Jupyter notebook to support 40 different languages and easy-to-use interface, makes it a perfect tool for scientists. Enabling users with different backgrounds to tune their software and access the cloud resources without the requirement to have a detailed knowledge on how it works. Using Apache Spark, we have created a computational infrastructure inside cloud.



On the front end, user is supposed to specify the type of scientific application and the number of workers required. User will get an IP address and using it inside browser, will give access to Jupyter notebook. The above framework is deployed on the concept of Master-Slave pattern. Spark master automatically distributes workload on the respective workers. The advantage of using Spark is its capability to run all computations inside memory; if the application is small. Or running some portions on memory and keeping some on the disk; if application size is larger than the memory.

3. Spark Master

Apache Spark provides high-level APIs in Java, Scala, Python and R. R Kernel inside Spark master deals with the user instructions from his local machine. It is also linked with SparkR; an R package that provides a light-weight frontend to use Apache Spark from R. Using Spark's distributed computation engine allows us to run large scale data analysis from the R shell. The Jupyter Server is linked with Jupyter notebook and provides all the necessary support for dealing with the instructions from notebook.

4. Software Stack of SNIC cloud

The cloud that is been used throughout this project, is based on the OpenStack software. OpenStack provides tools for managing compute, storage and networking resources. It offers a Python Application Programming Interface (API) that is used for communication with the cloud and allow users to execute operations inside the cloud. When a VM (Virtual machine) is created, user has the option to choose between different flavors. Flavors denotes for example the amount of RAM, the number of CPU cores and the amount of available disk space that will be used with a chosen flavor. There are five different flavor types and the flavor that is used in this project is called m1.medium and is using 4GB (gigabyte) of RAM, 2 CPU cores and 40GB of disk space [5].

5. Front End

The front end of this project is based on Jupyter Notebook and it is worth mentioning here, that it is completely independent of cloud infrastructure that we have designed. It means that the user of our application does not have to deal with the complexities of cloud.

2.3 Cloud as a Service for QTL Analysis

A quantitative trait is a phenotype or organism characteristic with continuous measurement, such as product yield and quality in agriculture species or risk factors for disease in animal and human populations. It is usually complex in that it is influenced by the actions and interactions of many genes and environmental factors and geneticists are interested in identifying and understanding the role of the genes involved.

QTL analysis is conducted by scientists to locate regions in the genome that can be associated to quantitative traits i.e. traits where individuals in a population exhibit continuous distributions. With a powerful statistical model, efficient numerical algorithms, and a suitable environment for computational experiments, it is then possible to identify the QTL position and the corresponding statistical significance levels. Mathematically, the search for the positions of QTL corresponds to solving a multidimensional global optimization problem, where the objective function is the statistical model fit. Standard tools for multiple QTL analysis, using standard computational algorithms, are not able to cope with massive computations required. Significant development, both in terms of algorithms and implementations, is needed to provide accurate and efficient tools for simultaneous search of multiple interacting QTL. Geneticists have generally not been main users of high end computing resources. While this is changing to some extent, many groups still do not own or have access to their own computational resources. Furthermore, the need for such resources for QTL analysis is intermittent in nature. Analysis and finding a proper model will only be relevant during a specific phase of the executing of a study. We therefore propose that cloud computing is ideal for this user group.

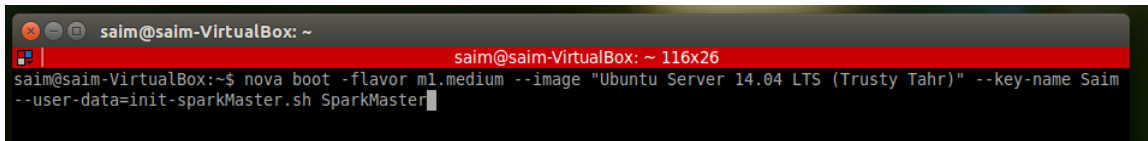
2.4 Service Automation

Next we mention a library we have used for the process of automation.

Cloud-init: is a Ubuntu package that provides boot time customization for cloud and virtualization instances. Services run early during boot, retrieves user data from an external provider and

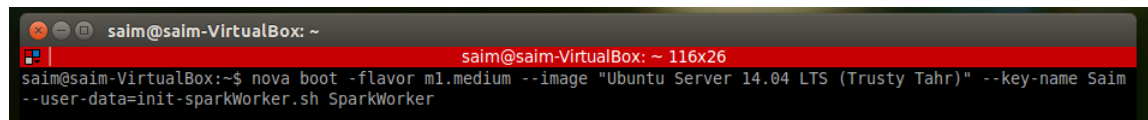
perform actions. Cloud-init's behavior can be configured via user-data i.e. `--user-data` argument.

We were able to automate the creation of Spark Master and Spark Worker.



```
saim@saim-VirtualBox: ~  
saim@saim-VirtualBox: ~ 116x26  
saim@saim-VirtualBox:~$ nova boot -flavor m1.medium --image "Ubuntu Server 14.04 LTS (Trusty Tahr)" --key-name Saim  
--user-data=init-sparkMaster.sh SparkMaster
```

All the dependencies of OpenJDK, firewall rules and variables are handled inside the code file of `init-sparkMaster.sh` and similarly to automate the spawning process of workers, we use an analogous instruction but with a different code file for Spark worker, `init-sparkWorker.sh`.



```
saim@saim-VirtualBox: ~  
saim@saim-VirtualBox: ~ 116x26  
saim@saim-VirtualBox:~$ nova boot -flavor m1.medium --image "Ubuntu Server 14.04 LTS (Trusty Tahr)" --key-name Saim  
--user-data=init-sparkWorker.sh SparkWorker
```

2.5 Code Snippets

Below we include code snippets for Spark Master and Worker creation [6]:

1. Spark Master file name: `init-sparkMaster.sh`

```
#!/bin/bash  
#  
# Cloud-init script to get Spark Master up and running  
#  
SPARK_VERSION="1.5.0"  
APACHE_MIRROR="apache.ub.no"  
LOCALNET="192.168.0.0/24"  
  
# Firewall setup  
ufw allow from $LOCALNET  
ufw allow 80/tcp  
ufw allow 443/tcp  
ufw allow 4040:4050/tcp  
ufw allow 7077/tcp  
ufw allow 8080/tcp  
  
# Dependencies  
sudo apt-get -y update  
sudo apt-get -y install openjdk-7-jdk  
  
# Download and unpack Spark  
sudo curl -o /tmp/spark-$SPARK_VERSION-bin-hadoop1.tgz  
http://$APACHE_MIRROR/spark/spark  
$SPARK_VERSION/spark-$SPARK_VERSION-bin-hadoop1.tgz  
  
sudo tar xvf -C /opt -f /tmp/spark-$SPARK_VERSION-bin-hadoop1.tgz  
  
sudo ln -s /opt/spark-$SPARK_VERSION-bin-hadoop1/ /opt/spark  
  
sudo chown -R root.root /opt/spark-$SPARK_VERSION-bin-hadoop1/*  
  
# Configure Spark master  
sudo cp /opt/spark/conf/spark-env.sh.template  
/opt/spark/conf/spark-env.sh  
  
sudo sed -i 's/# - SPARK_MASTER_OPTS.* /SPARK_MASTER_OPTS="-  
Dspark.deploy.defaultCores=4 -Dspark.executor.memory=2G" /'  
/opt/spark/conf/spark-env.sh  
  
# Make sure our hostname is resolvable by adding it to /etc/hosts  
sudo echo $(ip -o addr show dev eth0 | fgrep "inet " | egrep -o  
'[0-9.]+/[0-9]+' | cut -f1 -d/) $HOSTNAME | sudo tee -a /etc/hosts  
  
# Start Spark Master with IP address of eth0 as the address to use  
sudo /opt/spark/sbin/start-master.sh -h $(ip -o addr show dev eth0  
| fgrep "inet " | egrep -o '[0-9.]+/[0-9]+' | cut -f1 -d/)
```

2. Spark Worker file name: init-sparkWorker.sh

```
#!/bin/bash
#
# Cloud-init script to get Spark Worker up and running
#
SPARK_VERSION="1.5.0"
APACHE_MIRROR="apache.uib.no"
LOCALNET="192.168.0.0/24"
SPARK_MASTER_IP="192.168.0.36" (IP of Spark Master you've created)

# Firewall setup
ufw allow from $LOCALNET
ufw allow 8081/tcp

# Dependencies
sudo apt-get -y update
sudo apt-get -y install openjdk-7-jdk

# Download and unpack Spark
sudo curl -o /tmp/spark-$SPARK_VERSION-bin-hadoop1.tgz
http://$APACHE_MIRROR/spark/spark-$SPARK_VERSION/spark-
$SPARK_VERSION-bin-hadoop1.tgz

sudo tar xvz -C /opt -f /tmp/spark-$SPARK_VERSION-bin-hadoop1.tgz

sudo ln -s /opt/spark-$SPARK_VERSION-bin-hadoop1/ /opt/spark

sudo chown -R root.root /opt/spark-$SPARK_VERSION-bin-hadoop1/*

# Make sure our hostname is resolvable by adding it to /etc/hosts
sudo echo $(ip -o addr show dev eth0 | fgrep "inet "
| egrep -o '[0-9.]+/[0-9]+' | cut -f1 -d/) $HOSTNAME
| sudo tee -a /etc/hosts

# Start Spark worker with address of Spark master to join cluster
sudo /opt/spark/sbin/start-slave.sh spark://$SPARK_MASTER_IP:7077
```

3 Cloud for Numerical Simulations - Performance Analysis

Due to the advantages of high computing power, cheap cost of services, high performance, scalability, accessibility as well as availability, cloud computing has become a highly demanded service or utility. Companies can scale up as computing needs increase and scale down again as demands decrease. Using the cloud allows companies, for example, to avoid upfront infrastructure costs and maintenance and instead use the "pay-as-you-go" model only paying for the computing power they need at a particular moment. In the scientific world, especially in life sciences, a large number of applications are well aligned with the cloud computing model, such as permutation testing. In this second direction of our project, we are trying to quantify the overhead when solving a numerical experiment on the cloud.

Virtualization has been around almost as long as computers, and it refers to the act of creating a virtual version rather than an actual. For example, multiple virtual hard disks can be created inside an actual hard disk. In cloud computing however, the virtualization software, or hypervisor, creates the illusion that a single virtual machine (VM) is running with its own set of hardware. In reality, the underlying hardware supplies resources for multiple VMs at a time through the hypervisor. There are multiple virtualization techniques available, such as full-virtualization, paravirtualization and hardware-assisted virtualization. We are using the Kernel-based Virtual Machine (KVM) hypervisor which utilizes the full-virtualization technology, which means that a virtual machine is not aware that it is a virtual machine and such, no changes are needed in the guest operating system (OS) in each VM.

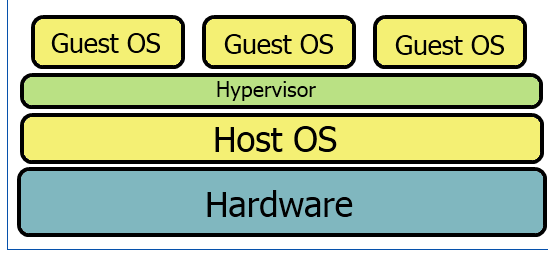


Figure 1: An illustration of a hypervisor

Multiple guest OS's can be run simultaneously using the same set of hardware; thanks to the hypervisor, as seen in Figure 1. Virtualization give users flexibility, reduction in cost, scalability and elasticity but it also comes with a performance overhead. The act of virtualization adds an extra communication layer that will impact performance. Another cause for performance degradation is consolidation. Each VM assumes it has its resources available to themselves at all times, but that is not the case. As more and more VMs are running on the same hardware, the hypervisor will have to switch the computing power between multiple machines. The goal in the performance part of the project is to quantify the performance overhead when using the virtualization technology.

Parallelizing a serial code is a great way to increase performance. Parallelizing a code means that you let multiple cores inside a computer's processor work at the same time. By increasing the performance one is able to run the same code in less amount of time, or increase the size of the problem and be able to run it in the same amount of time. However, most codes are not completely parallelizable as there are parts in the code that can not be done independently. Communication and synchronization between the cores are often the greatest bottlenecks when trying to get good parallel performance. It is also important to have an even load balance between the cores when solving tasks in parallel. Parallel computing and cloud computing go hand in hand and that is why we want to test the performance in run time when solving a highly parallel task using the cloud.

3.1 SNIC Science Cloud

The SNIC Science Cloud (SSC) with OpenStack Juno as the cloud software, provided by UPPMAX was used in all performance tests. Up to four VMs could be used at a time with eight cores per node and 16 GB of RAM. SSC is a shared virtual environment, meaning that the hardware is shared between multiple users at the same time.

Instances

Host =

<input type="checkbox"/>	Project	Host	Name	Image Name	IP Address	Size	Status	Task	Power State	Time since created	Actions
<input type="checkbox"/>	c2015007	sm7	joel-3	snaptest2	192.168.1.88 Floating IPs: 130.238.29.7	m1.xlarge	Active	None	Running	1 month	<input type="button" value="Edit instance"/> ▾
<input type="checkbox"/>	c2015003	sm7	rhino-cub-13	CoreOS	10.0.10.211	m1.medium	Active	None	Running	1 month, 1 week	<input type="button" value="Edit instance"/> ▾
<input type="checkbox"/>	c2015032	sm7	Kalyan	Rhadoop-lab2	192.168.1.84 Floating IPs: 130.238.29.152	m1.large	Active	None	Running	2 months, 3 weeks	<input type="button" value="Edit instance"/> ▾
<input type="checkbox"/>	c2015020	sm7	ubuntu-test	Ubuntu 15.10 (Wily Werewolf)	192.168.1.4 Floating IPs: 130.238.29.128	m1.small	Active	None	Running	4 months, 2 weeks	<input type="button" value="Edit instance"/> ▾

Displaying 4 items

Figure 2: Multiple users on a single node

One example is figure 2 where four separate instances have been created on the same host. If every user would be running intensive work at the same time, performance would suffer. This is something we have taken into account when we evaluate the performance. Also, SSC is currently not a production grade system. The underlying virtualisation layer is based on a non-optimised version of the KVM hypervisor with resource overcommit.

We also ran some tests using a dedicated virtual environment. In a dedicated virtual environment, users are in full control on which applications are running on the hardware. This allowed us to run our experiments by ourselves, with no other application running in parallel.

A bare-metal machine was also available where each node has the same specifications as each VM. By comparing the performance on the cloud versus bare-metal we are able to quantify the performance overhead caused by the added communication layer with the hypervisor in the cloud.

3.2 Test problem

To test the performance we ran already existing parallel code that solves the Laplace equation

$$\begin{aligned} -\Delta u &= f & \text{in } \Omega &= [0, 1]^2 \\ u &= 0 & \text{on } \partial\Omega, \end{aligned}$$

with f defined as

$$f(x, y) = \begin{cases} 1 & \text{if } y < \frac{1}{2} + \frac{1}{4} \sin(4\pi x) \\ -1 & \text{otherwise.} \end{cases}$$

The Laplace equation is discretized using finite elements and solved by the Algebraic Multigrid method (AMG) [13]. The open source scientific library Deal.II [9] is used for the simulation. Deal.II

provides the mesh generation and the discretization. For comparison purposes, two different packages are used for the AMG solver namely PETSc [11] and Trilinos [12]. The code is then parallelized using MPI. The behaviour of the solution depends on the right hand side of the equation and the solution has a singularity along the sinusoidal line flowing through the domain. This behavior is unwanted in this case when our aim is to test the performance on the cloud. We therefore made a small change in the code and removed all singularities and replaced it with a constant right hand side. Now, the solution will be continuous throughout the domain and will improve the load balance between the cores. If not stated otherwise, all experiments were run on a problem size of 16M degrees of freedom.

All experiments are performed on both the cloud and on bare-metal. We would like to see how well the run time scales with an increasing number of cores. Another interesting thing to look at when trying to quantify the performance overhead caused by the hypervisor is to look at the inter versus intra communication. Which is the communication between the virtual machines versus the communication between the virtual cores inside each virtual machine. In the graphs, we use the standard metrics speedup $S(p)$ and parallel efficiency $E(p)$, defined as

$$S(p) = \frac{t_1}{t_p},$$

$$E(p) = \frac{t_1}{pt_p},$$

where t_1 is the run time for one core and t_p is the run time for p number of cores.

4 Results

4.1 Scale Up

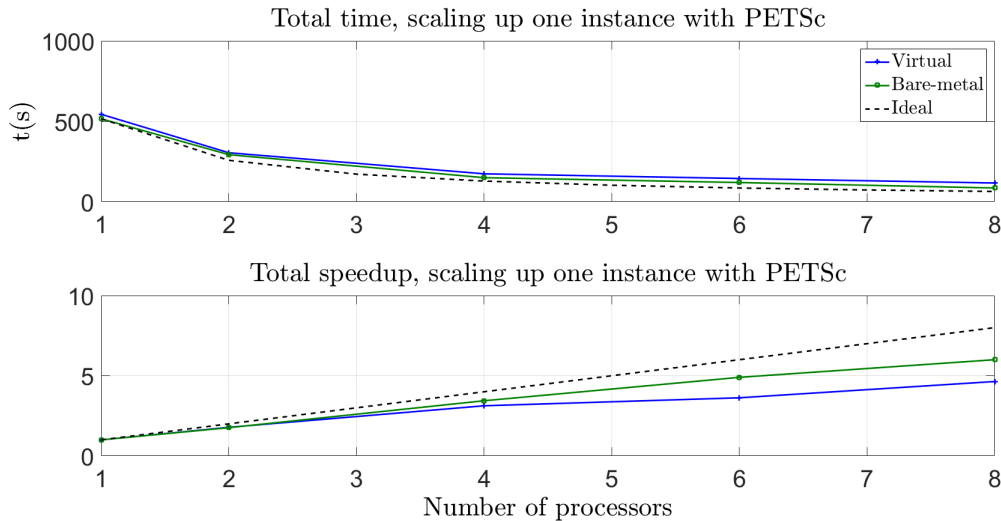


Figure 3: Plot that shows total time and speedup when scaling up on one instance with PETSc, shared virtual vs bare-metal

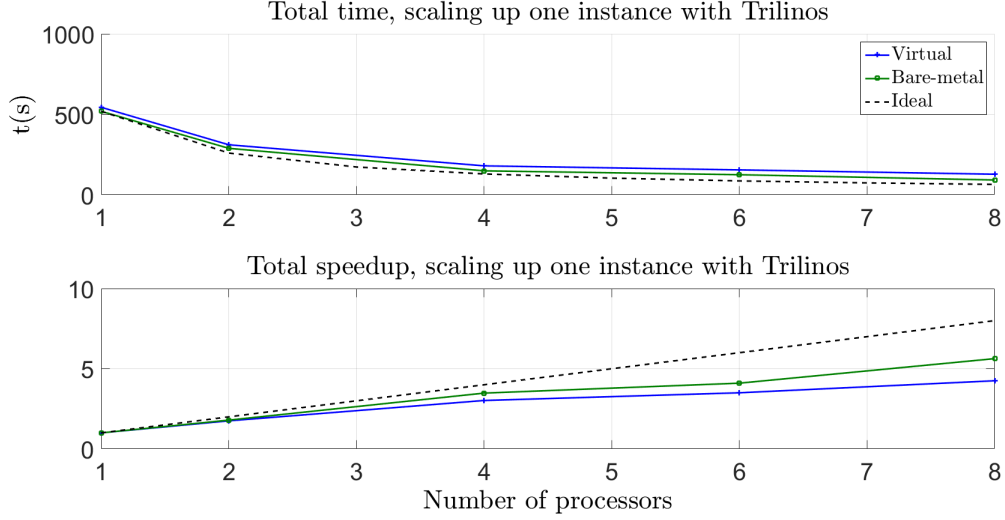


Figure 4: Plot that shows total time and speedup when scaling up on one instance with Trilinos, shared virtual vs bare-metal

In Figure 3 the run time is 15% longer on cloud compared to bare-metal when using two cores per node. The run time gets worse as the number of cores per node increase. Scaling up to eight core per node yields a 40% longer run time on the cloud. Figure 4 shows a similar behaviour.

4.2 Scale Out (shared virtual vs bare-metal)

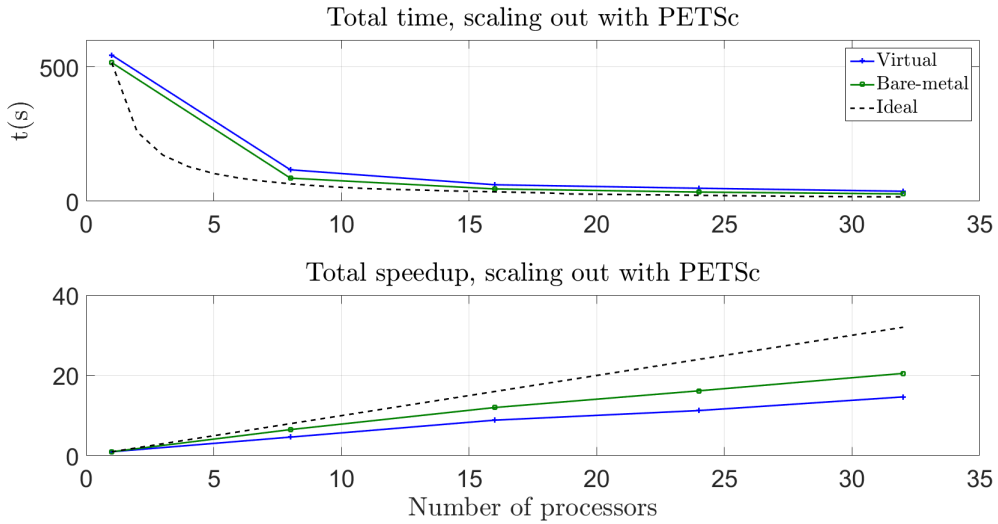


Figure 5: Plot that shows total time and speedup when scaling out to 32 cores with PETSc, shared virtual vs bare-metal

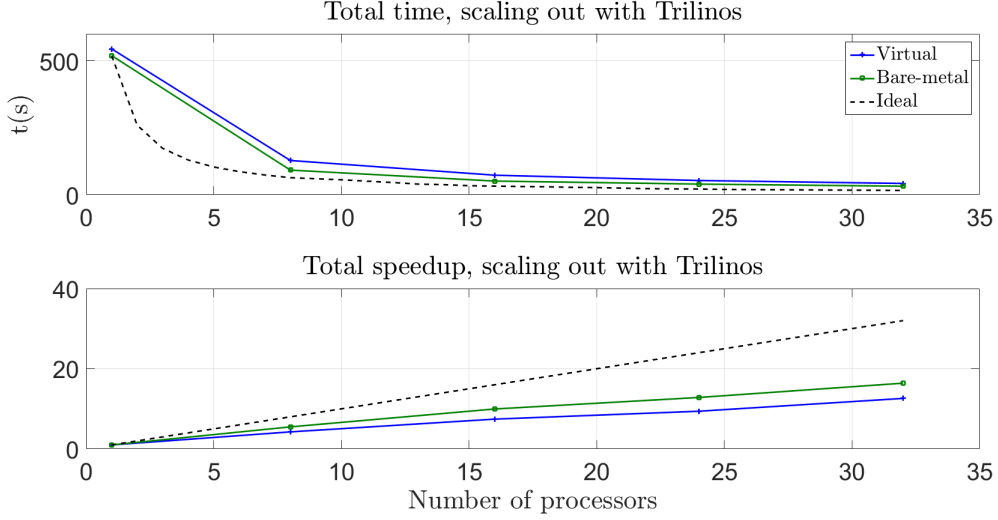


Figure 6: Plot that shows total time and speedup when scaling out to 32 cores with Trilinos, shared virtual vs bare-metal

Figures 5-6 shows the same behaviour as in the previous section (4.2). The performance on the cloud decreases compared to bare-metal as you increase the number of cores per node.

4.3 Scale Out (dedicated virtual vs bare-metal)

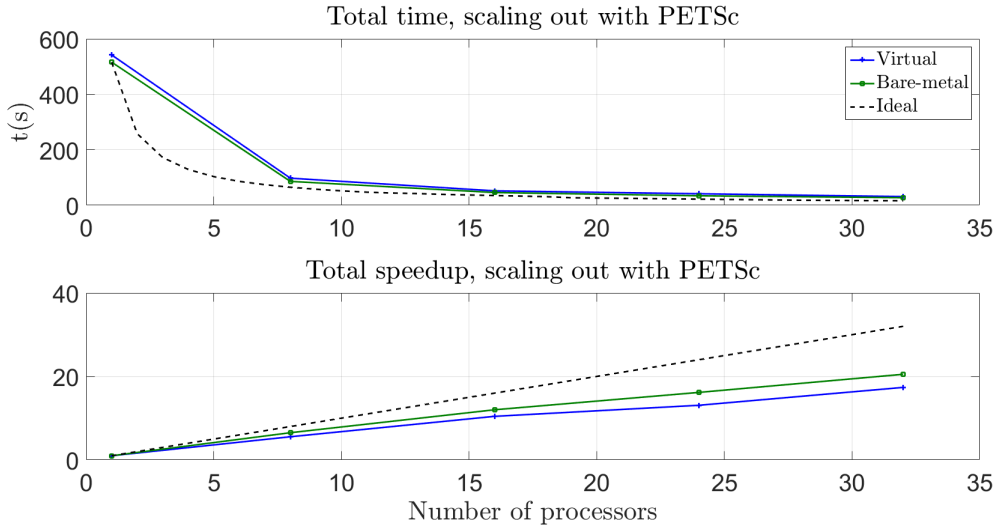


Figure 7: Plot that shows total time and speedup when scaling out to 32 cores with PETSc, dedicated virtual vs bare-metal

4.4 Inter vs. intra communication

All remaining results are based on only shared virtual environment.

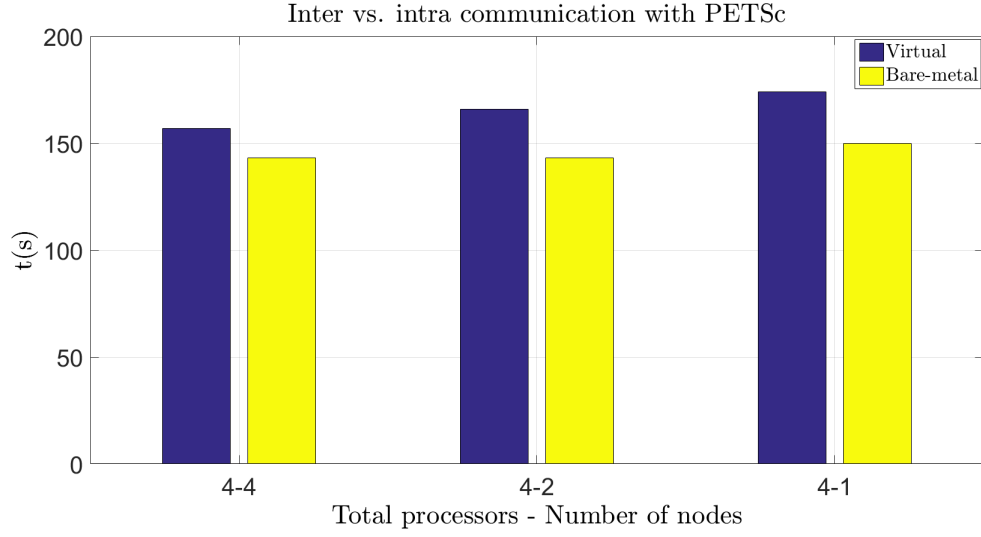


Figure 8: Plot that shows inter vs. intra communication with PETSc, shared virtual environment

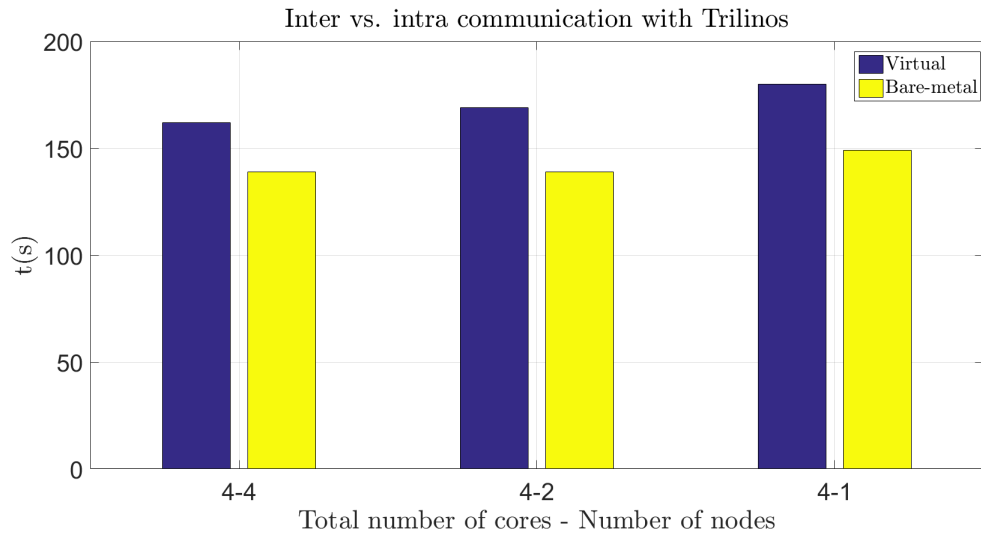


Figure 9: Plot that shows inter vs. intra communication with Trilinos, shared virtual environment

Figures 8-9 shows that the performance is worse when a VM communicates between its cores compared to communicating between VMs.

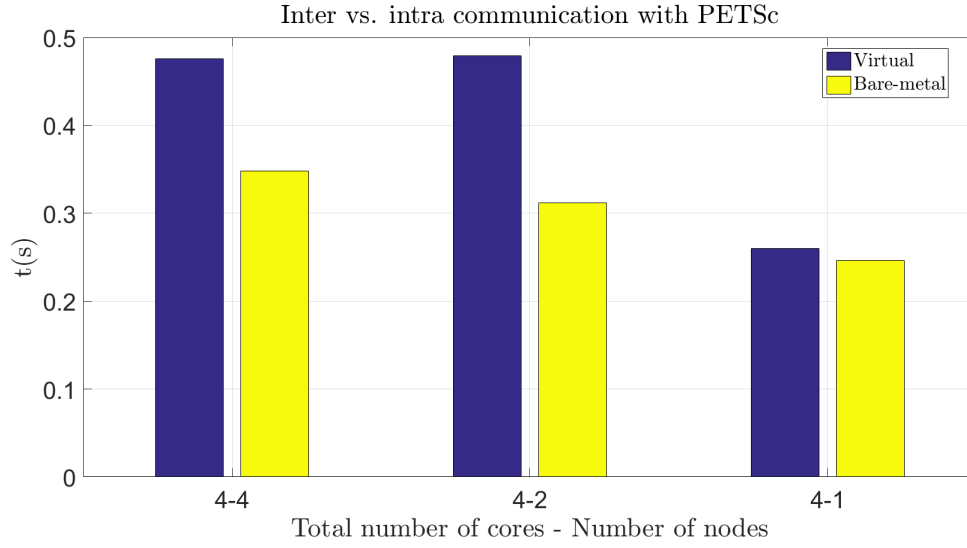


Figure 10: Plot that shows inter vs. intra communication with PETSc for a small problem size, shared virtual environment

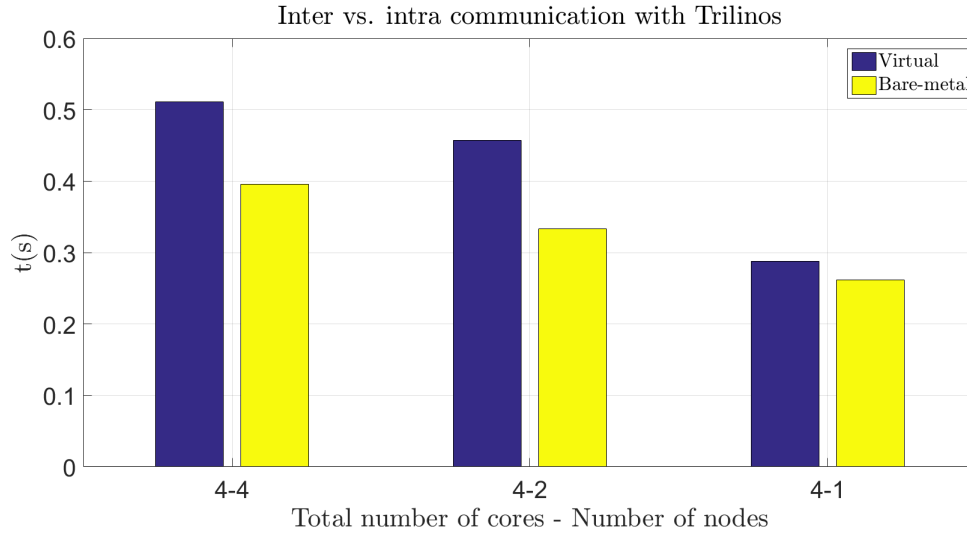


Figure 11: Plot that shows inter vs. intra communication with Trilinos for a small problem size, shared virtual environment

This time, in Figures 10-11 the communication is fastest between the cores.

4.5 Parallel Efficiency

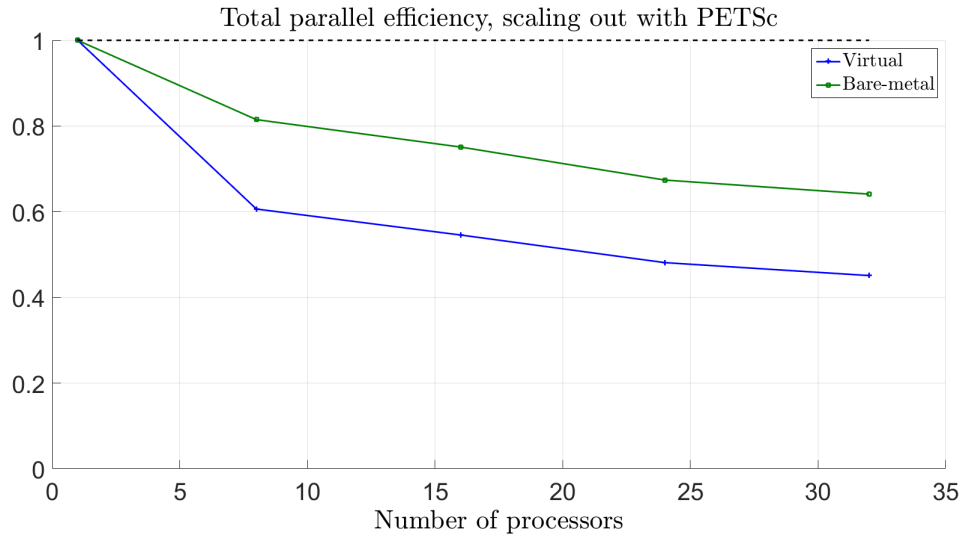


Figure 12: Plot that shows the parallel efficiency with PETSc, shared virtual vs bare-metal

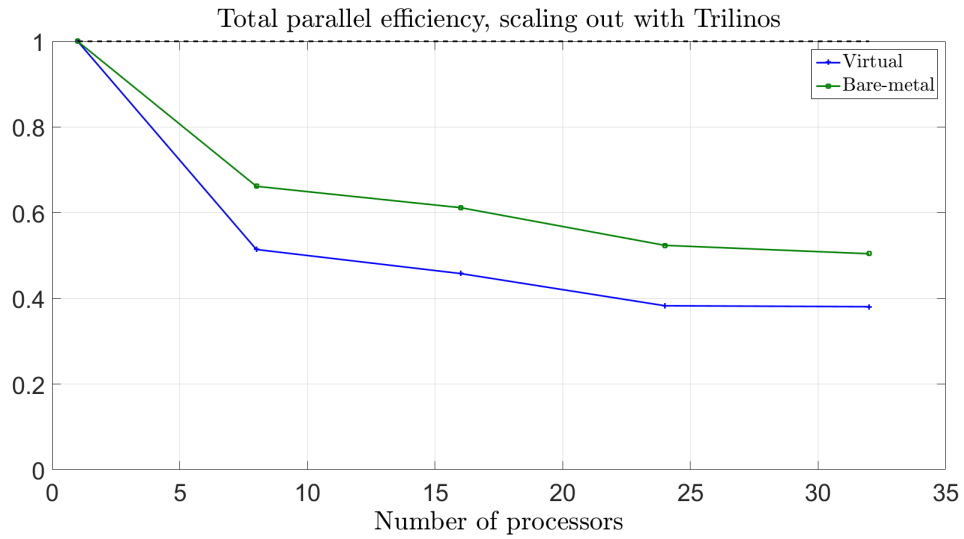


Figure 13: Plot that shows the parallel efficiency with Trilinos, shared virtual vs bare-metal

From figures 12-13 we can see that the parallel efficiency rapidly decreases when scaling up on one VM. The efficiency then follows the same trend for both bare-metal and the cloud.

5 Conclusion and Future Work

Among the many goals of this project were to make our framework vendor agnostic, i.e., a user must not be locked into the services of a single cloud provider and make it a one-click-application. We have succeeded in achieving our goals. Currently this application is based on OpenStack cloud suite and it can be made runnable on other cloud provider platforms. Another notable point about our work is, it is extendable for other scientific applications as well.

When comparing the performance between the bare-metal machine and the cloud, we can conclude that the performance when using the cloud is acceptable. There are some variations though, as the run time can take up to 40% longer when using the cloud. This is mainly due to SSC being a shared virtual environment. There are usually other users running experiment at the same time which affects performance. When looking at the results from the inter versus intra communication from Figures 8-9, the performance is actually worse when using four cores on one node compared to four cores on four different nodes. The expected results would have been the other way around. That is why we included the results in Figures 10-11 where we decreased the problem size to just a few thousand degrees of freedom. Now, the performance is what you would expect it to be. For the larger problem size, there might have been factors other than the communication that affected the performance such as cache misses for example. In Figure 7, when using dedicated virtual environment we can see an improved performance compared to Figure 5. The difference in run time compared to bare-metal is now only around 5%. This is a significant improvement compared to the results when using a shared virtual environment.

6 References

- [1] OpenStack, open source cloud computing software. <http://www.openstack.org/>
- [2] R language, about <https://www.r-project.org/about.html>
- [3] Spark, Apache spark. <http://spark.apache.org/>
- [4] Jupyter Notebook, Jupyter. <http://jupyter.org/>
- [5] OpenStack, flavors. <http://docs.openstack.org/openstack-ops/content/flavors.html>
- [6] <http://arnesund.com/2015/09/21/spark-cluster-on-openstack-with-multi-user-jupyter-notebook/>
- [7] <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>
- [8] <http://www.snic.se/>
- [9] <http://dealii.org/>
- [10] <https://www.uppmax.uu.se/smog-cloud>
- [11] <http://www.mcs.anl.gov/petsc/>
- [12] <https://trilinos.org/>
- [13] https://www.scai.fraunhofer.de/fileadmin/download/samg/paper/AMG_Introduction.pdf