MIKELANGELO

D2.3
Final Use Cases Implementation Strategy

Workpackage	2	Use Case & Architecture Analysis	
Author(s)	Peter Chronz		GWDG
	Benoît Canet, Nadav Har'El		ScyllaDB
	Gregor Berginc		XLAB
	Yosandra Sandoval, Nico Struckmann		USTUTT
	Matej Andrejašič		PIPISTREL
Reviewer	John Kennedy		INTEL
Reviewer	Joel Nider		IBM
Dissemination Level	PU		

Date	Author	Comments	Version	Status
2017-06-22	All	Initial draft	V0.1	Draft
2017-06-23	All	Document ready for review	V1.0	Review
2017-06-30	All	Document ready for submission	V2.0	Final

Executive Summary

This document presents the implementation strategies for the four use cases in MIKELANGELO. The use cases are the cancellous bones use case, the cloud bursting use case, the virtualized data analytics use case, and the aerodynamic maps use case. In this report, we describe our plan to evaluate the MIKELANGELO stack through the use cases.

This evaluation is of special importance, because the use-case evaluations are performed together with use-case stakeholders on real data. Furthermore, the use-case evaluations use the fully integrated MIKELANGELO stack instead of running isolated lab tests. In this document we present the evaluation plan. The actual results of the evaluations will be presented in D6.4 at the end of the project.

The implementation strategies in this document are described per use case. Although all use cases evaluate the MIKELANGELO stack, each of them makes use of particular subsets of features. Furthermore, the MIKELANGELO stack currently spans, disjoint architectures such as cloud and HPC systems. Thus, the evaluation needs to happen for each use case separately. For each use case we describe the experimental setup, the data to be used in tests, and the end-user validation. The experimental setup describes how the MIKELANGELO stack is being used by specific use cases. Additionally, the experimental setup contains descriptions of use-case specific architecture and implementations. The data to be used in the use-case evaluations is described briefly and its relevance and validity are explained. The end-user validation, finally, describes the end users and how they are going to evaluate their use case, when running on the MIKELANGELO stack.

The findings of the report can be summarized as follows. The implementation strategies currently mandate that the virtualized data analytics use case, the cancellous bones use case, and the aerodynamics use case be run in the integrated cloud testbed. Additionally, the cancellous bones use case and the aerodynamics use case will also be evaluated on the integrated HPC testbed. The cloud bursting use case will be run on an external cloud with hardware more suited to its need (more machines and an SSD attached to each VM), but we shall also attempt to run this use case on our integrated cloud testbed. Real-world data for system validation is available to the consortium for the cancellous bones use case, the virtualized data analytics use case, and the aerodynamics use case. End users are involved either directly as partners or collaborate closely with the project for the cancellous bones use case, the virtualized data analytics use case, and the aerodynamics use case. Furthermore, the current implementation of the cancellous bones use case, the virtualized data analytics use case, and the aerodynamics use case on at least one of the MIKELANGELO testbeds are available at least as prototypes.

This report introduces our general approach to implement the use cases for evaluation in the introduction. There we also provide an overview of evaluated components per use case. In the main part of the report we present the implementation strategies for the individual use cases: cancellous bones use case, cloud bursting use case, virtualized data analytics use case, and aerodynamic maps use case. Finally, the document concludes with a summary of the main results.

Acknowledgement

The work described in this document has been conducted within the Research & Innovation action MIKELANGELO (project no. 645402), started in January 2015, and co-funded by the European Commission under the Information and Communication Technologies (ICT) theme of the H2020 framework programme (H2020-ICT-07-2014: Advanced Cloud Infrastructures and Services)

MIKELANGELO

Table of contents

Table of Figures

MIKELANGELO

Table of Tables

1 Introduction

The purpose of this report is to describe the implementation strategy for the MIKELANGELO stack through the project's use cases. For each of the four use cases, we describe the experimental setup, the data to be used in experiments, and the planned validation by end users. This document describes the evaluation plans which we intend to implement after the feature freeze in project month 33. The results of the evaluations will be described in month 36 in report D6.4.

The use cases span a variety of applications to run in the cloud testbed and on the HPC testbed. Both testbeds have integrated most of the features from the MIKELANGELO stack. Each of the MIKELANGELO features is integrated in at least one of the testbeds. Furthermore, some of the features in the MIKELANGELO stack are specialized, so that their application is specific to individual use cases. In Table 1 we show which features are evaluated by which use case. Some features cannot be evaluated by all use cases. These restrictions are inherent to some of the use cases. For example, vTorque is designed to be used on an HPC cluster, therefore vTorque cannot be evaluated for use cases that focus on a cloud deployment. Some of the components are marked tentatively for evaluation. Reasons for a tentative marking are implementation timelines and challenges to overcome. We will pursue tentatively marked evaluation, but are not sure about their feasibility. More specific reasons discussing the rating of evaluations are given for each use case in the specific sections.

Table 1. Coverage of MIKELANGELO components by the use cases.

Use Case / Component	Cancellous Bones	Cloud Bursting - O	Cloud Bursting - NG	Data Analytics	Aerodynamics
IOcm	y	y	n	y	y
ZeCoRx	m	m	n	m	m
SCAM	m	y	n	y	y
vRDMA/OSv	m	n	n	n	m
vRDMA/Linux	m	n	n	n	m
OSv	y	y	n	m	y
MCM	m	y	n	y	y
Scotty	y	y	n	y	y

LEET	m	y	n	m	y
UNCLOT	y	y	n	m	y
vTorque	y	n	n	n	y
Snap	y	y	n	y	y
Seastar	n	y	y	n	n
ScyllaDB	n	y	y	n	n

The remainder of the document describes the implementation strategies for each of the four use cases in detail.

First, for the cloud bursting use case, a benchmarking setup for the cloud is presented. The benchmark compares a bursting scenario for Cassandra and ScyllaDB. Both experiments first deploy the respective databases on 3 VMs and then scale up to 6 VMs. The benchmark uses a Cassandra stressor that operates on 1.1TB of data. Details and the rationale for the two variants of the cloud bursting scenario are presented in the corresponding section.

Second, the virtualized data analytics use case presents the integration with Scotty and the use cases overall architecture with OpenStack, Heat, MongoDB, Spark, memcached, nginx, and Docker. Containers are used within VMs for easy deployment.The setup will use GWDG's cloud testbed and Scotty to perform parametric studies by turning MIKELANGELO's features on and off. The validation of the MIKELANGELO stack will be performed by the stakeholders of the real-world data analytics stakeholders.

Third, for the cancellous bones use case the integration with Scotty to automate experiments is described. This integration builds on USTUTT's HPC testbed. The workload generators for the cancellous bones use case allow the scaling of workload size based on node count and job size. The data used in the experiments is about 40GB large and served via NFS. The use case will perform parametric studies based on MIKELANGELO's features.

Finally, the aerodynamic maps use case will present its two types of experiments, to evaluate the MIKELANGELO stack via its integration with Scotty. The first experiment runs many small OpenFOAM cases in parallel. In contrast, the second experiment runs just one, albeit very computationally intensive, OpenFOAM case. Both experiments will be conducted on both USTUTT's HPC testbed and GWDG's cloud testbed. The evaluation for both experiments will be based on a parametric study by switching MIKELANGELO's features on and off. The end-user validation will be performed by PIPISTREL, who are the use case's main stakeholder and project partner.

2 Cancellous Bone Simulation Implementation Strategy

The purpose of the cancellous bones simulation is to analyze the structure of cancellous, or spongy, parts of the human bone in order to efficiently produce more accurately fitting longer lasting implants and safely attach them to it. The application is very compute intensive and by its parallel and scaling nature a perfect match to validate the MIKELANGELO software stack's integration into HPC environments.

In the following subsection the experimental setup for the Cancellous Bones Use Case is described in detail, explaining how the Use Case is integrated into the continuous integration (CI) platform Scotty and how experiments are executed in the HPC backend.

The validation is executed on a reduced data set with lower resolution in order to gain results in a timely manner as elaborated in the succeeding section, describing the actual data-sets used.

The last section focuses on the actual validation from an end-user's perspective. This comprises in first place the usage of vTorque, MIKELANGELO's middleware layer for HPC environments, integrating the various host and guest level components and providing control over them. This already includes vRDMA and IOcm which have been made available by the corresponding workpackages (WP3 and WP4). ZeCoRx will further be integrated and evaluated as soon it becomes available. All these components improve the I/O virtualization from different perspectives. Each can be switched on and off for an experiment run. Besides standard linux guests with vRDMA support, OSv guests with vRDMA support are also being used. Snap-telemetry is integrated into vTorque in order to provide the required data for in-depth analysis of the individual experiment runs. UNCLOT is also targeted to be integrated with vTorque and to be validated.

Table 2. Coverage of MIKELANGELO components by the cancellous bones use case.

MIKELANGELO Component	Use Case Cancellous Bones
IOcm	y
ZeCoRx	m
SCAM	n
vRDMA/OSv	y
vRDMA/Linux	y
OSv	y

MCM	n
Scotty	y
LEET	y
UNCLOT	y
vTorque	y
Snap	y
Seastar	n
ScyllaDB	n

2.1 Experimental Setup

Like the other use cases, Cancellous Bone uses the Gerrit code review tool [20] for performing internal code reviews, and with Scotty and Jenkins several experiments are performed in an automatic manner. The integration of the CI makes it possible to collect and document the use case experiments in an easy way and allows the rerunning of different experiment setups and also compare them to previous versions and bare metal base line measurements. For a valid analysis consistent performance data is the key, thus it is important to execute experiments in an automated way.

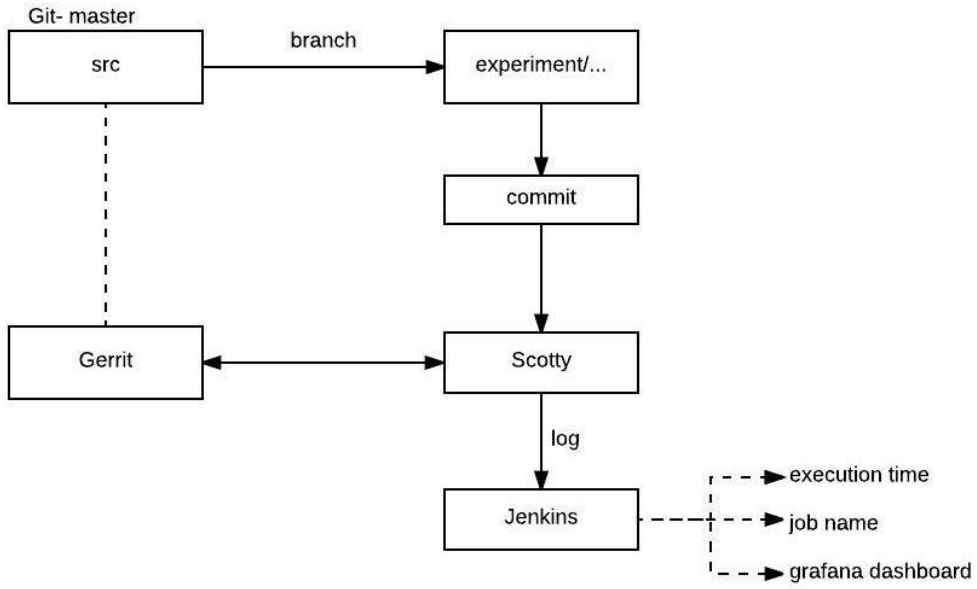

Figure 1. Continuous integration of Cancellous Bones

The repositories used in MIKELANGELO are located at XLAB and at GWDG. The figure above illustrates the workflow. The experiment configurations for the three distinct setups are

stored in a git repository. The actual execution is triggered by a commit to Scotty. Scotty then utilizes Jenkins to run the validations and stores the results in gerrit. There are 3 distinct experiment setups, each one with its own focus. There is one setup for baseline bare metal executions, another for the validation with standard linux guests and finally a setup that uses OSv guests:

1. Bare metal execution with Torque
2. VM execution with standard linux guests and vTorque
3. VM execution with OSv guests and vTorque

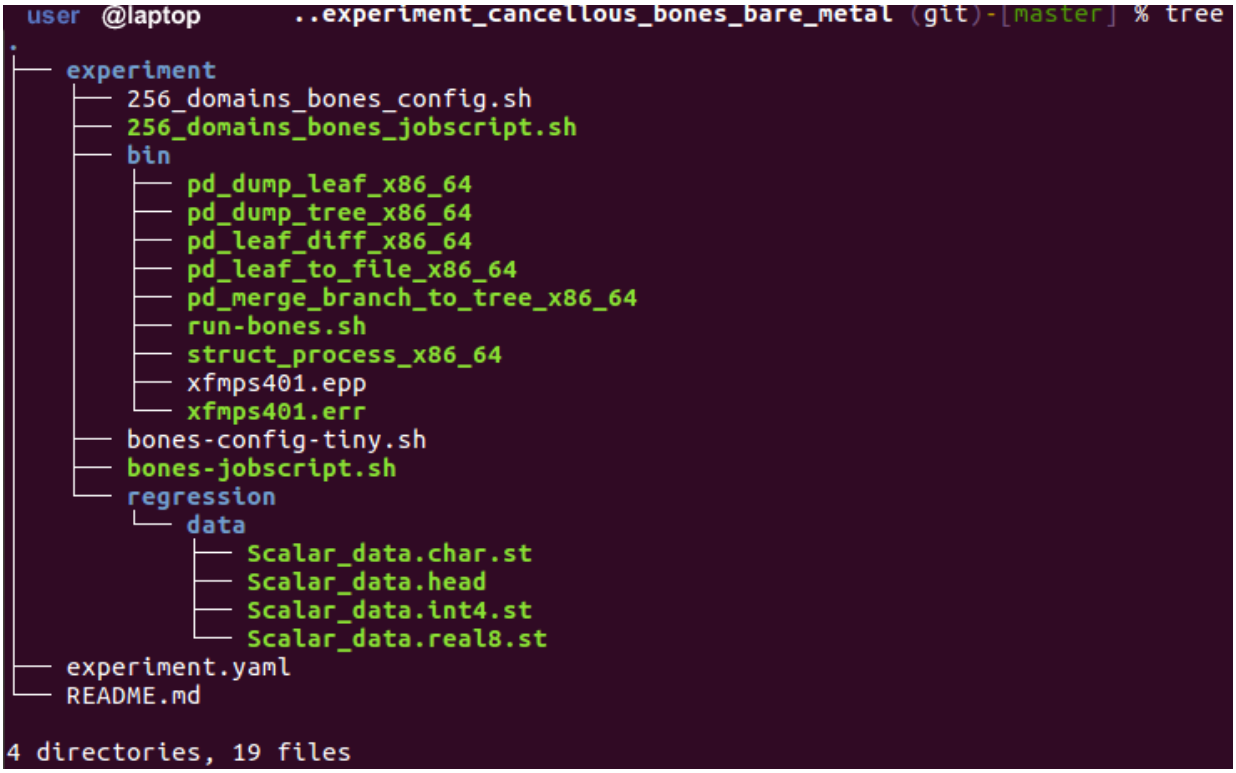

Figure 2. Structure of the experiment directory.

The structure of these experiment directories is presented in Figure 2. These contain all files required for an execution of the Cancellous Bones use case. Executable binaries are located in subdirectory `experiment/bin` and input data is located in subdirectory `experiment/regression/data`. In addition there are two files; one with suffix `_config.sh` configuring the use case's problem size and iteration steps, and another one with suffix `_jobscript.sh` that executes the use case validation in an HPC batch-system. The file `experiment.yaml` defines the different parameter combinations to run several combinations of arguments for one experiment setup. Each entry in that file corresponds to one experiment run. An example of this YAML [16] experiment configuration file is shown below.

The workflow of execution is defined as follows:

1. Create the repository in Gitlab and Gerrit
2. Checkout the different repos on the local machine

3. (Optionally) Change the bone configuration (input parameters) or the environment (.yaml file)
4. Commit the changes (git commit & git review)
5. Run the experiment through Scotty by posting run_experiment comment in Gerrit

The HPC-Workload-Generator was developed as an HPC backend for the Scotty CI to interact with a PBS/Torque HPC batch systems. This provides us with a way to submit HPC jobs directly out of the CI tool chain. The HPC-Workload-Generator is a middleware located in the layer between Jenkins or Gitlab CI and an HPC backend. To automate testing, the HPC-Workload-Generator also requires a YAML configuration file to specify the components and their configurations to be used in an experiment run.

```
- workload:
    name: bare_metal_256_domains_half_node
    generator: hpc-stuttgart
    params:
      job_script_name: 256_domains_bones_jobscript.sh
      experiment_dir: experiment
      qsub_number_of_nodes: 1
      qsub_number_of_processes_per_node: 8
      application_log: ~/experiment_log/log
- workload:
    name: bare_metal_256_domains_1_node
    generator: hpc-stuttgart
    params:
      job_script_name: 256_domains_bones_jobscript.sh
      experiment_dir: experiment
      qsub_number_of_nodes: 1
      qsub_number_of_processes_per_node: 16
      application_log: ~/experiment_log/log
...
```

The above is an example of how such an YAML experiment configuration file for the Cancellous Bones use case is structured. Each `workload` section describes the configuration of an experiment run. Besides the `name` of the experiment, and the `backend` to be used - in this case HPC - also batch-system parameters are provided defining the resources for an experiment run (`qsub_number_of_nodes`, `qsub_numer_of_processes_per_node`). Further parameters are the job script starting the actual application and also the log file recording an experiment's execution.

2.2 Data to Be Used in Experiments

One big challenge of HPC applications is scalability. To write an application that can make use of more than 10,000 cores is not a straightforward task. The scalability of an application depends on its runtime characteristics Some applications are CPU bound, which scale relatively easily, whilst some are I/O bound or have a lot of communication between processes. I/O bound applications scale to the point where more CPUs are waiting for I/O

operations to be completed, than CPUs that are computing. The cancellous bone simulation is mostly CPU bound but in some phases it is I/O bound.

The Cancellous Bone simulation uses its own file format (called PureDat) to store data in an efficient way. The basic idea behind the PureDat format was to develop a file format that separates the data types in memory in order to minimize the necessary system and library calls. PureDat is used as input file format as well as for the output.

The average size of a real world data sets are in the area of up to several hundred TBs. Thus, to gain performance measurement results in a timely manner the input data set used for validation is heavily reduced, but still causing the same workload pattern. It is split into 4 files with a total size of about 19 GB for a medium sized set (maximum data set is approximate 40 GB). One of the files is the main data set and contains most of the data used. The data for the calculation will be provided via a shared file-system (i.e. NFS or Lustre) which is mounted on every compute node as well as the frontend. The compute nodes communicate over MPI and read from/write to the shared storage intermediate data for processing. The master process splits the domains and assigns them to the workers. The workers then pick up the prepared data from the shared storage.

The basic idea is to have a simple persistent storage of computed data on the file system. It should store data space efficiently (less data transfer) and reduce the amount of single files (fewer file read/writes). This all helps the simulation to handle I/O operations in the most efficient manner.

For each data type there is a dedicated file in order to reduce function calls required to convert them into the correct data type. In addition, these files are stored in a binary format to reduce the overall size of files.

PureDat can handle six data types at the moment, but can be expanded if needed:

1. Signed Integer with 1 Byte
2. Signed Integer with 2 Byte
3. Signed Integer with 4 Byte
4. Signed Integer with 8 Byte
5. Floating Point Values with 8 Byte
6. ASCII Data with 1 Byte

Therefore, six separate files are written on the file system, each for a specific type of data. A brief example: during the simulation each of the compute nodes (workers) reads and writes coordinates. To achieve this in an efficient way, up to three different types of data are needed:

1. A list of node numbers (Integer)
2. A list of node coordinates (Floating Point)

3. The amount of nodes (Integer)

Each of these three data-sets can be stored in a separate file (which can be written in parallel). If we compare this with a "traditional" way of storing coordinates we can see why the PureDat approach is more efficient. Traditionally the node number (Integer) gets stored followed by the coordinates (Floating Point), which leads to more library calls per node writes or reads. Depending on the lengths of the list this can scale to a few million library calls.

To get a single block data out of this data "stream", the data is stored in a tree structure. This tree can store raw data in different locations. As in other tree data formats, there are leaves and branches. The information about this tree is stored in a separate file, to have all data fully separated.

2.3 Validation by End Users

To validate the improvements resulting from developments within the MIKELANGELO project, KVM will be swapped to the newly developed sKVM and the standard Linux based guest OS will be replaced by the lightweight single user cloud operating system OSv. At this stage the use case is fully integrated into the MIKELANGELO HPC stack and can be executed directly on the compute nodes as well as inside a virtual machine. To evaluate the functionality of developed MIKELANGELO components by the help of the the use case, it has to be executed in in three different configurations: bare-metal (Torque), standard linux guests (vTorque) and OSv guests (vTorque). In the following text part at first the general approach for the automated CI based evaluation of the Cancellous Bones use case is presented, followed by descriptions of manual execution for each of the configurations.

CI based automated execution

The execution of the experiments is similar for all cases, the only difference between virtual and bare-metal execution is the set of arguments provided for the batch-system steering the execution. Once the repositories are configured correctly, users login to the Gerrit web interface (see Figure 3). From the web interface users can then trigger the automated execution via scotty by responding to the patch-set with comment "run_experiment".

In the Jenkins output log contains detailed information about all the steps performed during the execution of each experiment. In particular: i) the branch that will be tested, ii) the parameters that receive the workload generator through the yaml file, iii) link to the see the job performance data - grafana GUI, iv) output of the simulation and finally v) summary of the job execution.

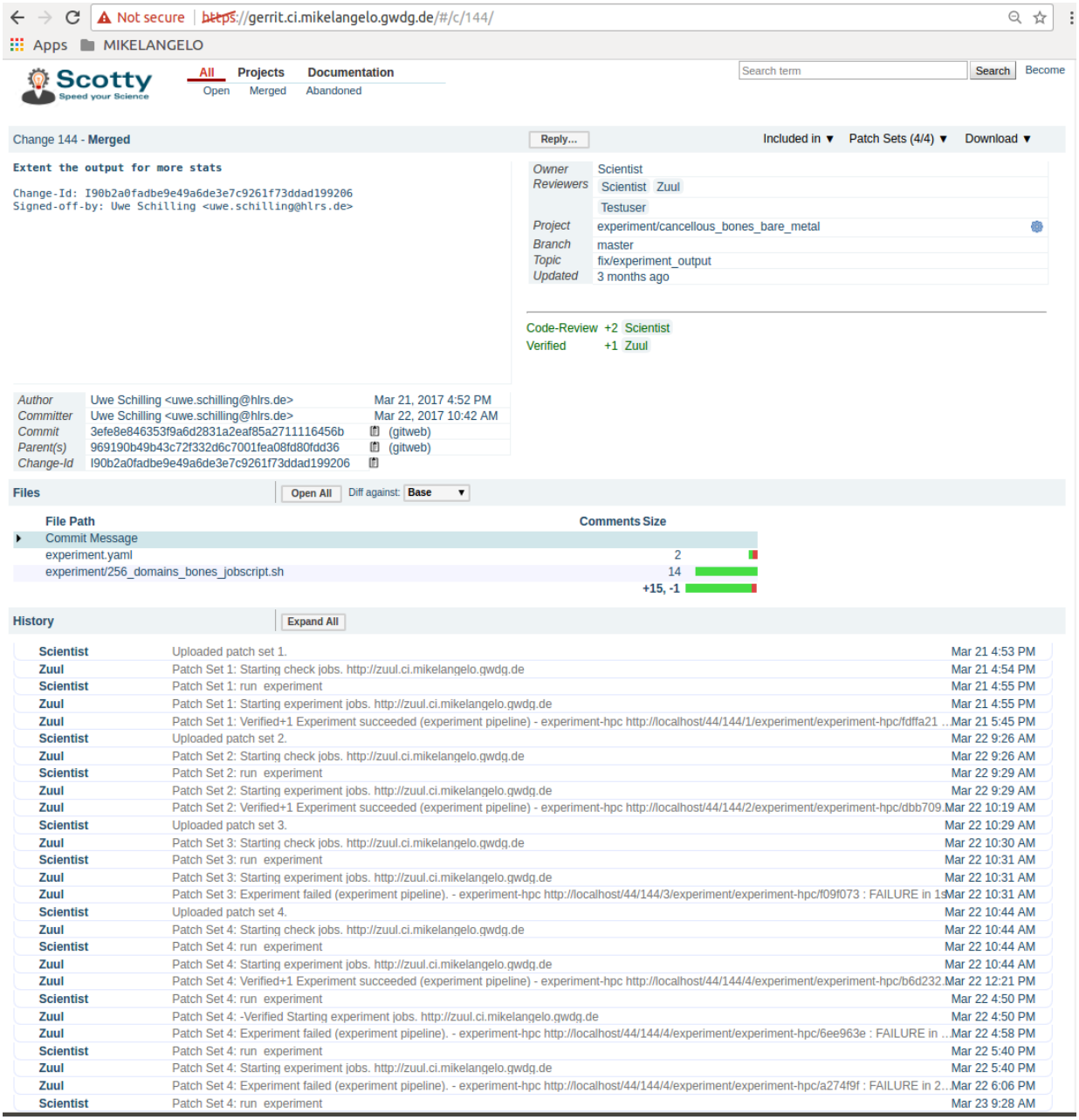

Figure 3. Gerrit web interface for the project experiment/cancellous_bones_bare_metal

The output / exec log is available via Jenkins or grafana GUI (the link to grafana GUI is included in the exec log), as shown in Figure 4 to Figure 7.

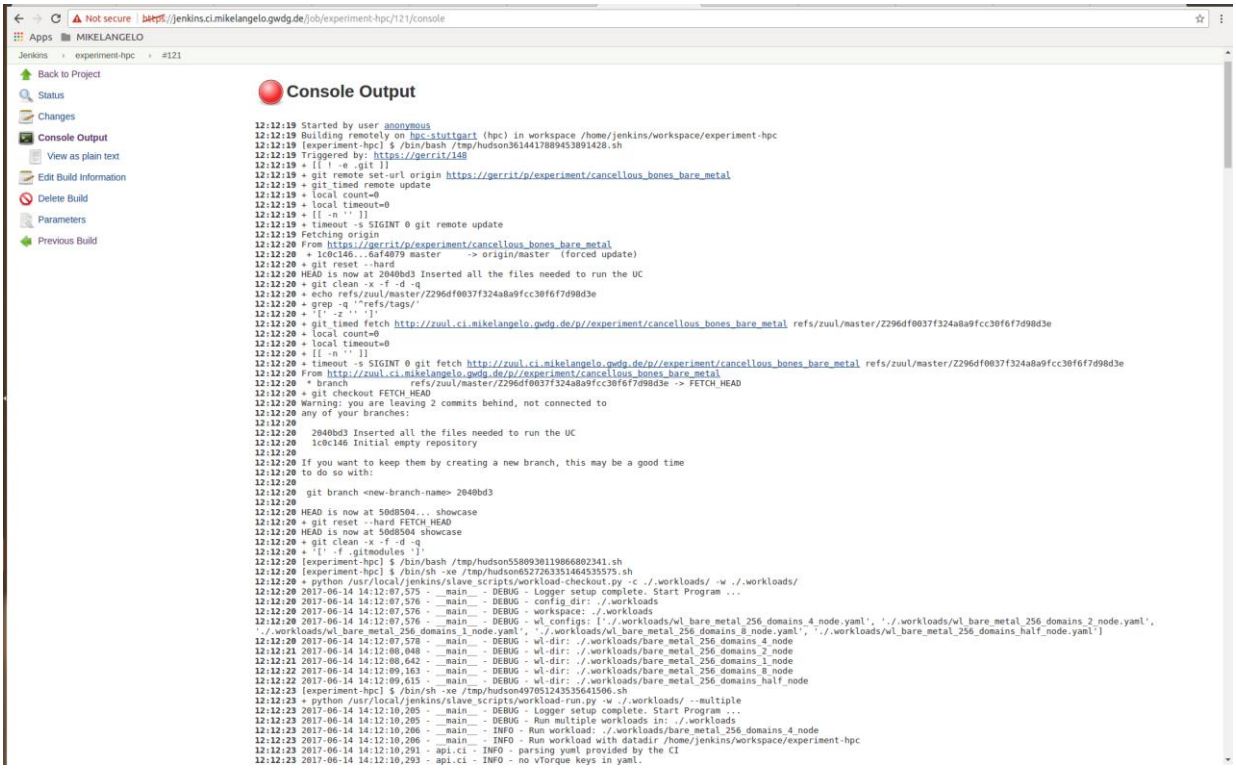

Figure 4. Output console Jenkins. Part I

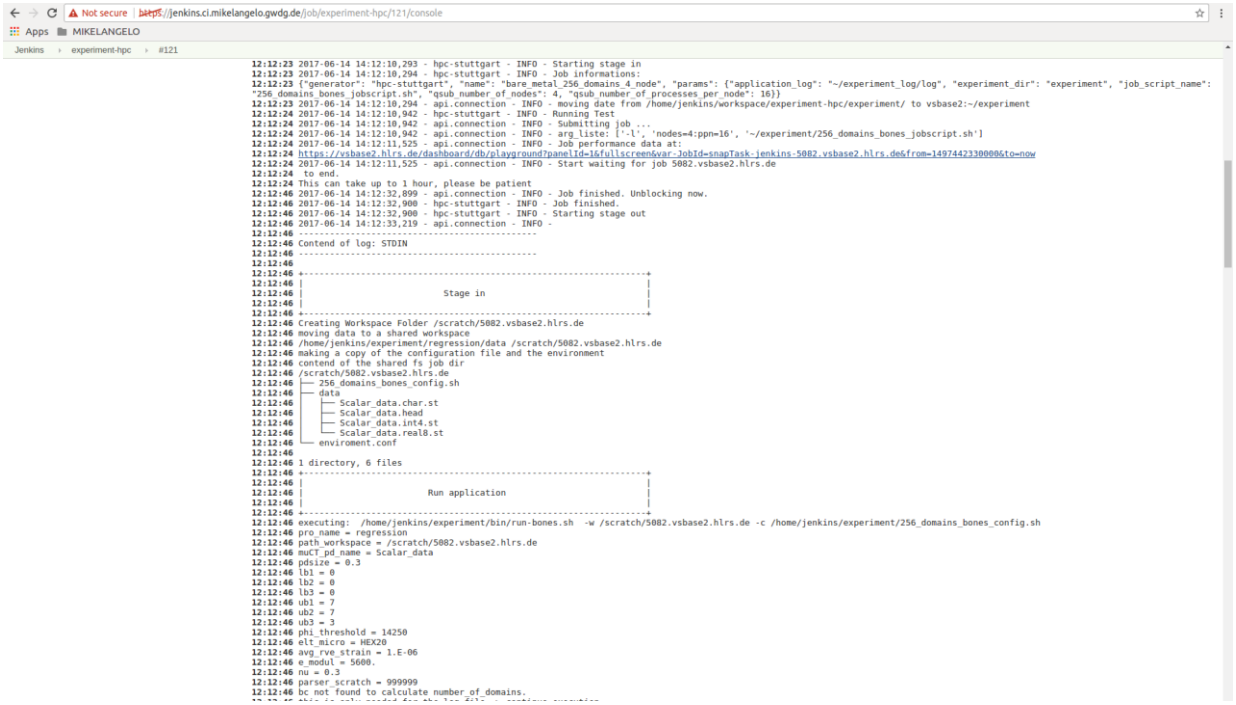

Figure 5. Output console Jenkins. Part II

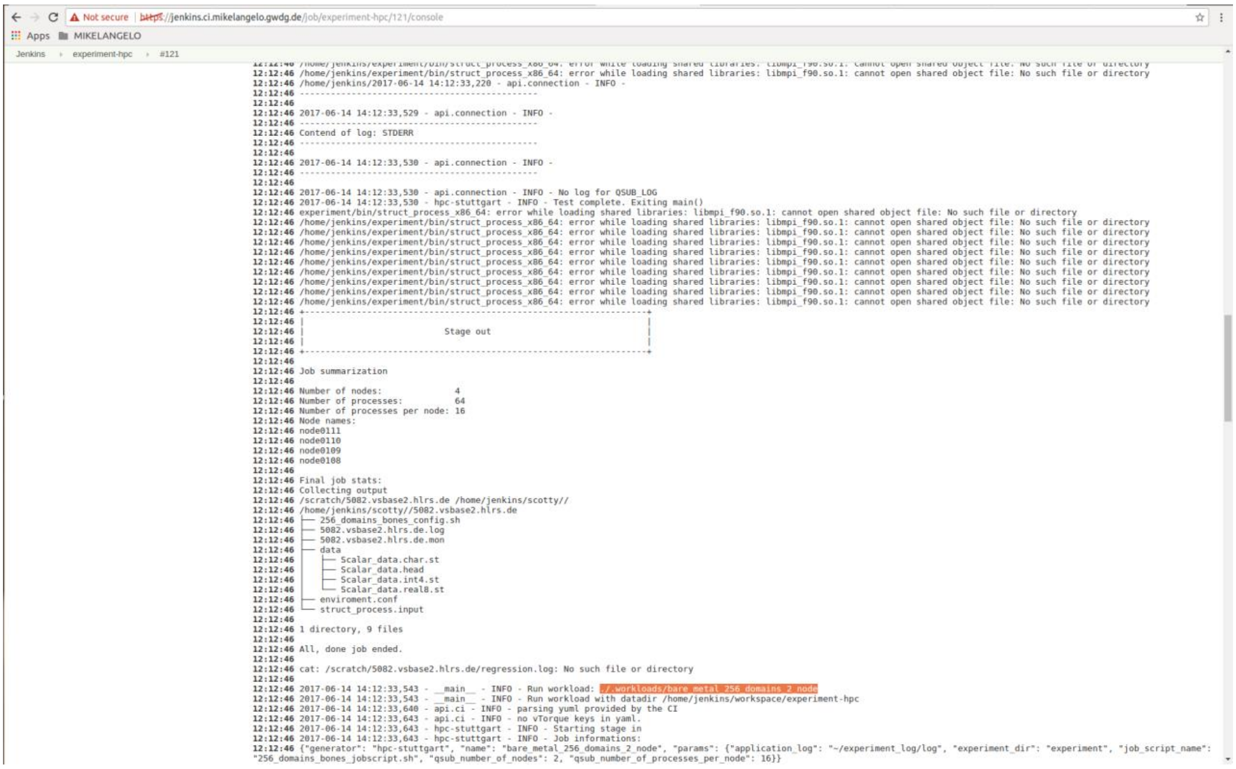

Figure 6. Output console Jenkins. Part III

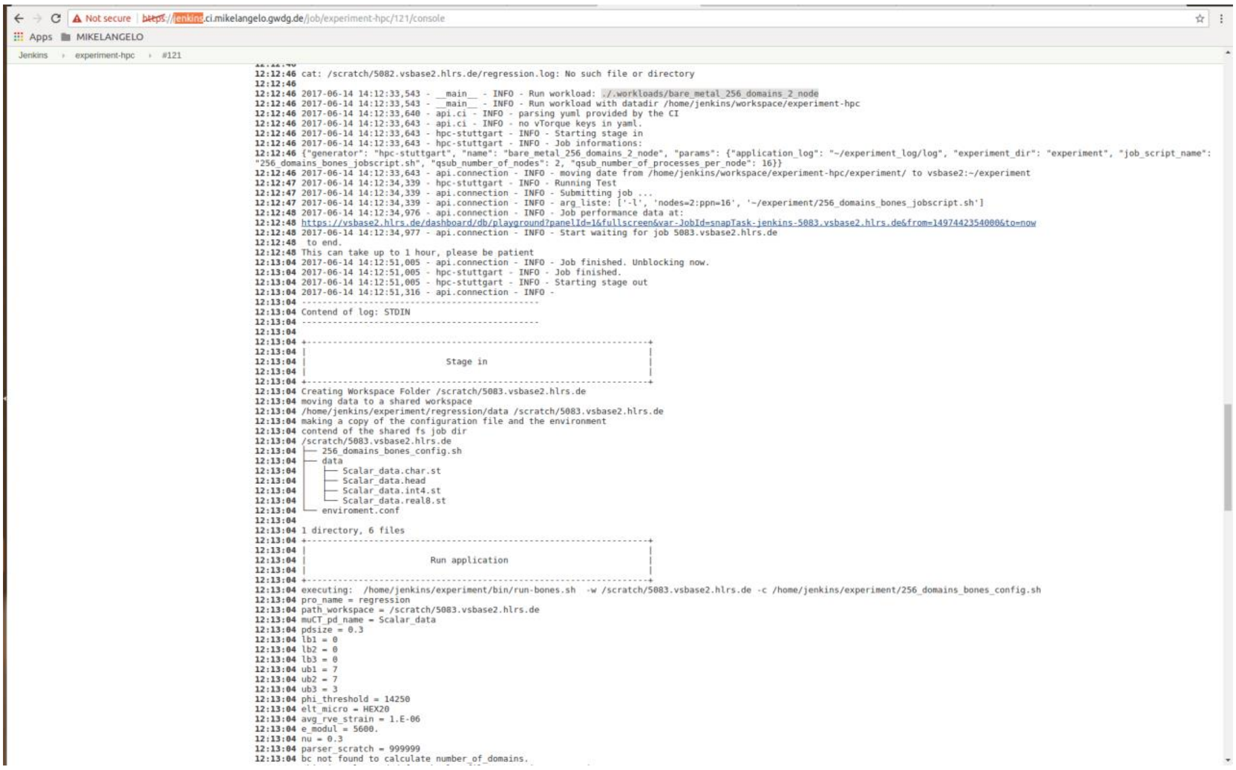

Figure 7. Output console Jenkins. Part IV

Bare metal execution with Torque

This is executed manually via the following command:

```
qsub -l nodes=1:ppn=16 /home/user/projects/mikelangelo-bones/bones-jobscript-metal.sh
```

Where nodes define the host environment, in this case one node with at least 16 cores (ppn stands for processes per node). The job script runs on the first allocated node. The job script handles the data transfer between the home file system and the shared scratch, defines a config file and starts the execution of the simulation.

Automated CI execution is based on the YAML files described in section 2.1.

Virtualized execution with Standard Linux guests and vTorque

A VM image is needed to execute the job, either it is provided or a default image defined by cluster administrators in vTorque's config is utilized. The image contains the application binaries and its dependent libraries needed to successfully run the simulation. The data, the configuration and the job script are not part of an image. This modularity makes it easy to set up different configurations without touching the images at all. This image can be understand as "fat", a single file which contains the smallest set of setup to run the simulation.

```
user@vsbase2 ~]$ /opt/dev/vTorque/src/vsub -l nodes=1 -vm vcpu_pinning=yes -vm
vcpus=4,img=/images/pool/ubuntu_bonesV01.51c.img ./experiment/bones-jobscript.sh
[vsbase2|2017-06-20T19:54:06|vsub|INFO] Generating files for VM job, please be patient..
5104.vsbase2.hlrs.de
[node0107|2017-06-20T19:54:13|prologue|INFO] Root prologue script started
[node0107|2017-06-20T19:54:14|snap-start|INFO] Setting up SNAP monitoring..
[node0107|2017-06-20T19:54:17|snap-start|INFO] Setting up SNAP monitoring done.
[node0107|2017-06-20T19:54:17|prologue|INFO] Root Prologue script finished
[node0107|2017-06-20T19:54:17|vmPrologue|INFO] User prologue wrapper script started.
[node0107|2017-06-20T19:54:30|sshd:vmPrologue.parallel|INFO] User prologue.parallel wrapper
script started.
[node0107|2017-06-20T19:54:45|sshd:vmPrologue.parallel|INFO] User prologue.parallel wrapper
script finished.
[node0107|2017-06-20T19:54:46|vmPrologue|INFO] User prologue wrapper script finished.
[node0107|2017-06-20T19:54:56|jobWrapper|INFO] Starting user job script in VM.
[node0107|2017-06-20T20:01:19|epilogue|INFO] Root epilogue script started.
[node0107|2017-06-20T20:01:22|epilogue|INFO] Root epilogue finished.
```

The command is similar to the bare metal execution, but uses vsub instead and may contains additional parameters preceded by `-vm`. The vsub command will run this job in a virtual environment. In the example above '`-vm img=..`' defines the VM image to be used, while '`-vm vcpus=4`' defines the number of virtual CPUs assigned to each guest.

The use case has been executed with several different core counts and has run multiple times with the same settings. The gathered performance measurements are shown in Figure 3.

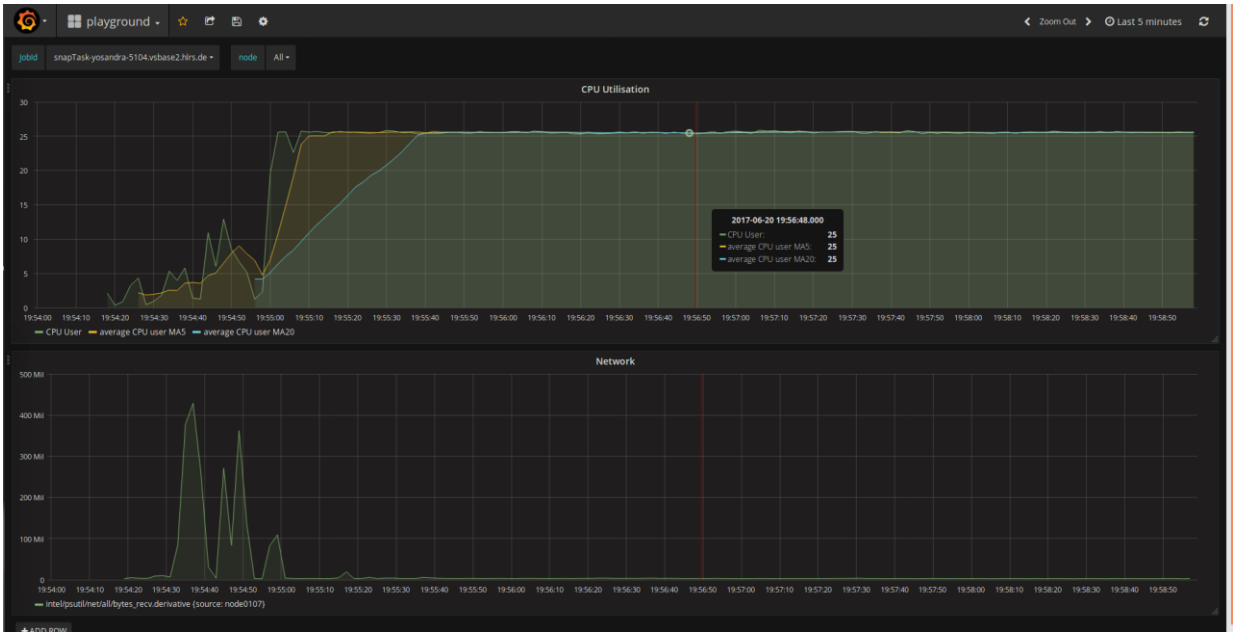

Figure 8: Performance measurements for Use Case Cancellous Bones execution.

Virtualized execution with OSv guests and vTorque

Cancellous Bones was also ported successfully to OSv. As intermediate step, the manual execution was performed both in debug and release mode, the output of the debug mode is shown below. To run the simulation manually, the "run.py" command is executed. The automated execution also takes places by the help of command `vsub`. A adaptation of this command will be include on the integration with CI in order to execution it in the same way that the other two cases.

```
root@node0102:/home/user/osv# sudo scripts/run.py --memsize 8G --vcpus 4 --api -d -V -v --
novnc -b br0 -e 'tools/mount-nfs.so nfs://172.18.2.3/storage/mikelangelo/ssd_scratch/
/scratch; /usr/lib/orterun.so --allow-run-as-root -np 5 /tools/struct_process_x86_64.so
/scratch/cancellous-bones/struct_process.input';
OSv v0.24-312-gd3866580
4 CPUs detected
Firmware vendor: Bochs
bsd: initializing - done
VFS: mounting ramfs at /
VFS: mounting devfs at /dev
net: initializing - done
vga: Add VGA device instance
...
+-----------------------------------------------------------------------------
+--: Output path
+--> /scratch/cancellous-bones/
+-----------------------------------------------------------------------------
+--: Project name
+--> regression
+-----------------------------------------------------------------------------
```


```
+--: muCT puredat pro_path
+--> /scratch/cancellous-bones/regression/data/
…
+--: Output Format
+--> FMPS
...
+-------------------------------------------------------------------------------
+--: Output amount
+--> DEBUG

<=============================================================================>
Starting chain link :struct process
MM Opened new log-file :                                                    MM
MM /scratch/cancellous-bones/regression.log                                 MM
MPI rank :          2 ; Domain number :          1 ; Domain path : /scratch/cancellous-
bones/regression_results/1/2/
MPI rank :          1 ; Domain number :          0 ; Domain path : /scratch/cancellous-
bones/regression_results/1/
MPI rank :          3 ; Domain number :          2 ; Domain path : /scratch/cancellous-
bones/regression_results/1/3/
MPI rank :          4 ; Domain number :          3 ; Domain path : /scratch/cancellous-
bones/regression_results/1/2/4/
 GENERATE QUADMESH
 GENERATE QUADMESH
 GENERATE QUADMESH
 GENERATE QUADMESH
 WORD   ITYP            IBUF                RBUF   CBUF
    1      0               0    0.0000000000E+00   ANAL
 WORD   ITYP            IBUF                RBUF   CBUF
    1      0               0    0.0000000000E+00   PART
…
…
WORD   ITYP            IBUF                RBUF   CBUF
    1      0               0    0.0000000000E+00   AT_I
    2      0               0    0.0000000000E+00   FROM
    3      1               1    0.1000000000E+01
    4      0               0    0.0000000000E+00   TO
    5      1             100    0.1000000000E+03
 WORD   ITYP            IBUF                RBUF   CBUF
    1      0               0    0.0000000000E+00   FIN

 MPI Program ends with code:           0
 MPI Program ends with code:           0
 MPI Program ends with code:           0
 MPI Program ends with code:           0
 MPI Program ends with code:           0
Requested termination of /libhttpserver.so, waiting...
VFS: unmounting /dev
VFS: unmounting /proc
VFS: unmounting /
Powering off.
```

3 Cloud Bursting Implementation Strategy

"Cloud bursting" is a sudden increase of incoming traffic to a cloud service. Gracefully handling such bursts is an important concern in many useful domains like e-commerce, news, weather information, Internet of Things (IoT), telecommunications and many more. Preparing for such a burst is a inherently a hard problem, primarily for two reasons. First, it is typically impossible to predict when the burst will occur and second, costs of maintaining the infrastructure required by such spikes is unjustifiable. Therefore, if a service cannot *prepare* to a cloud burst event, it would like to quickly *react* to one. In other words, during a cloud burst we would like to grow, as quickly as possible, the amount of resources (such as CPUs and disks) assigned to this service, until the new resource assignment is enough to handle the new load.

In our "cloud bursting" use case, we focus on cloud bursting in one specific domain: the NoSQL database Apache Cassandra [2]. Cassandra is a popular, open source fully distributed database, providing high availability with no single point of failure. It excels at fast, low latency writes, with slightly slower reads. Cassandra is often used as part of a cloud service, providing robust support for clusters spanning multiple data-centers. During a cloud burst, the Cassandra cluster of a given size, which was enough to handle the original load, must now grow to a bigger size as required to handle the new load. This growth involves not only starting new VMs which run Cassandra, but also requires moving huge amounts of data to the new nodes, since a new empty node cannot answer read requests before being filled with the data it is supposed to own. Moreover in addition to the speed of this data movement - and ultimately - to the speed the cluster grows - a second important concern is that *during* this growth process, the cluster must not become bogged down with this operation and become even slower than it used to be. Of particularly concern to Cassandra applications is that the tail (99th percentile) latency does not considerably increase during this growth.

In this use case we plan to measure MIKELANGELO's improvement to both aspects of the cloud bursting performance - the amount of time it takes the Cassandra cluster to grow, and the performance hit (especially latency) suffered during the period of growth. We plan to demonstrate that out of all of MIKELANGELO's components, the one most providing most benefit to the cloud bursting performance is the **Seastar** library developed by WP4. We re-implemented Cassandra using the Seastar library (the result is the open-source Scylla database[1]), and have already shown in Y2 significant improvements to all aspects of cloud bursting performance by using Scylla instead of Cassandra. This year, we plan to repeat the same measurements with the improved Seastar (and Scylla) we have now, and hope to see even bigger improvements. We especially hope to see even lower latencies, because this is an

[1] ScyllaDB GitHub repository, https://github.com/scylladb/scylla

issue which a lot of recent improvements to Seastar (such as its new CPU scheduler) were meant to address [4, 5].

Beyond evaluating the performance gain from Seastar, we also plan to try benchmarking the original Cassandra on the MIKELANGELO cloud test bed which includes additional improved components, to see how much performance some of these components might bring to this use case. However, we may not be able to effectively do these benchmarks because MIKELANGELO's current testbed is not configured for this type of workload: The Cassandra workload requires scalable random-access disk I/O which requires that computation nodes have local solid-state disks (SSDs) - but those are not currently available in the testbed. Note that in any case, we expect the gains which Seastar brings to this workload will be much higher than the gains which any of the other MIKELANGELO components might bring, so measuring the gains from the other components to this use case is of marginal importance.

The following table shows the components used in this use case.

Table 3. Coverage of MIKELANGELO components by the cloud bursting use case.

Use Case / Component	Cloud Bursting - Original	Cloud Bursting - Next Gen
IOcm	y	n
ZeCoRx	m	n
SCAM	y	n
vRDMA/OSv	n	n
vRDMA/Linux	n	n
OSv	y	n
MCM	y	n
Scotty	y	n
LEET	y	n
UNCLOT	y	n
vTorque	n	n
Snap	y	n
Seastar	y	y
ScyllaDB	y	y

The above table currently contains two variants of the cloud bursting use case. The **Original** use case focuses on the evaluation of most of the MIKELANGELO components on top of the MIKELANGELO testbed. The **Next Gen** use case evaluates the cloud bursting use case in an optimal scenario in an environment comparable to the environment typically used for large Cassandra deployments. The following benchmarks and plans focus primarily on the Next Gen use case. The evaluation of the Original scenario depends on the applicability of the internal MIKELANGELO testbeds for the application needs (e.g., ScyllaDB requires fast, local storage to achieve optimal performance).

Depending on how the early benchmark we mention later in this document behave we may test some of the technologies aforementioned in the above matrix.

3.1 The Cloud Bursting Benchmark

ScyllaDB [3] will redo the cloud bursting benchmark on AWS that was done for the 2nd review.

This benchmark measure how fast a huge database can grow under a heavy load. A benchmark loader is filling the database as fast as it can while the cluster grow. It simulate how people will actually grow the database in real world use case.

The latter benchmark is done in four phases on a continuous write-only workload.

1. Prepare a test dataset. A 1.1TB per node database of 3 vms is prepared using ScyllaDB and cassandra-stress and the data files are copied into a safe place for later usage.
2. Start Cassandra with this dataset, load cassandra with cassandra stress and grow the cluster vm count from 3 to 6 increasing one vm at a time. Along the way collect all relevant metrics (latency, timing).
3. Reiterate this experiment with ScyllaDB as a database while setting the load generator to produce the same maximum throughput that Cassandra was able to handle in the previous test.
4. Compare and analyse

The test measures how long it takes to double the cluster capacity from 3 to 6 nodes while measuring throughput (op/sec) and quality of service indicator (latencies).

3.1.1 Early Benchmark

The ScyllaDB team has opted to adopt an adaptive approach: first we will run some basic manual benchmarks with Cassandra stress in the MIKELANGELO evaluation test bed.

Since Scylla Cluster Test - the test suite that was used to automate the benchmark shown at the project's second year review meeting in Brussels - is a heavyweight framework that is very cumbersome to work with, ScyllaDB will start by running a simple benchmark in a manual way on MIKELANGELO test infrastructure.

3.1.2 Cloud Bursting Results Refresher

This section presents a summary of the results of this use case in March 2017 - an update from the 2nd Review data.

3.1.2.1 Throughput

	Scylla	Cassandra
Time to complete	3h	10h

In Figure 9, the square steps show the node count of each database increasing each time a new VM is spawned.

We can see that ScyllaDB throughput is capped at 4kop/s with any number of nodes. 4kop/s is the maximum Cassandra was able to achieve with 6 nodes. One sample is taken every 10s.

We can see that the test takes around 10 hours to complete for Cassandra and around 3 hours for Scylla. Scylla throughput stays at its requested value while Cassandra tries to cope with the task of growing the cluster with varying results.

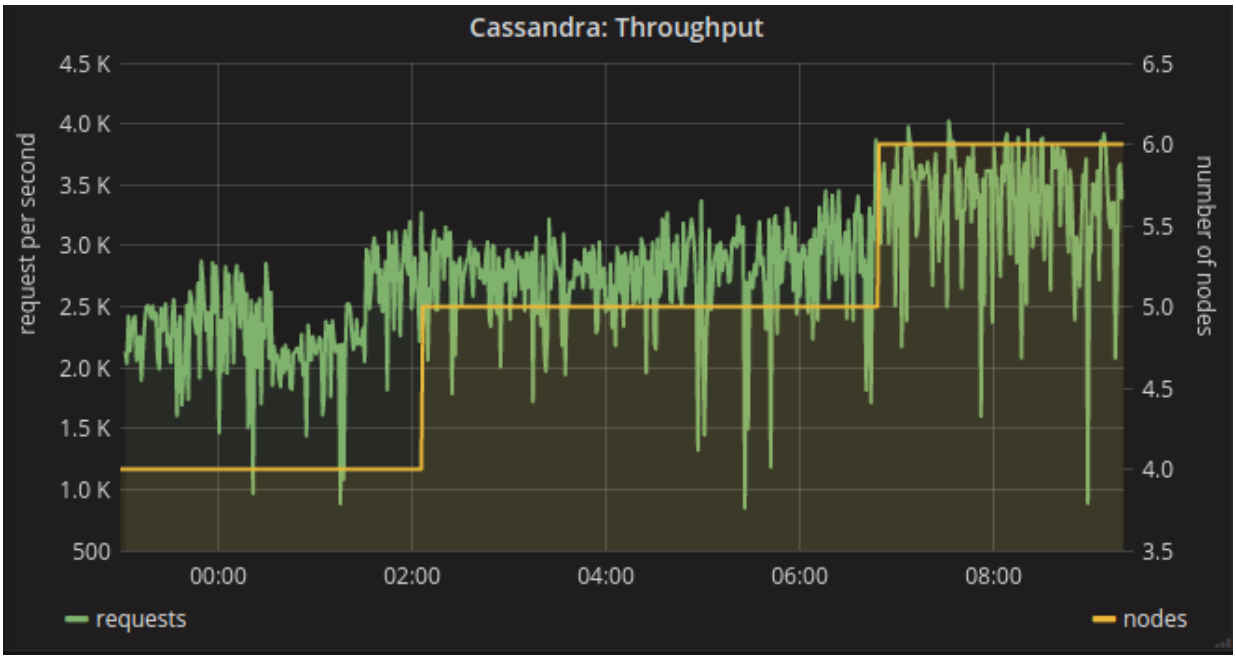

Figure 9. Throughput of Cassandra cluster during cluster growth.

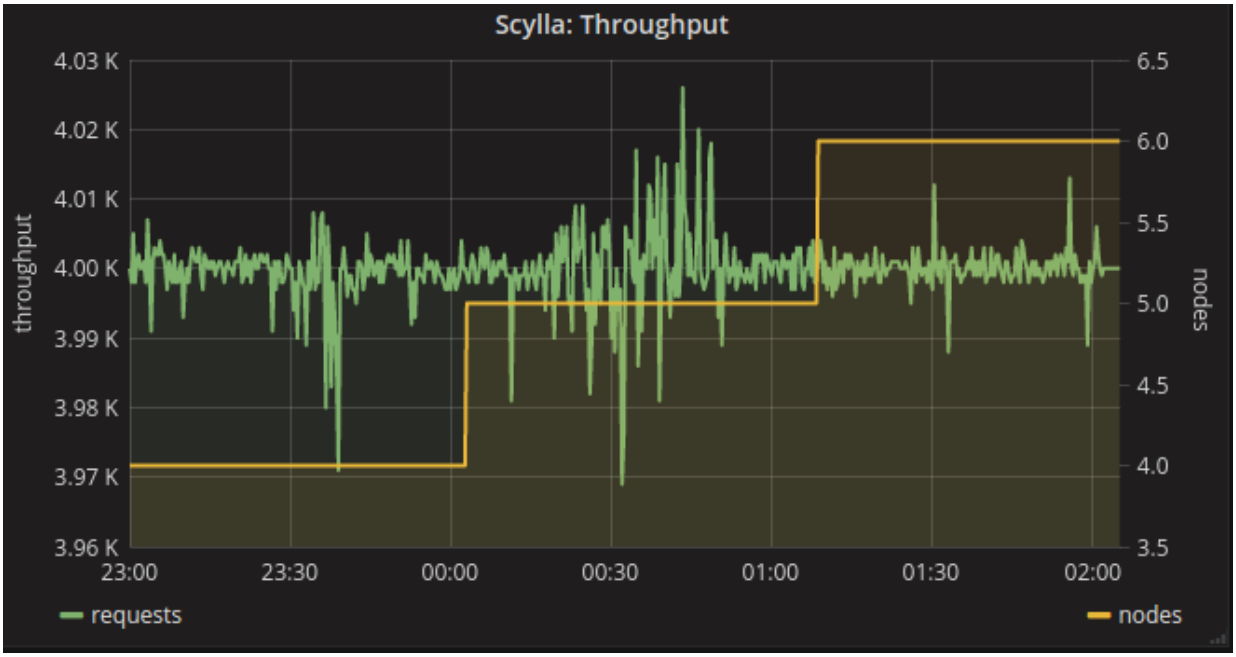

Figure 10. Throughput of ScyllaDB cluster during cluster growth.

3.1.2.2 Latencies

Let's examine the percentile latencies which are the latencies in which x% of the request are completed. These latencies are computed by cassandra-stress one time per 10 second hence providing a lots of samples that can be used to graph the evolution of the database response to the test.

	Scylla	Cassandra
Max 95% latency	100 ms	1500 ms

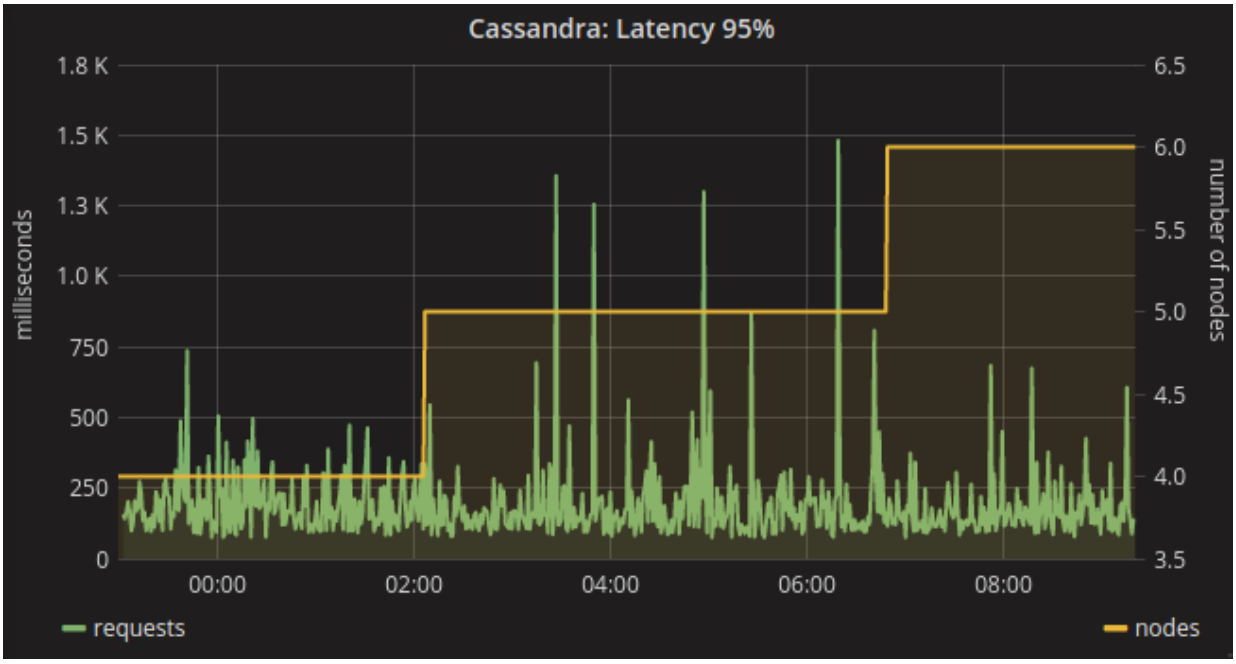

Figure 11. The 95 percentile latencies of Cassandra cluster during cluster growth.

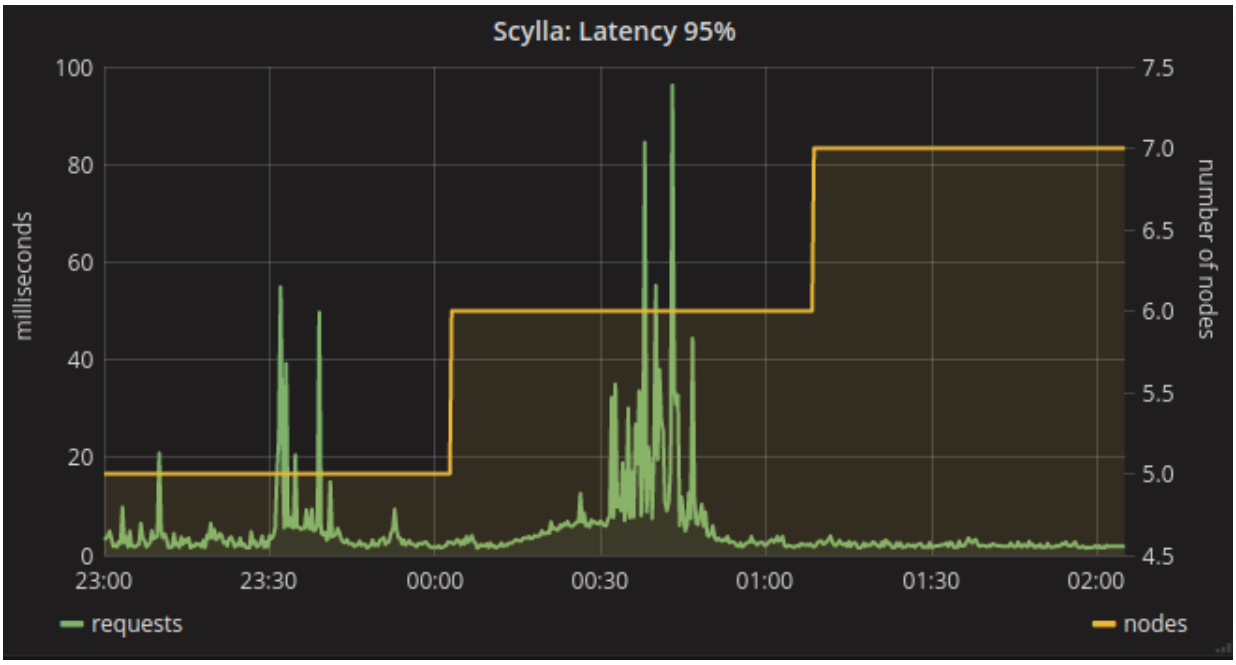

Figure 12. The 95 percentile latencies of ScyllaDB cluster during cluster growth.

ScyllaDB percentile latencies are significantly lower than Cassandra's. They are also more stable during the growth of the cluster.

The following table compares mean latencies of ScyllaDB and Cassandra clusters.

	Scylla	Cassandra
Max mean latency	26 ms	220 ms

Mean latencies show even bigger difference between ScyllaDB and Cassandra with an obvious advantage for Scylla.

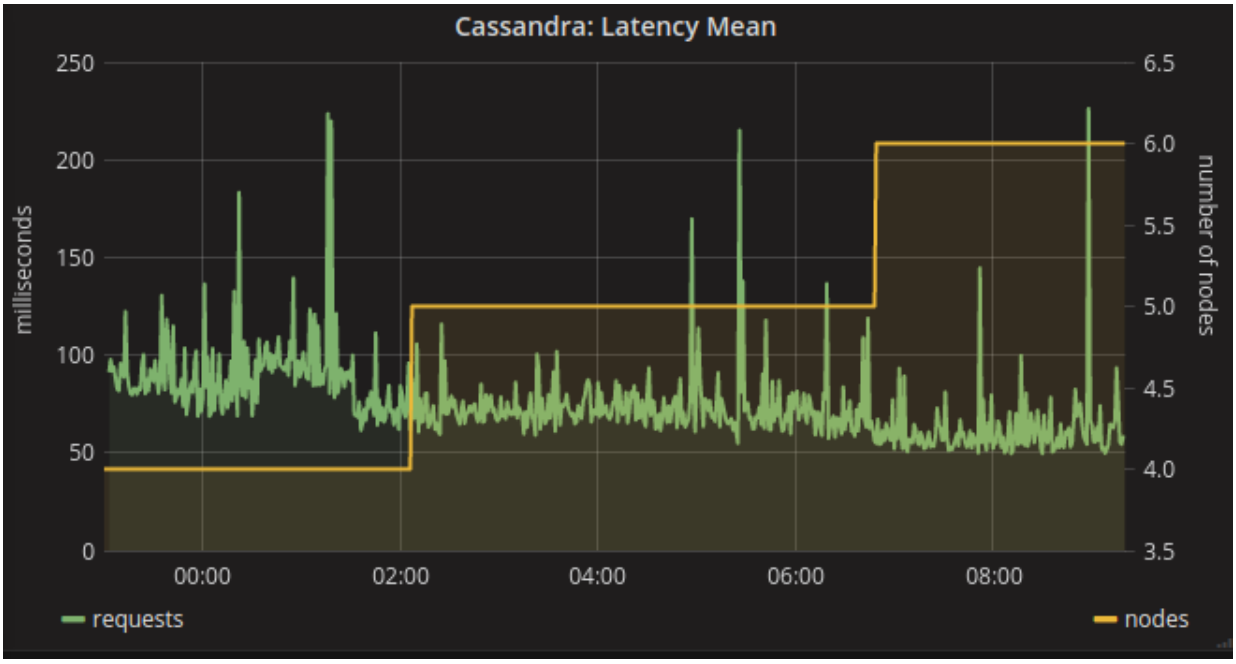

Figure 13. Mean latencies of Cassandra cluster during cluster growth.

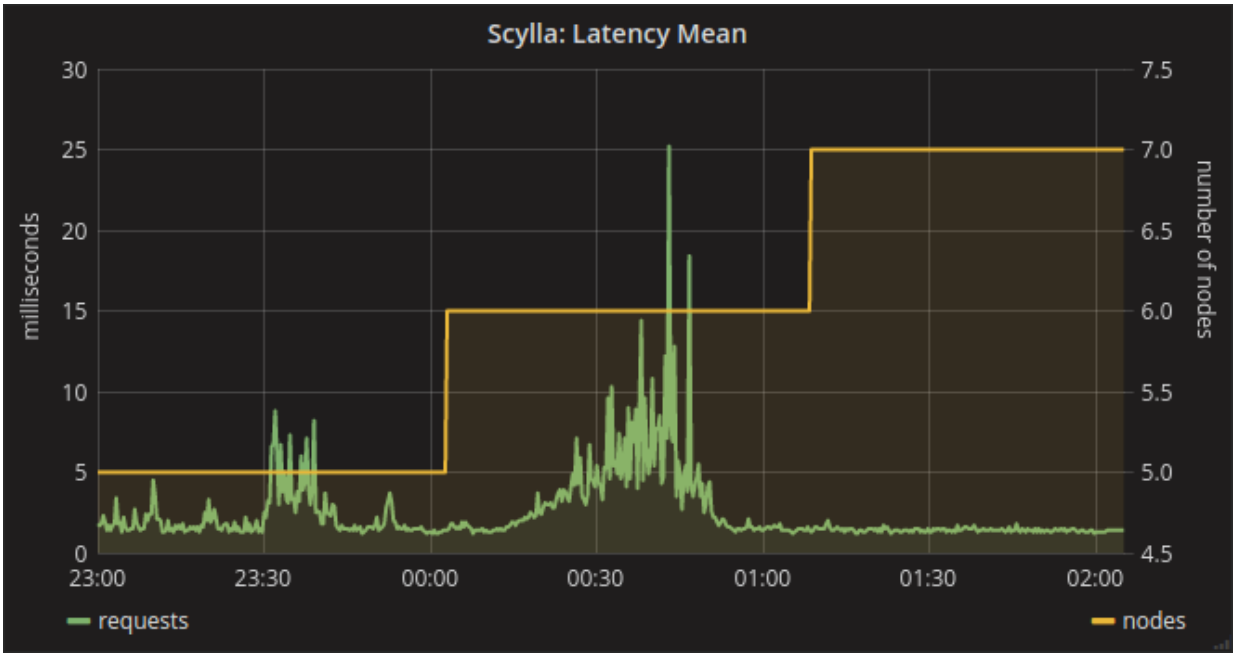

Figure 14. Mean latencies of ScyllaDB cluster during cluster growth.

3.2 Data to be used in experiment: cassandra-stress schema

Cassandra stress is Cassandra benchmark test workload. It inserts or reads random data structured by a known schema in a Cassandra or ScyllaDB database. The cassandra stress schema is a couple of blobs (binary large object values) gathered in a column family - the ScyllaDB equivalent of a relational database table. These blobs will be filled as fast as the stress tool can provide the data. The query per second count will be measured while the cluster is grown (by adding new machines to it) as fast as the database can allow it. Using the cqlsh command the CQL schema used by cassandra-stress can be dumped.

```
cqlsh> DESC SCHEMA
CREATE KEYSPACE keyspace1 WITH replication = {'class': 'SimpleStrategy',
'replication_factor': '3'}  AND durable_writes = true;
CREATE TABLE keyspace1.standard1 (
   key blob PRIMARY KEY,
   "C0" blob,
   "C1" blob,
   "C2" blob,
   "C3" blob,
   "C4" blob
) WITH COMPACT STORAGE
   AND bloom_filter_fp_chance = 0.01
   AND caching = '{"keys":"ALL","rows_per_partition":"ALL"}'
   AND comment = ''
   AND compaction = {'class': 'SizeTieredCompactionStrategy'}
   AND compression = {}
   AND dclocal_read_repair_chance = 0.1
   AND default_time_to_live = 0
   AND gc_grace_seconds = 864000
   AND max_index_interval = 2048
   AND memtable_flush_period_in_ms = 0
   AND min_index_interval = 128
   AND read_repair_chance = 0.0
   AND speculative_retry = '99.0PERCENTILE';

The crux of this schema is the following:

It create a keyspace which is a kind of name space used to isolate the various database
tables.
It create a CQL table that can store four Blob (Binary large object) in the database and
set up sensibles parameters for this table.
```

3.3 Validation by end-users

The cloud bursting is a synthetic benchmark that was designed specifically for the purposes of the MIKELANGELO project. It is designed to mimic a typical sudden increase of the workload to the existing service and evaluate the behaviour of this service and the underlying infrastructure. The use case offers an ideal benchmark for the evaluation of the changes

made to the virtualization layer (sKVM) and the improvements of the guest operating system (OSv) efficiencies.

The Original use-case: The dynamic nature of the burst itself is an example of a workload that potentially could benefit from the increased allocation of I/O cores during the burst, handled automatically by the IO core manager. Zero-Copy Receive furthermore reduces latencies by removing unnecessary copies of the data travelling the virtualised network stack. vRDMA and UNCLOT on the other hand are not going to be evaluated by this use case. The former because it will not be integrated into the cloud test environment, and the latter because it only affects collocated OSv instances which are not suitable for this kind of use cases.

The Next Gen use-case: This use case is furthermore a true demonstrator of the capabilities of a modern programming model along with a production-grade framework taking all the latest developments of CPUs and networks. This entire time, Seastar API has been improving towards a stable alternative to existing APIs and has been enabling ScyllaDB to get the optimal performance of the underlying infrastructure, including during the burst scenario. The evaluation of this scenario will benefit ScyllaDB in our efforts to further exploit the technology.

Finally, we repeat this use-case is intended to demonstrate improvements in the key ingredient of Cloud technology: bursting. Furthermore, it uses its cornerstone software - the Cassandra database (or its high-performance drop-in replacement: ScyllaDB). Such demonstration is of particular importance in onboarding of end-users to MIKELANGELO technologies, as it clearly demonstrates performance gains and benefits.

4 Virtualized Data Analytics Implementation Strategy

The virtualized data analytics use case revolves around synthetic and real-world data analytics workloads in a cloud testbed. The central value proposition for the use case of the MIKELANGELO stack lies in the improved IO performance achievable by harnessing sKVM and the cloud integration.

The use case features two synthetic data analytics workloads and two real-world workloads. The synthetic workloads have been taken from the CloudSuite benchmarking suite [15]. The first benchmark is called data-caching workload. The data-caching workload focuses on stressing multiple instances of memcached with requests, and thus features heavy network IO and memory pressure. The second benchmark is called web-serving workload. The web-serving workload combines nginx, a PHP-instance, and memcached with an HTTP load-generator to simulate a web application in the cloud. The simulated web application processes graph-based data and stresses the network subsystem.

The real-world use cases are called mining-software-repositories use-case (MSR) and digital-humanities use-case (DH). Both use cases have been chosen and are being implemented in collaboration with partners of GWDG. The MSR use-case features a data analytics application for research in mining software repositories. The use case is implemented and evaluated in collaboration with the informatics institute of the University of Göttingen. The DH use case comes from a collaboration with colleagues within GWDG. The DH use case originates from a project that analyzes data from Europeana [17]. This analysis focuses on machine learning to assess the quality of metadata on assets from digital humanities. These assets include ancient books and paintings.

The overall data analytics use case runs experiments based on the two synthetic and two real-world workloads in GWDG's integrated cloud testbed. There are two main parts to the implementation of the use case. First, the workloads have to be prepared to be run in the cloud testbed. Second, the workload needs to be automated via Scotty to allow for simple experimentation. When we achieve both goals, we will be able to run a multitude of experiments with Scotty and collect resulting data automatically. This approach will allow us to analyze the advantages and disadvantages of the MIKELANGELO stack for our four data analytics workloads.

Table 4 shows which features of the MIKELANGELO stack we will evaluate in the data analytics use case. Out of 14 features 5 will certainly be evaluated. An additional 4 features will be evaluated if possible. Finally, 5 features are out of scope and will definitely not be evaluated in this use case.

The features definitely to be evaluated are IOcm, SCAM, MCM, Scotty, and Snap. Since the data analytics use case runs on the cloud testbed, all components in the cloud testbed will automatically be evaluated. These automatically evaluated components include IOcm, SCAM, MCM, and Snap. Additionally, the use case will definitely evaluate Scotty because the use case's automation relies on Scotty.

The tentatively covered features are ZeCoRx, OSv, LEET, and UNCLOT. For ZeCoRx the estimated completion date is to be confirmed. Thus, the evaluation of ZeCoRx hinges on a sufficiently early completion of the feature. Furthermore, ZeCoRx uses a newer kernel than IOcm, vRDMA, and SCAM, because the effort is focused on upstreaming compatibility, which requires developing on the latest kernel. Thus, the cloud testbed will require a full upgrade including the host kernel (from 3.18 to 4.), the operating system (Ubuntu 14.04 LTS to Ubuntu 16.04 LTS), and OpenStack (Liberty to Ocata). The evaluation of OSv, LEET, and UNCLOT through the data analytics use case hinges on the availability of Spark in OSv. Experiments in the past have shown that OSv will not improve performance for MongoDB and for HDFS. Thus, XLAB currently pursues the integration of Spark with OSv. If Spark will run in OSv and promises at least similar performance to Ubuntu we will evaluate the OSv integration for the data analytics workloads as well. LEET and UNCLOT can only be evaluated if Spark runs in OSv, since both build on OSv.

The definitely excluded features from this use case evaluation are vRDMA with OSv, vRDMA with Linux, vTorque, Seastar, and ScyllaDB. Since vRDMA does not work without RoCE-capable NICs, it cannot be run in the cloud testbed. By extension both versions, OSv and Linux, of vRDMA cannot be evaluated by the data analytics use case. Since the data analytics use case relies on the cloud testbed vTorque will not be evaluated here as well. ScyllaDB will not be evaluated in the data analytics use case, because the four mentioned workloads use other databases by design. The synthetic benchmarks use memcached as a cache and MySQL for permanent data store. ScyllaDB, as a NoSQL-DB does not fit their profile and thus is not suited as a replacement. The two real-world use cases rely on MongoDB. Although an integration with Cassandra and ScyllaDB would have been beneficial for the evaluation of the MIKELANGELO stack, the use case stakeholders are set on MongoDB. By extension Seastar will also not be evaluated, since ScyllaDB is the only application that builds on Seastar.

Table 4. Coverage of MIKELANGELO components by the virtualized data analytics use case.

Use Case/Component	Data Analytics
IOcm	y
ZeCoRx	m

SCAM	y
vRDMA/OSv	n
vRDMA/Linux	n
OSv	m
MCM	y
Scotty	y
LEET	m
UNCLOT	m
vTorque	n
Snap	y
Seastar	n
ScyllaDB	n

Validation by end users will be performed by the collaborators in the MSR and DH real-world use-cases. Experiments with both workloads will be performed, managed by Scotty. These experiments will be evaluated based on a set of metrics derived by the end users. These metrics will measure improvements in QoS and management of data analytics workloads in the cloud.

4.1 Experimental Setup

The experimental setup for the data analytics use case has two major parts: the setup based on Scotty and the implementation of the use cases. The development of Scotty has been driven foremost by this use case and thus can be seen as extension of the use case. Furthermore, the development of synthetic and real-world workloads to run in Scotty accounts for the main effort in the data analytics use case.

The general experimental setup with Scotty consists of three main parts: Scotty's architecture, the MIKELANGELO components to be evaluated, and the experimental configurations to test.

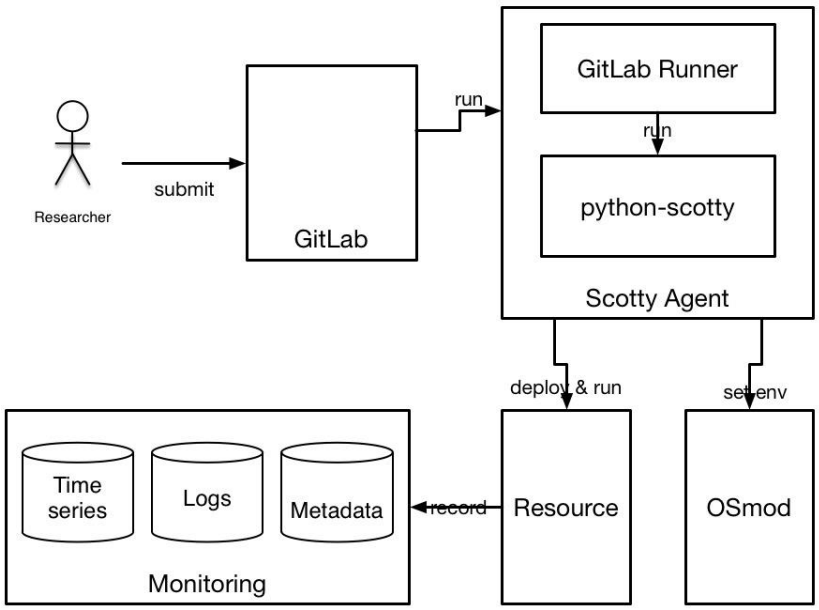

Figure 15. Architecture diagram for Scotty.

The MIKELANGELO components to be evaluated are shown in Figure 16. The illustration reflects the coverage matrix in Table 4. Here we briefly described how the components are going to be evaluated as part of the data analytics use case. The green colour refers to MIKELANGELO components, which will be tested with certainty. The orange colour refers to components for which it is uncertain whether we will be able to evaluate them for data analytics. The grey components refer to external resources, which have not been developed by MIKELANGELO.

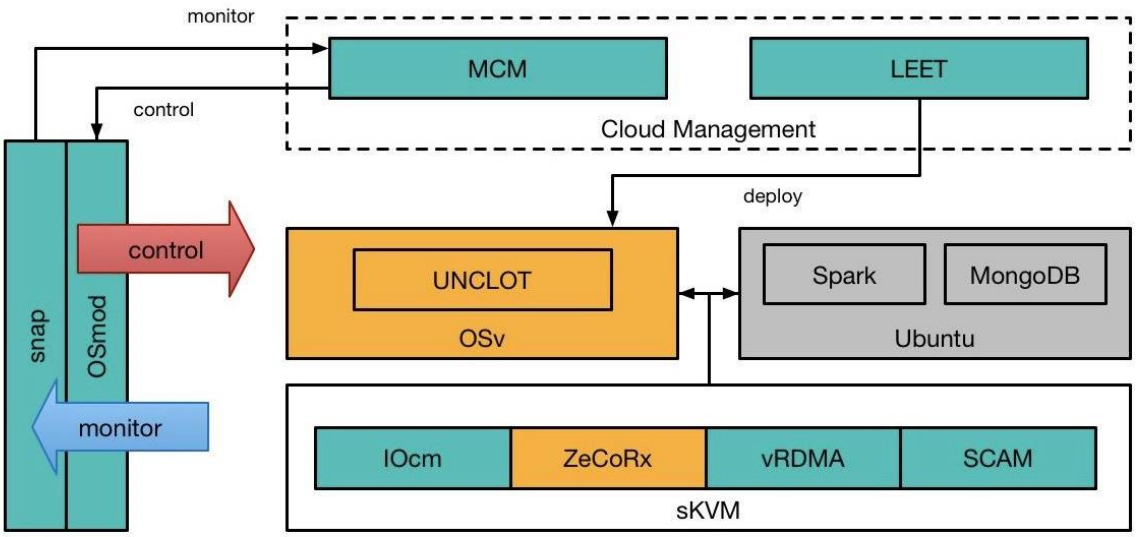

Figure 16. General architecture for the virtualized data analytics use case.

IOcm will run on each of the hosts in the testbed. Experiments with all four workloads will be performed with IOcm turned off and on. We expect to see improved IO throughput in our workloads when running IOcm on. The throughput will be measured by requests per second, transactions per second and raw throughput on virtual NICs.

SCAM will run on all hosts in the testbed as well. SCAM will be switched on and off to evaluate the impact of attack monitoring with SCAM on the QoS of the running workloads. Furthermore, SCAM attacks may be simulated in parallel to other workloads to assess the detection rate of the SCAM monitoring system.

ZeCoRx will run on all hosts and it will be turned on by default. Thus, for an evaluation of ZeCoRx we will rerun experiments previously performed on the IOcm-based testbed. The challenge to evaluate ZeCoRx will be to obtain comparable results in time. ZeCoRx can be evaluated in a similar manner to IOcm. By loading kernel modules with and without ZeCoRx, we can evaluate ZeCoRx's impact on the use case. One uncertainty regarding the evaluation of ZeCoRx in this use case remains. ZeCoRx builds on a newer Linux kernel than IOcm and SCAM. Due to the newer Linux kernel the whole testbed, including libvirt, OpenVSwitch, and OpenStack will need to be upgraded as well. One potential approach will be to deploy the upgraded testbed once with ZeCoRx and once without and to run parallel tests on those configurations.

Snap will be evaluated by using a snap installation by default for all metering in the testbed. Additionally many of the collectors developed based on MIKELANGELO requirements will be run in the experiments.

OSmod will be evaluated in two scenarios. First, OSmod will be used by Scotty to set the system parameters as required by the experiment. Second, OSmod will also be used by MCM as a resource management system. Thus, running Scotty-based experiments and evaluating MCM strategies will evaluate OSmod's functionality directly.

MCM will be evaluated in some of the experiments. MCM will implement resource management algorithms, which should improve arbitrary objectives. These algorithms will make use of all of the aforementioned components. Furthermore, MCM algorithms can control all features implemented in OSmod. These features currently span the control of MIKELANGELO components. However, we intend to extend those features for the final evaluation to other parameters, such as host hardware configuration (TurboBoost, Cache Partitioning, CPU Pinning), VM configuration (e.g. vCPUS, memory), and host OS configuration (e.g. IO and CPU scheduler, cgroups strategies).

Furthermore, OSv, LEET, and UNCLOT may be evaluated by the data analytics use case. OSv may provide improved IO performance and quicker start-up times for data analytics clusters.

LEET should provide a seamless deployment of clusters, such as Spark clusters for data analytics. Finally, UNCLOT should provide improved communication between VMs on the same host by using IVSHMEM.

The mentioned experiments will be run in isolation and in combination. To assess beneficial combinations of components, we will devise experiments that combine configurations that span multiple components. Thanks to Scotty, running those experiments and collecting data from the experiments will be fairly easy.

Both types of workloads, synthetic and real-world, require implementations with some complexity themselves. In the subsequent paragraphs we introduce their architectures and briefly discuss their setup.

The two synthetic benchmarks, CloudSuite Data Caching and CloudSuite Web Serving share a similar architecture of the computing infrastructure. However, the application layer is different for both. Both workloads first rely on a virtual infrastructure provisioned by OpenStack. The infrastructure is provisioned via OpenStack Heat. We have built a Heat template that creates virtual machines and a network for each instance of a workload. Additionally, Heat orchestrates the installation scripts for software within the virtual machines. Inside the virtual machines both synthetic workloads run a Docker Swarm cluster. Thus the first step of the Heat-based software installation takes care of installing and configuring Swarm. This step includes the installation of a key-value store, of the Swarm Manager and of Docker Machine instances on the worker nodes. Beyond this common infrastructure both workloads install different application-layer components.

The architecture of the data caching workload is shown in Figure 17. There are two main components for the workload: a memcached stressor and and multiple memcached instances. The stressor contains a memcached client and runs on the same VM as the swarm manager. The memcached instances run in the worker VMs. The client as well as the memcached servers are deployed as Docker containers. The swarm manager manages the provisioning and distribution of the individual containers. Besides the automation to deploy and run the workload, we have integrated snap collectors within the workload. One of those collectors has been custom-developed to provide the benchmark results live to snap. The memcached benchmark can be scaled arbitrarily by allocating more memcached instances. The experimenter can set the scaling parameter in Scotty's experiment.yaml. The data caching workloads features generates a lot of IO operations. Since the workload is distributed the IO operations translate to network operations, which in turn should see a speedup with IOcm and ZeCoRX.

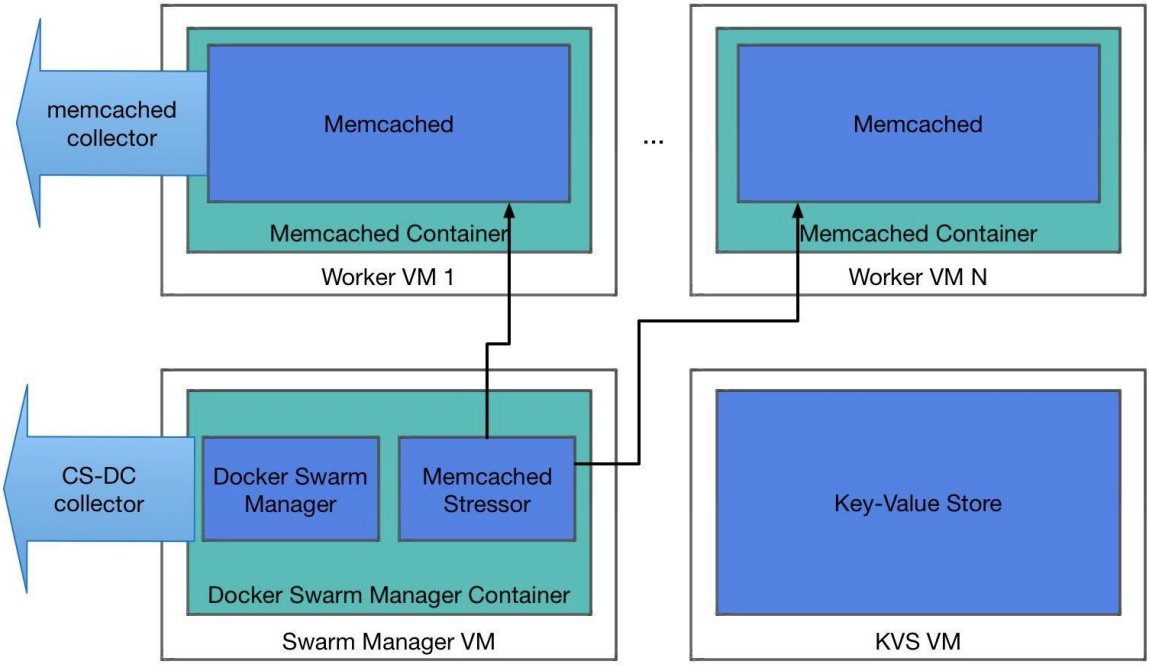

Figure 17. Architecture for the data analytics workload CloudSuite Data-Caching.

Figure 18 shows an example for a Scotty definition in YAML format [16] with the data caching workload. The snippet describes the experimental environment in terms of IOcm setting, MCM strategy, and experiment tag. Furthermore, the workload generator is set to the data caching workload generator with parameters that are specific to the data caching workload.

```
workload:
  name: Cloud_Suite_data_caching
  generator: csdc
  params:
    server_no: 4
    server_threads: 4
    memory: 4096
    object_size: 550
    client_threats: 4
    interval: 1
    server_memory: 4096
    scaling_factor: 30
    duration: 0
    fraction: 0.8
    connection: 200
  environment:
    iocm_state: true
    mcm_strategy: default
    snap_tag: exp-id
```

Figure 18. Example for a Scotty experiment YAML for the data caching workload.

The web serving workload builds on the same Docker-based infrastructure setup as the data caching workload. The Swarm Manager again runs the stressor for the experiment. The stressor is an HTTP request generator, called Faban client [12]. The worker nodes host a distributed web application, called Elgg [13]. Elgg is a framework for social media web sites, which is used regularly for benchmarking. Elgg is a PHP application, which in our setup is served by an nginx web server. The application uses a memcached instance to buffer requests. Finally, an instance of MariaDB server [14] is used as a permanent data store. The workload cannot directly be scaled, as is the case with the data caching workload. However, we can run multiple instances of the workload at the same time, which simulates scaling.

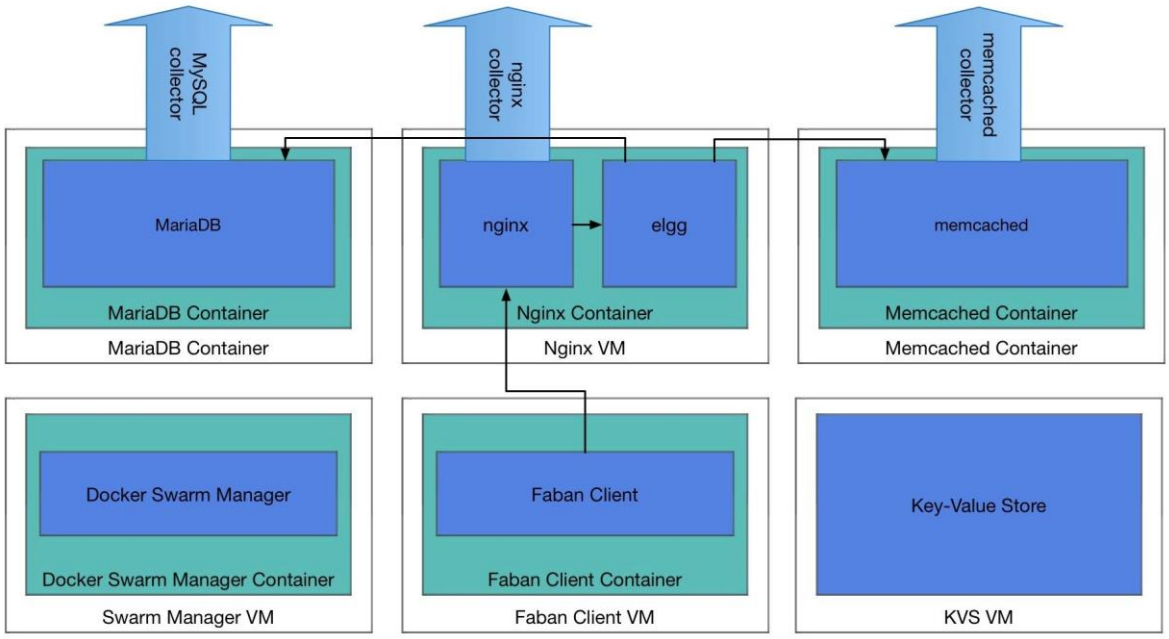

Figure 19*:* Architecture for the data analytics workload MSR.

Figure 20 shows an experiment.yaml for the web serving benchmark. Besides the automation of the benchmark, we have integrated snap collectors for MariaDB and for nginx in the workload. The experiment is analogous to the data caching experiment above. However, the workload section refers to the web serving benchmark and its parameters.

```
workload:
  name: Cloud_Suite_Web_Serving
  generator: csws
  params:
    server_no: 3
    fpm_max-child:80
    load_scale: 7
  environment:
    iocm_state: true
```

```
    mcm_strategy: default
    snap_tag: exp-id
```

Figure 20. Example for a Scotty experiment YAML for the web serving workload.

Both real-world workloads rely on OpenStack and Heat. The main components of both workloads are MongoDB and Spark. Both run as a cluster across multiple VMs. The MSR workload, as shown in Figure 21, provides a web application as a frontend. The web application is operated outside of the use case. In the production setting the web application manages the execution of jobs in an HPC cluster and in Spark. The HPC cluster lies outside of our use case implementation as well. The HPC cluster crawls git repositories off the internet and feeds the raw repositories in the distributed MongoDB installation. The MongoDB installation runs in the MIKELANGELO cloud testbed. The Spark cluster analyzes the data in MongoDB with machine learning algorithms to answer research questions. This research, for example, tries to find indicators for successful software projects from the behaviour of contributors. The Spark cluster runs on the cloud testbed as well. Currently, the setup on the cloud testbed is used for a production deployment of the workload. To provide comparable results, we will create automated workloads based on the production deployment. These automated workloads will be managed by Scotty and used to evaluate the MIKELANGELO stack like the synthetic benchmarks.

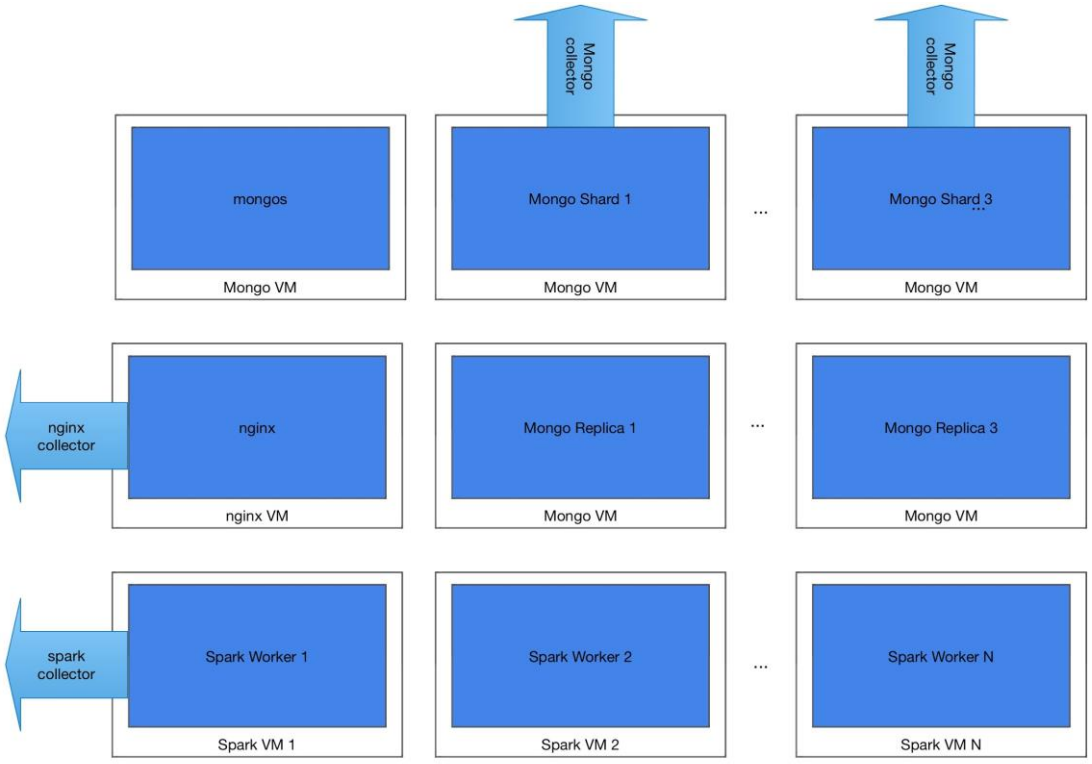

Figure 21. Architecture for the data analytics workload MSR.

The DH workload relies on MongoDB and spark as central components as well. Additionally, again there is a web application that serves as the user-frontend. However, the web application is hosted on machines outside of the MIKELANGELO cloud testbed. The use case works on a growing collection of metadata on a large set of assets from the digital humanities. The metadata is available as JSON-based files. In the DH use case machine learning provides estimators for the quality of the metadata. To provide comparable results, across runs with the data, we will create an experiment using Scotty. These experiments will run the same analytical jobs with different configurations of the MIKELANGELO stack.

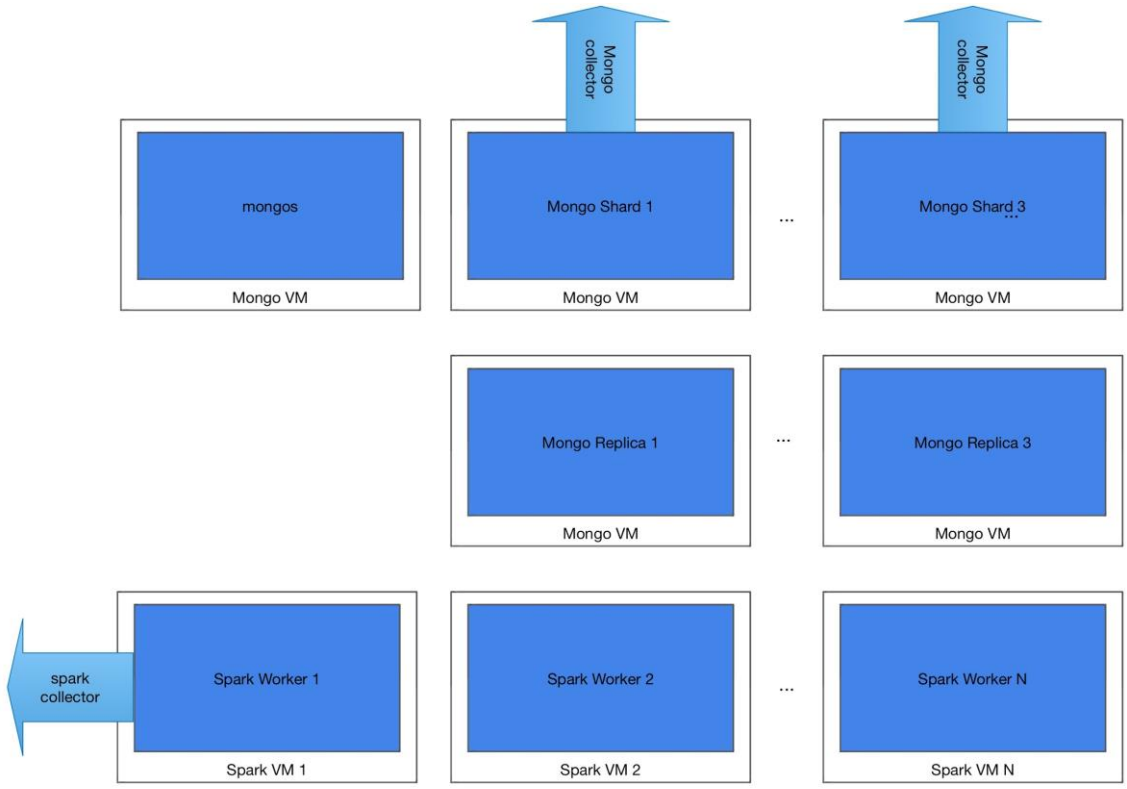

Figure 22. Architecture for the data analytics workload DH.

4.2 Data to Be Used in Experiments

The four workloads in the data analytics use case process different types of data. The synthetic data caching workload operates on a pre-loaded data set with graph-based data from Twitter. The Twitter data set has been collected and published for research and is publicly available [19]. The dataset contains users as nodes and relations betweens users as edges. There are 12 million nodes and 85 million edges in the Twitter data set. When running the data caching workload the dataset is scaled by sub-sampling or super-sampling to adjust

the execution time. The web serving benchmark generates HTTP requests using the Faban client [12]. The distribution of the requests can be adjusted when running the workload.

The data for the real-world workloads is a subset of the use cases actual dataset. Due to its size, we cannot perform experiments running over the whole dataset. The MSR workloads processes raw git repositories that are publicly available on the internet. The digital humanities workload processes Europeana metadata in the form of JSON files. The datasets for the real world workloads have been described in greater detail in their respective sections in D6.3 [18].

4.3 Validation by End Users

The validation of the MIKELANGELO stack through the data analytics use case will happen through the synthetic and real-world workloads. The synthetic workloads have been chosen to represent relevant workloads that are often found in a (semi-) public cloud. Thus, we expect that an improvement in the two chosen CloudSuite benchmarks will translate in an overall improvement in a typical cloud setting. The two real-world workloads have been chosen as actual use cases for the MIKELANGELO stack. Both use cases would already be developed for the GWDG cloud. Instead of running them there directly, however, we chose to develop them in the MIKELANGELO cloud testbed and to use the MIKELANGELO testbed to improve their performance. Once the project ends, these use cases will be transferred into the GWDG cloud for production. Thus, both real-world workloads are directly relevant to its stakeholders.

Validation of all four workloads relies on the improvement of application-specific markers. For the data caching workload we expect an improvement in the throughput of requests to memcached. For the web serving benchmark we expect an improved throughput of http requests to elgg. For the MSR workload we expect an increased throughput for storing new git repositories in MongoDB. For the DH workload we expect a reduced execution time to process a given set of metadata objects. The metrics for validation have been discussed with the stakeholders. The experiments will be performed and measured using Scotty. The final evaluation of the experiments will be conducted together with the use case stakeholders.

5 Aerodynamic Maps Implementation Strategy

5.1 Introduction

The Aerodynamic maps use case represents one of the two use cases of the MIKELANGELO project that are typically exploiting HPC infrastructures for their performance and increase the size of the domains being processed. One of the main objectives of this use case has been to steer the requirements for technical work-packages and offer an easy-to-use validation and evaluation base. The latter is particularly important because it can be used to evaluate components individually or fully integrated into target stacks (cloud and virtualised HPC).

Pipistrel, as the owner of the use case, entered the project expecting the results of the project to bring several benefits. Even though the MIKELANGELO is mainly about improving the performance, the main expectation for this use case is improved flexibility. This includes agility in terms of simulation management and elasticity of application deployment. It also requires omnipresent monitoring, i.e. getting data from all layers of the processing stack: infrastructure, virtualisation, networking and, above all, application level monitoring (for example, monitoring of OpenFOAM metrics).

This use case is going to evaluate most of the components of the MIKELANGELO technology stack. The most notable exceptions are the ones related to the Cloud Bursting use case, i.e. the Seastar library and ScyllaDB database, as they are irrelevant for the case of OpenFOAM simulations. ZeCoRx and vRDMA components are going to be evaluated if they are going to be provided on time. All other components will be evaluated for performance and flexibility. Exact details on how the components are going to be evaluated are given in section 5.4.

Table 5. Coverage of MIKELANGELO components by the aerodynamics use case.

Use Case/Component	Aerodynamics
IOcm	y
ZeCoRx	m
SCAM	y
vRDMA/OSv	m
vRDMA/Linux	m
OSv	y
MCM	y

Scotty	y
LEET	y
UNCLOT	y
vTorque	y
Snap	y
Seastar	n
ScyllaDB	n

The implementation of this use case consists of two approaches, one specifically targeting the cloud environment (OpenFOAM Cloud) with an integration with OpenStack and the other focusing on integration, validation and evaluation in the MIKELANGELO HPC environment.

OpenFOAM Cloud is a web-based application that has been introduced as part of the integration efforts early in the project. The initial implementation already allowed for a significantly simplified specification of OpenFOAM-specific workloads. The application consists of a frontend and backend. The frontend is based on the OpenStack Horizon project[2]. The version that was initially used was based on OpenStack Liberty, which is a bit outdated but still serves the purpose of demonstrating of the OpenStack integration. The backend is a standalone Django-based application. The main focus of the current project phase was the implementation of the infrastructure provider support. Initially, the backend was tightly coupled with the OpenStack IaaS as it was merely an extension of the frontend. However, it quickly became evident that this tight coupling prevented some potential exploitation options beyond this use case implementation. In the second project year, we contacted a representative of SimScale[3] and held a web conference presenting the MIKELANGELO project and the concepts behind it. SimScale has been identified as a potential exploitation path for this use case as it relies on extensive compute resources, currently running on Amazon Web Services[4] IaaS. To this end, we refactored the backend and made it provider-aware. The backend introduced a custom scheduler and an implementation of OpenStack and Amazon AWS provider connectors allowing users to deploy simulations on private and public OpenStack deployments and Amazon infrastructure.

The backend furthermore added support for parallel execution of the use cases using MPI-enabled OSv instances. Current implementation requires that the input data is preprocessed

[2] OpenStack Horizon, https://docs.openstack.org/developer/horizon/
[3] SimScale homepage, https://www.simscale.com/
[4] Amazon Web Services, https://aws.amazon.com/

by the user, i.e. the input 3D domain is already decomposed into the required number of subdomains, however the plan is to allow automatic decomposition (pre-processing) and result reconstruction (post-processing) in the backend itself.

Figure 23 shows a high level design of the OpenFOAM Cloud architecture. As described in the previous paragraphs, the frontend is a dashboard integrated into OpenStack Horizon. The backend implements the API server and a dedicated simulation database. The former provides a unified REST API towards the compute resources, configuration of simulations, monitoring and collection of the simulation results. Being offered through an API, it is possible to integrate the backend into an arbitrary frontend. The simulation database stores all relevant information about the simulations, the states and results.

As the diagram depicts, the backend is tightly coupled with Capstan and UniK tools (part of LEET). Capstan is used to compose VM images on the fly based on the simulation specification. UniK, and in particular the OpenStack provider developed by MIKELANGELO, is used to communicate with OpenStack, staging composed images, deploying simulations and configuring the networking. The MPI component is optionally used for parallel simulations and is the main entry point used by the backend towards the OSv-based workloads.

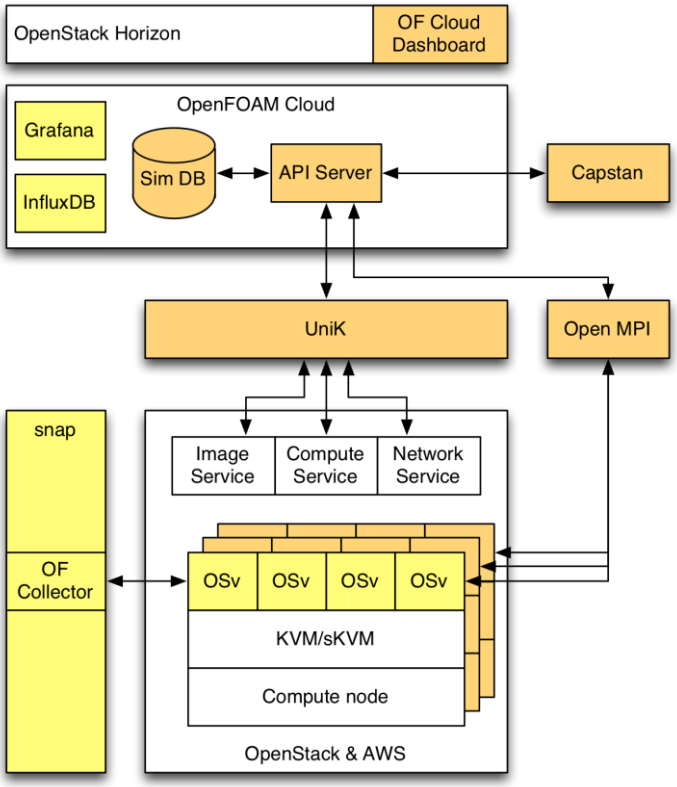

Figure 23. Architecture diagram of the OpenFOAM Cloud.

Finally, snap is being used to monitor the metrics relevant to the simulations. A dedicated OpenFOAM data collector[5] has been slightly updated for a more robust collection of simulation metrics.

One of the main advances since previous releases of the OpenFOAM Cloud has been the introduction of job schedulers that can be used to deploy OSv-based workloads in any of the pre-configured cloud providers. Currently OpenStack and Amazon Web Services are supported. Multiple providers of the same type can be configured (for example, it is possible to configure an internal OpenStack, external OpenStack cloud and mix them with Amazon cloud, all at the same time). Every provider has its own quotas giving full control over expenses (first, all private resources will be used, followed by public clouds in the order of priority).

The scheduler monitors the instances throughout the execution lifecycle ensuring instances are active only for the duration of the actual work necessary. As soon as resources are freed, any pending job will be brought into life, thus ensuring the requested parametric study optimally utilises the cloud resources available.

Contrary to the Cloud version of this use case, the efforts made in the HPC environment mainly revolve around making the use case evaluate as many components of the MIKELANGELO technology stack as possible. The use case started experimenting with OSv early in the project to ensure requirements for OSv emerged as soon as possible. This allowed us to showcase an OpenFOAM application running as a unikernel by the end of the first year. Initial experiments provided specific requirements for OSv operating system. On one hand these requirements were related to the missing functions of the system. However, the majority of them were targeting the parallel execution using Open MPI[6].

In the last project phase we thus focused on the evaluation of the achievements of work-packages WP4, WP5 and WP6. To ensure comparable results we have integrated the use case with the Scotty experimentation framework as partly explained in section 5.2 (Experimental Setup).

5.2 Experimental Setup

Two different types of aerodynamic analyses have been planned for this use case. The first analysis deals with numerous similar OpenFOAM cases run simultaneously. This requires a parametric study of a typical simple aerodynamic industrial configuration (for example, a 2D airfoil and a 3D propeller in front of a wing) in order to obtain results at all required sets of

[5] OpenFOAM snap collector, https://github.com/intelsdi-x/snap-plugin-collector-openfoam
[6] Open MPI homepage, https://www.open-mpi.org/

parameters. Since a single OpenFOAM case examining a single parameter isn't computationally very demanding, a large number (depending on the number of available computer resources) of such simulations can be run at the same time. Relevant studies can contain from 100 to 1000 such OpenFOAM cases. The integrations already done in MIKELANGELO as well as those planned for the last phase of the project allow Pipistrel to deploy and monitor all simulations simultaneously and, at the end, to collect and display all relevant results. The transition from current scripts on a small in-house cluster to a MIKELANGELO stack deployed on a much larger computing infrastructure will, using a user-friendly dashboard, simplify and accelerate Pipistrel's workflow and possibly also reduce fixed operating costs in the future. Since these kinds of simulations are so typical in scientific areas, the results of this use case are also potentially exploitable by other interested stakeholders.

The second type of analysis deals with a single computationally very intensive simulation (distributed propulsion system with multiple propellers in front of a single wing) using all available computing resources. Such a need arises when a set of parameters is already chosen but a physically or numerically more precise (high fidelity) simulation is needed. This case is a typical HPC problem and is ideal to evaluate the MIKELANGELO technology stack as a baseline in order to help increase the virtualised I/O efficiency of the MIKELANGELO cloud stack.

Initially all experiments have been performed manually which proved to be a rather cumbersome task, even when executed seldomly. With the more flexible integration of the cloud and HPC testbeds, providing more MIKELANGELO components, repeating experiments became even more error-prone and resulted in unmanageable benchmarks. To this end we have partnered with GWDG and USTUTT to provision repeatable experiments using Scotty. Scotty uses an approach common to modern configuration managers, such as Ansible and Puppet, where everything is code (Configuration as a Code). It allows one to specify one or more experiments in a way that enables reuse of experiments under varying conditions. It is also fairly easy to run the same workload multiple times and extract performance results, collected by snap.

For the HPC workloads, Scotty integrates the HPC workload generator. This acts as an intermediary worker between Jenkins CI and vTorque submitting the HPC jobs to the testbed. The experiment metadata defines exactly which experiment to run as well as what configuration of the HPC testbed to use, such as number of nodes, number of cores per node, use of IOcm and vRDMA.

This use case provides several experimental setups that are used throughout the project. So far, mostly smaller (faster to execute) use cases have been used. These fast experiments are periodically executed to ensure functional requirements of the integrated testbed are

properly satisfied. Approaching M30 of the project, experiments with bigger input cases have also been prepared and evaluated. Due to the large amount of data these experiments exploit the HPC infrastructure much more giving insights into the performance data of technologies such as vRDMA and IOcm. The simplest Scotty configuration of an experiment is presented in the following listing. The experiment defines four workloads varying only the number of physical nodes the experiment is executed on (1, 2, 4 and 8). In all cases, all 16 cores of a node are used totaling to 16, 32, 64 and 128 cores per simulation. No virtualisation is requested by this experiment.

```
- workload:
    name: 004-biggerCase-100-1_node
    generator: hpc-stuttgart
    params:
        job_script_name: 004-biggerCase-100.sh
        experiment_dir: experiment
        qsub_number_of_nodes: 1
        qsub_number_of_processes_per_node: 16
        application_log: ~/experiment_log/log
- workload:
    name: 004-biggerCase-100-2_nodes
    generator: hpc-stuttgart
    params:
        job_script_name: 004-biggerCase-100.sh
        experiment_dir: experiment
        qsub_number_of_nodes: 2
        qsub_number_of_processes_per_node: 16
        application_log: ~/experiment_log/log
- workload:
    name: 004-biggerCase-100-4_nodes
    generator: hpc-stuttgart
    params:
        job_script_name: 004-biggerCase-100.sh
        experiment_dir: experiment
        qsub_number_of_nodes: 4
        qsub_number_of_processes_per_node: 16
        application_log: ~/experiment_log/log
- workload:
    name: 004-biggerCase-100-8_nodes
    generator: hpc-stuttgart
    params:
        job_script_name: 004-biggerCase-100.sh
        experiment_dir: experiment
        qsub_number_of_nodes: 8
        qsub_number_of_processes_per_node: 16
        application_log: ~/experiment_log/log
```

Figure 24. Scotty configuration for the Scotty experiments with the aerodynamics use case.

The job script referred to in the workloads is shown next. It specifies the name of the OpenFOAM case, total number of simulation steps and where to get the input case from.

Smaller cases provide the input data as part of the code itself, however in this case, the input data is approximately 1.7 GB, rendering it impossible to be attached to the experiment configuration directly.

```
#!/bin/bash

################################################################################
# Start of experiment configuration                                            #
################################################################################

# Name of the input case data set
CASE=biggerCase

# Number of time steps to run in the simulation
TIME_STEPS=100

# Remote location of the input case
CASE_URL="https://xlab.koofr.net/content/links/59164cc6-8f47-4254-81fc-
d11c919286ef/files/get/biggerCase.tgz?path=&force"

################################################################################
# End of experiment configuration                                              #
################################################################################

# application base directory
if [ "$PBS_JOBID" ]; then
    # run with torque
    userBaseDirectory="${PBS_O_WORKDIR}"
else
    # run without torque
    userBaseDirectory="$(pwd)"
fi

# Aerodynamics start script
${userBaseDirectory}/experiment/aero-jobscript.sh -c $CASE -s $TIME_STEPS -u $CASE_URL
```

Figure 25. Control script executed via Scotty.

Several other experiment configurations are provided for virtualised jobs (Linux and OSv) as well as for different IOcm and vRDMA configurations. The focus of the remaining project phase is to provision additional experiments validating and evaluating more MIKELANGELO technologies as they become available. It is also planned that the experiments are executed regularly with all the benchmarks (snap metrics and summary overviews) automatically extracted from these executions. This will also significantly contribute to the project's OpenData pilot.

5.3 Data to Be Used in Experiments

As a pioneer in electric and hybrid propulsion aircrafts Pipistrel is already for some time interested in opportunities offered by distributed electric propulsion (DEP) systems. DEP

promises efficient multi-disciplinary coupling of the current state of the art electric propulsion systems with aerodynamics, high lift systems, acoustics, aircraft control and aeroelasticity. For Pipistrel it is therefore interesting to see how does a larger number of smaller propellers distributed along the wing influence the aerodynamic characteristics of that wing and moreover how could the complete system be designed in order to improve the efficiency in all flight regimes.

Pipistrel therefore prepared (besides smaller cases that are being used for validation purposes) a computationally more demanding OpenFOAM case with 4 propellers in front of a wing of constant airfoil. Left hand side propellers rotate in a clockwise direction, whereas the right hand side propellers in the counterclockwise one, all with frequency 4060 RPM. The speed of the incoming air is 50m/s. The simulation using a steady state incompressible flow solver simpleFoam and two-equation k-omega turbulence model was run until a satisfying convergence was achieved. Pressure distribution over the wing and all four propellers together with the velocity distribution over a slice through the aft most propeller can be observed in Figure 26 (left)

For each propeller its complete 3D geometry is taken into account. Propellers are therefore not simulated as pressure difference discs, whereas a more complex multi reference frame (MRF) simulation is being applied. In MRF zones the propellers and their near vicinity are being calculated in a rotating frame of reference that is rotating relative to an inertial reference frame (the rest of the computational volume outside of the MRF zone). MRF zones of all four propellers are depicted in Figure 26 (right). Using the MRF approach the simulation can remain steady state since there are no physically rotating parts.

Even though we are dealing with a rather big case (22 million cells in a mesh) the near-term plan is to use the MIKELANGELO cloud stack and do a small parametric study with this computationally more demanding simulation. Parameters to be explored are the direction of rotation and frequency of each propeller and the position of each propeller relative to the wing. Objectives to be observed are the resulting aerodynamic forces on the wing. This study will give Pipistrel initial pointers regarding which direction to proceed in, in order to explore DEP on its future aircraft. Furthermore, the case is prepared in such a way as to support an arbitrary number of propellers. By varying the number of propellers an OpenFOAM case of an arbitrary size (discretely though) can be prepared for additional MIKELANGELO cloud stack evaluation.

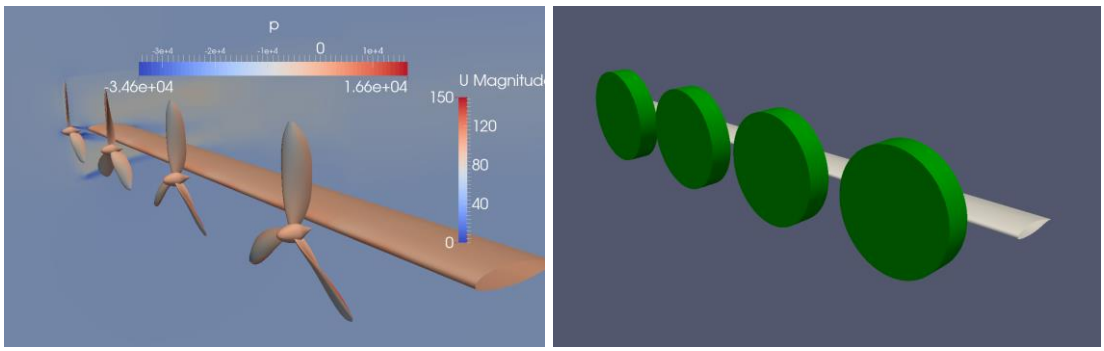

Figure 26. OpenFOAM simulation of an airflow passing a wing and four propellers (left) and MRF zones in cylinder shape, one for each propeller, denoted in green (right).

5.4 Validation by End Users

The aerodynamics use case regularly tests the newly integrated components to assert that their implementation and integration is properly enabled for the end users. This section details the coverage of the components of the MIKELANGELO technology stack and the means of evaluation and validation of these components. As indicated in the introduction, the aerodynamics use case is interested in evaluating all the components that are readily available except Seastar and ScyllaDB, which are out of scope of this use case. Scotty experiments will be prepared for all these cases in order to enable easily reproducible runs. The following list briefly explains the experiments for each of the components.

IOcm. IOcm can come in extremely handy when dealing with numerous smaller packets that are being exchanged between the workers. Due to the nature of OpenFOAM calculations, in particular those working on large input cases, the dynamic reconfiguration of IO cores may result in some performance improvement over standard hypervisor.

- Linux guests are the primary target of this experiment. OSv may be tested as well, but with the same NUMA configuration.
- In this experiment we are going to deploy between 2 and 12 worker nodes consuming all available physical cores (hyper-threading is going to be disabled).
- IOcm will either be disabled (baseline) or enabled with the maximum available cores.
- The same base kernel version will be used. For IOcm this means kernel version 3.18 because IOcm has not been ported to the latest 4.x series.
- HPC environment is the primary target, however cloud environment may also be tested.

ZeCoRX (Zero Copy Receive). This may even further improve the performance of the networking operations due to removal of unnecessary data copies when transferring data between virtual machines. The main reason for marking the evaluation as optional is the fact that it depends on the status of the implementation.

- Linux guests are the primary target of this experiment. OSv may be tested as well, but with the same NUMA configuration.
- The experiment will compare ZeCoRX disabled (baseline) against the hypervisor with the ZeCoRX enabled.
- The same base kernel version will be used for baseline and ZeCoRX. Since ZeCoRX targets upstreaming, the exact kernel version might change, but it is going to be at least 4.9.y (4.9 is the current long-term support version).
- Experiments will be performed in HPC and cloud environments.

SCAM (Side-Channel Attack Monitoring & Mitigation). SCAM is only going to be evaluated from the perspective of penalty hit. Because SCAM is focused on a specific kind of attack, it is not useable in the aerodynamics scenario. However, it is important to understand whether the SCAM monitoring itself affects execution of the OpenFOAM calculations.

- Linux or OSv guests will be used.
- The baseline is the hypervisor with no SCAM module enabled. This will be compared to a case when only SCAM monitoring is active. Since SCAM is monitoring for a specific type of side-channel attack which is not possible in this use case, we do not expect this to have any significant effect on the overall execution time.
- For completeness, SCAM mitigation will also be activated manually to assess the performance deterioration of active cache protection.
- Experiments will be performed only in a cloud environment.

vRDMA. Efficient inter-host communication is a prerequisite for efficient scaling of applications. vRDMA promises to bring RDMA interconnects, such as Infiniband, into virtual machines offering near-host performance. Albeit vRDMA has been shown to work in a synthetic benchmark, there are still some hurdles in the operation of Open MPI which is used by OpenFOAM. To this end, the OpenFOAM experiment with vRDMA is currently marked as optional.

- Linux and OSv guests will be used.
- Two baselines will be used: (a) host with direct support for Infiniband; (b) virtual machines with standard TCP/IP communication.
- vRDMA prototype 2 and prototype 3 will be evaluated.

OSv. OSv is going to be compared against the standard Linux guests. The comparison will focus on the overall footprint overhead (memory, disk), image size and deployment in distributed systems, and performance impact. The latter has been shown to be negative in our past experiments, most significantly due to lack of NUMA awareness of OSv. This is now being addressed with UNCLOT component which will be evaluated as well.

MCM. MCM will be evaluated indirectly using the various scheduling strategies set inside Scotty experiments. The evaluation will be based on a common experiment configuration only changing the MCM strategy.

Scotty. Scotty is going to be used as the main tool for configuration of experiments and their submission to the underlying infrastructures (HPC and cloud). Based on our current use of Scotty, it is evident that it will become particularly important to produce stable and comparable results. All the data, configuration and raw results are going to be published as OpenData.

LEET. LEET (Lightweight Execution Environment Toolbox) is already being used extensively for building application packages for this use case, composing target virtual machines containing the minimal set of required components and deploying them to the cloud. Further evaluation of LEET is going to be provided in the final evaluation deliverable. No additional experimentation is planned as LEET is constantly being used.

UNCLOT. In the context of this use case UNCLOT (UNikernel Cross Level cOmmunication opTimisation) might address the NUMA-awareness problem in OSv. With optimised communication channel between OSv virtual machines running on the same host, it should be fairly trivial to deploy one OSv instance per NUMA node and establish communication between workers on the same host based on shared memory. The UNCLOT experiments will be performed to evaluate the following:

- Compare one large OSv vs. multiple smaller OSv instances. Smaller instances are deployed in such a way that they consume one NUMA-node of the host.
- Compare OSv with Linux. We already know that OSv without UNCLOT performs significantly worse than Linux (20-30 % slower execution), thus we plan to compare one OSv instance per NUMA node with UNCLOT enabled.

vTorque. No specific experiments are planned for vTorque apart from the actual submission of virtualised jobs, both Linux and OSv. In the last evaluation report, this use case will provide an evaluation of the usability of the newly developed interface (CLI) offered by vTorque.

snap. Snap is being used to collect various performance metrics relevant to this use case. Snap is configured automatically via Scotty and exported based on the specific job id. Snap is also used by the OpenFOAM Cloud application where it collects data from the actual simulation offering the essential simulation state to the end user. This use case will thus provide an end-user evaluation of the usability of snap reports as well as an updated evaluation of the integration.

6 Conclusions

This document presented the implementation strategies for the four use cases in MIKELANGELO. The main part described the implementation strategies for each of the use cases separately.

The implementation strategies have laid out how the use cases will consume the MIKELANGELO stack in the cloud and HPC environments. All use cases build on automation to run a multitude of experiments to assess the impact of MIKELANGELO features on the use case workloads. The impact will be analyzed from data collected to perform parametric studies.

Most of our use cases will perform experiments using real-world data. In the case of the cancellous bones use case and the aerodynamic maps use case the data is anonymized and transformed due to privacy and confidentiality concerns. Most use cases also involve use case stakeholders closely, and perform the evaluations in accordance with their requirements. The descriptions have shown that the implementation strategies are tailored to improve performance of the workloads for the use case stakeholders, mostly by improving system throughput, request latencies, and workflow agility.

The implementation strategies have already been executed to varying degrees by the use cases. During the last six months of the project the use case implementations will be finalized. The good progress in the technical work packages, the ongoing integration effort for cloud and HPC, and the progress on use case implementations let us conclude that the use case implementations are on track.

7 References and Applicable Documents

[1] Amazon Lambda , https://aws.amazon.com/lambda/details

[2] Apache Cassandra, supplying tools for Scylla, https://github.com/scylladb/scylla-tools-java

[3] ScyllaDB, http://www.scylladb.com

[4] Designing a Userspace Disk I/O Scheduler for Modern Datastores: the Scylla example (Part 1), http://www.scylladb.com/2016/04/14/io-scheduler-1

[5] Designing a Userspace Disk I/O Scheduler for Modern Datastores: the Scylla example (Part 2), http://www.scylladb.com/2016/04/29/io-scheduler-2/

[6] https://en.wikipedia.org/wiki/Scheduling_(computing)#First_come.2C_first_served

[7] https://en.wikipedia.org/wiki/Scheduling_(computing)#Earliest_deadline_first

[8] https://en.wikipedia.org/wiki/Scheduling_(computing)#Shortest_remaining_time_first

[9] https://en.wikipedia.org/wiki/Scheduling_(computing)#Round-robin_scheduling

[10] https://en.wikipedia.org/wiki/Scheduling_(computing)#Multilevel_queue_

[11] Grafana: https://grafana.com

[12] FabanClient: http://faban.org/

[13] elgg: https://elgg.org

[14] MariaDB: https://mariadb.com

[15] CloudSuite: http://cloudsuite.ch

[16] YAML: http://yaml.org

[17] Europeana: http://www.europeana.eu/portal/en

[18] D6.3: D6.3 First report on the Use Case Implementations, https://www.mikelangelo-project.eu/wp-content/uploads/2017/01/MIKELANGELO-WP6.3-GWDG-v2.0.pdf

[19] Twitterdataset: http://socialcomputing.asu.edu/datasets/Twitter

[20] Gerrit Code Review: https://www.gerritcodereview.com/

[21] Schneider R. (2013) Storage and Indexing of Fine Grain, Large Scale Data Sets. In: Resch M., Bez W., Focht E., Kobayashi H., Kovalenko Y. (eds) Sustained Simulation Performance 2013. Springer, Cham