MIKELANGELO

D6.3
First report on the Use Case Implementations

Workpackage	6	Evaluation and Validation	
Author(s)	Peter Chronz		GWDG
	Peter Kiraly		GWDG
	Steffen Herbold		CSI (GWDG)
	Gregor Berginc		XLAB
	Matej Andrejašič		PIPISTREL
	Uwe Schilling		USTUTT
	Carlos Diaz		USTUTT
	Benoit Canet		SCYLLA
	Nadav Har'el		SCYLLA
Reviewer	Holm Rauchfuss		HUAWEI
Reviewer	Matej Andrejašič		PIPISTREL
Dissemination Level	PU		

Date	Author	Comments	Version	Status
2016-11-11	Peter Chronz	Initial document structure	V0.0	Draft
2016-12-12	All authors	First draft, ready for review	V1.0	Review
2016-12-16	All authors	Document ready for submission	V2.0	Final

Executive Summary

The first report on the use case implementations documents the progress of the MIKELANGELO use cases in the second project year. As the use cases commenced work in project month 12, the document contains information span twelve months of work.

There are four use cases that aim to validate the project results and that provide a perspective to transition project results into active exploitation. The four use cases are: the big data use case, the aerodynamic maps use case, the cancellous bones use case, and the cloud bursting use case. These use cases span the most important areas of application for virtual computing infrastructures: cloud computing, big data and HPC. Cloud computing is tackled by the big data use case, by the aerodynamics use case, and by the cloud bursting use case. Even the cancellous bones use case considers to use cloud computing for scale-out. Big data is specifically handled by the big data use case. HPC is covered by the aerodynamic maps use case and the cancellous bones use case.

The use cases validate that the project's results in work packages 3 to 5 are relevant. The use cases' relevance presents itself from two different viewpoints. First, from an outside view, the use cases represent essential fields of computing. Second, from MIKELANGELO's internal view, the use cases provide opportunities to leverage MIKELANGELO's computing stack. Our use cases evaluate all of the features from work packages 3 to 5.

All use cases have made good progress during their first year of implementation. The big data use case deployed a virtualized big data platform in the cloud test bed, it has automated the deployment of synthetic workloads, and it has defined real-world workloads for validation. The aerodynamics use case has created a simulation platform with a dashboard supporting the submission of experiments, defined test cases, implemented application packaging, and integrated vTorque. The cancellous bones use case ported its simulation to OSv, it integrated vTorque, and it deployed the MIKELANGELO stack in an HPC test bed. The cloud bursting use case has developed a new IO scheduler for ScyllaDB, an IO tuning component for ScyllaDB, it improved an RPC framework, and it enhanced the efficiency of data streaming for database replication.

This report presents each use case with a parallel section structure. Each use case presents the overall plan for year 2 and year 3. Then the work performed in year 2 and work in progress are presented. Finally, an outlook to year 3 is given. The use cases as presented in the order: big data use case, aerodynamic maps use case, cancellous bones use case, and cloud bursting use case.

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

1 Introduction

MIKELANGELO has four use cases, which validate and exploit the technical outcomes from work packages 3 to 5. The use cases cover the most important fields of application for higher performance virtual infrastructures. These fields are cloud computing, big data, and high performance computing. Cloud computing is covered by the big data use case, the aerodynamics use case and the cloud bursting use case. Big data is covered by the big data use case. High performance computing is covered by the cancellous bones and aerodynamic maps use cases.

In this report we present the progress of the four use cases in the second project year. Since the use cases began in month 12, the progress spans a timeframe of about 12 months. For each use case, we provide a short description, the overall plan for M12-M36, the work performed in M12-M23, work in progress in M24, and the work plan for M25-M36.

Each use case outlines how it validates the work products from work packages 3 to 5. In specific, a focus lies on the validation of MIKELANGELO's core promise to improve the IO efficiency of virtualized applications. Furthermore, the use cases provide an exploitation opportunity for the MIKELANGELO's technical results. Finally, the use cases for the final element in the project management process that starts with the requirements process in work package 2. The use cases and their evaluations in work package 6 provide valuable feedback to work packages 3 to 5. The requirements from work package 2 provide a common reference for features and their evaluation.

Where possible, the use cases provide baseline measurements of their workloads without the use of MIKELANGELO. We then compare the baseline measurements to those obtained with the MIKELANGELO stack. Furthermore, for each use case we evaluate which components it validates.

To put all four use cases into context, introductory excerpts are provided next. The diagrams contain all of the components comprising the MIKELANGELO technology stack. Details on the status of these components is presented in reports D3.2 [1], D4.5 [2] and D5.2 [3].

The big data use case. The big data use case leverages MIKELANGELO's advancements in virtual infrastructures to foster big data computation in the cloud. The use case hinges on an improved agility, provided by MIKELANGELO's stack.

The use case comprises work on a virtualized big data platform and on workloads for validation. The work on the platform focuses on agility and on ease-of-access for end-users. The goal of the platform is to run big data workloads sensibly in the cloud. This includes the use of stale resources without interrupting other workloads, rolling out clusters on demand,

and providing self-service access to end-users. All this work leverage MIKELANGELO's stack. As shown in Figure 1, we expect that snap, IOcm, ZeCoRx, vRDMA, SCAM, MPM and MCM will improve agility and efficiency of big data deployments. The virtualized big data platform deploys VMs with big data middleware installed on top of our stack. Finally, workloads run in the platform.

The components used by the big data use case are shown in Figure 1. Gray components will not be considered for the big data use case. Orange components are planned to be integrated in year 3. Green components are already integrated.

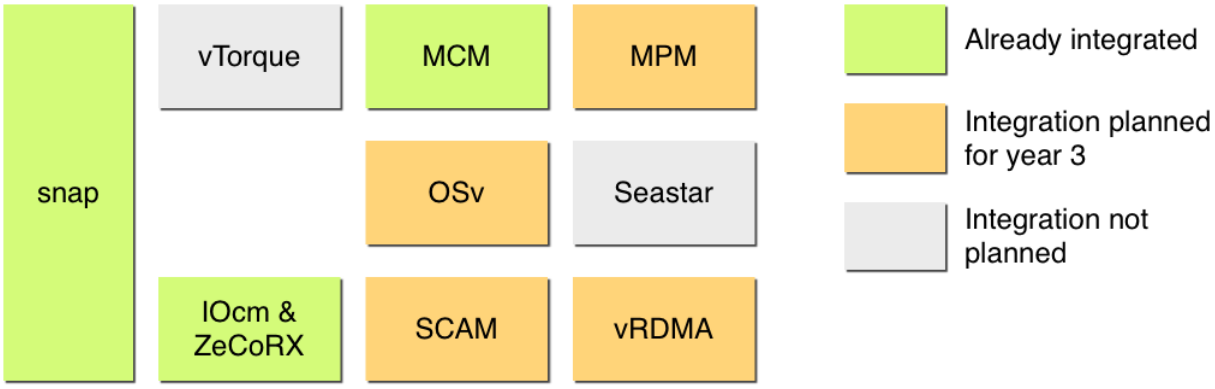

Figure 1. MIKELANGELO Components Used by the Big Data Use Case.

The aerodynamic maps use case. The main objective of the Aerodynamic Maps use case is to transfer aerodynamic simulations run by Pipistrel from HPC cluster to MIKELANGELO cloud infrastructure. Expected benefits of this transition are acceleration of the aerodynamic analyses, improved agility and elasticity of application deployment and its monitoring and finally, simplification of the post processing phase, as was already explained in report D2.2 [4].

Two different types of analyses are planned to be run. The first one is dealing with parametric study of typical simple aerodynamic industrial configurations in order to accelerate the workflow and to obtain results at a large set of parameters. The second one is dealing with a single computationally very intensive simulation using all available computing resources in order to present possible improvement of the virtualized I/O efficiency of MIKELANGELO cloud stack [4], [5]. Computationally more intensive case differs from the simpler propeller-wing aerodynamic simulation mostly by taking into account the complete geometry of the propeller. It also simulates more than one propeller. Each propeller is therefore not simulated as a pressure difference disc, as at the simpler case, but a more complex multi reference frame (MRF) simulation is being applied. The simulation is still using an incompressible flow in steady state (simpleFoam solver), but the propellers and their near vicinity are being calculated in a rotating frame of reference. The case is prepared in such a

way to support arbitrary number of propellers. By varying the number of propellers an OpenFOAM case of an arbitrary size (discretely though) can be prepared.

The components used by the aerodynamic maps use case are shown in Figure 2.

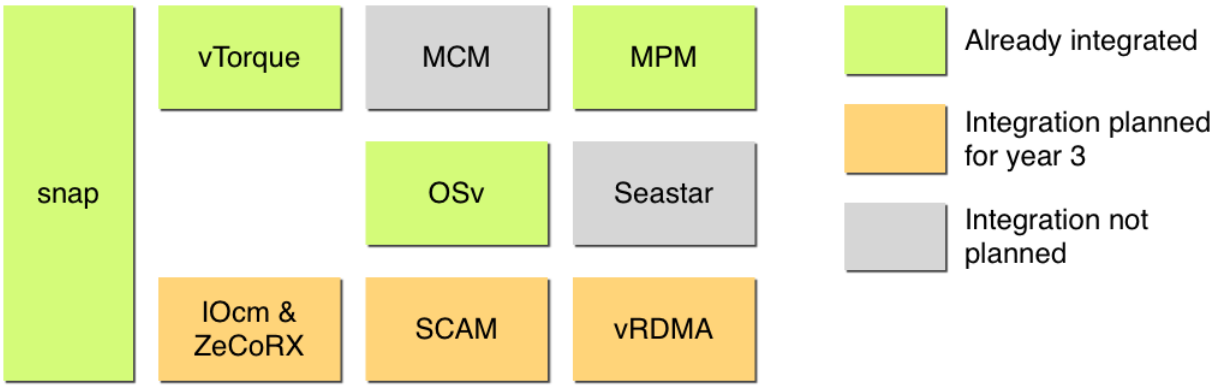

Figure 2. MIKELANGELO Components Used by the Aerodynamic Maps Use Case.

The cancellous bones use case. The purpose of cancellous bones use case is to understand the structure of the cancellous, or spongy, parts of the human bone, in order to efficiently and safely attach required implants to it. Using this simulation, implants can be produced in a more accurate fashion, which increases production yield and at the same time improves the healing time for bone implant surgeries as well as the lifespan of the implant. To understand the structure of the human bones it is important to get more insights, than it is possible with a scan of the bones. This simulation generates a more accurate model and shows how the bone react to pressure. This has an impact on the medical aspect of understanding the patient and his problem, helps to understand how implants should be design, gives an overview where they can be attach to increase the lifespan and helps students to get a general knowledge how a bone react to stress.

The components used by the cancellous bones use case are shown in Figure 3.

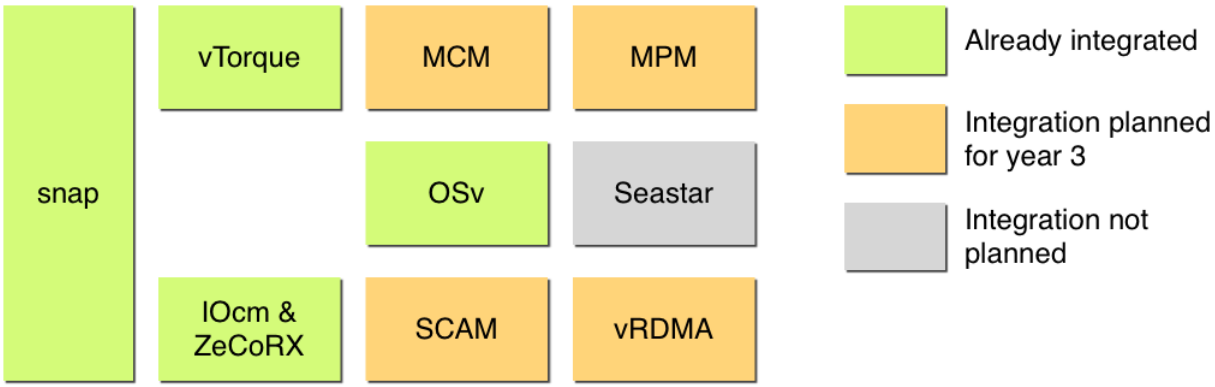

Figure 3. MIKELANGELO Components Used by the Cancellous Bones Use Case.

The cloud bursting use case. From the business side the "cloud bursting" use case is about being able to serve an unusual burst of customers, in events such as Black Friday. From the technical side, the cloud bursting use case is about fulfilling one of the initial promises of the cloud: elasticity. Elasticity in a cloud is the ability to grow or shrink the underlying computing resources of your cloud application without interrupting the regular behavior of the service. One of the ways to provide elasticity is to add more machine to the cluster serving the customer requests then re-balance the workload among them. In a cloud, more VMs would be added or removed from the cluster. Adding more machines to scale does not come for free: In most cluster applications the data would need to be rebalanced among the cluster machines, and it is difficult to achieve this rebalancing efficiently, without hurting the cluster's capacity to handle regular requests.

The components used by the cloud bursting use case are shown in Figure 4.

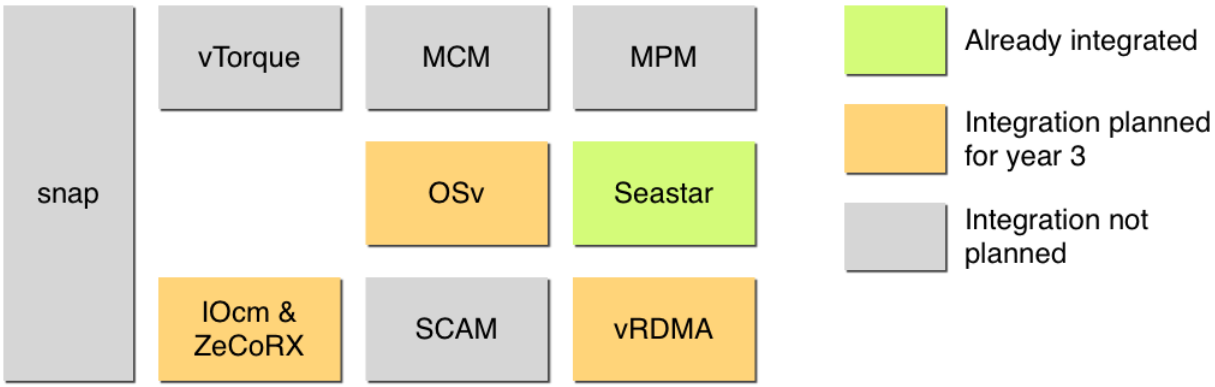

Figure 4. MIKELANGELO Components Used by the Cloud Bursting Use Case.

This report presents each of these use cases in turn. Each of the use cases follows the same structure. Each use case first presents a short description, then the overall work plan for year 2 and year 3, then the work performed in year 2, then the work in progress, then the work plan for year 3. Finally, each use case summarizes its progress and plans in a conclusion. The document begins with the big data use case, which tackles big data and cloud computing. Then the aerodynamics use case presents how our work bridges the worlds of HPC and cloud computing. Then the cancellous bones use case describes how it leverages our stack for pure HPC jobs. Finally, the cloud bursting use case show it is possible to scale ScyllaDB in the cloud.

2 Use Case: Big Data

The big data use case leverages MIKELANGELO's advancements in virtual infrastructures to foster big data computation in the cloud. The use case hinges on an improved agility, provided by MIKELANGELO's stack.

The use case comprises work on a virtualized big data platform and on workloads for validation. The work on the platform focuses on agility and on ease-of-access for end-users. The goal of the platform is to run big data workloads sensibly in the cloud. This includes the use of stale resources without interrupting other workloads, rolling out clusters on demand, and providing self-service access to end-users. All this work leverages MIKELANGELO's stack. As shown in Figure 5, we expect that snap, sKVM (incl. IOcm, ZeCoRx, vRDMA, and SCAM) and MCM will improve agility and efficiency of big data deployments. The virtualized big data platform deploys VMs with big data middleware, such as Spark, installed on top of our stack. Finally, workloads run in the platform.

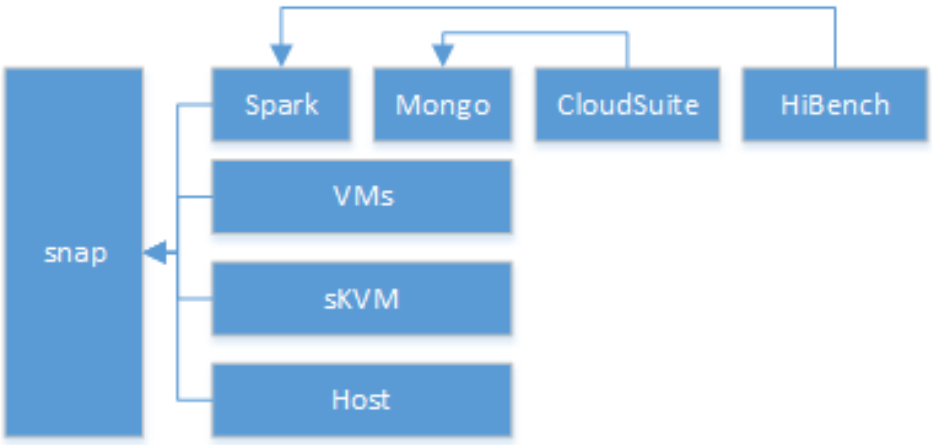

Figure 5. Virtualized Big Data Architecture.

In this use case we repeatedly refer to the terms agility and efficiency. In the context of the big data use case, agility refers to the speed of adaption of the big data platform to customer needs. Efficiency refers to the efficient use of the computing infrastructure. Efficiency includes considerations such as increasing VM packing density and resource utilization while maintaining QoS.

Our efforts on running workloads for validation consist of synthetic and real-world workloads. Synthetic workloads cover a broad spectrum of representative workloads. Furthermore, researchers can parameterize and run synthetic workloads quickly. Synthetic workloads build on the open source benchmarks Cloudsuite, HiBench, and YCSB. To validate our work in real-world scenarios, we further test the infrastructure with real-world workloads. These real-world workloads originate from two real-world use cases. The first real-world use case performs

analysis of software repositories. We refer to this use case, as Mining Software Repositories (MSR) workload. The second real-world workload analyzes metadata from digital archives in the humanities. We refer to the second use case as Digital Humanities (DH) use case or DH workload.

The big data use case is of practical importance as it combines cloud computing with big data. Both technologies represent growing markets according to MIKELANGELO's market trend analysis [6]. Furthermore, cloud computing and big data analysis fit well together. While cloud computing suffers notoriously from low utilization, big data's computing requirements outmatch the growth of computing resources. At the same time big data suffers from rigid infrastructures that need to be dedicated for big data computation. By combining both concepts, we can utilize computing resources beyond today's possibilities and we help to satisfy big data's thirst for computation. MIKELANGELO's stack supports this plan by providing holistic metering information, live-scheduling, and optimized IO operations.

The big data use case is relevant to MIKELANGELO as it builds on MIKELANGELO's core developments. We designed the use case to take advantage of MIKELANGELO to improve agility and efficiency. The components in work packages 3-5 deliver those benefits. Thus, the use case also evaluates most features of the components developed in the project as follows. IOcm is validated by running VMs with net IO-heavy workloads. VRDMA is evaluated by using the inter-VM, intra-host communication offered. The cloud integration is being tested via MCM and its resource management algorithms. Furthermore, the cloud integration provides the automation required to deploy and manage big data clusters. SCAM will be tested in future for its performance impact. ZeCoRx will be tested for performance impact of IO-heavy workloads. Finally, OSv has been already partially evaluated for performance by running HDFS with it.

The following sections introduce an implementation plan and summarize the progress for the big data use case. This description includes the work performed in year 2, the work in progress as of writing this report, and the plans for year 3.

2.1 Plan for Year 2 and Year 3

The overall plan for the big data use case comprises plans for a virtualized big data platform, for synthetic workloads, and for real-world workloads. In year 2, we carried out the original plan for the big data use case as described in the grant agreement with few deviations. The original plan for year 2, in the context of the big data platform, called for an automated deployment of big data clusters using a web interface or CLI. The implementation was to be preceded by a review of architectures and the design of an architecture. For workloads the plan foresaw the deployment of realistic synthetic workloads on the cloud testbed and the definition of real world workloads. The plan for year 3 plans to extend the platform by

improving usability and the feature set. The plan for year 3 calls for the extension of real-world workloads.

While the overall plan for the big data use case remains we have only adapted three aspects about the scope of the architecture. First, we do not target YARN as a primary middleware anymore. Discussions with the real-world use case stakeholders have revealed that they prefer Spark over other platforms. Second, due to the preference for Spark, we do not focus on providing support for MapReduce anymore. Currently, none of GWDG's partners plan to use it. Third, the real-world use cases requested MongoDB for data storage. In exchange they do not require HDFS. Additionally, the integration of HDFS does not fit with the current progress and plans in work packages 3 and 4. The work on sKVM optimizes IOcm for net access, not for block-devices. Thus, a speedup with IOcm for HDFS could only be shown if HDFS ran remotely; for example in central storage. This approach however contradicts HDFS's central idea of distributing storage to the nodes where the computation happens. Although, we do not plan to invest additional effort for support of YARN or HDFS, we still support their use for evaluation via HiBench. HiBench operates on a CDH cluster with YARN and HDFS in our synthetic benchmarks.

In year 2 we have set up OpenStack Sahara in the cloud testbed, we have developed a Spark collector plugin for snap, and we have performed a parametric study of the performance of a virtualized big data cluster. The major part of effort went into synthetic workloads. This work includes the automation and integration of HiBench, YCSB, and partially CloudSuite with Scotty. Finally, conceptual and implementation work has begun for two real-world workloads. The implementations currently focus on providing functional prototypes.

Work in progress for the platform revolves around the integration of Spark 2 with Sahara, the integration of snap collectors, performance tuning for real-world workloads, and the creation of additional snap collectors. Regarding synthetic workloads, we focus on the automation and integration of benchmarks. For real workloads we focus on providing functional prototypes. In year 3, we will extend the platform with a web UI aimed at end-users, create concepts for exploitation, and resource management algorithms. We will extend synthetic workloads to include all CloudSuite benchmarks and to fully automate and integrate all benchmarks in our portfolio. Furthermore, we will investigate the extension of our portfolio by including more benchmarks for collaboration with networking projects, such as H2020 Nephele and BMBF SENDATE (EU Celtic). Finally, we will invest a large chunk of our effort to extend the real workloads. For both use cases benchmarks with a subset of data will be created and integrated with Scotty. Both use cases will be tuned for performance. Finally, we will investigate whether the inclusion of even more use cases will make sense.

2.2 Work Performed in Year 2

In year 2 we have set up and partially automated the virtualized big data platform, we have automated synthetic benchmarks, and we have defined two real-world workloads. This sections lists features implemented for the big data platform and the big data workloads in year 2. These features reference scenarios and requirements in the requirements sheet, as developed in work package 2. The appendix contains the referenced list of scenarios and requirements for the big data use case.

Currently, we are mostly testing the functionality of the implemented features, with performance tests planned for year 3. The main reason why we lack performance tests lies in the necessity for functional prototypes before performance can be tested and tuned. Furthermore, performance tests require a complex setup, evaluation of a wide variety of settings, and sufficient repetition to provide meaningful results. These kinds of tests will be facilitated by the ongoing automation provided with Scotty, in work packages 5 and 6.

To date, we have collected performance data in a study of the impact of parameters on virtualized big data performance, and through performance tests with the three integrated benchmarks CloudSuite Web Serving, CloudSuite Data Caching, and YCSB.

We describe the use case's progress separately for the big data platform and the workloads. We subdivide the workloads further into synthetic and real workloads.

2.2.1 Implementing and Setting up the Big Data Platform

The implementation and provisioning of the big data platform progressed in four major areas. First, we have set up Sahara on the cloud testbed. Second, we wrote a snap collector plugin for Spark. Third, we performed a parametric study of a virtualized big data cluster. Fourth, we have integrated the automated provisioning of MongoDB.

To automate the deployment of big data clusters on OpenStack, we have set up OpenStack Sahara. The deployment of Sahara requires customization for the corresponding infrastructure. As a dependency, we had to configure OpenStack Heat, which facilitates the automation of many Sahara workflows. Furthermore, plugins for each big data platform need configuration. In our case, we have configured the default plugin, which offers a vanilla version of Apache Hadoop. As an extension, we have configured the CDH plugin, which offers an alternative to the vanilla Apache big data stack. HiBench requires CDH specifically to runs tests. The scenarios S6.BDUC.001, S6.BDUC.002, S6.BDUC.004, S6.BDUC.006, S6.BDUC.007, and S6.BDUC.012 require a working installation of Sahara.

To capture data from Apache Spark, we have developed a new snap collector plugin. Spark is, with MongoDB, one of the two central components of our big data platform. While a snap

collector plugin for MongoDB exists, there was none for Spark. Our new spark plugin allows the collection of all Spark metrics exposed by the native monitoring system. These data allow analysis of system dynamics and by extension facilitate the development of advanced resource management algorithms. The spark collector will be published on MIKELANGELO's github page. The collector supports the scenarios S6.BDUC.010 and S5.SNAP.009. The use of snap comes naturally as it is a central component in the project.

To analyze the impact of system parameters of a virtualized big data deployment, we have performed a parametric study. As workloads, we used HiBench's benchmarks. The configuration parameters in the study span hardware, middleware, and the application. We have run several thousand of automated experiments and collected data for each. These data points provide insights into the sensitivity of benchmark makespans, data throughput, and latencies. The data points include metadata and time series from snap. The study will form a basis for the development of resource management algorithms for the big data platform in year 2. We expect that this type of study can be performed regularly based on Scotty, as developed in work packages 5 and 6. We will exploit these results to develop new resource management algorithms. These algorithms will be implemented using MCM, MIKELANGELO's live scheduler for OpenStack. Furthermore, We plan to publish the results of this study at a scientific conference. More details on our plans to exploit these results are described in Section 2.4. The data collected in this study can be accessed in our open data repository [7]. This feature contributes to scenario S6.BDUC.009.

To offer MongoDB for real-world workloads and synthetic benchmarks, we have integrated automated provisioning of MongoDB. Our work builds on existing puppet scripts [8]. We have integrated those scripts with Heat in order to automate the whole roll-out process including the provisioning of the virtual infrastructure. In our scenarios, we build on a sharded and replicated deployment of MongoDB, which can be used in production. We have integrated a MongoDB cluster based on this automation with Scotty as a testing resource for benchmarking. This feature supports the scenarios S6.BDUC.003, S6.BDUC.011, and the requirements R6.BD.006, R6.BD.007.

2.2.2 Integrating and Running Synthetic Big Data Workloads

The integration and execution of synthetic big data workloads made four significant advancements in year 2. These cover the automation and integration of big data benchmarks with the testbed and Scotty. The covered benchmarks contain the complete HiBench suite, CloudSuite's web serving benchmarks, CloudSuite's data caching benchmark, and YCSB.

HiBench is a workload suite that contains 11, so called, micro-workloads [9]. Each of those micro-workloads is a synthetic workload. The workloads operate on data that is generated in a loading phase. Data generation uses common probabilistic distributions depending on the

microbenchmark. For example, TeraSort sorts uniformly distributed data. Microbenchmarks that operate on text-data use a Zipfian distribution that models texts as found on the web We have used HiBench in the past. However without the extensive automation available to us now, it was not feasible to use it for repeated tests. In year 2, we have extended the automation of HiBench. This automation uses puppet to obtain, build, and deploy HiBench. Further puppet modules allow to configure and start HiBench. Furthermore, we have integrated our deployment mechanism with OpenStack Heat to allow the preparation of the infrastructure. Thus, in the current state Heat provides required VMs and networks and triggers puppet with the right profiles on each host. Puppet then installs HiBench. In addition, Sahara allows us to roll-out a CDH cluster. Finally, HiBench connects to this cluster and executes its benchmarks. This feature facilitates scenarios S6.BDUC.002, S6.BDUC.009, and S6.BDUC.012.

The integration of HiBench with Scotty is another feature. Since HiBench contains a multitude of benchmarks the automation of HiBench requires more effort than other workload generators. The HiBench integration requires that Scotty can roll-out a CDH cluster by itself. Furthermore, the integration requires a flexibly configuration of HiBench from the outside of the benchmark. Without the automation in year 2, changing HiBench's configuration meant changing configuration files manually in the directory structure of the HiBench installation. This feature contributes to scenario S6.BDUC.002 and is covered by the requirements R6.BD.002-R6.BD.005.

CloudSuite web serving is a benchmark in the CloudSuite benchmarking suite [10]. As CloudSuite rolls out custom infrastructure for each benchmark, we have to invest more effort to automate and integrate each of its benchmarks than with HiBench. Thus, we treat the CloudSuite benchmarks as separate features.

CloudSuite web serving simulates a web application with nginx, MySQL, and memcached as components. The workload puts considerable load on the network, which allows to validate the outcomes of work package 3-5 well. We have implemented this feature to a degree where CloudSuite can be installed automatically in a clustered mode with bash scripts. Further work will include a potential automation with snap, an integration with Heat, data collection with snap, metadata collection via Scotty, and an integration as workload generator with Scotty. The web serving benchmark supports the scenario S6.BDUC.008, and the requirements R6.BD.014 - R6.BD.019.

CloudSuite data caching simulates a setup with memcached as central components. Again the workload uses a large load of IO operations, which allows good validation of features from work packages 3 and 5. We have automated the deployment of the benchmark such that it can be installed with bash scripts in a cluster of machines. Further work on this

benchmark will result in a automation with puppet and Heat, metering with snap, and metadata collection and integration with Scotty. This feature is reflected by scenario S6.BDUC.010, and requirements R6.BD.010 - R6.BD.014.

YCSB is a benchmarking system for NoSQL databases [11]. In our work, we focus on MongoDB, as our real-world use cases require MongoDB. YCSB can be configured to submit varying types of loads to a NoSQL database. As a result, experiments measure latencies at various rates of throughput. For this feature we have automated the installation of YCSB as a resource for Scotty. Furthermore, we have created an example of an experiment that uses the YCSB workload generator. This feature covers scenario S6.BDUC.012 and contributes to requirements R6.BD.044-R6.BD.048.

2.2.3 Defining and Integrating Real-World Big Data Workloads

In year 2 we have defined and commenced integration of two real-world big data workloads. Functional integration with those use cases has begun. In this report, we introduce these real-world use cases with definitions. Both real-world workloads introduce the problem they tackle, their approach, expected results, and their data. First, we define the MSR workload, then we define the DH workload.

2.2.3.1 Defining the MSR Workload

The MSR workload deals with the problem of replicable and reproducible research in the software engineering community. The focus lies on empirical studies based on data collected from software repositories, For example, we use source code versioning systems, issue tracking systems, and mailing lists of projects. The use case is carried out in a collaboration of GWDG and the computer science institute (CSI) of the University of Göttingen.

Empirical studies based on repositories became one of the largest subtopics of empirical software engineering during last decade. Most of the research uses open source projects. Thus researchers can access the data easily. Nevertheless replication of results hardly ever happens. The two most important reasons for this failure to replicate are: a lack of sharing extracted data and a lack of sharing implementations. This leads to challenges when comparing results and when measuring the impact of new proposals.

SmartSHARK, the use case's public branding [12], [13], aims to directly combine the three phases of repository mining studies. These phases are: data collection, data sharing, and data analysis. We have parallelized data collection using HPC clusters. The collection stores all data into a single MongoDB that combines data from different sources. We then open the MongoDB to the research community to provide a common foundation for empirical studies.

To provide a solution for sharing implementations directly on the platform, we provide an Apache Spark based API for defining analysis of the data.

SmartShark's architecture is shown in Figure 6. Users interact with the system via a web server. The web server can trigger the execution of analytical jobs on software repositories in Spark. The results from this analysis are then sent to MongoDB for storage. Furthermore, users can query results in MongoDB via the same web server. Finally, newly grabbed data is ingested by HPC jobs.

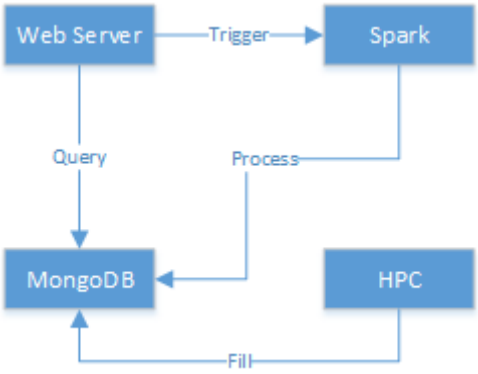

Figure 6. Architecture for the MSE Workload.

Currently, the use case has accumulated less than 100GB of data. However, we expect the data to grow to more than 10TB. The data is coming in burst from the data collection on the HPC. The HPC cluster can run more than 100 processes at the same time leading to a large amount of data in a short amount of time. We expect peeks of as much as 1 GB/min, possibly even more. MongoDB stores all incoming data, i.e., the data format is JSON. The JSON contains both large text-blocks (e.g., issue descriptions, source code), as well as numerical metadata (e.g., timestamps, software metrics). The incoming data is raw and directly reflects the information contained in the software repositories. The algorithms that analyze the data may introduce uncertain data. For example, computed labels for bugfixes, bug inducing commits using heuristics, or results of machine learning approaches for intention mining. We will use the data to answer research questions about software development processes.

2.2.3.2 Defining the DH Workload

In the DH workload we analyze cultural heritage metadata records in order to infer the quality from structural features. In large and heterogeneous collections, such as the European digital library, called Europeana [14], we cannot assess the quality manually [15]. Our research provides a tool to filter out specific problems.

Without metadata we cannot fulfill the functional requirements of the services built on them, and thus the quality of the service will be suboptimal. Finding the records starts the process of fixing issues or improve different features of the records.

At first step we do a feature extraction and apply known and newly introduces metrics on record level. Then we build a model on those metrics and features to get an overall evaluation. We do statistical analysis on group of record, which were created or curated by the same institution. Finally we provide a detailed report with data visualization, covering different aspects and levels of the quality indicators [16].

We expect that the reports highlight selected data quality features. As a consequence, both the data aggregators and data providers receive a proper feedback. This feedback can be used to plan the next phase: improving the data quality.

Currently the data set consists of 46 million records, formatted as JSON [17]. The volume amounts to 420 GB. We expect the data to grow by 10-20 percent annually. In this phase of the project we add new data only once a year. In the next phase data will come in monthly cycles or continuously. The data format is called Europeana Data Model (EDM), and is serialized as JSON. Originally it was harvested via OAI-MPH protocol. EDM provides about 130+ data fields, and lots of flexibility, so within this schema the individual records shows big heterogeneity. Originally more than 3500 individual institution have created the data. During their life cycle various users have transformed the data from one format and data schema to another. The research aims to find reliable and less reliable data.

2.2.4 Performing Experiments

To validate the MIKELANGELO platform, we have performed experiments for the big data platform and the big data workloads. These preliminary experiments serve as groundwork for reproducible experiments with Scotty, to be implemented in year 3.

We have run experiments to test the platform as well as workloads. To test the platform, we have run functional tests for platform components and performance tests for OSv.

The first set of platform experiments have validated that the big data platform works with various configurations as required. The tests have been performed with Spark 1.3.7 [18], Cloudera Distribution of Hadoop [19], YARN [20], HDFS [21], and MapReduce [22]. To verify that the platform works, we have run HiBench on the platform. The platform itself runs in VMs and is packed tightly, so that they utilize most of the host systems well. To assess the impact of running the experiments in a virtual infrastructure, we have also run the experiments with Docker-based containers directly on the hosts. These experiments have shown that both approaches lead to comparative results. Using virtual machines over Docker

containers does not leads to a significant slowdown. The experimental data can be accessed in our open data repository [23].

The second platform experiment aimed to assess the performance speedup by using OSv for big data applications. The experiment evaluated the performance of HDFS in VMs with DFSIO, the standard benchmarking tool for HDFS. The evaluation has taken into account three different conditions. In all cases we have controlled for effects of sizing, distribution, and configuration of DFSIO and HDFS. The first condition ran HDFS in Ubuntu to establish a baseline. The second condition ran HDFS in OSv. The third condition ran HDFS in Ubuntu with ZFS. ZFS has been tried out in Ubuntu to control for the fact that OSv uses ZFS by default. Thus, we wanted to assess how much of the expected performance gap between OSv and Ubuntu results from the use of ZFS. ZFS in general should perform better than the default ext-filesystem in Ubuntu. The experimental setup is shown in Figure 7. We have repeated the experiments multiple times and recorded the data to control for statistical variations [24]. Statistical analysis reveals that OSv, contrary to our expectation, performs worse than Ubuntu for HDFS. In the best cases OSv, even with tweaks by the developers performs on average with a speedup of 0.5 over Ubuntu. In the worst cases, OSv performs with a speedup of 0.1 over Ubuntu. The changed variable that affects the speedup is the HDFS block size. Current investigation lead to a possible problem in the implementation of the ZFS file system in addition to the deployment scenario used in these benchmarks. The VM storage, as indicated in the Figure below, is provided by a central Ceph. Further analysis is required to understand the root cause of this performance deterioration. Due to the current low performance of OSv for HDFS, we have re-focused our efforts to first leverage the performance gains promised by sKVM and cross-layer optimization. Further investigation on the potential use of OSv for big data will be performed in year 3 for MongoDB and Spark.

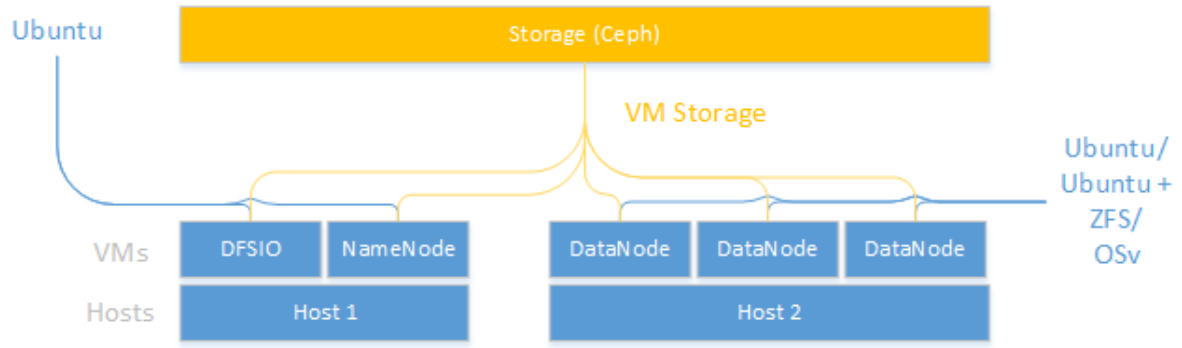

Figure 7. Experimental Setup for HDFS on OSv.

For synthetic and real world workloads we have performed experiments to verify functionality and the workflow. For synthetic benchmarks, we ran CloudSuite data caching and CloudSuite web serving. For both we have recorded summary and live benchmarking data, as well as system metering. At the current stage, however we have not performed any comparisons

between the MIKELANGELO stack to a more traditional stack. Due to the parameter space and complexity to adapt to changed parameters, we need to finish the integration with Scotty prior to running complex experiments. This work will be tackled at the beginning of year 3.

For real benchmarks, we have provided a Spark cluster and a MongoDB cluster to our use cases partners. The final users have evaluated both systems. The system performs well enough for the digital humanities use case. Thus, right now we are implementing the use case for full functionality. The MSR workload requires additional modifications to the platform. newer versions of both MongoDB and Spark have been requested. A newer version of MongoDB will boosts performance significantly. The new version of Spark offers a new, vastly improved API.

2.3 Work in Progress

As of writing this report, we are working on the big data platform, synthetic benchmarks and real-world benchmarks.

For the platform we focus on three areas: integration of Spark 2, integration of snap collectors, and performance tuning of MongoDB.

The integration of Spark 2 came up as a requirement from the MSR workload. There the developers require an up-to-date version of Spark, since there have been many changes to Spark's API. These changes facilitate easier implementation for the use case implementers and they improve the performance. This requirement poses a challenge, as the current OpenStack version in the testbed comes with a Sahara installation that provides Spark 1.3.7 only. We see multiple options to integrated Spark 2. First, we can modify existing Sahara images to use Spark 2. This approach requires the least effort in the short term. However, it breaks with the mainline deployment of Sahara. Second, we can upgrade the whole testbed to the most recent version of OpenStack. This approach would be "the right thing" to do. However, that approach would also consume considerable resources. As a result, this approach would delay the development of the cloud-related aspects in the project. The big data use case would be one of the delayed work products. Thus, we target the quick-fix of modifying existing Sahara images to use Spark 2. During year 3, once the use cases reach full support, we will work to upgrade the cloud platform seamlessly.

The integration of snap collectors into the big data platform will allow deep monitoring of the big data platform. This integration requires the installation of the snap daemon and all required plugins in VM images. Furthermore, at boot time tasks in the VMs need to be started with appropriately configured processors and publisher endpoints. Currently, we are integrating collectors for Spark, CloudSuite, YARN, and MongoDB.

Performance tuning of MongoDB is an important issue for our real-world use cases. As they treat the big data use case as a platform to transition into production deployment, they require high performance from the database. Both use cases rely on MongoDB. Especially the MSR use case requires high performance from MongoDB, as they write data in bursts from the HPC cluster. Currently, their bottleneck is the MongoDB instance in use. It is a single-node installation on one machine. As of writing this report this single-node installation outperforms our test installation in 3 VMs on the Mikelangelo cloud test bed. We aim to spread out the database in a sharded and replicated design across the whole cluster and to outperform the monolithic installation significantly.

Our work in progress on workloads concerns both synthetic and real-world workloads. For synthetic workloads we work on an integration with Scotty. For real-world workloads we work on onboarding both real-world use cases.

As synthetic workloads we currently are integrating the CloudSuite benchmarks data caching and web serving. We plan to gradually automate and finally integrate both workloads with Scotty. Automation refers to automated provisioning of virtual resources, installation of the whole benchmark, and running of the benchmark. Integration with Scotty means that the benchmark can be rolled-out, executed, and its results collected by Scotty. Currently, both benchmarks have progressed to the same stage of development. Both can be installed automatically given the infrastructure. Both have also custom-made snap collectors to extract live data during benchmarking. The next implementation phase will use Heat to provision the infrastructure upfront. Then, finally we will integrate the benchmarks with Scotty.

For real-world workloads, we foremost consider functional aspects. We start by providing all necessary resources to the use cases so that they can start operation. For the MSR workload we need to upgrade the platform to provide Spark 2. Furthermore, this workload requires more performance from MongoDB. For both real-world workloads we have also begun with the integration with Scotty. This will allow the use case owners to submit jobs to test their implementations. Furthermore, the integration with Scotty will allow the MIKELANGELO project to use benchmarks created from real workloads for further evaluation.

2.4 Work Plan for Year 3

Future work on the big data platform and workloads will serve to maximize the benefits for real-world workloads. In the case of the big data platform, we plan to provide a web-based UI, a concept for accounting and billing for a virtual big data service, and resource management algorithms to run big data collocated with other cloud workloads. For workloads we plan to take on-board more workloads, automate existing ones, and run the existing workloads on a larger scale.

The web UI for the big data case will allow users to create big data clusters on-demand in self-service fashion. We will consider three options for implementation. First, we might use the existing integration with OpenStack Horizon via Sahara. Second, we will consider to extend the existing interface to allow customization if required by use case stakeholders. Third, we will consider using the CDH interface. The interface might be used either standalone, or it might be integrated with the common workflow to manage the big data platform.

For accounting and billing we plan to research relevant concepts. Bringing in big data from the perspective of a research provider poses new questions. Usually customers at GWDG can each get three VMs. More VMs can be requested, but require additional compensation. Even though our cluster runs near its advertised capacity, the VMs lead to a low CPU utilization. This would allow us to offer spare resources for big data computation. However, then we need to relax the constraint of 3 VMs flexibly. In addition, a fair scheduling policy to serve different customers needs to be developed as well. We require new concepts to manage access and arbitration for a number of tenants. Also algorithms need to be developed to keep performance degradation for latency-critical services high.

To keep performance degradation of cloud workloads low when they are collocated with big data workloads, new resource management algorithms are required. A well-known problem when colocating workloads, called resource interference, arises. In our case we need to ensure that performance degradation of the default cloud workload does not take over when running big data workloads on the same hardware. A lacking specification of cloud workloads complicates the problem of running big data workloads in collocation. Also, no direct and reliable metrics for performance degradation exist, as any type of service can be run in the cloud.

We plan to extend our synthetic benchmarks by automating and integrating all benchmarks in CloudSuite. Furthermore, we will investigate to onboard new synthetic benchmarks to be used in related national and EU projects. Two projects under consideration are the Horizon 2020 project Nephele and the German BMBF project SENDATE. Both projects could reuse the results of MIKELANGELO and benefit from the big data workloads. At the same time they can provide network-related benchmarks, which can be of use to MIKELANGELO to evaluate IO performance.

For real-world workload we plan the creation of benchmarks, the integration of benchmarks with Scotty, and potentially to on-board more use cases. The real-world workloads currently run for a long time and consume significant resources. Such workloads do not lend themselves for infrastructure tests. Thus, we plan to create benchmarks from the real-world use cases that run on a reduced data set. These benchmarks can then be run repeatedly for

infrastructure tests with a short duration. We will integrate these real-world workloads with Scotty in the next step. This integration will enable reproducible and orchestrated infrastructure tests. Finally, we will evaluate whether to onboard more real-world use cases or whether to invest more effort into the existing ones. This decision will depend on the performance of the existing use cases and other potential use cases.

2.5 Conclusions

The big data use case strives to create a well-performing, virtualized big data platform. We aim to combine the benefits of cloud computing and big data. That is, we want to bring the agility of cloud computing to big data. At the same time we want to perform big data computations on notoriously underutilized cloud computing infrastructure. Our work can be separated into work on the platform and on big data workloads. The workloads can be further divided into synthetic workloads and real-world workloads.

In year 2 both the platform and workloads have progressed well. The platform runs most required components. Its functionality has been verified. Investigation on the effects of configuration on big data workloads has been performed. Synthetic workloads have been automated and partially integrated with Scotty. Concrectely, HiBench, YCSB, CloudSuite web serving, and CloudSuite data caching have been worked on. Two real-world workloads, the MSE workload and the DH workload, have been defined. Both workloads rely on the same setup, consisting of Spark and MongoDB.

For year 3, we will continue our work on the platform and the workloads. For the platform, we plan integrate Spark 2 and fully deploy snap for data collection. New collectors for the big data platform will be developed as required, Furthermore, we will tune the performance of our virtual MongoDB deployment. Furthermore, we will work on an easier access to the big data platform for end-users. Synthetic workloads will be extended by integrating as many Cloudsuite benchmarks as possible. Furthermore, we will explore using YCSB with more databases. For real-world workloads, we will create benchmarks and run them on a larger scale.

3 Use Case: Aerodynamic Maps

The main objective of the Aerodynamic Maps use case is to transfer aerodynamic simulations run by Pipistrel from HPC cluster to MIKELANGELO cloud infrastructure. Expected benefits of this transition are acceleration of the aerodynamic analyses, improved agility and elasticity of application deployment and its monitoring and finally, simplification of the post processing phase as was already explained in report D2.2 [4].

Two different types of analyses are planned to be run. The first one is dealing with parametric study of typical simple aerodynamic industrial configurations (Figure 8 and Figure 9) in order to accelerate the workflow and to obtain results at a large set of parameters. The second one is dealing with a single computationally very intensive simulation (Figure 10) using all available computing resources in order to present possible improvement of the virtualized I/O efficiency of MIKELANGELO cloud stack [4], [5]. Computationally more intensive case (Figure 10) differs from the simpler propeller-wing aerodynamic simulation (Figure 8 right) mostly by taking into account the complete geometry of the propeller. It also simulates more than one propeller. Each propeller is therefore not simulated as a pressure difference disc, as at the simpler case, but a more complex multi reference frame (MRF) simulation is being applied. The simulation is still using a steadystate incompressible flow solver (simpleFoam), but the propellers and their near vicinity are being calculated in a rotating frame of reference. The case is prepared in such a way to support arbitrary number of propellers. By varying the number of propellers an OpenFOAM case of an arbitrary size (discretely though) can be prepared.

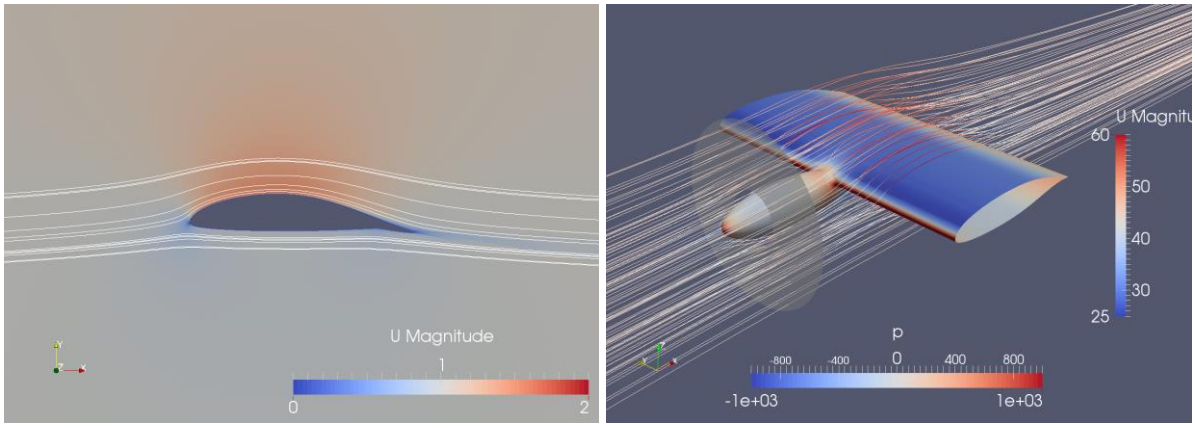

Figure 8. Typical 2D aerodynamic simulation of an airfoil (left) and 3D simulation with a propeller in front of a wing section (right).

The key difference between the experiments is that the first one belongs to the Embarrassingly Parallel[1] class of problems, while the second one is a typical, tightly coupled,

[1] https://en.wikipedia.org/wiki/Embarrassingly_parallel

MPI-aided problem. The first one demonstrates HPC-Cloud bursting, while the second one presents the virtualized I/O efficiency of MIKELANGELO.

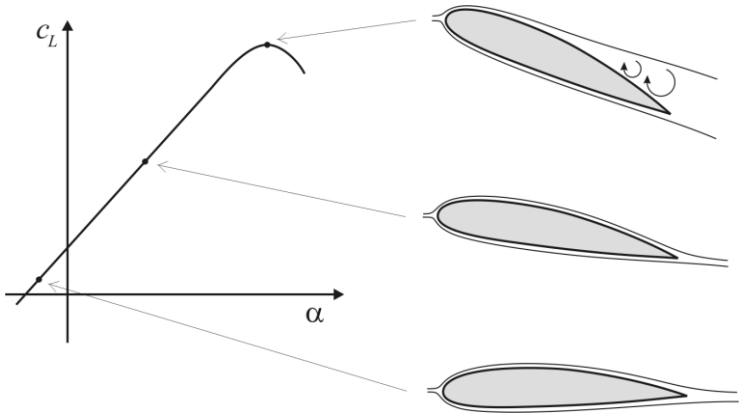

Figure 9. Lift coefficient with respect to the angle of attack as a typical end result of a parametric study of a 2D airfoil (wing section) analysis.

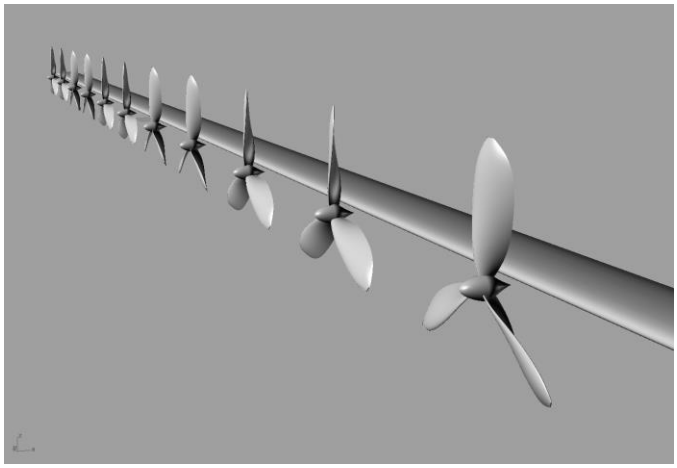

Figure 10. Geometry that is planned for computationally more intensive simulations dealing with distributed propulsion system with multiple propellers in front of a single wing.

The use case uses OpenFOAM [25] for running computational simulations. OpenFOAM is one of the most widely used solutions for all kinds of simulations, ranging from biomedical engineering (for example blood flow) to aerodynamics and large-scale flood simulations. It is an open source toolbox facilitating all stages of the simulation process: input data preprocessing, execution of the simulation and post-processing of the data. The preprocessing stage is crucial as it guarantees that the data is decomposed in a way to allow for efficient parallelisation in HPC environments. This decomposition furthermore guarantees the equivalence of the simulation regardless of the number and topology of the workload execution (single process simulation is always guaranteed to produce exactly the same result as many process simulation, possibly distributed over hundreds of nodes).

In context of this use case, the OpenFOAM toolbox is used for delivering high-end simulations of the airflow past the designed aircrafts and their underlying parts. In order to reduce design time and costs, makers of aircrafts rely heavily on the computer assisted design (CAD). These designs are then analysed for various aerodynamic features, such as lift, drag, moment, heat transfer, etc. The following figure shows typical workflow in the design of a new aircrafts or their parts.

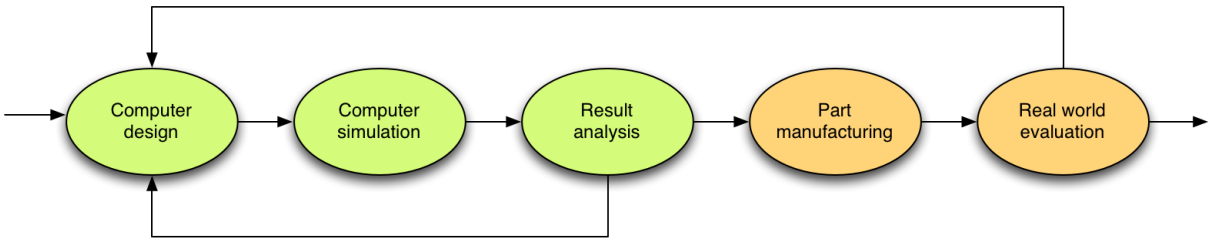

Figure 11. Typical workflow for designing new parts of the planes.

The entire inner loop is done using computers. Computer simulations can either be performed locally or in an HPC environment, depending on the resources required. After the result analysis approves aerodynamic features, concrete physical parts are made and verified in a real world environment. It is imperative that the outer loop is far more expensive and time demanding.

Relevance of this use case for the MIKELANGELO project are presented next.

Flexible simulation environment. As we have seen before, OpenFOAM can be deployed and used on a single (desktop) computer, internal small-scale cluster or large-scale infrastructure, such as HPC. Required computational resources vary significantly over different types of simulations. Since SMEs are lacking the resources necessary for execution of large and detailed simulations they are trying to offload these workloads to HPC clusters. However, the delays in execution due to the low priority of these kind of jobs are seldom acceptable. The management layer of the MIKELANGELO stack and the implementation of this use case will allow for a more flexible deployment of simulations in (HPC-)cloud environments. Even though the queuing system of the traditional HPC will still delay low-priority jobs, users will be able to deploy the same simulations in a public cloud instead.

Based on a widely popular open source toolbox. Projects like MIKELANGELO can benefit when they can demonstrate their value using popular open-source solutions, such as OpenFOAM. OpenFOAM is very popular for cloud environments. UberCloud [26] and SimScale [27] are just two examples simplifying execution of OpenFOAM simulations in public clouds. With this use case we are building an environment where lightweight simulations can be deployed quickly and without unnecessary resource consumption.

I/O intensive at large scale. Our experiments show that communication overhead becomes important when running simulations in parallel. MIKELANGELO promises to reduce the overhead of virtualised I/O, which this use case is continuously evaluating, in an attempt to steer the development of the underlying technologies.

Software management. In order to support as many workloads as possible, HPC providers need to maintain different tools and even different versions thereof. One of the promises of the MIKELANGELO project is to relieve infrastructure owners from having to manually maintain and optimise supported software. The flexibility of the application management and virtualisation technologies allows infrastructure providers to focus on their business, while the application developers and users are free to use their software packages.

Real-world demonstrator. Although the OpenFOAM Cloud application supports the execution of the use case we are mostly interested in, this use case seeks broader purpose. First, by providing an open source releases of an end-to-end application this use case serves as a live demonstrator of the many possible ways of the MIKELANGELO components integration. Second, it provides a reference implementation for the OpenFOAM simulation backend. This can then be integrated into other environments having access to multiple cloud providers.

3.1 Plan for Year 2 and Year 3

The initial plan for this use case was to experiment with numerous small OpenFOAM cases on a remote and distributed infrastructure, i.e. to perform parameter search studies. Project proposal described the goal of this use case is *to achieve quick deployment of relatively small OpenFOAM cases on remote machines, which provides basis for more efficient simulation.*

While this goal allows this use case to evaluate the application package management and improvements in the cloud/HPC management layers it does not support evaluation of the I/O improvements MIKELANGELO is delivering. This is because these workloads are inherently CPU bound (their main limitation is the performance of the central processing unit). I/O activity is extremely low as the data is read only at the beginning and written periodically at requested intervals (potentially only at the end). To remedy this, the plan was slightly extended by introducing more complex scenarios (Figure 10). Single compute node requires hours or even days to complete such scenarios, while multiple nodes reduce this to just few hours. Because this also increases network communication it allows us to analyse the gains of the core MIKELANGELO technologies, such as IOcm, vRDMA and ZeCoRX (refer to report D3.2 [1] for details).

3.2 Work Performed in Year 2

Up until M18, simpler OpenFOAM cases have been prepared in order to start evaluating separate MIKELANGELO components. Integrations of Open MPI with OpenFOAM on different configurations (Ubuntu host, Ubuntu guest and OSv guest) have been tested and compared. In most cases Ubuntu outperformed OSv, because the NUMA information is not exposed in OSv and consequently Open MPI is not able to optimise the deployment of worker threads in OSv in the same way as in case of Ubuntu. Additional evaluation comparing the Ubuntu and OSv image (time to copy VM and time to start VM) has been conducted showing that OSv outperformed Ubuntu [28].

In order to evaluate and validate the MIKELANGELO technology stack using the Aerodynamic maps use case an OpenFOAM Cloud application has been designed and implemented as an OpenStack dashboard [28]. The OpenFOAM core and simpleFoam solver have been provided as OSv packages early in the project, along with several others that play important roles in the management of distributed workloads. The OpenFOAM Cloud application is still in a prototype phase serving primarily for testing purposes, but the user is already allowed to specify the resources, input data and provide a set of case customisations to perform parameter search experiments. Once launched, the application keeps track of the allocated resources. Actions, such as Grafana visualisation or log retrieval are available.

Continuation of the work, along with the plan since M18 release has been presented in report D2.2 [4]. The high level work plan for the implementation of this use case in the second year defined in this report can be summarised with the following topics

1. Evaluation and benchmarking of OpenFOAM simulations. To the extent possible, this use case has evaluated various core components of the MIKELANGELO project. Being the use case that was able to run on OSv early in the project, this has been our primary focus. Some benchmark results are presented in Section 3.3.
2. Supporting MPI applications. Use case was involved in the verification of the implementation of MPI support in work packages WP4 and WP5. Consequently, we are now able to run parallel MPI workloads without known limitations.
3. Extending the list of aerodynamics simulations. As presented in the following subsections, this use case has requested additional OpenFOAM applications that were successfully ported for execution on top of OSv.

Following subsections provide further details of the work performed and already provides several possible paths for improvements.

3.2.1 OpenFOAM Application Packages

Previous implementation of this use case exported *simpleFoam* application solver as the only option for the end user. This has now been extended to a list of commonly used solvers, as defined by the business case partner, Pipistrel. All application packages are built using modular approach, extending the OpenFOAM Core application package. This allows composition of virtual machine images comprised of several solvers, if necessary. Furthermore, all packages have been extended to provide default run configurations [2] ensuring users are able to quickly understand the workflow. Three configurations are provided by default: usage, single process and MPI parallelisation:

- Usage will show usage information as shown by the particular OpenFOAM solver (as if the user would have executed: `/solver -help`)
- Single process runs the solver using a predefined input case directory. End user is required to put input data into that directory (either by uploading it into the VM or by mounting a shared workspace) in order to be able to launch it.
- MPI parallelisation shows an example of running the application using the `mpirun` command with some presets.

All of these commands are provided as part of self-documentation. Users can try them out automatically, however the main idea is that based on these commands users will be able to construct their own commands based on their experience with OpenFOAM.

All of these packages have been uploaded to the MIKELANGELO application package repository from where they can be used by any interested stakeholder.

3.2.2 Integration with Capstan

The recent release of the Capstan tool provides several significant improvements over the initial version. First improvement that has been made to the backend services of the OpenFOAM Cloud application is the integration of the application package repository. The repository is a central place where all application packages that are part of the MIKELANGELO distribution are shared. Exploiting this repository enhances the simulation instantiation process by allowing users to choose from the workload to deploy in the cloud. Based on user's specification, backend is going to dynamically request specific application packages and compose required VM images. If packages have not been found in a local repository, they will be downloaded and cached for future use.

Runtime configurations (consult report D4.5 [2] for more information) are also partially integrated into the backend services. Instead of using hard-coded commands these are created on-the-fly based on user's inputs. At the time of writing this report, the most significant feature requiring this functionality is specification of workload topology. The

topology defines how the parallel (MPI) processes should be deployed on the underlying infrastructure. This is defined by the overall number of processes (threads), number of nodes and number of cores on each of the nodes. This information is used to provision required instances, as well as to prepare the boot command for the main instance, delegating the work to the worker nodes.

3.2.3 Integration with UniK

Initial release of OpenFOAM Cloud application manually deployed and launched OSv-based instances in OpenStack. With the advances in the integration of the application management and cloud provider made in WP4 and WP5, the backend services have been partly migrated to the new capabilities of Capstan and UniK. Experimental VM image deployment and instance creation is already handled by this integration.

The diagram in Figure 12 shows the architecture of this use case. Components marked with orange color have been revised or added to the diagram to support the advances in the implementation of the use case or underlying technologies.

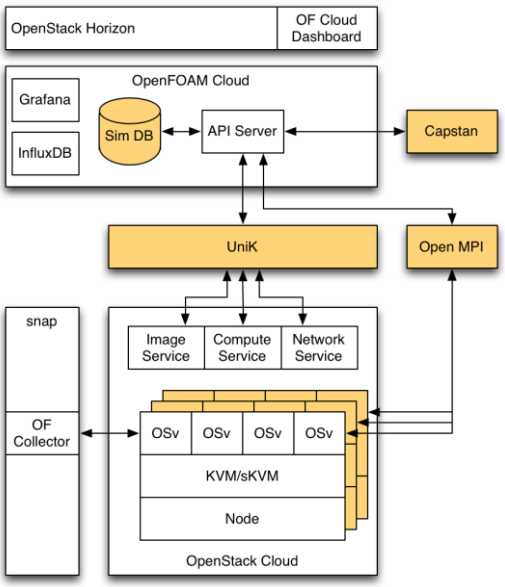

Figure 12. Architecture of the OpenFOAM Cloud application. The architecture has been revised since the initial version. Orange components have been added or updated.

3.2.4 Extended Dashboard Application

Previous enhancements focused mainly on the backend, whereas this feature describes the changes in the graphical user interface, i.e. the frontend of the OpenFOAM Cloud application. The dashboard has been extended to support all backend features (e.g., selection of simulation application and specification of simulation topology). Dynamic visualisation of

user's (tenant's) quotas has been built directly into the simulation orchestration dialog. This allows users to quickly see what resources are available and how the desired simulation is going to affect the quota.

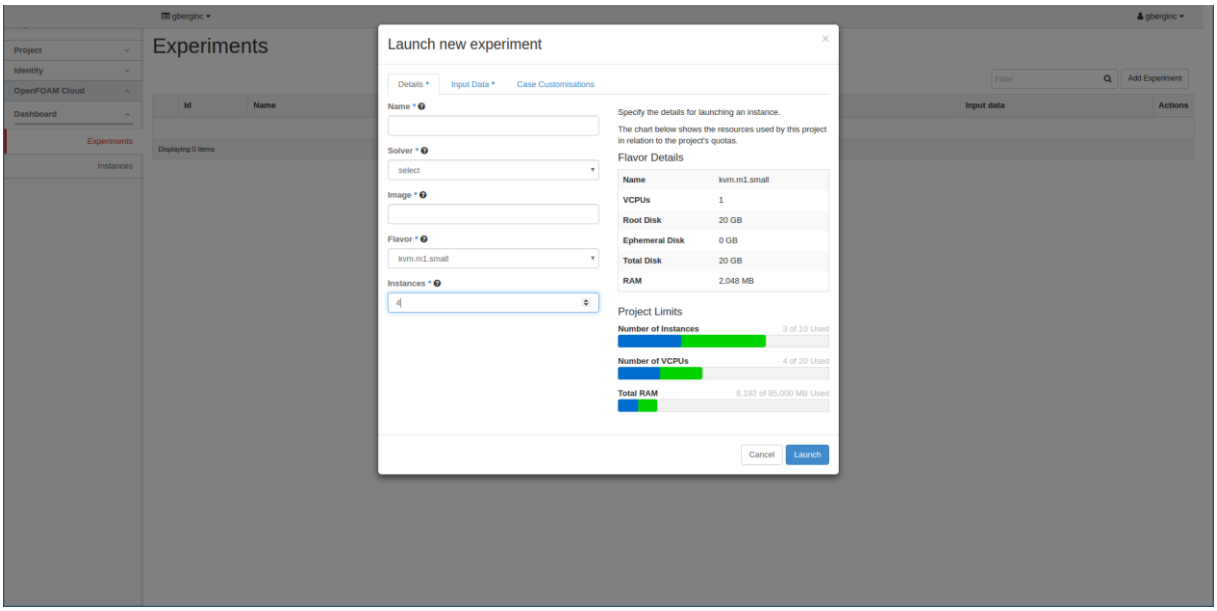

Figure 13. OpenFOAM experiment submission dialog showing dynamic user quotas.

Work has also started on the scheduling of simulations when overall required resources exceed user's quota. As it was described in the introduction to this section, one of the main interests for users like Pipistrel is execution of numerous parametric studies where the core of simulation is the same and only certain parameters as varied. Currently, it is only possible to create a parametric study whose overall resources are within user's quota. The scheduling is going to allow the user to request a larger parametric study. The only requirement will remain that a single simulation instance does not exceed the quota. Scheduling is going to monitor simulation execution and will launch new studies as resources from the previous case are freed.

3.2.5 Integration with vTorque

The capabilities and the components of vTorque are thoroughly presented in report D5.2 [3]. Because OpenFOAM simulations are frequently deployed in traditional HPC, several experiments and benchmarks have been performed using vTorque. Specific integration is not necessary because the same application packages that are used in cloud environment can be deployed in HPC. The most notable difference is the contextualisation of virtual machines. Contrary to the OpenStack cloud that uses an approach similar to Amazon AWS, vTorque provides this information in a separate file that is attached to the deployed instance.

3.2.6 Orchestration of OpenFOAM Appliances

Term orchestration refers to deploying all the components single application requires for proper operation (e.g., web server, application server with application and the database) and establishing the relations between them (e.g., the application is configured to connect to the database). In OpenStack orchestration support is enabled via OpenStack Heat project [29].

The benefits of orchestration for OpenFOAM simulations are two-fold. First, it greatly simplifies the evaluation process. Simulation environment is always bootstrapped in the same way. By enabling specific configuration (orchestration template) parameters, we are furthermore allowing deployment of different simulation topologies. Second, by using the orchestration we are able to deploy self sufficient simulation environments, easily deployable by our clients. OpenFOAM Cloud requires to be deployed on the target OpenStack cloud to be used. On the other hand, deploying simulations using orchestration templates will work on any OpenStack cloud.

Orchestration templates have been designed for both Linux- and OSv-based workloads. Current version of the templates performs the following steps

1. Create a volume of requested size that is to be used as the shared storage.
2. Deploy a Linux VM and mount the given volume.
3. Download OpenFOAM input data from the given location and decompose it based on the requested simulation topology (decomposition of the input case is a pre-processing step that prepares the data for parallel execution).
4. Deploy Linux or OSv virtual machines and automatically configure NFS mount for shared storage.
5. Initiate MPI execution on the master worker node.

To use the template, one can again use OpenStack Orchestration dashboard and fill template parameters from the graphical user interface (Figure 14). Upon selecting the OpenFOAM template, the service provides a dialog with all the mandatory parameters. Parameters are pre-populated with a set of allowed values, as defined in the template.

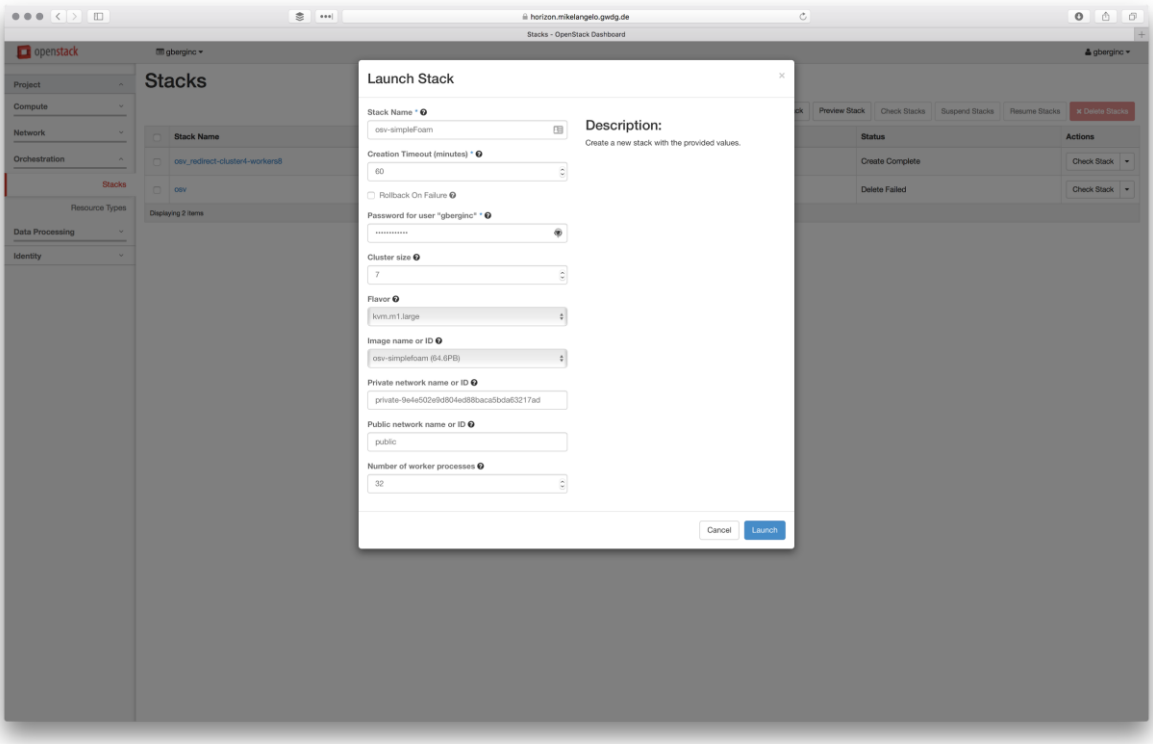

Figure 14. OpenStack orchestration dashboard.

Alternatively, the template can be pushed to the orchestration engine using command line tool. Example command:

```
openstack stack create -e lib/env.yaml -t heat-1.yaml --parameter="image=e2ead08a-7197-
4907-a557-75c678201b8a;flavor=kvm.m1.medium;cluster_size=3;worker_processes=8"
osv_redirect-cluster4-workers8
```

3.3 Work in Progress

At the time of this report two main activities are being conducted, namely implementation of the simulation job scheduler and use case evaluation.

We have presented in the previous section the need to express simulations requiring resources exceeding those available to the user. Initial implementation of an *ad hoc* scheduler is currently under internal testing. This allows for simple monitoring of pending simulation jobs and submitting them as soon as enough resources are available. The scheduler represents the basis for our future work on this area presented in the section on the work plan for the next year.

With the evolution of continuous experimentation platform (Scotty) presented in report D5.2 [3], evaluation of the use case is being automated. The following experiment dimensions are being investigated in these benchmarks:

- **Environment**: HPC, managed by vTorque, and cloud, managed by OpenStack
- **Input data case**: simple use case is being used for functional validation, while more complex tests drive the performance evaluation (we have 15 minute, 1 hour and 4 hour case, as well as a complex scenario with rotating elements during the simulation)
- **Number of nodes (HPC)**: in HPC environment we are able to allocate physical nodes exclusively. This allows us to control how virtual machines are deployed on the infrastructure.
- **Guest operating system**: Linux and OSv. It is important to understand the differences in the two operating systems. Our current evaluations have shown that OSv is in most cases slightly slower than Linux, thus further evaluation is needed.
- **IOcm configuration**: IOcm (D5.2 [3]) allows for setting the minimum and maximum number of cores that can be allocated to I/O operations by the policy manager. We are going to investigate two different options
    - Static allocation of IO cores (equal minimum and maximum): in particular in HPC environment where virtual machines are exclusively allocated to a physical node, it make sense to allocate IO cores statically.
    - Dynamic allocation of IO cores, up to 2. This experiment will be performed in cloud and HPC.
- **vRDMA configuration**: once prototype II of vRDMA, presented in report D4.5 [2], is fully deployed in HPC environment, it will be possible to compare the use of virtual RDMA interfaces. Virtual performance in Linux and OSv will be compared against bare metal performance.

Overall simulation time as reported by the OpenFOAM itself is the most relevant benchmark being used. Further metrics are being collected with the use of snap telemetry [3].

The following three graphs compare three different aspects of OSv versus Linux guests. Figure 15 shows the size of the Linux guest image compared to the OSv image. In case of Linux, Ubuntu cloud image [30] was used to provision an instance. OpenFOAM 2.4.0 installation procedure for Ubuntu [31] was then used to install complete OpenFOAM (without graphical tools). Finally, a disk snapshot was made on a suspended virtual machine. The resulting image size was approximately 2 GB. On the other hand, an OSv virtual machine image, composed from OpenFOAM solver package required 65 MB (including the OSv kernel).

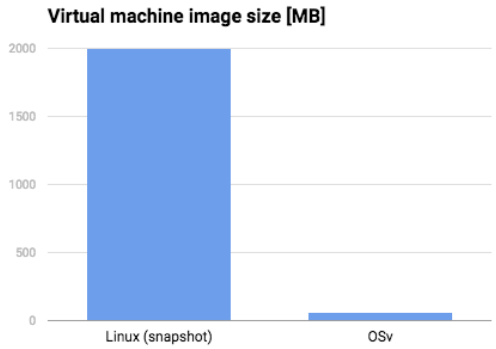

Figure 15. Comparison of the virtual machine image size (Linux vs OSv).

Boot time comparison is presented in graph in Figure 16. Linux was provisioned twice to evaluate the effects of image caching by compute nodes. The leftmost bar titled Linux (first boot) measures the time different phases took when the OpenFOAM Linux guest is provisioned from the previously created snapshot for the first time. The middle bar title Linux (cached) shows the same measurements when the Linux image has been cached on the compute node. Finally, the rightmost bar shows the times used for OSv virtual machines.

Phases presented in the graph are build, spawn and prompt. Build and spawn are the phases shown by OpenStack dashboard and were measured manually. Following the spawning phase, OpenStack reports the state of the instance as "running". However, the instance is not accessible immediately because several initialisation steps are still in progress. Additional time required for the machine to be reachable is thus shown as "prompt".

The results are very promising. OSv instances boot significantly faster than those based on Linux. Caching of machine images reduces the time needed to deploy Linux guests, but it is still slightly slower than OSv.

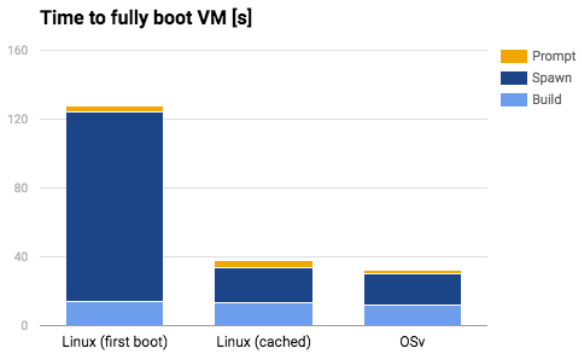

Figure 16. Comparison of the boot times for Linux and OSv virtual machines.

Finally, Figure 17 compares overall execution time of the same simulation using Linux and OSv based instances. Smaller cases have been used for this evaluation. First two test configurations (leftmost and the middle bars) show that OSv performs slightly worse than Linux (approximately 2% performance loss). On the other hand, the rightmost bar shows a significant performance degradation (~30 %) when running on a NUMA-aware CPU. NUMA stands for Non-Uniform Memory Access and is commonly used in multi-socket processors. The memory is divided into a number of regions (typically this number equals the number of sockets). Each socket then has faster access to one of these memory regions compared to the other.

When the virtual machine is configured to reflect the processor architecture, i.e. the NUMA specifics are passed onto the guest, Linux will be able to provide this information to the application running within. On the other hand, OSv does not have this capability. This means that even when given correct NUMA specification, OSv will treat the processor as a single socket processor with all cores equivalent in terms of memory access.

Because OpenFOAM uses Open MPI for parallelism, lack of correct processor configuration causes this degradation of performance.

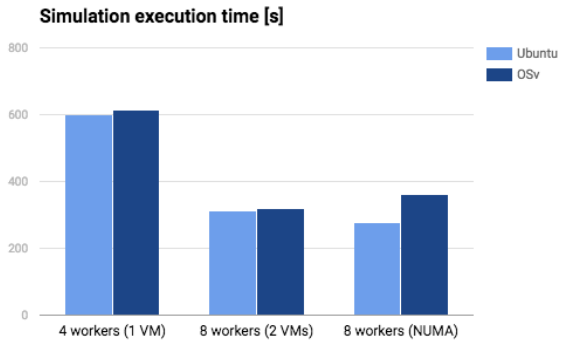

Figure 17. Comparison of OpenFOAM execution time for Linux and OSv virtual machines.

Similar evaluation has been conducted in the cloud environment. Figure 18 compares overall simulation times for OSv and Linux guests. Four different worker configurations were used: 2 or 4 virtual machines, and 8 or 16 worker processes (these are MPI processes doing the OpenFOAM simulation). Each virtual machine allocated the number of vCPUs required for the specific configuration (e.g. in configuration *2-16* VMs have 8 vCPUs, while in *4-8* they have 2 vCPUs). 2GB of RAM was allocated for each vCPU, i.e. 8 vCPU machine was allocated 16 GB of RAM. Same OpenStack flavors were used for OSv and Linux based guest.

OpenFOAM version 2.4.0 have been used. In case of OSv, precompiled MPM packages was used from the MIKELANGELO application repository [32]. In Ubuntu 14.04 was used and a default OpenFOAM packages were installed [31].

Results presented in Figure 18 are preliminary and show an interesting differences in OSv and Linux. For two virtual machines, OSv is able to outperform Linux, even up to 40%. On the other hand increasing the number of virtual machines, performance in OSv deteriorates drastically. OSv virtual machines are fully utilised throughout the entire simulation, so this performance drop does not seem to be caused by some blocking I/O calls. This is going to be thoroughly investigated.

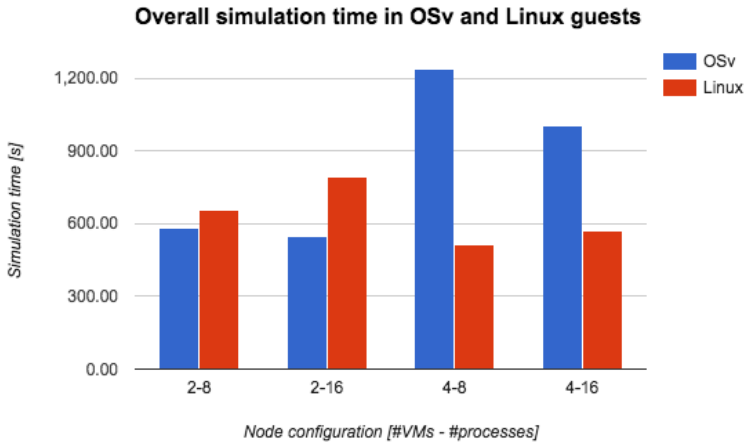

Figure 18. Comparison of the overall simulation time under varying guest operating systems and node configurations.

3.4 Work Plan for Year 3

Future work on the aerodynamics use case involves further integration of MIKELANGELO stack, enhanced simulation platform and thorough evaluation. Integrations is going to focus more on HPC environment. We plan to ensure that OpenFOAM workloads are working effortlessly in the virtualised HPC. The simulation management platform will see improvements in terms of defining simulations and parametric studies, monitoring and visualising the result. Evaluation will focus on continuous benchmarking of individual components of the MIKELANGELO stack as well as fully integrated target environments.

So far our main target for integration has been the cloud environment. We will continue to deliver new features implemented in core technical work packages into OpenFOAM applications. Primary integrations are related to application and package management, and monitoring. Other components are mainly integrated into target environments and are

indirectly addressing the needs of this use case. Nevertheless, they play a significant role in this use case, enabling efficient use of resources in virtual environments.

Application package management is evolving into a complete management suite. It is no longer focusing only on the composition of virtual machines. By integrating OpenStack into UniK it now provides means to manage complete lifecycle of applications: composition, deployment, monitoring and termination. Integration with these new capabilities is ongoing and will continue in the last year of the project. One of the main reasons for this is that OpenFOAM cloud backend services will not rely on just OpenStack in the future. We plan to separate it from the frontend provided by OpenStack dashboard. Users, administrators and DevOps should be able to use just the simulation environment, regardless of the underlying infrastructure.

The snap collector for OpenFOAM has given us the ability to track high-level application specific metrics in real time, without any changes to the application or the way users submit their simulations. In the next phase, this metric collection will be made even more flexible. Statically defined set of collected metrics will be replaced with a dynamic analysis of the data provided by simulations. Users will be able to specify a subset of metrics they are interested in in advance. This change is also going to allow external use cases to monitor their own simulation-specific metrics. We also plan to implement aggregation of simple metrics across multiple simulations comprising a single parametric study. This will allow for efficient analysis of simulation parameters under varying initial conditions.

Enhancements to the simulation platform will be made to the frontend (UI) and backend. Backend will be enhanced with more robust simulation scheduling and more generic deployment options. Scheduling will allow deployment of arbitrary simulation configurations, even exceeding available resource quotas. Deployment of workloads in different infrastructure providers will be mainly done through integration with UniK. Additional work is required to support configuration of these providers for the end user.

Frontend is the main demonstrator of this use case and is also one of the important exploitation deliverables of the entire project. It is therefore crucial for frontend to provide compelling presentation. Frontend improvements have been proposed by Pipistrel through a series of interviews and experiments with some of the simpler workloads. Frontend will be extended to support capabilities presented above (integration and backend). We also plan to improve simulation data handling to make it more intuitive. Instead of having to upload input cases into separate object storage (for example, Amazon S3), users will be allowed to upload data directly, as part of the simulation configuration workflow. Results (intermediate and final) will be stored and presented to the user as part of the simulation detail.

Finally, we plan to evaluate the use case periodically to benchmark the influence of improvements in the MIKELANGELO components. So far, the benchmarking has been manual. However, for the last year we plan to integrate them with the overall continuous integration system provided by GWDG and USTUTT. Besides simplifying the entire process it will also ensure benchmarks are regularly updated and always executed in a similar way.

3.5 Conclusions

The aerodynamics use case has been used as one of the main driving force for the HPC enhancement in OSv since the beginning of the project. Getting OpenFOAM and Open MPI run seamlessly in OSv has given us the chance to focus more on the implementation of advanced capabilities relieving some typical pain points of OpenFOAM (or even HPC) workloads.

This section has presented two main foci, cloud and HPC. Cloud infrastructure allows us to work with flexible simulation workloads. Users are not relying only on the compute power, but rather ease of use, in particular when working with a multitude of (small-scale) simulations in the early stages of the design process. Although the graphical dashboard is the most visible part of this use case, it is the backend service that carries the true value. It is responsible for deploying and managing simulations, retrieving the data and collecting relevant metrics, both from performance and simulation point of view. Primary target for the next phase is therefore the backend, while the dashboard will primarily be used for demonstrating the overall results.

Being one of the typical HPC workloads, this use case is ideal for validating and evaluating HPC integration. This involves specific components of the MIKELANGELO project, such as IOcm, vRDMA, OSv and MPM on one hand. On the other, the evaluation considers the overall integration of all components with a widely popular HPC frontend - Torque PBS. Further plans for the development of this use case mainly include additional application packages and improvements to the Open MPI, in collaboration with WP5. The latter is of vital importance to facilitate acceptance of the results in the HPC world.

4 Use Case: Cancellous Bones

The purpose of this simulation is to understand the structure of the cancellous, or spongy, parts of the human bone, in order to efficiently and safely attach required implants to it. Using this simulation, implants can be produced in a more accurate fashion, which increases production yield and at the same time improves the healing time for bone implant surgeries as well as the lifespan of the implant. To understand the structure of the human bones it is important to get more insights than it is possible with a scan of the bones. This simulation generates a more accurate model and shows how the bone reacts to pressure. This has an impact on the medical aspect of understanding the patient and his problem, helps to understand how implants should be designed, gives an overview where they can be attached to increase the lifespan and helps students to get a general knowledge how a bone reacts to stress [33].

HPC software is mostly built around libraries and hardware technologies that will simplify the complexity of HPC technologies. E.g. MPI [34] helps to build complex communication structures and make use of technologies like RDMA. To a certain degree, most of the HPC applications are optimized to run in a given environment with highest performance. A programmer doesn't have a chance to change the hardware within the HPC system. He has to put the focus of optimisations of his own code.

The cancellous bones use case (UC), as other HPC simulations, is built and optimised with a set of libraries that are commonly used. It has long lasting sequential parts, that are typically independent of each other. Solve the equations inside the simulation has to be done in serial, but many of them can be computed at the same time. The communication between processes is low in comparison to the data collected from the storage. Interprocess communication is used to manage the distribution of the work. The data for calculation is collected from the shared storage. In many cases the storage is also the limiting factor, i.e. if the storage can not provide the data fast enough additional compute nodes will not speedup the overall runtime.

The Cancellous Bones use case is a widespread example pattern for HPC applications. It is based on MPI, written in Fortran and has the capability to scale up to hundreds of nodes. It is built around the master / worker pattern. The high level overview of the cancellous bones architecture can be seen in Figure 19. Consequently, it requires at least two threads. The master will distribute the information for the calculation. One or more workers will take this information, fetch the data and start calculating. It scales to the point where the shared workspace file system, where the data is stored. If the storage is not able to provide the data the worker will wait then and no scaling effects will be possible.

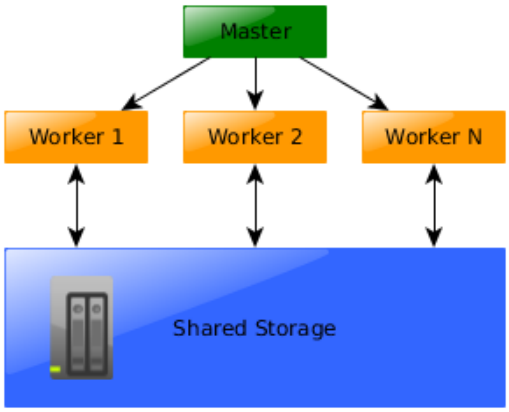

Figure 19; High level overview over the cancellous bones architecture

The problem with virtual machines in HPC is mostly associated with I/O in general. Since technologies like VT-x [35]/AMD-V [36] make the virtualization of CPU load less problematic, storage access via network is still an issue.

To evaluate the HPC components of MIKELANGELO as shown in Figure 20, this use case can be used to cover the following functional tests and system verification tests:

- The software stack behaves, from a user perspective, without any noticeable differences.
- The implementation of the components inside the software stack are running successfully.
    - vTorque makes it possible to run VM on HPCs
    - vRDMA enables RDMA for VMs
    - IOcm speedup the I/O
    - OSv shrink VMs and improves the speed of VM executions
    - Snap monitors the performance
    - SCAM secures VMs

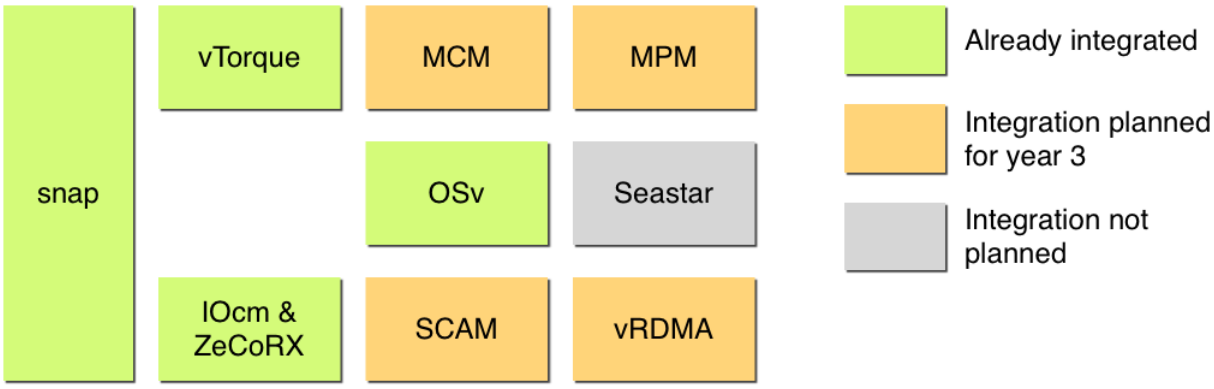

Figure 20. MIKELANGELO Components Used by the Cancellous Bones Use Case.

In addition to the plain functional evaluations, the performance of the simulation can also be observed. Snap, as a monitoring solution, is deployed on the test environment to capture performance metrics. This enables multiple metrics collected in a central and general way. The main advantage of this approach is that it does not depend on the application it is monitoring[3].

4.1 Plan for Year 2 and Year 3

The higher objective of the MIKELANGELO is to converge cloud and HPC with a common software stack. The virtual environment must be near to the bare metal node performance with the components developed within MIKELANGELO. For this UC the plan is to run it on the bare metal nodes, inside a standard Linux VM and inside the new cloud operation system OSv. This involves, first building the application inside the one environment and observe what is needed to run the simulation regarding the software stack.

The initial plan for the implementation, integration and evaluation of this use case has been described in previous reports. The latest update has been provided as part of report D2.2 [4]. Because the use case has been thoroughly defined since the beginning of the project no major changes have been made. Changes are made in procedures of how and what should be measured to see differences in runs. Running the same use case several times has shown that there are some irregularities in the runtime of the simulation [28]. The focus of getting more data leads to a metadata collector [3]. Snap on the other hand has been extended to have more insights.

One of the major updates that happened in this period, is the port of the cancellous bones use case to OSv. This is a huge step towards understanding how a uniform operating system looks like. Furthermore, to have a system to automatically trigger tests and run experiments in huge quantities, the development of the continuous integration (CI) [3] has progressed. Parts of the development have started in year 2 and will be integrated in year 3. In order to be able to submit a greater amount of experiments, it is necessary to integrate the CI to a degree, that it is possible to submit jobs via the CI into the vTorque implementation. This development, the workload generator, is in a first prototype available and will be finalized soon. For a more informations please see Section 4.2.

The initial set of measurements have shown that more and more fine grained measurements are needed. This use case serves as a blueprint for the integration of the continuous integration/experimentation in cooperation with the HPC testbed and the Scotty integration. The combinations of different components has to be examined. To cover all permutations of the various parameter combinations, and to execute them several times in order to be able to compare collected information and benchmarks to earlier results, automatization is inevitable. The data itself is collected by snap and the metadata collector and tagged by the

vTorque implementation to trace and document the experiment with a unique identifier. For a more informations please see Section 4.3.

4.2 Work Performed in Year 2

In year 2 the main work was to run the use case inside OSv. The operating system differs from the general purpose Linux guest due to the fundamental design, e.g OSv is a single user single process operating system. HPC applications run usually with a full featured operating systems. The process of porting the use cases is described in the following section. Also covered in this section is the integration work of this use case, the scenarios that has been worked on and the work in progress.

4.2.1 Use Case Port to OSv

The Cancellous Bones UC has been ported to OSv operating system. The porting process started with the setup of the source code within the MIKELANGELO applications directory, that is, the source code is currently under the path "~/osv/mike-apps/cancellous-bones/serial_fmps", where "serial_fmps" contains the code for the Cancellous Bones solver.

```
user@node0102:~/osv/mike-apps/cancellous-bones$ ls
GET  Makefile  serial_fmps  usr.manifest
```

To fully integrate the UC in the OSv building process, only a few configuration files (Makefile, GET and usr.manifest) had to be added to the UC directory. The Makefile is automatically called during the building process of the OSv image. GET script is invoked to check if the source code of the Cancellous Bones solver exists, it exports the necessary environment variables for building the solver binary and creates the usr.manifest file that contains a description of all the libraries that need to be included in the OSv image. In this way the solver binary would be able to run in OSv. Once these configuration files were ready the Cancellous Bones UC was built using the default OSv build system, part of the OSv source tree:

```
scripts/build mode=debug image=OpenMPI,cancellous-bones,cli,httpserver
    nfs=true
```

An OSv image is composed of several modules that are specified on a module list. OpenMPI package provides modified Open MPI implementation to support OSv threading model, while cancellous-bones module provides the libraries and source code of the use case simulation. CLI and HTTP server modules are added for debugging and monitoring purposes.

These two modules do not affect the execution. Finally, the NFS is enabled to allow access to the shared workspace.

The "cancellous-bones" directory was added in the module list of the OSv image building process. Once the OSv image is ready, the Cancellous Bones solver has been executed as follows:

```
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;
```

The binary file for the Cancellous Bones solver has the name of "`struct_process_x86_64.so`". The solver is an MPI program. To run MPI programs in OSv the "`orterun.so`" command needs to be used. This command is equivalent to the "mpirun" command more commonly used in Linux systems. Parameters are passed to the solver by means of an input file called "`struct_process.input`". This input file contains different kind of parameters, e.g. the process name, the path to the input data, etc. The most important parameter is the grid resolution to be used in the experiment. This grid defines the blocks of input data processed by the MPI workers:

```
0 0 0
5 5 5
```

The grid resolution presented above means that 6x6x6 blocks of input data will be used to obtain a 3D model. That is a total of 216 blocks that will be distributed among the MPI workers of the Cancellous Bones solver.

However, there have been some technical issues when running the Cancellous Bones solver in OSv. Some C/C++ functions were missing in the GNU C library (libc.so) that is provided in OSv by default. To solve this, the critical missing functions have been implemented and the rest have been just stubbed. Another important issue was regarding running the experiment in a 64bits system. 64bits systems have memory addresses of 64 bits as positive integers from 0 to 2^{64} - 1. In Fortran 90, the programming language used to write the code of the Cancellous Bones solver, there is no data type to represent positive integers greater than 2^{32} - 1. This problem is not specifically related to OSv, but it did culminate with this port because OSv is 64-bit by default. This is a real problem when porting Fortran 90 code to a 64bits system. The workaround used in our case consisted of moving all operations that use memory address into a C/C++ function that is called from the Fortran code. In C/C++

positive integers of 64bits are possible and if the value returned by the C/C++ function to Fortran is less than 2^{32} - 1, this implementation makes it possible to run the use case.

The experiment was first run in the debug mode with the mentioned resolution. The experiment completed, that is, all the MPI workers finished the work, after approximately one hour. 4 virtual CPUs were used for running 4 MPI workers + 1 MPI master. There is no sense to have an additional virtual CPU for the MPI master as this process is only active when the others are waiting for it to deliver them more work. In this way, all the virtual CPUs used were always busy. Finally, 2GB of memory was required for the experiment as seen in Figure 21.

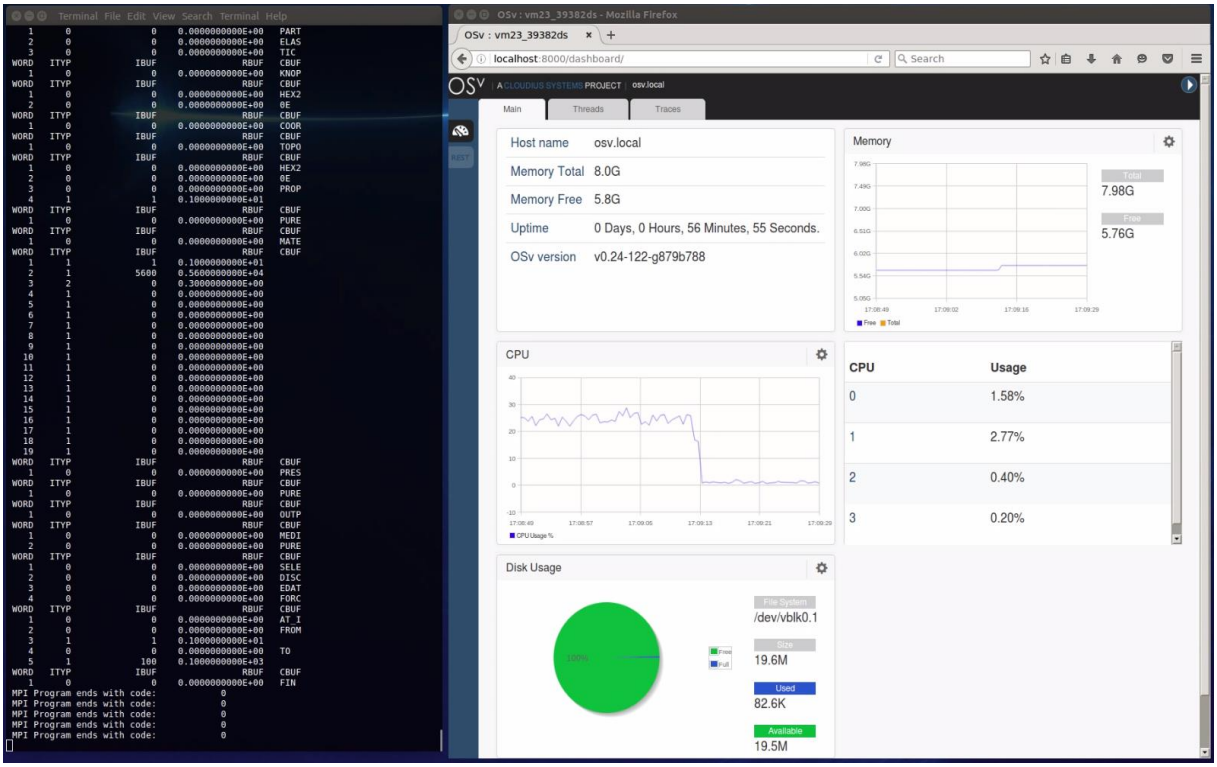

Figure 21. On the left, a console with the output of the MPI processes. On the right, the OSv dashboard with the memory, virtual CPUs and disk usage charts.

The output of the experiment, besides the logging data, contains also the 3D model of the cancellous bone. For each block of input data the corresponding part of the 3D model is generated. The Paraview tool was used to render all these parts and to visualize the whole 3D model. Figure 22 represents the experiment grid and the 3D model of a cancellous bone generated by Paraview.

Although the experiment finished well, that is, the 3D model was generated successfully, some technical issues needed to be solved. For example, the MPI processes (workers and master) finished but the OSv process kept running, that is, the OSv virtual machine didn't

shut down when the MPI program ended. After reporting this issue, the MIKELANGELO team working in OpenMPI for OSv solved it.

Unfortunately there is another issue that has not been solved yet, i.e. currently it is not possible to run the experiment when OSv is built in release mode. That is, the experiment runs well when OSv is built in debug mode but in release mode the experiment output freezes as soon as it starts, with no error message. It appears as if all MPI processes are paused. In order to solve this issue more effort will be needed. Luckily this issue is not blocking other tasks as the integration of the Cancellous Bones UC running in OSv on an HPC system can be done first in debug mode and just switched to the release mode when the issue is solved.

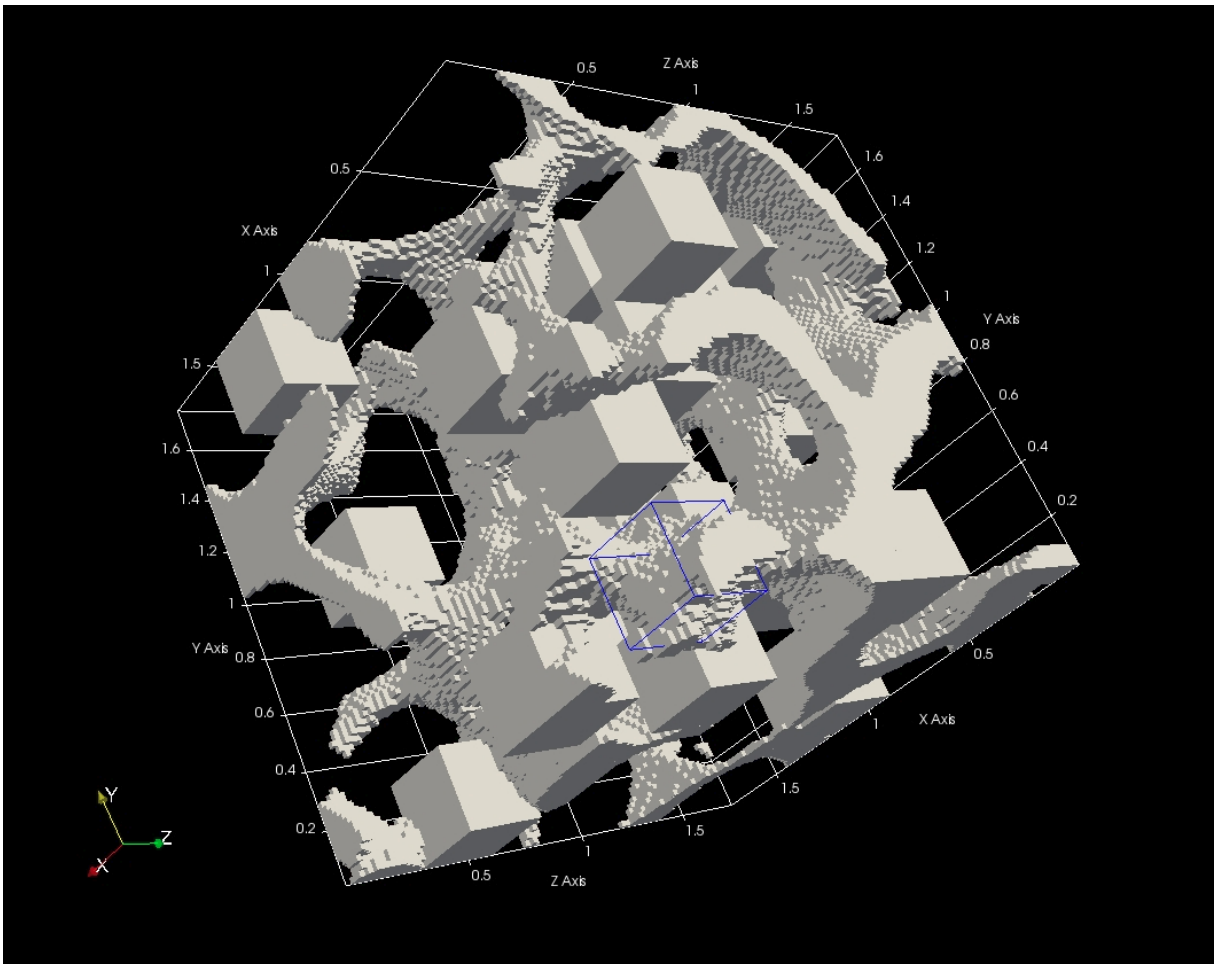

Figure 22. 3D model of a Cancellous Bone generated in Paraview. The grid used is shown in white. A single block of data (or domain) is highlighted in blue for illustrative purpose.

4.2.2 Scenarios Covered

Scenarios are introduced as additions for low level requirements, they abstract the requirements in a way that is clear why they are neede. In addition it merges some

requirements into one scenario. From the perspective of the use case, there are several scenarios that are important for a 100% successful run, inside an HPC environment. Not all of them are fulfilled yet or they are in a "work in progress" state. Missing parts that are related to the infrastructure or other MIKELANGELO components will be documented in the requirements document. These scenarios give other work packages the opportunity to improve their components. The following list of scenarios gives an overview over scenarios that are close or are completely fulfilled.

S6.BoUC.01 Configure the hardware resources needed by a job and **S6.BoUC.10** Run jobs in the cluster using OSv-based VMs

S6.BoUC.01 is a scenario which is focused on the resources, required by the job, either for Linux- or OSv-based VMs. The configuration should allow specification of the infrastructure. To complete these two scenarios it is important to know what is needed by OSv to ensure transparent support for OSv and Linux. After testing with the use case and OSv the information to implement OSv into vTorque is passed to Work Package 5 (WP5).

S6.BoUC.07 Parametric study

Parametric study is a form of running the use case. It runs with different sets of input data or with different settings. This is needed to collect data in an automatic way with less configuration than in a manual execution. From the use case perspective, this is prepared. As soon as the CI integration is in place this can be executed. To find suitable values regarding the use case more testing is intended.

S6.BoUC.12 Get resource metrics per job

This scenario describes the need to have additional performance data. With snap and the metadata collector it is possible to collect the performance counter out of the influxDB and the environment out of the job log.

4.2.3 Integration with MIKELANGELO Components

The use case is able to use almost all components developed within MIKELANGELO, so it is possible to evaluate and validate them. On the host side it shows the working implementation of vTorque and can compare the performance of KVM to sKVM. On the guest side it can run now in two different environments and is therefore able to compare OSv to a standard Linux guest. The use case shows also how Snap and the metadata collection can be used to trace and document experiments. The performance collection of Snap will show the performance difference between the different technologies and help to improve the development within MIKELANGELO.

For now we can see that it is possible to run the use case inside OSv as shown with the porting and first tests with the new operation system. To get more confident with the performance, more tests and experiments has to be performed. As soon as more components are deployed, integrated into the test environment, refined and the experiment pipeline is in place it is possible to collect more information.

A quick baseline test was taken for the use case on raw nodes and standard Linux guests previously [28]. To have a more solid ground the baseline will be remeasured with the CI to have a similar setup as for the performance measurements. This will be integrated in a more general testing environment and will be automated. This has the advantage to save time and not focus on the experiment setup, but more on the experiment itself. The results of the experiments will then be more in the center and will generate reproducible results.

The ability to run the use case inside OSv has shown that the core functionality of the use case can run, without major rework of the code. The modification has mainly happened within the MPI implementation [37]. MPI has been modified previously to run threads instead of processes. OSv is a single user, single process operation system. This leads to the problem that MPI can not spawn processes, it hast to work with threads instead. This shows no improvement so far, but it shows that it is possible to run HPC workload inside OSv. This supported the porting, due to the fact, that handling of processes for this use case is managed by MPI. We were not able to see any improvements or disimprovement so far. The performance test of OSv with our use case starts as soon as the integration into vTorque is in a more advanced implementation and the CI is integrated. This will show the functionality of vTorque and OSv as well as OSv and the use case.

4.2.4 Work in Progress

There is integration work that is close to completion but has not been fully evaluated or fully integrated. One of them is the metadata collector [38], which is implemented, but not fully integrated into the testing environments. The other component is the workload generator for the CI. The use case will be integrated in Scotty to make use of its new functionalities.

To give an quick outlook for the future work: vTorque has to be able to collect the metadata and Scotty has to work with vTorque.

4.3 Work Plan for Year 3

Albeit the functionality of this use case has been tested in the current deployment scenarios, there are missing parts allowing the use case to be properly finished. From the perspective of this use case two integrations need to be made simpler and more flexible: integration of OSv with vTorque and vTorque with Scotty, our experimentation platform.

Following these integrations this use case will be able to set up the experiments that cover combinations of different components, like: OSv and IOcm and compare the results to other experiments like Ubuntu guest and IOcm. Such an experiment will show the difference between the guest systems regarding the I/O performance.

On the one hand this use case will be prepared with parameters that enables fast execution times to see the implementation works in general. On the other hand more realistic parameter with longer execution time will be prepared, to see performance and scaling effects.

To execute this it is planned to provide several experiments to the CI and different workloads. The amount of experiments will increase over the project time.

With snap and the metadata collector, two collectors for performance and environment data are almost in place. They have the capability to quantify the results of the development and show the use case behavior in different environments. They will enable to get more insights into the software, show the hardware load and document the environment in which the experiments have run in.

This use case has no needs in overhauls, rewriting code or implementing new features at this point. It runs now inside three different environments without major drawbacks. The future plan is to utilize the use case to evaluate the performance of the separate components of MIKELANGELO. The largest amount of work in year 3 will be done on the automation and setup of experiments, where this use case has a central part. Changes will be made mostly for the environment and the configuration of the use case, to run different data sets and run on more nodes. If the use case is lacking of additional functionality regarding the experiment setup, this functions will be implemented and a new binary will be build and distributed.

In a later state of the project it will be evaluated, how to change the use case setup to run inside a OpenStack cloud environment similar to the OpenFoam use case. This will show what modifications have to be done in order to have an application that can be run in nearly any environment. This will also show how to converge cloud and HPC to a truly common stack. To see the impact it is needed to see how an HPC application runs inside a cloud setup to conclude the advantages or disadvantages over such a setup. Performance is the most important aspect of using HPC environments over cloud installations.

4.3.1 New Features

A new feature results directly out of the Scotty integration. It enables to set up a larger amount of experiments. This use case will provide a set of experiments to evaluate the components of MIKELANGELO.

The amount of combinations possible, with components developed inside MIKELANGELO is too large to run them by hand, document them, collect the data and rerun them several times.. For example, to see scaling effects it should be tested inside the three different environments (bare metal node/ OSv/ Linux guest) in three different configurations (IOcm/vRDMA/without both). To see scaling effects, I/O and CPU performance. This set of tests should to be executed on one NUMA node, on one complete node, on two nodes, on 4 nodes, on 8 nodes and on 14 nodes. This will result in about 42 tests. Due to the capability of vTorque to start more than one VM on a single node it would be interesting to see if the use case gain some performance with vRDMA and vms pinned on NUMA nodes.

To run all these tests, Scotty will supply this use case with a system that will automatically compose the experiments, submit them via vTorque to the USTUTT test system and provide the log of the test, corresponding metadata and evaluation results back to the integration management system.

4.3.2 Risks of a huge data collection

To have a realistic view of the use case and the new developed components, the use case should be configured in a way that represents real-world HPC applications. HPC experiments can run for weeks, but this is not possible in order to collect data that is detailed enough to evaluate the components of MIKELANGELO The task is now to find a small example that is representative, but has a short enough runtime to be executed multiple times, to have an accurate set of representative data. On the other hand the experiments should not block the test system the whole time for this single use case. Other use cases like the "Aerodynamic Maps" case should have the possibility to run on the test systems as well. It is important to automate the two experiments to run them and collect the execution data to have enough time to analyse the output of the experiments.

4.4 Conclusions

To summarize the work on the use case, only small modification had to be done to run the use case inside OSv. For now the porting to OSv was successful in a first setup, this will be extended and executed with vTorque. But the first test has build the confidence to progress and extend the use of OSv in the next period.

With the collection of more metrics this use case will produce comparable results regarding the different components and settings for the environment/use case. The integration of the CI makes it possible to collect and document the use case experiments in an easy way and allows to rerun different experiments over and over. For a valid analysis solid performance data is the key. It is needed to run the experiments in an automated way. This automation for

will be used in year 3 for the comparison and validation of the MIKELANGELO components like IOcm, vTorque and vRDMA.

In a later state, it has to be evaluated if it is possible to run the use case inside VM in a cloud environment like OpenStack. The different types of a purely cloud environment has to be analysed and tested if it is possible to extract the use case out of an HPC environment and run in cloud environment.

5 Use Case: Cloud Bursting

From the business side the "cloud bursting" use case is about being able to serve an unusual burst of customers, in events such as Black Friday.

From the technical side, the cloud bursting use case is about fulfilling one of the initial promises of the cloud: elasticity. Elasticity in a cloud is the ability to grow or shrink the underlying computing resources of your cloud application without interrupting the regular behavior of the service. One of the ways to provide elasticity is to add more machine to the cluster serving the customer requests then re-balance the workload among them. In a cloud, more VMs would be added or removed from the cluster. Adding more machines to scale does not come for free: In most cluster applications the data would need to be rebalanced among the cluster machines, and it is difficult to achieve this rebalancing efficiently, without hurting the cluster's capacity to handle regular requests.

To give an example of this use case, imagine an online merchant, which serves requests of, say 100.000 simultaneous customers, on a regular basis. Then, Black Friday approaches, when 1.000.000 simultaneous customers must be served. More VMs for HTTP frontend servers are added to serve additional web requests. That is the easy (and solved) part of the scaling. But it leaves the merchant with the issue of scaling the database where the online transactions and product catalogue is stored. More virtual machines can be added to the database cluster, but a database machine is not useful without its *data*, so the existing data now needs to be *streamed* (copied) from the existing nodes, to the new node(s).

This - *quickly* scaling a scalable database - is the "Cloud Bursting" use case which we will consider. We chose to implement the cloud bursting use case at the database level because databases are often the bottleneck regarding the latency and throughput metrics. As explained above, it is easy to add load balancers and additional HTTP frontends but it's very hard to make a database that can scale well - let alone quickly. The faster the database cluster growth can be done, the more the database cluster will be able to be reactive to the traffic burst generated by the merchant's customers and the more the virtual machines billing will be fit to the real usage and end user needs.

It is not enough that increasing the scale of the cluster happens quickly - the second goal of the cloud bursting use case is to demonstrate that performance of the cluster does not significantly deteriorate during the period of quick growth because of the huge amount of data *streaming* (sending of data between nodes) which needs to happen in this period.

5.1 Apache Cassandra and ScyllaDB

For this use case, we chose the Apache Cassandra [39] NoSQL database, as one of the most popular and versatile *scalable* databases used in the cloud.

NoSQL databases are often categorized along two directions:

- Data model:
  - Key Pair DB (like Redis, AWS DynamoDB),
  - Document DB (like Mongo),
  - Wide Row, or columnar, row store, DB (HBase, Cassandra and ScyllaDB), and
  - Graph DB (like Titandb).
- Consistency, Availability and Partition-tolerance:
  - Consistent and Available (CA),
  - Available and Partition-tolerant (AP) or
  - Consistent and Partition-tolerant (CP).

Cassandra uses Wide Column data model and is Available and Partition-tolerant (like DynamoDB, Riak). According to the CAP theorem [40] having all three properties is impossible.

Initially, our plan was to demonstrate the performance improvements of running Apache Cassandra on top of OSv, but we were only able to demonstrate modest improvements with that approach. This is why ScyllaDB (one of the MIKELANGELO partners) embarked on a new approach: Rewriting Cassandra using C++ and a newly designed set of asynchronous APIs, **Seastar** [41], which can result in much better performance than the traditional Linux APIs. We describe Seastar in much more detail in deliverable D4.5, released in parallel with this one. The name of the new Cassandra rewrite is ScyllaDB [42] (same name as the company), and just like Apache Cassandra and Seastar, ScyllaDB is also open source.

As mentioned above, one of the advantages of the Cassandra design is that it scales well to clusters of many nodes. Since ScyllaDB uses the same distributed algorithms, it also scales well to many nodes - but as we plan to show, thanks to an improved design an implementation, and especially due to the new Seastar APIs, it scales much faster. Benchmarks in [42] also demonstrate that a ScyllaDB node can handle as much as 10 times more throughput than a Cassandra node, which means that a ScyllaDB cluster would not only scale faster than a Cassandra cluster - it will also have to scale to fewer nodes to achieve the same desired throughput.

This use case is important because web companies must be able to handle traffic spikes whenever they happen, since it has been proven that a customer that sees an HTTP error

page or experiences high latencies will have a very negative view of the company and will probably not come back to visit the web site.

In a general sense the cloud bursting use case is relevant for MIKELANGELO since the cloud is more than just virtual machines: some clouds provide VMs, some clouds provide infrastructures, some clouds provide APIs and some clouds provide lambda computation. We see the ScyllaDB database as a cloud CQL API provider ("CQL", "Cassandra Query Language", is Cassandra's SQL-like query language). This statement is backed by the fact that some of ScyllaDB partners like IBM Compose [43] do sell ScyllaDB database storage and traffic as a service. This fact is the proof that ScyllaDB is well placed to benefit from the API-fication of the cloud that is currently ongoing.

The cloud bursting use case validates the breakthrough of Seastar [41], a C++ asynchronous programing framework that is part of MIKELANGELO, by showing that its flagship application ScyllaDB can bring some fresh technical performance improvement to a problem that Datastax [44], the main Cassandra contributor, has been trying to tackle since 2008.

Additionally, since the cloud bursting use case is I/O intensive, it could benefit from the ongoing work on WP3 (IOCM and vRDMA) to improve I/O performance.

5.2 Plan for Year 2 and Year 3

From the start the plan was to evaluate and try every idea that could lead in a cloud bursting improvement since this feature is required by ScyllaDB customers. Cassandra being coded in Java and being in overall slower than ScyllaDB the particular cloud bursting use case has room for improvement. The improvement will try to be done mainly in the Seastar C++14 asynchronous framework that is the basis of ScyllaDB. However the ScyllaDB team will try to do some code reuse whenever it's possible if another feature of the database could benefit from the core Seastar features.

5.3 Work Performed in Year 2

This year, we have done a lot of work to improve Seastar (See D4.5) [2] and ScyllaDB for the cloud-bursting use case:

- Adding an I/O scheduler. One of the key requirements that arose in the "Cloud Bursting" use case was to ensure that performance did not deteriorate significantly during a period of cluster growth. When a Cassandra cluster grows, the new nodes need to copy existing data from the old nodes, so now the old nodes use their disk for both streaming data to new nodes, and for serving ordinary requests; It becomes crucial to control the division of the available disk bandwidth between these two uses. For this, we implemented this year an I/O scheduler for Seastar: The application can

tag each disk access with an I/O class, for example a "user request" vs. "streaming to new node", and can control the percentage of disk bandwidth devoted to each class. The I/O scheduler is internal to scylla and not related to WP3's IOcm.

- Adding IOTune - Seastar's disk API is completely asynchronous and future-based just like everything else in Seastar. This means that an application can start a million requests (read or write) to disk almost concurrently, and then run some continuation when each request concludes. However, real disks as well as layers above them (like RAID controllers and the operating system), cannot actually perform a million requests in parallel; If you send too many, some will be performed immediately and some will be queued in some queue invisible to Seastar. This queuing means that the last queued request will suffer huge latency. But more importantly, it means that we can no longer ensure the desired I/O scheduling, because when a new high-priority request comes in, we cannot put it in front of all the requests which are already queued in the OS's or hardware's queues, beyond Seastar's control. So clearly Seastar should not send too many parallel requests to the disk, and it should maintain and control an input queue by itself. But how many parallel requests should it send to the lower layers? If we send too few parallel requests, we might miss out on the disk's inherent parallelism: Modern SSDs, as well as RAID setups, can actually perform many requests in parallel, so that sending them too few parallel requests will reduce the maximum throughput we can get in those setups. The requirement to tune the I/O parallelism to what the disk can actually handle led to the development this year of "IOtune", a tool that runs on the intended machine, tries to do disk I/O with various levels of parallelism, and discovers the optimal parallelism. The optimal parallelism is the one where we get the highest possible throughput, without significantly increasing the latency. This is the amount of parallelism which the disk hardware (and RAID controllers, etc.) can really support and really perform in parallel. After discovering the optimal parallelism, IOtune writes this information to a configuration file, and the Seastar application later reads it for optimal performance of Seastar's disk I/O. The same remark than for the I/O scheduler must be done here IOTune may interact with IOcm.
- Enhancing an RPC framework to support efficient streaming of data. A number of cpu consumption optimizations have been done on the Remote Procedure Call framework used to stream the data from nodes to nodes in order to minimize the load put on the virtual machine by the streaming operations.
- Enhancing streaming to reduce effect on ongoing load. A number of enhancement on the streaming operation themselves has been done to minimize the impact on the regular workload. Here again the strategy was to reduce the processor consumption since ScyllaDB is very CPU intensive.

- Enhancing ScyllaDB to provide more consistent performance. Virtual dirty memory management was added to avoid hitting a memory wall, where no more memory is available and latencies increase drastically all of a sudden. Virtual dirty memory is a back-pressure mechanism. When more than half of the memory is used the number of request served is slowed down before all the memory is used in order to avoid to have to push the brake all of a sudden.

The experiments were conducted on Amazon EC2 manually, in order to show that a ScyllaDB cluster grows faster than a Cassandra cluster without impacting negatively the latencies. There are two types of server VMs in the test suite in this setup: load servers VMs to generate the workload and database servers VMs hosting the databases.

Two tests were conducted in the experiment:

1. The first test is to load Cassandra to it's maximum throughput and see how it behaves during cluster growth.
2. Then the second test is to see how a ScyllaDB cluster grows with the same workload (the same level of requests that Cassandra could sustain). The experiment simulate a database cluster growth during event such as Black Friday.

Here is a overview of the experiments:

- Test 1: 3 i2.8x servers (each has 32 vcpus, 244 GB of ram, 8x 800GB SSD) using Cassandra with replication factor (RF) 3 and a data set of 120GB (using partitions of 10K) under a write workload - increasing the cluster size to 4, 5, and 6 nodes. The loaders are set to maximize the throughput.
- Test 2: 3 i2.8x servers using ScyllaDB, again with a RF 3 and a data set of 120GB (using partitions of 10K) under a write workload - increasing the cluster size to 4,5,6 nodes. Loaders will be set to use the same throughput from Test 1

The number of operations per second (ops) resulting from Test 1 and Test2 as can be seen in Figure 23.

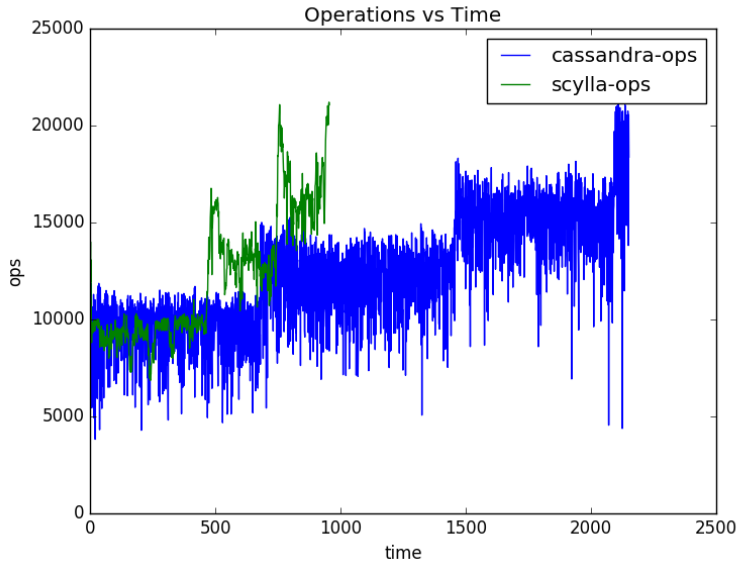

Figure 23. Operations vs Time

In Figure 23, we start (in time=0) with three nodes (VMs), add a fourth node and wait for the data it needs to be streamed to it, and then repeat this process adding a fifth and sixth node. The steps in performance correspond to the time where a new node became operational, after all the data it is responsible for was streamed to it. We can see that the streaming process in ScyllaDB (the width of a step in the green line) is roughly twice faster than Cassandra (blue line), and the entire growth to 6 nodes happens roughly twice faster. We believe that with more work, we will be able to demonstrate even more than this 2-fold improvement in performance.

We need to confirm that ScyllaDB doesn't achieve its twice-faster cluster growth by sacrificing the operation of regular requests during the growth period. Indeed, it does not: Both ScyllaDB and Cassandra are loaded with exactly the same number of requests per second (this is how we designed this experiment), and ScyllaDB achieves lower and more stable latency numbers: Figure 24 presents 95th percentile latency (95% of latencies are lower than this number) during the cluster growth period:

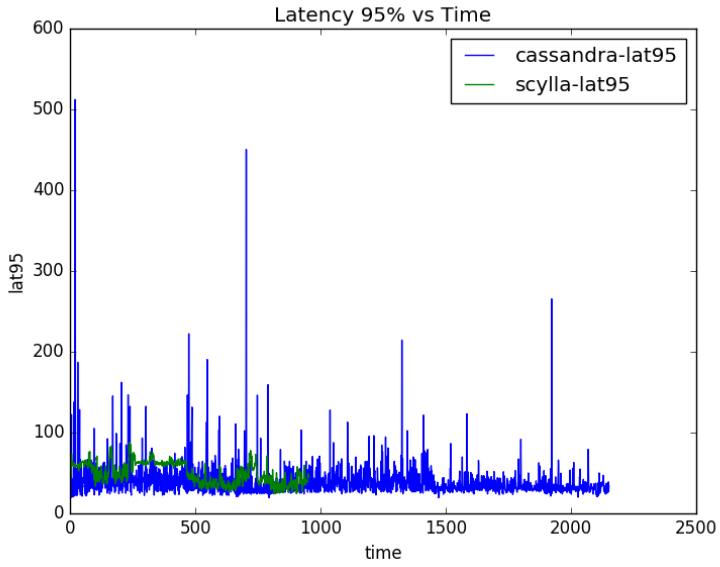

Figure 24. Latency 95% versus time

We can see that ScyllaDB latency spikes are lower, giving a better quality of service in a final cloud service. A similar plot with a 99th percentile latency is presented in Figure 25 below. Here the benefits on latency of ScyllaDB and its asynchronous Seastar base are even more evident. While Cassara's 99th percentile latency is significantly impacted by the ongoing cluster growth - reaching hundreds of milliseconds, ScyllaDB's 99th percentile latency is very stable and less than 50ms throughout the cluster's growth.

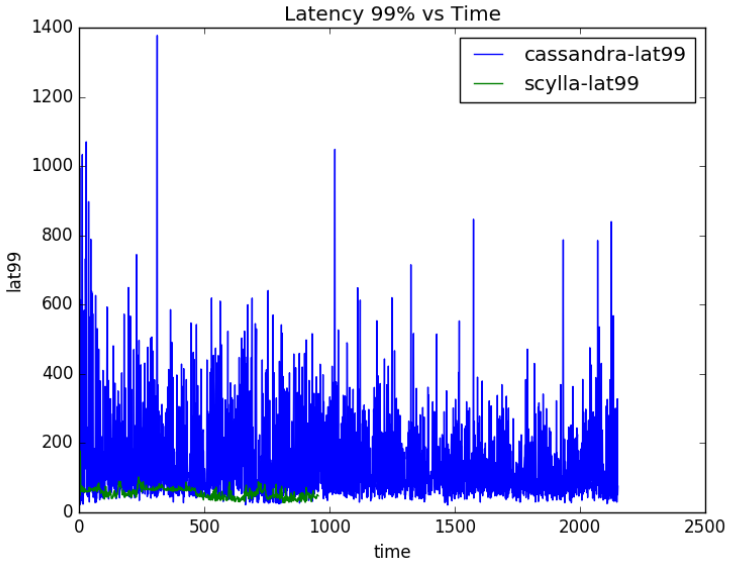

Figure 25. Latency 99% vs Time

5.4 Work Plan for Year 3

From January 2017 to the end of the project ScyllaDB will keep improving the cloud bursting performance by exploring the following leads and benchmark the database again at the end of the project:

1. Adding a network scheduler and a CPU scheduler to Seastar - Now that the disk accesses are scheduled and balanced ScyllaDB must do the same between the streaming network bandwidth and the regular request serving network bandwidth, and similarly for CPU time. This can further lower the latency of requests that happen during the long-term streaming process. Some interaction between the network scheduler and the internal IOCM algorithm can happen.
2. Adding a method for the user to control relative priority of different tasks. ScyllaDB has very few tunable but empowering the user to decide whether he wants to give priority to direct write access, table compaction, streaming or other task would be a very useful feature that the ScyllaDB support engineer could use to adapt better ScyllaDB to each particular customer workload.
3. Improving flushing of memtables and inserting data into the cache.
4. Better cache. A better cache implies fewer caches misses hence less disk access for the regular database operation and so more disk bandwidth for the streaming.
5. Better streaming. Streaming is at the heart of the cluster growth operation it look like a place where low hanging optimization fruits can be found.
6. Better and faster data serialization. The current data serialization format imposes a ten x bandwidth overhead on the streaming. So rewriting the data serialization format looks like a easy way to make huge performance improvement in the cloud bursting use case.
7. Cassandra's distributed mechanism of deciding which node owns which part of the data (using a ring of "tokens" which individual nodes own) currently requires that nodes be added to an existing cluster one node at a time. We would like to develop better algorithms which could allow to add several nodes at a time to the ScyllaDB. This could parallelize the streaming process, as well as lower the total amount of data streamed, and overall, reduce the growth time. We are considering distributed consensus algorithms (such as Paxos).
8. Better sharding algorithm ensuring data split more evenly between cores. Currently the sharding algorithm responsible to distribute the workload between the cores is suboptimal and this leads to some core to be fully used while some other are underused. Distributing the workload more evenly would liberate some CPU bandwidth for the cloud bursting use case.

These features will be evaluated, explored and developed until month 36. The potential risk is that some of these features are dead-ends and will not give us the performance improvement that we initially expected from them. ScyllaDB as a company works in a very iterative open source way and this prevents the team from working on a dead end for a long time. The team will keep on looking for improvement ideas as the project goes on since for ScyllaDB customers who need the cloud bursting feature this is the final result that is taken into account and not the cleanliness and predictability of the development process.

5.5 Conclusions

The cloud bursting use case is about growing a cluster of VMs as fast as possible in the cloud to respond to a traffic spike. A lot of work has been done at ScyllaDB in year 2 to improve this use case like streaming improvement and the I/O scheduler and its deployment in the code base. As a result ScyllaDB is already twice as fast as Cassandra on this use case and there is still plenty of room for optimization.

We will also work on a new encoding scheme and keep working on to make sure the resource usage of the different part of the database is well balanced using the network scheduler and the other leads presented above.

ScyllaDB as a company is always looking to improve the cloud bursting performance since it's a valuable and marketable feature for our customers.

6 Conclusions

The first report on the use case implementations presented the progress of the MIKELANGELO use cases in the second project year. As the use cases commenced work in project month 12, the document contains information span twelve months of work.

There are four use cases that aim to validate the project results and that provide a perspective to transition project results into active exploitation. The four use cases are: the big data use case, the aerodynamics use case, the cancellous bones use case, and the cloud bursting use case. These use cases span the most important areas of application for virtual computing infrastructures: cloud computing, big data and HPC. Cloud computing is tackled by the big data use case, by the aerodynamics use case, and by the cloud bursting use case. Even the cancellous bones use case considers to use cloud computing for scale-out. Big data is specifically handled by the big data use case. HPC is covered by the aerodynamics use case and the cancellous bones use case.

The use cases validate that the project's results in work packages 3 to 5 are relevant. The use cases' relevance presents itself from two different viewpoints. First, from an outside view, the use cases represent essential fields of computing. Thus, success in those use cases has wide implications for computing in general. Second, from MIKELANGELO's internal view, the use cases provide opportunities to leverage MIKELANGELO's computing stack. Most of the use cases use nearly all of the features from work packages 3 to 5.

All use cases have made good progress during their first year of implementation. The big data use case deployed a virtualized big data platform in the cloud test bed, it has automated the deployment of synthetic workloads, and it has defined real-world use cases for validation. The aerodynamics use case has created a dashboard, defined test cases, implemented application packaging, and integrated vTorque. The cancellous bones use case ported its simulation to OSv, it integrated vTorque, and it deployed the MIKELANGELO stack in an HPC test bed. The cloud bursting use case has developed a new IO scheduler for ScyllaDB, an IO tuning component for ScyllaDB, it improved an RPC framework, and it enhanced the efficiency of data streaming for database replication.

7 Appendix A: Scenarios Referenced by the Use Cases

7.1 Scenarios Referenced by the Big Data Use Case

ID	What	Why	Depends on
S6.BDUC.001	Deploy a spark cluster on demand	so that I can run computations on it.	R6.BD.001, R4.MPM.004
S6.BDUC.002	Run HiBench on a virtualized cluster	assess performance.	R6.BD.002-5
S6.BDUC.003	Deploy a MongoDB cluster	use it to store and access my data.	R6.BD.006-7
S6.BDUC.004	Submit jobs to the spark cluster	run an application on HPC (and retrieve the results).	R6.BD.008
S6.BDUC.005	Access results	analyze and interpret results.	R6.BD.009
S6.BDUC.006	Run an MSR benchmark experiment	assess performance.	S6.BDUC.001,3-5
S6.BDUC.007	Run an digital humanities benchmark experiment	assess performance	S6.BDUC.001,3-5
S6.BDUC.008	Run CloudSuite benchmarks to assess big data performance	assess performance.	R6.BD.010-44
S6.BDUC.009	Run synthetic and real workloads with varying parameters,	Analyze the big data performance on the MIKE stack.	S6.BDUC.008, 2
S6.BDUC.010	monitor spark with snap	review and analyze the performance.	S5.SNAP.009
S6.BDUC.011	monitor MongoDB	review and analyze the performance.	S5.SNAP.016
S6.BDUC.012	Run automated and reproducible experiments	review and analyze the performance.	S6.Scotty.01-14
S6.BDUC.013	Run YCSB benchmarks to assess big data performance	assess performance.	R6.BD.044-048

7.2 Requirements Referenced by the Big Data Use Case

ID	Functional Requirement	Required By	Developed By
R6.BD.001	Spark 2.0 runs with Sahara.	GWDG	GWDG
R6.BD.002	HiBench can be installed via Puppet	GWDG	GWDG
R6.BD.003	HiBench can be installed via Heat	GWDG	GWDG
R6.BD.004	Trigger execution of HiBench remotely	GWDG	GWDG
R6.BD.005	Collect HiBench result data	GWDG	GWDG

ID	Functional Requirement	Required By	Developed By
R6.BD.006	Deploy MongoDB with Puppet	GWDG	GWDG
R6.BD.007	Deploy MongoDB with Heat	GWDG	GWDG
R6.BD.008	Submit jobs to spark cluster	GWDG	GWDG
R6.BD.009	User can access results of computation	GWDG	GWDG
R6.BD.010	Deploy CloudSuite data caching via puppet	GWDG	GWDG
R6.BD.011	Deploy CloudSuite data caching via heat	GWDG	GWDG
R6.BD.012	Collect data from CloudSuite data caching	GWDG	GWDG
R6.BD.013	Configure CloudSuite data caching paramters	GWDG	GWDG
R6.BD.014	Run CloudSuite data caching via CI	GWDG	GWDG
R6.BD.015	Deploy CloudSuite web serving via puppet	GWDG	GWDG
R6.BD.016	Deploy CloudSuite web serving via heat	GWDG	GWDG
R6.BD.017	Collect data from CloudSuite web serving	GWDG	GWDG
R6.BD.018	Configure CloudSuite web serving paramters	GWDG	GWDG
R6.BD.019	Run CloudSuite web serving via CI	GWDG	GWDG
R6.BD.020	Deploy CloudSuite data serving via puppet	GWDG	GWDG
R6.BD.021	Deploy CloudSuite data serving via heat	GWDG	GWDG
R6.BD.022	Collect data from CloudSuite data serving	GWDG	GWDG
R6.BD.023	Configure CloudSuite data serving paramters	GWDG	GWDG
R6.BD.024	Run CloudSuite data serving via CI	GWDG	GWDG
R6.BD.025	Deploy CloudSuite data analytics via puppet	GWDG	GWDG
R6.BD.026	Deploy CloudSuite data analytics via heat	GWDG	GWDG
R6.BD.027	Collect data from CloudSuite data analytics	GWDG	GWDG
R6.BD.028	Configure CloudSuite data analytics paramters	GWDG	GWDG
R6.BD.029	Run CloudSuite data analytics via CI	GWDG	GWDG
R6.BD.030	Deploy CloudSuite graph analytics via puppet	GWDG	GWDG
R6.BD.031	Deploy CloudSuite graph analytics via heat	GWDG	GWDG
R6.BD.032	Collect data from CloudSuite graph analytics	GWDG	GWDG
R6.BD.033	Configure CloudSuite graph analytics paramters	GWDG	GWDG
R6.BD.034	Run CloudSuite graph analytics via CI	GWDG	GWDG
R6.BD.035	Deploy CloudSuite in-mem analytics via puppet	GWDG	GWDG
R6.BD.036	Deploy CloudSuite in-mem analytics via heat	GWDG	GWDG
R6.BD.037	Collect data from CloudSuite in-mem analytics	GWDG	GWDG
R6.BD.038	Configure CloudSuite in-mem analytics paramters	GWDG	GWDG
R6.BD.039	Run CloudSuite in-mem analytics via CI	GWDG	GWDG
R6.BD.040	Deploy CloudSuite media streaming via puppet	GWDG	GWDG
R6.BD.041	Deploy CloudSuite media streaming via heat	GWDG	GWDG
R6.BD.042	Collect data from CloudSuite media streaming	GWDG	GWDG
R6.BD.043	Configure CloudSuite media streaming	GWDG	GWDG
R6.BD.044	Deploy YCSB data caching via puppet	GWDG	GWDG

ID	Functional Requirement	Required By	Developed By
R6.BD.045	Deploy YCSB data caching via heat	GWDG	GWDG
R6.BD.046	Collect data from YCSB data caching	GWDG	GWDG
R6.BD.047	Configure YCSB data caching paramters	GWDG	GWDG
R6.BD.048	Run YCSB data caching via CI	GWDG	GWDG
R6.Scotty.001	Run CloudSuite media streaming via CI	GWDG	GWDG
R6.Scotty.002	Prepare pipeline	GWDG	GWDG
R6.Scotty.003	Prepare meta-job	GWDG	GWDG
R6.Scotty.004	Deploy inside VM	HUA, IBM	GWDG
R6.Scotty.005	Deploy on host	HUA, IBM	GWDG
R6.Scotty.006	Dynamic building of resources	HUA, IBM	GWDG
R6.Scotty.007	Component packaging	HUA, IBM	GWDG
R6.Scotty.008	Integrate with snap	GWDG, INTEL	GWDG
R6.Scotty.009	Deploy snap	GWDG, INTEL	GWDG
R6.Scotty.010	Integrate with mongodb for metadata	GWDG, INTEL	GWDG
R6.Scotty.011	Deploy mongodb	GWDG, INTEL	GWDG
R6.Scotty.012	Integrate with influxdb for time series	GWDG, INTEL	GWDG
R6.Scotty.013	Deploy influx	GWDG, INTEL	GWDG
R6.Scotty.014	Integrate with logstash for logs	GWDG, INTEL	GWDG
R6.Scotty.015	Deploy logstash	GWDG, INTEL	GWDG
R6.Scotty.016	Integrate with ceph.S3 for binaries	GWDG	GWDG
R6.Scotty.017	Deploy ceph.S3	GWDG	GWDG
R6.Scotty.018	Integration with HPC backend	HRLS	GWDG, USTUTT

7.3 Scenarios Referenced by the Aerodynamics Maps Use Case

ID	What	Why
S6.OFC.001	run my OpenFOAM simulations in OFC in a single process or in parallel executed by a multitude of worker processes.	obtain my simulation results in a faster and more user friendly manner compared to current bash scripts.
S6.OFC.002	choose the parameters to change and define all their possible values.	do a parametric study.
S6.OFC.003	choose the number of processes to be used for the simulation	obtain my results in a timely manner.
S6.OFC.004	choose the objective I would like to follow	obtain the dependance of the objective function with respect to the parameters chosen.
S6.OFC.005	check or save or download a collective set of results (e.g. CSV file)	do a proper, physically meaningful, CFD analysis.

ID	What	Why
S6.OFC.006	choose the amount of memory to be used for the simulation	satisfy app requirements and to obtain my results in a timely manner
S6.OFC.007	configure the mount points that are provided to the virtual guest	provide my users with shared storage mounted to a pre-defined path inside the guest (i.e. /scratch for the intermediate fast storage used during a simulation)
S6.OFC.008	provide additional (abstracted) hardware to my users, i.e. accelerator cards, usb-connector, ...	satisfy any additional (hardware-)requirements users may request
S6.OFC.009	Plot the results directly in OpenStack dashboard (or a connected application)	quickly check the results or even check the progress of convergence of simulations.
S6.OFC.010	monitor convergence of certain/chosen simulation variables (residuals and forces typically)	be certain that the simulations are converging or so that I can stop the simulation if I see that the simulations diverged.
S6.OFC.011	monitor the objective function convergence	be certin that the simulation is converging.
S6.OFC.012	continue the parametric study from one that already finished	add the behaviour of the objective function at new set of parameters and in this way improve the parametric study analysis
S6.OFC.013	quickly see the satus of each simulation (deployed, running, stopped, finished, error...)	quickly check the status of my simulations.
S6.OFC.014	save the deployed experiment (currently it is only possible to launch or cancel)	so that you can launch the simulation later and you don't loose all the settings you did so far.
S6.OFC.015	upload my OpenFOAM case to OpenStack dashboard in an easy manner (not through cyberDuck)	do the upload easy, quickly and more conveniately
S6.OFC.016	choose the container I need from the list or upload my own	be more flexible.
S6.OFC.017	deploy an arbitrary number of simulations	deploy all at the same time and not wait for the previous to finish.
S6.OFC.018	choose from a predefined list of OpenFOAM solvers	run my simulation using different app logic
S6.OFC.019	upload my own application solver as an MPM package	customise my simulation according to my needs
S6.OFC.020	request parametric study whose overall resources are higher than those appointed to my tenant	freely submit arbitrary study

ID	What	Why
S6.OFC.021	query only my own simulations	protect privacy of the results
S6.OFC.022	use the new functionalities of the Capstan tool (e.g. the remote package repository and improved composition capabilities)	simplify the deployment of the application management
SF.OFC.023	deploy OSv-based VM instances on OpenStack using UniK	eliminate direct dependency on the OpenStack
SF.OFC.024	use public cloud providers (e.g., Amazon Web Services or Google Compute Engine) to deploy simulations	get simulation results faster
SF.OFC.025	provision additional instance with preinstalled packages for OpenFOAM visualisation and configured to have access to intermediate results	visualise current status of the simulation

7.4 Scenarios Referenced by the Cancellous Bones Maps Use Case

ID	What	Why
S6.BoUC.01	Configure the hardware resources (nodes, cpus, memory, storage, etc.) needed to execute a job	Be aware of execution costs and performance
S6.BoUC.02	Transfer the input data for the jobs to the storage	The input data will be ready to run jobs in the cluster
S6.BoUC.03	Get feedback about the progress of a job submitted in the cluster	Be able to react on time when something is going wrong
S6.BoUC.04	Access the results stored in the cluster and transfer them to a local storage	Be able to make an analysis of the results and adjust the experiments parameters if needed.
S6.BoUC.05	Be able to store the data in a secured way where the input data or the produced data is protected from the access of unauthorized users.	Be able to keep sensitive or confidential data secure
S6.BoUC.06	Be able to run jobs protected against possible attacks that could read data from caches of memories when computing in the cluster	Be able to keep sensitive or confidential data secure
S6.BoUC.07	Be able to run multiple jobs in parallel in the cluster. Each job must have a unique	Be able to do an efficient parametric study

ID	What	Why
	identifier in the system and the generated data by each job must be stored properly.	
S6.BoUC.08	Be able to specify the desired level of parallelism, that is, number of processes per job	Get benefit of a parallel execution
S6.BoUC.09	Be able to run jobs of the Cancellous Bones UC in a HPC cluster	Get benefit of high performance hardware on the execution time (CPU and IO time)
S6.BoUC.10	Be able to submit jobs in the cluster using OSv-based VMs as hardware resources instead of physical nodes	Get benefit of virtualization and unikernel OS performance
S6.BoUC.11	Be able to submit jobs in the cluster using Linux-based VMs as hardware resources instead of physical nodes	Get benefit of virtualization in the use of specific libraries or software
S6.BoUC.12	Be able to get resource metrics of each job submitted in the cluster	Detect any performance problem

8 References

[1] MIKELANGELO Report D3.2, The intermediate Super KVM - Fast virtual I/O hypervisor, http://www.mikelangelo-project.eu/deliverables/deliverable-d3-2/

[2] MIKELANGELO Report D4.5, OSv - Guest Operating System – intermediate version, http://www.mikelangelo-project.eu/deliverables/deliverable-d4-5

[3] MIKELANGELO Report D5.2, Intermediate report on the Integration of sKVM and OSv with Cloud and HPC, http://www.mikelangelo-project.eu/deliverables/deliverable-d5-2/

[4] MIKELANGELO Report D2.2, Intermediate Use Cases Implementation Strategy, https://www.mikelangelo-project.eu/mikelangelo-wp2-2-ustutt-v2-0/

[5] MIKELANGELO Report D2.10, The first aerodynamic Map Use Case Implementation Strategy, http://www.mikelangelo-project.eu/deliverables/deliverable-d2-10/

[6] MIKELANGELO Report D7.3, The third plan and report for exploitation of MIKELANGELO project, http://www.mikelangelo-project.eu/deliverables/deliverable-d7-3/

[7] Virtual big data use case parametes and data http://opendata.mikelangelo-project.eu/public.php?service=files&t=d8af61ea0130ba14d44bb8fa015b26bc

[8] Mongo Puppet scripts, https://github.com/puppetlabs/puppetlabs-mongodb

[9] HiBench homepage, https://github.com/intel-hadoop/HiBench

[10] CloudSuite homepage, http://cloudsuite.ch/

[11] Yahoo! Cloud Serving Benchmark, https://github.com/brianfrankcooper/YCSB

[12] SmartSHARK, the Smart Software History Analysis and Retrieval of Knowledge platform, http://smartshark.informatik.uni-goettingen.de/

[13] Addressing Problems with External Validity of Repository Mining Studies Through a Smart Data Platform, Fabian Trautsch, Steffen Herbold, Philip Makedonski, Jens Grabowski, 13th International Conference on Mining Software Repositories (MSR), ACM, 2016

[14] Europeana digital library, http://europeana.eu

[15] Europeana data quality committe, http://pro.europeana.eu/page/data-quality-committe

[16] Metadata Quality Assurance Framework, http://pkiraly.github.io

[17] Metadata Quality Assurance Framework data set, http://hdl.handle.net/21.11101/0000-0001-781F-7

[18] Spark homepage, https://spark.apache.org/

[19] Cloudera CDH homepage, https://www.cloudera.com/products/apache-hadoop/key-cdh-components.html

[20] Apache YARN homepage, https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.htm

[21] HDFS homepage, http://hortonworks.com/apache/hdfs/

[22] Map Reduce tutorial, https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[23] HiBench results data, http://opendata.mikelangelo-project.eu/public.php?service=files&t=f6e52b21cd8d3da98af55dd2c73f5912

[24] OSv big data experiments data, http://opendata.mikelangelo-project.eu/public.php?service=files&t=cddadf54bfd392c4337c88c3e0ff2101

[25] OpenFOAM homepage, http://openfoam.com/

[26] UberCloud homepage, https://www.theubercloud.com/

[27] SimScale homepage, https://www.simscale.com/

[28] MIKELANGELO Report D6.1, First report on the Architecture and Implementation Evaluation, https://www.mikelangelo-project.eu/wp-content/uploads/2016/07/MIKELANGELO-WP6.1-INTEL-v1.0.pdf

[29] OpenStack Orchestration Engine, Heat, https://wiki.openstack.org/wiki/Heat

[30] Ubuntu Trusty (14.04) cloud image, https://cloud-images.ubuntu.com/trusty/current/trusty-server-cloudimg-amd64-disk1.img

[31] OpenFOAM installation procedure for Ubuntu 14.04, https://openfoamwiki.net/index.php/Installation/Linux/OpenFOAM-2.4.0/Ubuntu

[32] MIKELANGELO package repository, http://mikelangelo-capstan.s3.amazonaws.com

[33] MIKELANGELO Report D2.1, First Cancellous bone simulation Use Case Implementation Strategy, http://www.mikelangelo-project.eu/deliverables/deliverable-d2-1/

[34] MPI Forum homepage, http://mpi-forum.org/

[35] Intel VT-x, http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html

[36] AMD-V, http://www.amd.com/en-us/solutions/servers/virtualization

[37] MIKELANGELO Report D5.4, First report on the Integration of sKVM and OSv with HPC, https://www.google.com/url?q=https://www.mikelangelo-project.eu/wp-content/uploads/2016/06/MIKELANGELO-WP5.4-USTUTT-v1.0.pdf&sa=D&ust=1482346075427000&usg=AFQjCNH9rCJol8O6vSBC5ZE_uoXxYFdXw

[38] Node metadata collector tool, https://github.com/mikelangelo-project/node-metadata-collector

[39] Apache Cassandra homepage, http://cassandra.apache.org

[40] CAP Theorem, https://en.wikipedia.org/wiki/CAP_theorem

[41] Seastar framework homepage, https://github.com/scylladb/seastar

[42] ScyllaDB homepage, http://www.scylladb.com

[43] Hosted ScyllaDB at IBM Compose https://www.compose.com/scylladb

[44] Datastax homepage, http://www.datastax.com