MIKELANGELO

D2.7

The First Virtualised Big Data Use Case Implementation Strategy

Workpackage:	2	Use Case & Architecture Analysis
Author(s):	Peter Chronz	GWDG
	Christopher Menke	GWDG
Reviewer	Michael Gienger	USTUTT
Reviewer	Daniel Vladušič	XLAB
Dissemination Level	Public	

Date	Author	Comments	Version	Status
2015-08-20	Peter Chronz	Initial draft	V0.1	Draft
2015-08-20	Christopher Menke	Added data from experimental results.	V0.2	Draft
2015-08-22	Peter Chronz	Revision of the whole document.	V0.3	Revised Draft
2015-08-24	Peter Chronz	Document ready for review	V0.4	Review
2015-08-31	Peter Chronz	Document ready for submission	V1.0	Final

Executive Summary

This report describes the initial implementation strategy for the virtualised big data use case of the MIKELANGELO project. This use case will showcase the advancements achieved by MIKELANGELO in the field of big data.

The goal of this use case is to leverage MIKELANGELO's advancements in I/O efficiency of virtual machines and application management to run big data applications on virtual infrastructures. Besides integrating components from other work-packages in MIKELANGELO, this use case will extend existing middleware to run big data clusters on demand in a cloud. At the end of the project the goal is to reach a state that will allow resource providers to deploy an on-demand big data service on their cloud.

This report describes the use case itself along its two main stages. These stages are comprised of benchmarking and of running mini-use cases. The description of the use case consists of a description of hardware and software infrastructure, a description of data sets, a description of mandatory requirements, and a description of how the use case relates to the project's key performance indicators. Furthermore, we provide an implementation plan for the benchmarking stage and the mini-use case stage. For the benchmarking stage this report offers some more detailed descriptions as they will serve as the project's initial baseline measurements.

The work already performed for the big data use case has led to a number of tangible results. First, we present in this document an overview of the state of the art with regards to big data benchmarks. These form a basis for our first choice, HiBench, as big data benchmark. Second, we have run HiBench on a test bed to establish a baseline to be used for evaluation of MIKELANGELO's new developments. The data generated in these experiments can be reviewed in MIKELANGELO's open data repository. Third, we have formed an implementation plan for the remainder of the project for the big data use case. This implementation plan relies on a requirements analysis and on the analysis of the project's KPIs. The implementation plan specifies the required features for the benchmarking part and for the mini use-cases.

This report provides the most detailed description of the virtualised big data use case so far. This description recaps the current state of the implementation and an implementation plan to be executed during the next project stages. Conclusively, the use case has received most work on the conceptual side and progresses smoothly into the implementation phases.

Table of Contents

List of Figures

List of Tables

Abbreviations

HPC	High Performance Computing
I/O	Input-Output
KPI	Key Performance Indicator
KVM	Kernel-based Virtual Machine
RDMA	Remote Direct Memory Access
sKVM	super-KVM

1 Introduction

This deliverable describes the implementation strategy for the virtualised big data use case at month 8 of the project. The goal of this use case is to implement a management system for big data clusters that operate in a cloud. Specifically, end-users shall be capable to set up a big data cluster on the cloud by themselves with convenient high-level user interfaces. The use case is of importance to the whole project since it verifies and showcases the benefits of the MIKELANGELO stack for input-output-heavy (I/O) applications.

The problem statement of this use case is to allow end-users to run a big data cluster in a cloud. In our case OpenStack [24] serves as cloud middleware. Thus the integration of the big data cluster management will be carried out in conjunction with OpenStack. Currently, there already are initiatives to integrate big data applications with OpenStack. The most prolific one is the Sahara [25] project, which is part of OpenStack. OpenStack Sahara is in the early stages of development and aims to deploy and manage Hadoop and Spark clusters on top of OpenStack. In this use case we want to provide functionality that extends that of Sahara.

One of the major hurdles to use big data in a cloud environment comes from the high attrition of resources due to the low I/O efficiency of virtual machines. MIKELANGELO's biggest promise is to improve this I/O efficiency. In this use case, we are going to leverage the improved I/O efficiency to run big data applications in the cloud. Another major issue when rolling out a big data cluster on-demand in a cloud is the time to set up the cluster. The setup time consists mostly of transferring base images to target nodes, setting up virtual machines (VM), booting VMs, installing required software, and loading custom configurations. In MIKELANGELO this processes will be sped up significantly by leveraging OSv's small footprint and its packing mechanisms. A third potential issue for big data in the cloud are security concerns. Security researchers have shown repeatedly that VM isolation can be overcome by various means, such as side-channel attacks. In the case of big data, confidentiality plays an important role in many real-world scenarios. For some use cases a high level of data confidentiality is a mandatory requirement.

We approach these issues by making use of MIKELANGELO's improved stack for virtualized infrastructures. In specific, this use case builds on the integration of MIKELANGELO's lower level components with OpenStack as a cloud management middleware. This use case will evaluate MIKELANGELO's progress in the areas I/O efficiency, platform agility, and security. All major components of MIKELANGELO's stack will be used and tested in this use case. We expect increased I/O efficiency from super KVM (sKVM), OSv, and the use of remote direct memory access (RDMA) over Ethernet. We expect improved agility when setting up a cluster from OSv's low footprint and application packaging. Additionally the cloud management system will help to manage the big data cluster. We expect to see improved security from the security additions throughout the whole MIKELANGELO stack. The integrated telemetry system will provide the data necessary for advanced administration and to verify the improvements in all three areas.

From a management perspective this use case follows goals beyond the technical goals already stated. The use case aims to verify MIKELANGELO's work in specific to big data applications, which are growing in importance. This verification will happen using benchmarks that contain benchmarks and real-world implementations. In addition to running these benchmarks on the whole stack, the big data use case will provide input to the validation work-package as a whole. We will integrate the benchmarks with a continuous integration system so that developers can re-run those benchmarks as part of integration tests. From the real-world implementations in this use case, we will extract reference cases, which again will be integrated with continuous integration. Thus this use case will provide synthetic and real-world tests for developers to verify their implementations and to guide their development. Furthermore, we will strive to provide convenient high-level interfaces on top of an OpenStack cloud to deploy and manage big data clusters. These interfaces will be geared towards end-users who want to use the cluster. In this context users will be able to customize the cluster according to their needs. For GWDG, as the use case's stakeholder, the goal is to get actual customers on board and to bootstrap a big data service as part of our portfolio. We will leverage the experiences in this use case to create a real service that is going to last beyond the project's lifetime.

This report covers a broad area of topics from management to deep technical issues. For sake of brevity this deliverable will focus on the motivations, the approach and the goals of this use case. In addition, we will provide basic technical information that is not covered in other reports. This deliverable informs about big data benchmarks, mini use-cases, GWDG's infrastructure setup, the relation to GWDG's business model, and it describes the use case's implementation and validation plans. As part of the validation plan you can already find measurements from a benchmark, which will serve as a baseline to improve upon.

The use case is of special importance to GWDG and to the big data community as a whole. To GWDG the use case is important since we want to offer big data as a service. We have a chicken-and-egg problem that we will not be able to afford a large infrastructure for big data without customers. Without customers, however, we will not be able to economically run a large big data cluster. Thus, we would like to bootstrap this new area of involvement based on our cloud infrastructure. However, current technology poses significant hurdles to such a use case in the areas I/O efficiency, elasticity, infrastructure management, and security. For the big data communities at large there are benefits for providers and customers. Our work will empower providers to offer big data clusters on demand to their customers based on existing cloud infrastructure. Simple management on top of OpenStack will lead to a proliferation of big data offerings by small and mid-sized data centres. In turn wide adoption of on-demand big data offerings will give end-users the choice to use an offering of their choice with best quality of service, security, and price.

The report is limited in its technical depth, in the evaluation of the business context, and in the description of mini use-cases. The technical descriptions are kept at a high level in the initial iteration of this deliverable. Only peculiarities important to the project as a whole will be mentioned, described, and evaluated in more detail. All technical details left out of this report can be found in the respective technical reports and in the technical documentation. Business-wise we state the relation of this use case to our business model. In specific we relate the project's key performance indicators to this use case and to GWDG's business model. However, due to the early stage of the project we do not provide any economic calculations. To carry out reliable calculations, we will first need to make more technical progress. Similarly, the progress on real-world implementations is at an early stage. Currently, we are in contact with interested parties and early discussions are taking place. However, no concrete technical work has been carried out yet.

The rest of the document is structured as follows. Section 1 provides a more detailed definition of our use case. Section 2 describes the setup of this use case in terms of hardware and software infrastructure. Section 3 provides a requirements analysis. Section 4 provides the implementation plan. Section 5 describes the evaluation and validation plan. Section 6 concludes this document by summarising the most important insights.

2 Definition of the Use Case

This section describes the motivation, the approach, and the implementation strategy for the use case. The motivation covers business aspects and technical aspects. The technical aspects align themselves nicely with the goals of the overall project. These, in turn are leveraged to satisfy our business needs. The approach hinges on leveraging the technical merits of MIKELANGELO and building additional integration with big data systems on top. The implementation strategy describes how we intend to integrate the MIKELANGELO stack into a system that can be used readily for on-demand big data deployments. Thus the strategy includes the description of the benchmarking stage and the mini use-case stage.

2.1 Motivation

The motivation for the virtualised big data use case can be subdivided into business motivation and technical motivation. From a business perspective GWDG would like to increase the utilization of its cloud infrastructure, satisfy customer requirements for big data processing, and establish a lasting big data service on top of its cloud. From a technical perspective GWDG would like to improve the management of big data clusters on clouds, deal with diverse customer requirements flexibly, offer a high degree of security, and make big data workable on top of cloud computing in a reasonable way. The technical side of the motivation stems from a list of limitations of current cloud infrastructure, which we discuss at the end of this section.

GWDG is the IT provider for the University of Göttingen and for the Max Planck Society. In addition, we serve customers in the academic environment in and around Göttingen. One of GWDG’s more recent offerings is its cloud computing service. With this service customers can create virtual machines and perform basic configuration such as assigning IP addresses and configuring firewall rules. For GWDG one concern is to increase the utilization of physical machines, since idle resources consume power unnecessarily and stale IT infrastructure always means a poor use of investment. However, services installed in a cloud fluctuate significantly in use. On the other hand, many of our customers signalled the need for a similar service that would offer big data services. These customers would ideally want to run their big data applications based on our cloud offering, which already offers identity management and accounting. Many of our customers however either lack the expertise or the resources to set up a big data cluster by themselves. What complicates matters even more is that our customers have diverse scientific backgrounds, which in turn translates to diverse technical requirements for our big data platform. Finally, from a business perspective we want to establish a successful business model for a big data service and keep it going as a lasting service.

From a technical point of view there are a couple of hurdles to overcome when setting up an on-demand, configurable big data service. The first hurdle is the management of big data clusters in a cloud. Here OpenStack Sahara provides a good start, but more elaborate features are required to allow for an end-user driven deployment of big data clusters. Furthermore, since our customers have diverse requirements for big data clusters, we need to allow for a high degree of flexibility. This in turn leads us to the requirement for end-users to configure their system by themselves, even without deep expertise in system administration. In addition, we expect that our customers will require a high degree of security. Customers may require that their big data computations will run on isolated hosts. Furthermore, we will need to have an edge with regards to cloud security to establish trust with customers, especially with regards to sensitive data. The requirements have been reported in WP2. Finally, the last technical hurdle is one of efficiency. Although currently it is possible to run a big data cluster in a cloud, such as ours, the deployment will be inefficient. There are multiple reasons for this inefficiency, which in the end boil down to low I/O efficiency and poor cloud agility. These issues lie at the core of MIKELANGELO's motivation. I/O efficiency of VMs, the operating system, and for collocated VMs is not as high as it could be. Another technical hurdle in the cloud arises with the deployment of a fresh big data cluster. Setting up a fresh big data cluster is akin to cloud bursting. A lot of VMs require VM images on multiple hosts and then they need to be brought up as fast as possible. Cloud bursting is a poorly solved problem in cloud computing, which leads us to believe that it will pose significant limits on our use case as well.

We can elaborate on these technical hurdles by listing the concrete technical limitations of current clouds and virtual infrastructures in general. There are multiple areas of limitations that we want to overcome. We want to leverage the gained benefits obtained by WP3-5, which mostly translate into gains of I/O efficiency, faster bursting, better application management, and improved security. On top of that we want to augment big data platforms with the benefits of cloud computing.

I/O efficiency is a major hurdle to run big data applications in cloud computing. Cloud computing typically builds on full virtualization. This in turn features low I/O performance. At the same time, big data is I/O-heavy. Efficient access to memory, local storage, and remote storage via the network are important for a smoothly running big data application. Furthermore, there are no proper mechanisms to leverage colocation of virtual big data nodes in cloud computing. In the project we are developing RDMA which can lead to significant speed up of communication of collocated VMs.

Fast bursting of services is another severe limitation of current cloud systems. The limits to how many VMs can be started up gracefully is reached very quickly. This limitation is due to transfer times of VM images, creation of VMs, and boot-times of VMs. Depending on the application/service deployment method some parts of the service might even have to be installed after the machine is booted. This installation process will again require network and computational resources.

Application management in cloud computing currently is limited by rigid deployment models. One has to use either predefined images or separate tools to configure applications. Then again there are orchestration tools such as OpenStack Heat which can take care of orchestration. This use case is limited by the current state of the art in that there is no integrated system which deploys VMs, applications in those VMs, and then integrates them through an orchestration layer.

Security is limited by the state of current hypervisor technologies and lack of more elaborate security concepts in cloud management systems. Recent publications have shown that side-channel attacks on kernel-based virtual machines (KVM) are performed with relative ease. Additionally there are few security mechanisms integrated in the cloud management systems. It is not straightforward to achieve isolation of services on physical machines. For example the OpenStack scheduler only schedules VMs once and it does not reschedule them later on.

Beyond the mentioned limitations, this use case is limited currently because big data platforms do not leverage the benefits of cloud computing. Lacking features according to the NIST definition [26] are on-demand self-service, resource pooling and multi-tenancy, rapid elasticity, and metering.

On-demand self-service currently is not possible with big data platforms. Such platforms have to be set up on dedicated hardware with a rigid configuration of components. Customers of big data services need to adapt their applications to that environment, even if a different set up would simplify the implementation and improve runtime performance. Features such as live migration, snapshotting, and explicit migration of data between hosts to improve performance or security is not possible currently.

Resource pooling and multi-tenancy are limited since most big data platforms assume that their deployments will be used mostly in-house. Since big data platforms are deployed on physical hosts directly, resource pooling becomes a problem, especially when users have varying requirements to the platform. Isolation of data and processing is hard to achieve. Thus security for multi-tenancy scenarios is not sufficiently given. Without multi-tenancy and resource pooling there is a waste of computational resources.

Rapid elasticity is not given in current big data platforms at all. For example YARN doesn't support dynamic scaling, that's why Spark doesn't support dynamic either. Static deployments need to be over-provisioned to satisfy customer needs according to agreed-upon terms. When customers do not run sufficient jobs the physical hardware will be idle, incapable to serve requests for other services such as cloud or high performance computing (HPC) services. There is no way to scale big data platforms up or down. However, elasticity is an important feature for some big data use cases. There are use cases that rely on bursting a big data cluster when new data comes in. Such use cases include large experiments that collect sensor data.

Service metering is not a major feature in big data platforms since the assumption is that big data processing happens in-house. A pay-per-use model is not at the heart of the current generation of big data platforms. Metering only really makes sense once big data platforms can be use on-demand by users. Because of varying user requirements on-demand only makes sense in a virtualized cloud environment.

This section has listed a number of limitations of the state of the art that pose hurdles on the way to virtualized big data services. Some of those limitations will be resolved in other work-packages in MIKELANGELO. These limitations have an impact on various fields of applications such as HPC and cloud computing in general. In this use case, we will leverage those solutions by integrating them in a flexible big data platform. The remaining limitations are specific to the field of big data. These issues will be tackled head-on in this use case. Still, the field of big data is large and it will grow in size and importance over the next years. With a proliferation of big data, having a flexible big data system will become inevitable. Thus this use case holds the potential for large-scale impact on a large scale.

Many small and mid-sized data centres share our requirements and need to cope with the listed limitations. Thus, we are confident that the work in this use case will have a significant impact on many service providers. By extension an increased adoption of big data on-demand services will lead to a proliferation of big data offerings, which in turn will be to the benefit of big data customers.

2.2 Approach

The technical approach in the big data use case makes use of increased efficiency of virtualized I/O, increased agility of application deployment, increased application compatibility, improved application management, increased security of the hypervisor, and the cloud integration. The approach in this use case is to leverage the technical results of work-packages 3-5 as well as possible. Thus, in this section we also mention the expectations we have towards the deliverables from work-packages 3-5. The technical approach targets the limitations in the previous section. Thus, this section parallels the structure of the previous section.

Increased efficiency of virtualized I/O in work packages 3 and 4 is required to make best use of physical infrastructure, hosts in specific. Since big data applications are I/O-heavy and virtualized I/O is poor in efficiency they are a bad match; especially when considering the importance of big data and the benefits we could have from combining it with cloud computing. We will gain improved I/O performance from optimizations in sKVM, integration with OSv, and use of RDMA, which may include smart colocation of VMs that communicate a lot with each other. Furthermore, we will explore whether we can make use of OSv-specific big data technologies to gain even bigger benefits than possible with current-generation technology without OSv. A database for big data, currently in development by CLOUDIUS, is the prime candidate to improve I/O efficiency. Current experiments show a speedup of 6 in comparison to Cassandra. The different types of I/O operations will be tested via different benchmarks and real-world implementations. For this we will identify suited tests for I/O to disk, I/O on in-memory systems, network-heavy applications, and for services that should benefit from colocation. There will also be tests that combine those separate tests by mixing workloads.

Since big data applications in general are I/O-heavy, often even I/O-bound, we expect these gains in efficiency to translate to shorter run times of big data workloads with constant size. We expect to gain further improvements from leveraging RDMA for inter-VM communication in combination with colocation. Furthermore, for interactive big data applications, we expect reduced query times. This should translate to enabling interactive real-time queries, which would otherwise not be possible in a virtual environment.

Increased agility of application deployment translates in our use case to being able to handle service bursts gracefully and to support a wide variety of big data services via increased application compatibility and application packaging. The expectation here is to get clusters running on a virtual infrastructure quicker than possible nowadays. This translates into creating larger clusters on-demand without slowing down the whole cloud beyond a certain threshold or into creating smaller clusters faster than possible with current systems. We expect to achieve these features by making use of OSv's small footprint. This should translate into fast transfer times of VM images, fast creation of VMs, and very fast boot times. In addition, we expect that optimised application management can improve bursting as well. Instead of creating many large, predefined VM images, one can use small base images. Then needed services can be distributed to be installed in running VMs onto the nodes for fast installation. On the target node one can create the customised base image in a VM. Then clone it as often as needed on that host. This will relieve strain on network and on the CPU on the target node and is expected to improve bursting capabilities.

Increased application compatibility refers to the integration of big data technology with cloud management and to running new software, relevant to the big data use case in OSv. The integration of big data with cloud management, most likely will be incarnated by using OpenStack Sahara on top of OpenStack with some additions. The other big point is running big data software in OSv as guest OS. Currently it seems that there are some limitations that prevent Hadoop from being run in OSv including all of its components. However, it may still be possible to run a subset of its components, such as YARN, HDFS, and HBase in OSv. Big data implementations can also run with other middleware, databases and processing services than offered by Hadoop. Among those are popular applications such as Mesos, Zookeeper, Solr, Elasticsearch, Spark, and Cassandra. Some of those components are already known to run in OSv [34]. In this use case we will provide feedback to the work-package on OSv so that additional development can take place to run more of those software solutions in OSv or to increase the performance of compatible components. We provided all the already-known requirements to OSv.

Improved application management is expected to improve the accessibility of big data to end-users. We expect that providing high-level interfaces will reach more potential users of big data. It should be possible to create a big data cluster through a web interface. Once deployed, users should also be made aware of their deployment via the integrated monitoring framework. Thus the performance and bottlenecks in performance should become obvious in the web interface already. The application management should also include the capability to customise big data systems quite easily. The high-level setup of a cluster will be based on using OpenStack Sahara with additions to narrow down or even close the gap between the current abstraction and what users really require. Initially, the system will likely provide a couple of pre-defined big data clusters that will be made available via special images and additional configuration files. The set of predefined clusters shall be researched properly to cover a wide range of real-world use cases. In later project stages we shall investigate whether an extended level of flexibility is required by users.

Increased security of the hypervisor is an important aspect to real-world implementations that process sensitive data, such as personal data. These security issues are of specific importance in our use case since we aim to run big data applications in a public cloud. We thus need to deal with multi-tenant setups. Here it needs to be possible for use cases with sensitive data to isolate their virtual big data nodes on physical hosts. Such a feature can be achieved by migrating VMs using live-migration and OpenStack's aggregate mechanism. Recent publications [27, 28, 29] have shown that current versions of KVM, which is the hypervisor technology that we extend, is prone to side-channel attacks. To provide a trustworthy service to our customers, such issues need to be alleviated as far as possible. Improved security is expected to gain increased trust from customers and to prevent side-channel attacks. In short, we expect to convince our customers that our platforms offer improved isolation in comparison to other state-of-the-art systems. We expect the security to be high enough to attract customers to run big data applications on our cloud even with sensitive data. For mini use-cases with sensitive data we expect to be able to run workloads on dedicated physical hosts that prevent access by third parties during execution of the big data workload.

Cloud integration of MIKELANGELO will bring benefits that revolve around on-demand self-service, resource pooling and multi-tenancy, rapid elasticity, and metering. Those benefits will be reached by building on the cloud integration and by adding a big-data integration into the system.

On-demand self-service is one of the biggest motivators for this use case. Providing a customised big data platform demand is a way to attract more customers to us as an infrastructure and service provider. We expect that in this use case we will be able to provide configurable big data clusters that can be set up within minutes by end-users. This means that users should be able to configure, deploy, manage, and monitor their systems all by themselves. In the background, features such as live-migration, snapshotting, and VM migration should be leveraged transparently to provide a good quality of service.

Resource pooling and multi-tenancy raise the expectation for us as a service provider to improve the utilization of our systems. For a service provider an increased use of the infrastructure, while maintaining a satisfactory quality of service, is always an important goal. We expect that the integration of big data with cloud computing will allow us to combine workloads from both systems. There should be a distinction between time-critical and loosely constrained jobs. This distinction would allow to mix both workloads in such a way that the infrastructure will be kept as busy as possible at all times. Such a distinction, of course needs to be accounted for and reflected in pricing.

We expect to reach ***rapid elasticity*** by leveraging MIKELANGELO's improved bursting capabilities. These capabilities need to be combined with scaling mechanisms for big data platform, which in turn need to be integrated with the cloud management system. Rapid elasticity thus should be available to set up new big data clusters as well to scale in and scale out horizontally. This flexible scaling should lead to reduced costs for customers and a real pay-per-use model for them. For the service provider it means that more meaningful computation is performed on their systems. In addition more customers can be served on the same infrastructure by time-sharing efficiently.

Metering is expected to allow an optimization of the infrastructure, big data clusters, and accounting. For the service provider, detailed, flexible, and dynamic metering allows for improved optimisation of the infrastructure. Here, it is important that the most important metrics can be explored quickly by administrators. Furthermore, it is important to dynamically adjust the resolution of metering as required to handle the large data size of monitoring systems. For end-users metering should be available and allow them to analyse the performance of their virtual cluster. Metering and exploration of the metered data should allow them to identify bottlenecks in their deployment quickly and easily. Finally, metering should provide a basis for accounting of big data clusters. Dynamically adjustable metering may allow us to charge customers in a more fine-grained way, which will reduce their investment.

In addition to the technical approach and our expectations, there are two major expectations from this use case from a business perspective. For one GWDG wants to offer big data as a service on its cloud infrastructure. We expect this service to be taken up by our customer base very quickly. Gaining old customers as new customers for the new service translates for us to more use of our infrastructure. When our infrastructure is used more for more services in a meaningful way, we can invest more into cloud infrastructure and proliferate our big data services. In addition, we want to strengthen our expertise in developing big data and cloud systems. Thus, within this use case we would like to establish ourselves as a contributor in the field of big data. The developments from this use case will be made publicly available so that other resources providers can make use of them as well.

With this described approach, we expect to leverage most of the features of the MIKELANGELO stack. These include improved I/O efficiency, improved application agility, improved application management, improved security, and cloud integration. Besides that we expect to reach a productive state of the use case by the end of the project.

2.3 Implementation Strategy

The use case implementation has two major parts, which translate to two major, albeit overlapping, phases of the use case implementation. The first part consists of benchmarks. The second part will consist of mini use-cases. In the initial project phase we will focus on setting up benchmarks specific to big data applications. In the second project phase we will focus on getting additional stakeholders on board and on providing real-world implementations and real users to our system. Both phases will overlap in part, since we have started to get in contact with additional stakeholders already, while the benchmarking phase has not yet concluded.

The implementation strategy for the benchmarking phase consists broadly of two sub-phases. In the first phase of the use case implementation benchmarks are researched and set up manually on a couple of hosts. The initial details of our technical setup (Section 3), the relevant benchmarks (Section 3.2.1), and the baseline results (Section 6.2) can be found in this document. In the second phase of the benchmarking phase we will re-run the benchmarks regularly on a larger infrastructure. Furthermore, we will, of course, extend the benchmarking to evaluate not only a baseline measurement, but also MIKELANGELO's stack for comparison. Here, we will target all features that are new to MIKELANGELO individually and in an integrated fashion. These features and components will include sKVM, OSv, RDMA, improved security, and monitoring. Once this setup is ready, we will integrate the benchmarks with the project's automated continuous integration system. This integration will allow developers to run test cases tailored to evaluate their components in isolation and in an integrated environment on-demand. To this end, parts of benchmarks will be run as separate test cases to avoid running complete long-running benchmarks.

In the second phase of the use case implementation we will tackle real-world problems in a set of mini use-cases. These mini use-cases will come from GWDG's network, which chiefly comprises the Max-Planck-Society, the University of Göttingen, and the SUB. In addition there is a long list of related institutes and organizations which also use GWDG as an infrastructure and service provider. The mini use-cases will first need to be recruited. The recruitment consists of eliciting potential mini-use cases, assessing their suitability to showcase MIKELANGELO's benefits, and the required effort to implement the mini use-cases. We hope to bring as many mini use-cases on-board our big data service during the project as possible. Since many of our related institutes do not possess deep technical skills, we will most likely help them to provide implementations on top of our big data platform. Initially, the mini use-cases will be run on a static cluster and the performance will be evaluated on a case-by-case basis. Later on, the integration with OpenStack will allow users to deploy a cluster by themselves and run their computations as they need. From the mini use-cases we will extract reference cases, which will focus on specific features of MIKELANGELO's stack. These reference cases will then be integrated with the continuous integration system, similar to the benchmarks, such that MIKELANGELO's developers will be able to run the reference cases to test their code. We aim to progress with the use case so far that by the end of the project, that we will be able to transition the mini use-cases and serve them as new customers of our virtualized big data platform.

3 Technical Setup

The virtualised big data use case consists of two major logical parts, a benchmarking part and a part with real-world implementations of mini-use cases. These two parts will be developed in two project phases. The benchmarking phase will help us to set up the infrastructure to gain experience with several important big data technologies and common workloads. The benchmarks will further serve as integration tests to guide development during later project stages. The real-world use cases will be developed on top of those experiences. In the early phases of the real-world implementation, we will focus on just one or two mini-use cases to gain experience. Later on we will automate as much as possible of the process to get more mini use-cases on board.

This distinction into two parts and two main stages of the project allows the use case to grow organically making use of previous experience. This distinction is reflected in the following sub-sections and throughout the whole document in the descriptions of the use case setup. Thus the initial benchmarking phase can be seen as a learning phase to assess the compatibility of MIKELANGELO with big data technologies and workloads. The second phase with mini use-cases can then be seen as extension of these experiences to real-world applications.

In the following sub-sections we present for both stages the physical and virtual infrastructure, software setup, the used and generated data, and security issues for the use case. These descriptions are based on the current state of the use case. Future issues are projected based on the current state. In specific, any information to the real-world implementations is based on expectations, estimates, and projections.

3.1 Physical and Virtual Infrastructure

This section summarises the physical and virtual infrastructure used in this use case in year 1. Furthermore, we describe our expectations for future deployments of the use case.

3.1.1 Physical Infrastructure

The extent of the physical hardware at GWDG dedicated to MIKELANGELO will grow during the project's runtime. Initially, for experiments there will be four physical machines. One of those machines will be a control node with relatively low specifications. The other three machines are properly dimensioned data centre servers. The specification of those three machines is provided in Table 1. These machines will serve to establish baseline measurements for the use cases. Furthermore, these machines will initially be used to deploy the MIKELANGELO stack for verification. In year 2 this set of machines will be extended by additional memory to host more virtual machines and to handle larger big data workloads. Furthermore, we will extend the setup of machines that are dedicated to MIKELANGELO in later project stages to provide more dedicated hosts. Even more important, GWDG will include, whenever possible, physical hosts from its cloud to the testbed. During times of low usage physical hosts of GWDG's cloud can be added to the testbed to run larger experiments. Currently, the cloud system runs on over three dozen machines.

Component	**Specification**
CPU	1 Intel Xeon E5-2630 v3 processor (2,4GHz, 8C/16T, 20MB Cache, 8GT/s

	QPI, 85W, Turbo, HT)
Memory	1.866MHz, 8 16GB RDIMM, 2.133MT/s, Dual Rank, x4 data bandwidth
Storage	2x600GB SAS 6Gbit/s 10K 2,5" HDDs, 13G, PERC H330 Integrated RAID Controller (RAID 1)
Network	Intel X520 2 Ports 10GbE DA/SFP+ Server Adapter, Low Profile

Table 1: Initial hardware setup for the testbed

3.1.2 Virtual Infrastructure

In the initial stage the goal is to provide early baseline measurements using KVM [30] as hypervisor and Ubuntu 14.04 LTS [31] as both host OS and guest OS. This initial setup will run benchmarks once on the above mentioned three hosts directly without a hypervisor. These experiments will provide the initial host-based baseline for the big data benchmark. These results should provide an upper boundary for the performance achievable on those machines. The other important part of the baseline measurements is the performance of a typical virtual setup. In our case we choose KVM as hypervisor and Ubuntu 14.04 LTS as guest OS. The choice for KVM is motivated by its dominance in OpenStack-based clouds. Furthermore, the direct comparison of sKVM and KVM will show very clearly the impact of or work on sKVM, since sKVM is a fork of KVM. We choose Ubuntu 14.04 LTS as it is the most recent long-term support version of Ubuntu. Ubuntu itself is a common and popular choice as guest OS. Its stripped-down server-version is popular for cloud computing since it provides a relatively small image and many common tools, which make the installation of software in it simple. Furthermore, Ubuntu is well tested with big data software. Our specific software setup for the initial benchmarks is detailed in Table 2.

Component	Version
Ubuntu Host (+ kernel)	3.16.0-30-generic
Ubuntu Guest (+ kernel)	3.16.0-30-generic
libvirt	1.2.2
KVM	1.2.0

Table 2: Software specifications for the initial baseline measurements.

In the automated testing stage we will leverage the benchmarks to cover more technologies and to provide targeted performance tests for single components. We will cover more technologies with the tests by not only running them with full virtualisation and directly on the hosts but also with container-based virtualisation. Currently, container-based virtualisation goes through a hype. One of the reasons for this hype stems from the low overhead of container-based virtualisation. Containers read near-native performance while still offering a certain level of virtualisation. Furthermore, we will extend the reach of our benchmarks by including additional benchmarks that cover anything from full-blown big data frameworks to individual big data technologies such as databases and search frameworks. We will offer individual micro-workloads from those benchmarks that will focus on certain aspects of the full benchmarks. These micro-workloads will be made available through an integration with the Jenkins, a continuous integration system. As an effect developers will be able to evaluate the performance of their development in an integrated environment of their choosing against specific workloads.

3.2 Software

The software setup of the virtualised big data use case can be separated into the software used in the benchmarking stage and the software used for the real-world implementations. However, we expect that both parts will overlap with regards to their software components significantly. The benchmarks will cover a variety of commonly used big data components such as YARN [10], HDFS [12], and Spark [16]. Further components that will be considered in the use case are Mesos, HDFS, Cassandra, Spark, Elasticsearch, Solr, Pig, and Hive. We expect that these technologies will also find their application in the real-world implementations.

3.2.1 Benchmarks

This section provides an overview of benchmarks for big data platforms and it describes the setup of our initially chosen benchmark HiBench. The analysis of the state of the art provides a reasoning for HiBench as our preferred choice. The description of our HiBench setup contains details on the deployment and on the micro-workloads found in HiBench.

3.2.1.1 State of the Art Analysis on Big Data Benchmarks

This section provides a short overview of benchmarks for big data platforms. These benchmarks cover a broad variety of big data technologies and workloads. Some of those benchmarks listed here have received a lot of engineering effort. The analysis of the state of the art will guide our decision for a suited benchmark. Our choice for the first benchmark to adopt is HiBench.

HiBench [1] is a benchmark specifically written to analyse big data platforms. The benchmark has been developed by Intel and is available as open source software on GitHub. HiBench operates with ten micro-workloads on generated data. The micro-workloads represent different but typical big data workloads. All tests evaluate speed of execution, throughput, HDFS bandwidth, system resource utilization, and disk access patterns. There are five types of generated data, which are re-used by multiple workloads. The data used by HiBench is described in Section 3.3.1. HiBench supports Hadoop 1.x, Hadoop 2.x, Spark 1.2, and Spark 1.3 as software components. Our concrete setup is described in the following section. HiBench is our primary choice because, it builds on standard big data components, it is open source, and its micro-workloads can be run independently of each other. Furthermore, the project is under active development according to its statistics on GitHub.

Big Data Benchmark for Big Bench [2] is another actively developed benchmark specific to big data platforms. The benchmark is available as open source on GitHub and it is also developed by Intel. However, this benchmark is developed together with Cloudera and the system under test is the Cloudera Distribution of Hadoop. The benchmark builds on HiBench and contains 30 micro-workloads. The software covered includes Hadoop 2.3, Mahout, Hive 0.12 and Java 7. The project is active on GitHub and its functionality extends beyond that of HiBench. However, we chose HiBench over Big Data Benchmark since HiBench covers a vanilla Apache Hadoop platform. Nonetheless, due to its extent we plan to use Big Data Benchmark later in the project.

BigDataBench [3] is another comprehensive benchmark for big data platforms. The benchmark operates on 14 real-world data sets that span structured, semi-structured, and unstructured data. There are 33 workloads that originate from various fields of application. Some of those workloads implement the same algorithm with multiple technologies. The software components in the benchmark span Hadoop, Spark, Nutch, HBase, MySQL, Hive, Shark, Impala, and MPI. However, some of the workloads are only implemented using MPI and thus should not be considered as big data implementations. It is possible to download the benchmark from the authors' website as open source software. The most recent version is from December 2014. One major drawback of this benchmark is the overhead to install it. The software platform, the data sets, and the workloads need to be downloaded and run separately. Due to the comprehensive nature of the benchmark we will consider the benchmark or at least parts of it for a future extension of our benchmarking setup.

The ***AMP benchmarks*** [4] by the AMP lab at Berkeley provide four queries, disk-based and in-memory, based on data generated by HiBench and sampled from Common Crawl. The software setup includes Redshift, Shark, Impala, Hive, and Tez. The whole benchmark has been run on Amazon EC2 and in general depends on AWS. This dependence on AWS prevents a direct application of this benchmark for our purpose, in specific to evaluate sKVM. We estimate that there would be substantial effort to run the benchmark on our setup. Furthermore, there are more elaborate and flexible benchmarks available. Finally, the project itself is not active on GitHub.. The last code has been pushed over a year ago.

TPCx-HS [5] specifies a big data benchmark, which resembles the features of HiBench. As software the benchmark is built on Hadoop with HDFS and custom algorithm for benchmarking. The data uses modified versions of the workload generators TeraGen, TeraSort, and TeraValidate. These are in part also used in HiBench. Specification and implementations of this benchmark are not directly available from public sources, such as GitHub. Furthermore, HiBench implements most of the features described in TPCx-HS. HiBench even goes beyond these features and it is freely available for installation.

Besides the mentioned benchmarks there are related benchmarks, which target either cloud computing, databases, or only isolated components. For example YCSB [6] is a benchmark that targets cloud computing and individual databases. Thus it does not target our specific needs. CloudSuite [7] is another benchmark that mostly targets cloud computing. Furthermore CloudSuite is not available under an open source license. There are various benchmarks for databases, which are typical for applications in big data. LinkBench [8] is a popular example for a benchmark for databases that operates on graph-based data. PigMix [9] in turn is a benchmark specific to Apache Pig as one example for technology-specific benchmarks. For future work we will consider to include benchmarks for important technologies, databases or others, into our benchmarking setup.

3.2.1.2 HiBench Setup

Our baseline measurements come from two similar sets of experiments. The first set of experiments runs HiBench directly on three hosts. The second set of experiments runs HiBench on the same hosts, but inside VMs running on top of KVM.

Figure 1 shows the setup for the direct deployment of HiBench on the three hosts. Hadoop, HDFS, Spark, and HiBench are installed directly on the Ubuntu host system. The first node acts as a master node and worker node at the same time. First, for each of the micro-workloads data is generated and then stored in HDFS. Then the algorithms for each of the workloads are executed on all three hosts. Hadoop uses remote procedure calls to distribute work among the workers, which then run independently. Each of the worker nodes in turn runs multi-threaded algorithms.

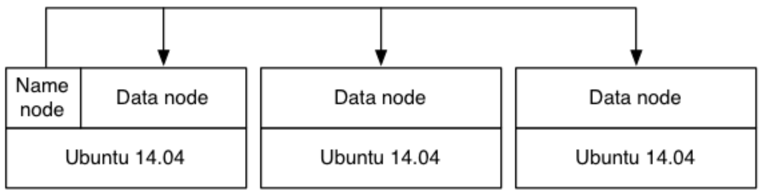

Figure 1: Deployment of HiBench directly on the hosts.

Figure 2 shows the benchmarking setup with VMs. Each host runs three VMs with Ubuntu as guest OS. The first host runs a relatively small VM, as specified in Table 3, for the master node. The worker VMs on all hosts have the same configuration. The configuration of both types of VMs is described in Table 3. In the second setup we keep the problem size and general benchmarking configuration the same as for the host-based setup.

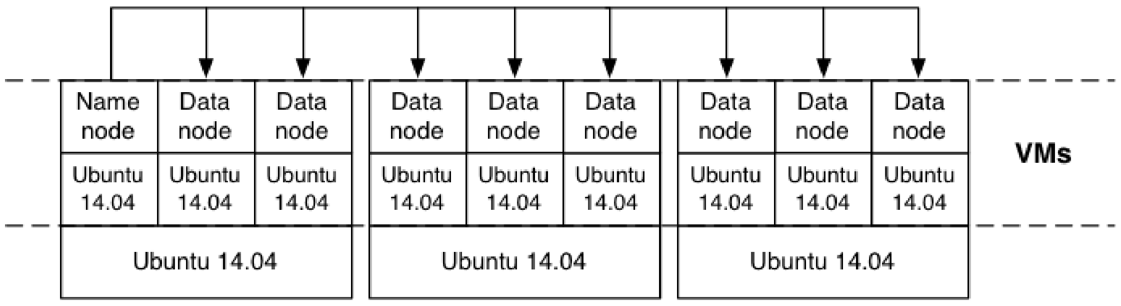

Figure 2: Deployment of HiBench in VMs.

Property	Name Node	Data Node
Quantity	1	8
Memory	8	40

Storage	40GB	40GB

Table 3: VM configuration for benchmarking

We now briefly describe the micro-workloads, before we turn to the results of the experiments in the next section. The concrete data used as input and the data generated by the experiments is described in Section 6.2. Here, we only provide an overview of the data sources. The description of both the data and the workloads is taken from the HiBench documentation at GitHub [1].

1. Sort – This micro benchmark sorts text data, which is generated by RandomTextWriter.
2. WordCount – This micro-benchmark counts the occurrences of words in the input data, which is generated by RandomTextWriter.
3. TeraSort – This micro benchmark sorts a large data set generated by TeraGen. This micro benchmark has been used to push the boundaries of sorting very large data sets.
4. Sleep – This micro benchmark sleeps for a set amount of time to test the scheduler.
5. SQL (Scan, Join, Aggregate) – This benchmark tests typical SQL commands using Hive queries. The benchmark operates on generated hyperlinks.
6. PageRank – This benchmark computes the PageRank on generated hyperlinks.
7. Nutch indexing – Nutch is an Apache search engine and implements an indexing algorithm. The benchmark operates on generated web sites with words and hyperlinks.
8. Bayesian classification – This benchmarks uses Spark MLLib and Mahout to perform Naive Bayesian classification. The data consists of generated text documents.
9. K-means clustering – In this benchmark the k-means clustering algorithm in Spark MLLib/Mahout is used to cluster generated input data.
10. Enhanced DFSIO – This benchmark tests the HDFS throughput by generating a large number of tasks that simultaneously write and read data from HDFS.

3.2.2 Mini Use-Cases

The software setup for the real-world implementations of mini-use cases is not yet known. However, the most likely technologies can be estimated already. The software components for the big data platform fall roughly into the three categories middleware, databases and storage, and processing components.

As middleware Hadoop YARN [10] is a likely candidate that covers a broad set of use cases. Especially the increased flexibility of YARN over MapReduce makes it a popular candidate for batch processing and real-time processing. Another likely candidate for a middleware is Apache Mesos [11], which is a resource manager for combined workloads from multiple frameworks.

Database systems and storage systems will likely cover HDFS [12], Ceph [13], and Cassandra [14]. In conjunction with HDFS and in the context of big data in general it is likely that HDFS will be used. In addition, the GWDG cloud uses Ceph as a storage middleware, which can benefit from I/O speed up in the context of this use case as well. The database system to be used will depend more on the mini-use cases. Cassandra is a likely candidate to be used due to its broad applicability and its good performance. For computation Hadoop [15], Spark [16], and Elasticsearch [17] are likely candidates. For batch processing Hadoop offers a mature big data technology, which should be supported to generate a significant impact and to reach a large customer base. Spark has become a very popular choice for real-time processing of data streams. With Spark we can test especially well the impact of I/O heavy applications with in-memory data. Additional technologies such as search technologies for large data sets via Elasticsearch and Solr [18] may come into play as well. The use of these and other potential technologies will depend on the actually chosen mini-use cases. For the real-world implementations there will be reference cases, which will be integrated with continuous integration. These reference cases will allow developers to test their software against real-world use cases in an automated way. The concrete choice of supported and tested technologies will be described in future iterations of this deliverable.

3.3 Data

This section describes the data associated with the big data use case. Again we distinguish between benchmarks and mini use-cases. Due to the early stage of the project, we provide concrete information on HiBench, which is our initially supported benchmark. Later in the project we will extend this section to offer information about the data in additional benchmarks. Furthermore, the information about real-world implementation is scarce and describes the expected data used and generated by the mini-uses cases. Future iterations of this deliverable will contain more information on this topic. In addition to the data itself, we briefly discuss data storage and data transfer for the benchmarks.

3.3.1 Benchmarks

The data associated with the benchmarks can be divided into input and output data. The input data refers to the data that the benchmarks operate on. For HiBench the input data consists nearly entirely of generated data. The output data can be divided further into output data of the algorithms and monitoring data. For the purpose of the benchmark the monitoring data is of special interest, since it provides a quantification of the benchmark performance.

There are broadly five types of generated data used by the workloads in HiBench. Table 4 and Table 5 in Section 6.2 list the data size for each workload. Thus we focus on the nature of the input data in this section. The first type of data is generated text from the Hadoop program RandomTextWriter. The algorithm generates random sequences of words. The generated data serves as input to the micro-workloads Sort and WordCount. The second type of input data originates from TeraGen, which is a generator of random data. TeraGen has been developed specifically for TeraSort. Thus in HiBench TeraSort operates on the TeraGen-generated data. The third type of generated data is web-data that follows a Zipfian distribution. Zipf's law offers a discrete probability distribution that is used in linguistics, since it fits the empirical distribution of words in written language well. Zipf's law is used to generate textual documents such as web-pages and links. The resulting data is used by the micro-workloads Scan, Join, Aggregate, PageRank, and Bayesian classification. The fourth type of data comes from GenKMeansDataset, which is a program that generates a large number of data points for the k-means algorithm. The generated data points follow either a uniform or a Gaussian distribution. The generated data is used exclusively for the k-means micro-workload. The fifth type of data uses the Linux dictionary at /usr/share/dict/linux.words. The dictionary is used for the micro-workloads Nutch-indexing and Bayesian classification.

The procedure of data generation and transfer is the same for all of the micro-benchmarks. First the data is generated by dedicated data generation algorithms for the micro-benchmark that is about to run. This data is then transferred to the Hadoop installation on HDFS. The data is replicated with a replication factor of 3. Thus, for the host-based experiments, each host has all of the data. For the VM-based experiments each data block is available at three different VMs. Data is thus stored locally at the nodes, distributed across the cluster. After computation the data is written back to HDFS.

There are two levels of detail for the output data. The first level of detail contains a high-level summary of the problem size, duration, and throughput for each micro-workload. This data can be found in Table 4 in Section 6.2. The second level of detail provides elaborate monitoring results, which give the same metrics as the summary, but for each task on its own. Using this detailed data once can search for more intricate patterns in the execution of tasks. The detailed logs are too long to be added to this deliverable. However, we attach a small excerpt of such a log to the appendix. The whole log files can be found in our project's data archive. For future iterations of the use case, we intend to extend the current monitoring data by performance monitoring of the host and the VMs. To get this performance data, however we first have to set up a monitoring system that can handle a high load. Furthermore, the performance monitoring data needs to be truncated and aligned with the available benchmarking data.

3.3.2 Mini Use-Cases

The real world use case implementations are not yet set. Thus the description in this iteration of the deliverable needs to be focus on our expectations about the data used in the real world implementations. It is clear to us that we want to cover a variety of use cases. This variety stems, among other aspects, from the variety of data. The data variety will show itself in the form of three of the five Vs of big data: velocity, volume, and variety. We would like to cover multiple instantiations across all of those dimensions. Thus there should be data with varying velocity. Volume of data should change between mini-use cases. In specific, we expect that use cases will offer data that varies in the distribution of its volume. While there should be use mini-use cases that offer many small files, there should also be mini-use cases that offer few, very large files. The variety of data between use cases should also be changing. The data can be divided into input and output data as in the case of the benchmarks. Based on our initial contact with the stakeholders of the mini-use cases we expect data from various domains of science. These domains include full-text search in digital humanities, sensor data from wind tunnels, medical statistics, bio-informatics, genome sequencing, and analysis of text data of ancient American cultures.
The output data of the real-world implementations will consist of the output data of the use cases themselves and of monitoring data. Whether the outputs of the use cases themselves can be shared will depend on the final mini-use cases. Use cases with an open data-sharing policy will be favoured. For the purposes of the project, however the monitoring data is of higher importance, since it will allow us to evaluate the impact of improved I/O on the execution of the mini-use cases. The monitoring output for the mini-use cases is expected to be similar to the output of the benchmarks. We expect to collect data about the duration and throughput of tasks and nodes. We also expect to capture monitoring data of VMs and hosts that run the worker nodes. During the project's later stages we may expand the monitoring data by probes that focus on I/O performance.

Results of all will be uploaded as far as possible according to privacy agreements. Mini use-cases with publicly accessible data will be favoured with all things being equal.

3.4 Security

The issues related to security can be divided into security issues with the benchmarking part of the use case and with the real-world implementations. With the benchmarking part there are no constraints regarding security. Partners and the public can access all data used, generated, and recorded in the experiments. During the initial benchmarking phase partners will not gain any access to the machines executing the use case. However, in later phases the benchmarks will be integrated with the system for continuous integration. There partners will be able to run the benchmarks themselves with a custom configuration. Partners will also get access to the data used, generated, and recorded as part of the benchmark execution. There are no additional security requirements for the benchmarks. Currently, no encryption is used for data. Nevertheless, we may extend the set of benchmarks to include benchmarks with encryption and generation of random numbers with high entropy in future releases.

For the real-world implementations the security implications are currently unclear. We assume that some of the mini-use cases will impose restrictions on data access. Especially the access to the input and output data of the use case itself may be restricted. Monitoring data, however will be available publicly. Furthermore, we will provide reference cases for each mini-use case as integration tests and to repeat experiments. In this context, we will make source code and at least anonymized data available if possible. To verify MIKELANGELO's improved security features, we aim to have at least one security-sensitive mini-use case. Such a use case may have security requirements such as isolation on a host or efficient generation of high-entropy random numbers.

4 Requirements Analysis

This section describes all of the project's requirements and KPIs that relate to this use case. Most requirements need to be provided to the use case by other work packages. Still, some of the requirements need to be implemented by this use case itself. Similarly, KPIs that relate to the use case in any way are introduced in the second sub-section. Most of the presented KPIs will be fed with results from the use case.

4.1 Requirements

This section lists the mandatory requirements for the virtualised big data use case. The requirements listed here refer to a subset of the project's overall requirements list for year 1 [32]. Here we only list those requirements which are mandatory to this use case. Each requirement comes with a description of its relation to the implementation of the use case and how it enhances the use case.

Requirement #18 – Integration of the modified hypervisor with OpenStack

The mini use-cases will include cluster on-demand features which quick cluster deployment to work on top of OpenStack. To show the benefits of the full MIKELANGELO stack, this use case will require the integration of sKVM with OpenStack.

Requirement #70 – DPDK integration on guest

If at all possible, OSv should be used to host services in the context of big data. In OSv in turn as much I/O throughput as possible should be made available. For this DPDK is a mandatory requirement.

Requirement #71 – vhost-net integration with virtual switch on host

The integration of vhost-net with the host-based virtual switch is required to set up low latency, high throughput networking between VMs. Such improved networking in turn will speed up workloads that rely on inter-VM communication.

Requirement #75 – Integration of RDMA components with Ubuntu guest

Ubuntu will be used besides OSv in the use case. RDMA integration with Ubuntu will provide improved I/O performance. This integration is expected to offset some of the disadvantages that come with using Ubuntu instead of OSv.

Requirement #89 – vhost support in hypervisor's user space

Running vhost in the hypervisor's user space helps to avoid context switches between the kernel and user space. Reduced context switching in turn leads to improved I/O performance.

Requirement #3 – Hypervisor command line API

The hypervisor command line API will be required in the initial phases to set up the use cases statically before the OpenStack integration will be fully available.

Requirement #7 – Multi VMs shared memory communication

In this use case there will be multiple VMs per host to use the full potential of the host system. Communication between collocated VMs currently imposes an unnecessary overhead. Reducing this overhead by shared memory communication is expected to lead to gains in performance for workloads in which VMs cooperate intensely.

Requirement #14 – Hypervisor API backward compatibility

Hypervisor API backward compatibility is mandatory to allow sKVM to run as part of an infrastructure that lacks the extensions of MIKELANGELO in OpenStack. Thus, this feature will allow sKVM to integrate seamlessly with legacy cloud installations.

Requirement #55 – Hypervisor virtual network connectivity

The network connectivity of the hypervisor with the virtual network is required to allow communication of VMs.

Requirement #58 – Hypervisor Ubuntu guest support

It is foreseeable that not all big data components will run in OSv. Thus, Ubuntu as a replacement needs to be supported.

Requirement #65 – Hypervisor virtual block device support

The hypervisor needs to allow access of virtual block devices to VMs. The big data use case in specific requires block device images to be stored locally on hosts, since data is distributed to VMs by Hadoop.

Requirement #101 – Libvirt integration for sKVM

The architecture of this use case builds on OpenStack, which in turn uses libvirt to manage VMs. Thus, this use case also requires libvirt integration of sKVM.

Requirement #96 – Hypervisor mechanism to detect or mitigate cross-VM information extraction via side-channels, requirement #962 – Prime&Probe meta-attack, #964 – Collect timing and counter information from cache

For mini-use cases with security-sensitive data, detecting and counteracting cross-VM information extraction is a mandatory requirement. This feature will allow the use case to be sold as secure to stakeholder of mini-use cases with sensitive data.

Requirement #20 – OSv support for environment variables

To deploy big data services in VMs running OSv environment variables are mandatory. Environment variables will be used to inject configuration into the VMs.

Requirements #23 – OSv support / API for monitoring the instances, #42 – Capture performance metrics of guest OS – OSv, #40 – Hardware monitoring, #2 – Hypervisor monitoring

In this use case end-users and admins need to have a good overview of the performance of their big data cluster. For this purpose monitoring support for OSv is required.

Requirement #29 – OSv support for multithreading

Big data services, which run in a single process, commonly use as many cores on a machine as possible by multi-threaded execution. Multithreading is preferred over running many processes, since data sharing can speed up execution. Thus, multi-threaded execution of OSv is a mandatory requirement.

Requirement #39 – Big Data stack running in OSv and sKVM

In this use case we would like to make the best use of all components of the MIKELANGELO stack. OSv and sKVM are at the core of the project's stack. Thus, we would like to run as many services from the big data stack in OSv as possible. Our goal is to keep the overhead of execution and setting up a big data cluster as low as possible. In specific, we want VMs to transfer, boot, and migrate as quickly as possible. Furthermore, we want to have full I/O integration and use OSv's application deployment mechanisms.

Requirement #73 – Integration of RDMA components with OSv guest

For the big data services running in OSv, we want to boost the I/O performance by using RDMA. Especially in situations where collocated VMs need to communicate a lot, we want to allow for reduced communication overhead by using host-based shared memory.

Requirement #10 – Hypervisor support for Ubuntu guest

Currently, not all major big data services run in OSv. To mitigate this issue and to provide performance comparisons, we will deploy the big data services in a competing cloud OS. Initially this other guest OS will be Ubuntu.

Requirement #41 – Capture performance metrics of host hypervisor – sKVM

The performance metrics of the hypervisor provide crucial information to administrators of cloud infrastructures. Furthermore, this monitoring data will allow the development of autonomic algorithms to adjust configuration and VM placement to manage performance.

Requirement #9 – Hypervisor support for OSv guest

In this use case we want to showcase as many features of MIKELANGELO as possible and we want to gain the most performance. Thus, we primarily target the combination of sKVM and OSv as primary technologies to execute big data services.

Requirement #45 – Monitoring GUI

The monitoring GUI is an important feature for administrators doing DevOps, development of the use case, and for end-users, who want to monitor the execution of their platform. This GUI should be easy to use and flexible. End-users should be able to identify bottlenecks in their deployment.

Requirement #43 – Services/Applications monitoring

This requirements extends the previous monitoring requirements by adding the service layer. The quality of service is the final monitoring metric that counts for the end-user. In addition, detailed monitoring of services allows to understand easier which parts of a service require optimisation.

These requirements form a large part of the year 1 requirements of the MIKELANGELO project. The requirements cover all layers of the MIKELANGELO stack from the hypervisor up to the cloud integration. The main requirements that have been left out are those that relate specifically to HPC infrastructure such as the integration of MPI and Infiniband. Naturally these requirements will be demonstrated in the two HPC use cases that deal with cancellous bones simulations and aerodynamic maps.

4.2 Key Performance Indicators

This section describes the project's key performance indicators that relate directly to the big data use case. For each KPI there is a subsection with the description of the KPI, the importance of the KPI for the use case, the impact on GWDG's business model, and how we are going to monitor the KPI. In total this use case relates to fourteen KPIs.

KPI1.1 – Relative efficiency of bursting a number of virtual machines

This KPI measures the characteristics of bursting various numbers of VM to form a big data cluster from scratch. The measurements should be clear enough to identify the bursting behaviour under various load scenarios of the infrastructure. This includes CPU, memory, storage, and networking resources.

This KPI is important to the big data use case since forming a big data cluster from scratch is akin to cloud bursting. Many new VMs, including custom services with custom configurations, need to be set up at once. Furthermore, there may be cases where small big data clusters need to be scaled up rapidly, which again is a classic example for cloud bursting. The actually required scale of bursting capabilities will depend on the mini use-cases. For example, a mini use-case that provides sensor data at unpredictable intervals might require a big data cluster for real-time processing to be set up within few seconds. Otherwise important data and insights may get lost.

The success with regards to this KPI will affect how many new customers the GWDG will be able to attract to the new big data services. The quicker we will be able to set up a big data cluster, the more customers we will be able to interest in our service. In contrast, if the bursting quality will be poor or unreliable, customers are expected to penalise us by leaving the service and by giving poor reviews.

This KPI can be measured by technical means. Experiments will be conducted to measure the duration of bursts with various configurations. These configurations will contain variations in cluster size, VM configurations, services to be installed, infrastructure load, and distribution of VMs. This KPI will be measured in even more detail in the cloud bursting use case, which focuses on this specific area.

KPI1.2 – Relative improvement of time to change the installed application on a number of virtual machines

In the context of our use case this KPI describes how long it takes to install big data services in running VMs. Thus, this KPI is a subset of KPI1.1. While KPI1.1 measures the whole duration to perform bursts, this KPI only measures the time it takes to install services on top of preconfigured big data VM images.

This KPI is especially important to the big data use case since we want to allow our end-users to customise their big data cluster. Customisability is one of the main selling points of our work. Customisability means that we are going to offer base images for big data installations. Based on these images the system will need to deploy additional software and configurations to form the customised cluster.

This KPI will allow GWDG to provide a customisable big data service to end-users. This flexibility will allow us to reach more customers and thus target more potential clients. Furthermore, by letting customers customise the cluster, we keep the administrative overhead for this service low.

Monitoring of this KPI will be similar to monitoring KPI1.1. Again this KPI can be monitored by technical means. We can measure how long it takes to perform various deployments of under various load scenarios. The deployments will differ in the software configuration, the number of VMs, and their distribution across hosts.

KPI2.1 – Relative efficiency of virtualized I/O between KVM and sKVM

This KPI lies at the core of MIKELANGELO. In the big data use case the relative efficiency of I/O operations between KVM and sKVM will determine the success of the use case. Improving the I/O efficiency will make it feasible to transform this experimental use case into a working service. We expect a speedup of our use case as measured by reduced execution times, which are expected to stem from increased I/O throughput and reduced I/O latencies. The increased efficiency in turn will allow us to run more big data applications on the same virtual infrastructure.

This KPI is one of the key metric for this use case. Specifically the increased I/O efficiency of sKVM is important, since not all big data services are expected to work with OSv. Thus, sKVM will lead to a large part of speedup in generic settings.

This KPI impacts GWDG's business model by enabling a virtualised big data service in the first place. Without success in this KPI running a virtual big data cluster would be too expensive. In contrast, with success in this KPI, we will be able to efficiently use spare resources of our cloud. Furthermore, we will be able to serve more customers on the same infrastructure. This in turn will allow us to receive additional funds to invest in additional infrastructure.

This KPI will be measured by technical means using benchmarks and reference cases from mini use-cases. The I/O efficiency will be measured by comparing durations of execution and throughput of the big data cluster for KVM and sKVM-based deployments with all other components fixed.

KPI3.1 – The relative improvement of efficiency of MIKELANGELO OSv over the traditional guest OS

This KPI evaluates the improved I/O efficiency of OSv over competitor guest OSs. In this use case we will compare the I/O efficiency of OSv based on baseline measurements with Ubuntu as guest OS. This KPI is in principle the same as KPI2.1 with the focus on the guest OS instead of the hypervisor. Thus, the importance for the use case, the impact on the business model, and the monitoring of the KPI are the same as those for KPI2.1.

KPI3.2 – The relative improvement of efficiency [size, speed of execution] between the baseline guest OS and MIKELANGELO's OSv.

This KPI refers to efficiency improvements of OSv other than I/O efficiency of applications running in OSv. In specific this KPI refers to the size of VM images, to boot times, and to migration times. This KPI measures mostly the gains in management speed that stem from OSv small footprint. Again, we are going to compare the speed along the mentioned dimensions to those metrics obtained with an Ubuntu guest.

This KPI is important since it allows a far more elastic and reactive cloud system. By extension this KPI will allow the big-data use-case to create more base images for big data clusters and to achieve shorter bursting times.

Our business model will be impacted by this KPI by enabling us to provide more resources and a more agile service to our customers. Resources that are freed by using a more streamlined OS can be made available to customers. Furthermore, having a reactive system will increase our reach to include customers who require reactive systems.

We will measure the KPI by monitoring the time it takes to transfer OSv images, to boot OSv-based VMs, and to migrate them.

KPI3.3 – The relative improvement of compatibility between baseline and MIKELANGELO versions of OSv

This KPI describes the improved compatibility of OSv with big data services. Currently, OSv is a new development, which supports a limited number of applications. To foster a more widespread adoption OSv's application compatibility will be developed further to support more applications. This KPI measures the improvement of support for additional applications. In this use case we will focus on services that are needed to run a big data cluster.

Since we want to gain the benefits of running OSv, such as fast migration, VM setup, boot times, and I/O operations, we aim to use OSv as much as possible. Thus, a widespread support of OSv for big data services is important. Running more services within OSv will allow us to provide a more reactive system and to use resources betters. This in turn will again attract more customers to our service.

This KPI will be monitored by comparing the amount of software running in OSv without MIKELANGELO extensions and with the improved version of OSv. As a basis, we assume that all those services run perfectly on mainstream OSs such as Ubuntu. We will evaluate the number of services running on OSv before and after extension of the OS. Furthermore, we will evaluate the issues that arise in the context of installing and running services using OSv.

KPI4.1 – A system for appropriately packaging applications in MIKELANGELO OSv is provided

This KPI measures the application packaging mechanism for OSv. In this use case we aim to provide thin base images for VMs that will then be extended by additional services upon deployment. This way we can keep big data base images locally on hosts. All of the custom services can then be installed once customers configure their cluster. This approach assumes a reliable application packaging mechanism.

For this use case to succeed in using OSv the application packaging mechanism is a mandatory requirement. It is not feasible to create many varying big data VM images. Thus, installing services once VMs have booted will be common procedure. This procedure however requires an application packing and deployment system. Overall, application packaging will thus allow GWDG to offer a customisable, instantly available big data cluster without extra administration overhead.

This KPI aligns itself with KPIs 3.1-3.3 in terms of business impact. Both need to provide satisfactory results for GWDG's business plan to succeed. First, OSv needs to provide improved performance over baseline OSs. Second, OSv needs to support as many big data services as possible. Third, OSv needs to provide an application management mechanism to enable us to deploy the services. Without application packaging it will become much more tedious to offer customisable big data services using OSv. The only workaround, to not having the application packaging mechanism, would be to use another guest OS or to provide custom installation scripts. Developing such scripts in turn would require extra effort and thus limits the breadth of services that can be offered to end-users.

This KPI will be measured by noting how many of the big data services in this use case can be managed using OSv's application packaging system. Following this approach there are two trivial baselines. One is the application packing system offered by a mainstream guest OS such as Ubuntu. Any mainstream OS offers nearly all regularly applications in a packaged format. The other trivial baseline is the current application packaging capabilities of OSv. For our application we will measure the applications that can be used with application packaging compared to the initial baseline. The ideal situation for this KPI would be to reach the same level of support as offered by a mainstream OS such as Ubuntu.

KPI4.2 – A relative improvement of efficiency [time] between deploying a packaged and non- packaged application.

This KPI measures the duration to deploy various big data components that are supported by the application packaging mechanism of OSv. This KPI includes both the effort to package a new application and the time it takes to deploy such an application in a running system. The first aspect refers to an administrator packaging an application and making it available to the application packaging system. The second aspect is about the actual automated deployment.

Both aspects are important to our use case. First, the packaging of services should require low effort. The lower the effort the more big data services can be offered to customers. Second, an efficient deployment of applications is important to facilitate bursting scenarios. Fast bursting translates to a quick setup of whole virtual big data clusters.

Both aspects, the effort for application packaging and the effort for application deployment, impact the business model of running on-demand big data clusters. First, efficient packaging of applications translates into more big data services offered to customers. Administrators are a constant bottleneck in a data centre. If it takes a long time to introduce a new big data service into the system, fewer services will be offered over time. Fewer offered services result in a smaller target audience, which in turn results in fewer customers. Thus, efficient packaging is an important part in the chain of offering a virtual big data service. Second, the efficiency of deploying applications is important to provide good bursting capabilities. Transferring large images takes a long time and limits bursting capabilities. However, with refined application deployment mechanisms one can make up for most of this time. Furthermore, good bursting performance again allows us to target more customers as it will allow them to scale very quickly according to their needs.

We will monitor both aspects separately. First, we will measure the effort to deploy an application in OSv manually by retrieving the sources, compiling, and running an application. This metric will be compared to the effort of using Apt [19] on Ubuntu and potentially also configuration management systems such as Puppet [20] or Ansible [21]. Second, we will measure deployment times using Ubuntu guests with the above mentioned tools, and OSv with and without optimisations for application packaging and deployment.

KPI5.1 – Demonstration of a management infrastructure on diverse physical infrastructures.

In the big data use case this KPI refers to the use of the MIKELANGELO stack on heterogeneous physical infrastructures. In general this KPI refers to using the stack on disparate systems such as cloud and HPC systems. The project showcases these features across the different use cases. In this use case we restrict the breadth and instead delve a little deeper by showing how the management infrastructure will run on a heterogeneous physical infrastructure. Heterogeneity in this case refers to having non-uniform systems with different hardware configurations along the lines of memory, CPU, and networking resources.

This KPI is of importance to the big data use case since typical cloud infrastructures grow over time. This growth comes with changes to the physical landscape. New components rarely are identical to older ones. Resource specifications and resources capabilities usually improve. Thus, this KPI is important to verify that MIKELANGELO will work in realistic cloud deployments.

By extension this KPI is important for the business model. As described above, a typical cloud business uses hardware of different generations. Thus, success in this KPI is required to build a feasible business model on top of it.

We will monitor this KPI by identifying problems that arise by deploying the big data stack using the MIKELANGELO stack on heterogeneous physical infrastructure. Issues will be categorised, counted, and analysed for root causes.

KPI6.1 – Enumerated security concerns solved on hypervisor level

This KPI refers to identifying and solving security issues in the current generation of KVM. Furthermore, issues that arise through the extensions of sKVM over KVM need to be dealt with as well. Vulnerabilities in KVM that are published during the project's runtime need to be addressed.

This KPI is of special importance to mini uses-cases that operate on sensitive data. In this use case big data clusters will be offered that run in a public cloud, which are co-tenant by design. Thus to offer sufficient security the hypervisor needs to isolate VMs as well as possible.

The better we can justify that our deployment is secure, especially with regards to VM isolation, the more customers we can attract. Especially German laws and IT culture require high standards with regards to data privacy. Improvements over the baseline for this KPI will again allow us to attract more customers.

The success of this KPI will be reflected by the work on security issues. There will be a list of discovered and resolved vulnerabilities. Also a list of known vulnerabilities in KVM will be compared to the resolved issues in sKVM.

KPI7.1 – All use cases appropriately demonstrated

For the virtualised big data use case this KPI refers to running benchmarks and mini use-cases on a sufficient scale. Furthermore, the performance of those mini-use cases needs to be compared to the performance of baseline measurements without the use of the MIKELANGELO stack.

Successfully implementing this use case means being successful in fulfilling this KPI. Success in this KPI in turn is important to exploit the results of this use case by rolling out a productive service.

This KPI will be monitored by evaluating the success of all other KPIs associated with this use case. The appropriate demonstration of the KPI depends on the demonstration of its constituent KPIs.

KPI7.2 – Documentation of using MIKELANGELO in Use cases - Best Practices tutorial

This KPI evaluates the documentation of the use case with regards to the use of the MIKELANGELO stack. This documentation shall describe in detail the experiences and best practices working with sKVM, security enhancements, RDMA, OSv, cloud integration, and the big data integration itself.

This KPI is important to allow outside parties to reproduce the results of the project. Third parties will be able to make use of the results of MIKELANGELO and to improve on them. Good documentation is also a prerequisite to build a community and foster the continued development of the MIKELANGELO stack for big data deployments.

The business impact is influenced by two separate aspects, one internal to GWDG and one external. The internal business impact builds on the documentation to maintain and improve the big data service with time. The external aspect refers to gaining interest in the community and to foster additional development by third parties. This in turn may lead to improvements that can be of use to GWDG's service in future.

This KPI will be measured by identifying the aspects that improve the documentation beyond documentation in source code, scripts, and publicly available documentation of individual components. Furthermore, the KPI should give an estimate how much time the documentation saves in rolling out a service similar to ours by third parties. Should there be third parties rolling out our big data stack themselves, their feedback on documentation will be elicited and evaluated as part of this KPI as well.

KPI7.3 – Documentation of using MIKELANGELO in use cases – documented benefits

This KPI measures the quality and quantity of the documentation about the benefits of the big data use case.

The documentation of benefits is important to guide the development of the use case. Effort should be spent in a way to augment benefits and to alleviate problems with the state of the use case. Furthermore, documented benefits shall be used to advertise the big data stack to third parties to popularise the virtual big data stack and with it the whole MIKELANGELO project. These external parties may include end-users and other service providers.

At GWDG the documented benefits will drive strategic decisions on how to deploy the results of MIKELANGELO in production. Furthermore, we will see how we can best contribute to the use case to improve its impact.

The KPI will be monitored by quantifying the documented benefits. Additionally the quality of those descriptions needs to be monitored as well. Concretely, we will monitor the quantity of text and feedback from target audiences about the quality of the documentation besides the quantity of documented benefits.

Conclusively, this section has described fourteen of the project's key performance indicators that relate to the big data use case. Success in some of those KPIs is required for this use case to succeed and all of the KPIs will be advanced with regards to big data deployments. The KPIs revolve around showcasing the improved efficiency of running big data applications on top of GWDG's cloud infrastructure. Further advancements that will be measured are improved security, infrastructure management, and a high quality documentation.

5 Implementation Plan

This section describes how we plan to run benchmarks and mini use-cases to evaluate the MIKELANGELO stack. The implementation plan follows the division into benchmarking and mini use-cases. As of writing this report the benchmarking stage is in an advanced state already. Most of the work associated with the benchmarking stage is technical. This technical work includes setting up the benchmarks, collecting, and analysing the performance data. The mini use-cases will go beyond setting up and running services. The mini use-cases will include talking to stakeholders and winning them over, helping them to implement their algorithms on our infrastructure, and moving data to GWDG's site. We briefly describe the implementation plans for these two parts of the use case in the following sections. These plans contain information on how we plan to set up the use case, potential issues that we foresee, and how we plan to overcome them.

5.1 Implementation of Benchmarks

The benchmarking stage of the use case itself consists of three phases. The first phase deals with researching benchmarks for big data platforms. This phase has been concluded and the results have been presented in Section 3.2.1.1. This phase gave us an overview of the popular big data benchmarks and the metrics that they cover. Based on this analysis we have chosen HiBench as initial benchmark and we will consider to extend the benchmarking setup by Big Data Bench and Big Bench.

The second phase deals with setting up the chosen benchmarks. As of writing this report the second phase is in progress. As part of this phase we have already set up HiBench directly on a host system and by using VMs as described in Section 3.2.1.2. Using this setup we have already collected information on the baseline of running big data benchmarks directly on hosts and in VMs as will be described in Section 6.2. In this phase the addition of more benchmarks such as Big Data Bench and Big Bench is still open. A restriction of this phase is the relatively static setup. Benchmarks will be run by GWDG personnel with a manual configuration. The setup is static in that it requires dedicated hosts and VMs are set up manually using libvirt.

The third phase deals with the integration of benchmarks into the continuous integration system. The goal of this phase is to provide a system that can be used by developers interactively to evaluate their work on specific micro-benchmarks. The implementation of this phase requires a successful cloud integration, which will then be leveraged by using Jenkins [33] to provide continuous integration. A benefit of the cloud integration will be that we can use more physical hosts for big data experiments during low usage times of GWDG's cloud infrastructure. The continuous integration system will then allow to run big data benchmarks as jobs in Jenkins. This phase is planned to be executed during the remainder of year 1.

We do not foresee major issues in the implementation of the benchmarking phase with the exception of OSv compatibility. This is an expected issue in the early phases of the project. It is known that some big data applications do not work in OSv right now. However, these issues will be addressed during further project stages.

5.2 Implementation of Mini Use-Cases

The implementation of the mini use-cases will dominate year 2 and year 3. Once the benchmarking part is implemented, we will shift our focus to the implementation of real-world problems using our big data platform. The use case stage with the mini-use cases comprises three phases.

The first phase consists of identifying the mini-use cases. This includes the identification of potential partners for mini-use cases, establishing contact, eliciting requirements, and choosing the right mini-use cases in the right order for implementation. The order of implementation will be crucial to gather experiences quickly and to get as many different mini use-cases on board as possible. Thus, the initial implementation phases will be marked by only one or two mini use-cases with relatively similar configuration. Once these mini-uses cases run satisfactorily, we will on-board more mini-use cases, which should be possible quite quickly. In the end of the project the process of on-boarding should only rely on consultation by GWDG to implement algorithms using big data technologies for those partners who do not possess this expertise themselves. For partners who know how to develop big data applications themselves the process should be automated so far that GWDG does not need to be involved explicitly.

The second phase of the mini use-cases stage consists of the actual implementation of mini use-cases. We expect that customers will vary greatly in IT skills. For those stakeholders that have experience in IT operations we will offer guidance to choose the right big data technologies and how to set up a big data cluster. From those partners we expect to gain deep insights into required improvements of the system to gain performance, especially in the early phases. The partners with less IT experience will be supported by GWDG to implement their algorithms and to transfer their data to GWDG's systems. We expect to gain insights from those partners regarding the requirements to automate our system to let those users gain more autonomy. Ideally at the end of the project these partners should be able to set up their mini use-cases with only a low amount of consultation to implement algorithms for their use case. Considering technologies such as Hive [22], which offer high-level queries on data, even partners with a relatively weak IT background should be able to implement their mini use-cases all on their own on GWDG's virtual big data system. Based on each of the mini use-cases we will offer reference cases that can be used with the continuous integration system. These reference cases will feature a fixed data set and allow evaluation of component and the whole integrated platform such as the micro-benchmarks in the first part of the use case.

In the final phase of the mini-use case stage we intend to push the automation of the virtual big data platform based on the experiences gained in the first two phases. The goal of the third phase to arrive at a system that can be spun-off into a productive service. End users should be able to use this service to set up customized big data clusters according to their needs. They should also be able to implement their use cases on their own using high-level systems such as Pig [23] and Hive. We expect that in the third phase the on-boarding of mini use-cases will be optimised so far that there will be a growing number of stakeholders using our platform.

This section has described the implementation plan of this use case. The implementation plan consists of two phases. The first phase is the benchmarking phase and will dominate the first project year. Year 2 and year 3 will be spent on the implementation of mini use-cases and by extending the virtual big data system to allow it to be spun-off into a service on its own.

6 Evaluation and Validation Plan

In this section we present the evaluation and validation plan, which relates benchmarks and mini use-cases to KPIs. In this iteration of the deliverable we will describe how the current progress on the use-case implementation provides a baseline for a core set of KPIs. Furthermore, we describe how future work on extending and integrating benchmarks with continuous integration and the implementation of the mini use-cases will cover more KPIs thoroughly.

6.1 Relation of KPIs to Experimental Metrics

In this subsection we list the KPIs related to this use case as in Section 4.2. For each KPI we provide a short description on the KPI's relation to performance metrics obtained in experiments we ran, future experiments, and other actions of this use case.

KPI1.1 – Relative efficiency of bursting a number of virtual machines, KPI3.2 – The relative improvement of efficiency [size, speed of execution] between the Baseline Guest OS vs. the MIKELANGELO OSv.

These two metrics are closely related and will be measured by dedicated experiments in later project stages. Currently, it is open whether these experiments will build on mini use-cases or whether there will dedicated synthetic experiments to test the bursting capability of the use case. This KPI can best be measured in a meaningful way on a cluster that goes beyond a couple of hosts. However, the required resources will only be available to the test bed in later project stages. Concretely, the KPI will be based on the deployment times of big data clusters on virtual infrastructure. For KPI1.1 only the transfer and booting of VM images will be taken into account. The deployment of software will be measured in KPI3.2. The measurements will be conducted with various image sizes, numbers of images, numbers of hosts, and under varying load scenarios. This wealth of configurations will provide a thorough insight into the limits of bursting capability. These experiments will be repeated with and without OSv to show a relative improvement over the state of the art.

KPI1.2 – Relative improvement of time to change the installed application on a number of virtual machines

This KPI complements KPI1.1 in that it deploys applications within already running VMs. Both KPIs together form the full bursting scenario. As with KPI1.1 the experiments to measure this KPI will be provided in later project stages. The experiments will measure the time it takes to install big data clusters on base images that only contain the guest OS. Further experiments may complement the findings by using base images that have common configurations of big data services that lack only small additions. To provide a relative improvement of using MIKELANGELO the experiments will be conducted with KVM and Ubuntu as baseline and with the MIKELANGELO separately.

KPI2.1 – Relative efficiency of virtualized I/O between KVM and sKVM, KPI3.1 – The relative improvement of efficiency of MIKELANGELO OSv over the traditional guest OS

These two KPIs are very similar in nature and thus will be measured in the same way. We will execute the benchmarks and mini use-cases using our baseline stack, and sKVM and OSv separately and in combination. This will allow us to compare the relative improvement achieved by sKVM and OSv separately and in combination. The metrics collected by experiments that cover this KPI are the duration of execution of I/O-heavy workloads, latencies caused by I/O operations, and the aggregate throughput of the cluster.

KPI3.3 – The relative improvement of compatibility between baseline and MIKELANGELO versions of OSv

This metric will be measured in later project stages by reviewing how many big data services can be installed with ease using OSv in the beginning and how this number improves with the progress of the project. This will also be integrated with the benchmark that are already running. The main reason for the metric not being available as relative improvement already is our focus on providing baseline measurements first.

KPI4.1 – A system for appropriately packaging of applications in MIKELANGELO OSv is provided

This KPI will be evaluated by the number and ease of deployment of OSv-compatible big data services with the application packaging system. This metric will be measured in later project stages.

KPI4.2 – A relative improvement of efficiency [time] between deploying a packaged and non-packaged application.

This KPI will be evaluated by measuring the effort to deploy an application via OSv's packaging mechanism in contrast to the current manual deployment mechanisms. We will evaluate this effort for a first-deployment scenario. Once scripts and automated deployment systems are available the benefits of the application packaging are mostly offset. Thus, the manual installation and creation of installation scripts or integration with configuration management need to be evaluated primarily.

KPI5.1 – Demonstration of a management infrastructure on diverse physical infrastructures.

This KPI will be verified by using increasingly diverse physical infrastructure to run the big data use case. Initially, the use case runs the benchmarks on just three homogeneous nodes. As the project progresses GWDG will add more nodes to the test bed and allow the use of its cloud resources, which are inherently heterogeneous. The friction or lack of friction introduced by the heterogeneity of the physical infrastructure will be quantified and used as basis for this KPI.

KPI6.1 – Enumerated security concerns solved on hypervisor level

Security issues will be tested by running security tests in MIKELANGELO using the continuous integration system. If at all possible these issues will be demonstrated in tests with use-case-specific scenarios.

KPI7.1 – All use cases appropriately demonstrated

This KPI will be measured by evaluating the improvement of I/O efficiency, bursting capability, and ease of use of the big data platform. The first two aspects will be measured already by some of the KPIs above. The ease of use will be evaluated by the success or issues end users face in deploying and using their own big data cluster on demand.

KPI7.2 – Documentation of using MIKELANGELO in Use cases - Best Practices tutorial, KPI7.3 – Documentation of using MIKELANGELO in Use cases - Documented Benefits

These KPIs will be measured by the quantity of documentation and its quality as evaluated by reviewers.

6.2 Results from Baseline Measurements

In this section we present the results of our initial benchmarking runs. First of all we distinguish between benchmarking directly on hosts and benchmarking using VMs. Each of those benchmark instances consists of ten micro benchmarks. Some of those micro benchmarks provide multiple implementations. These differ in language of implementation and framework. The frameworks includes Hadoop and Spark. The Spark-based implementations feature code in Java, Python, and Scala. In this report we only list the aggregated benchmarking results, which mention the duration, aggregate throughput, and throughput per node for each implementation of micro workload. These results and all remaining experimental data can be found in MIKELANGELO's open data repository at http://opendata.mikelangelo-project.eu.

Table 4 lists the results for the benchmark run directly on the host systems. Table 5 lists the results for the benchmark as run in VMs. Comparing both tables one can see that both benchmarks run mostly the same micro workloads with identical or at least very similar problem size. The last three experiments encountered problems related to the implementation of HiBench. Thus in Table 4 one can see that the host-based experiments ran the micro workloads with Bayesian classification, but they did not run the micro workloads for TeraSort. The VM-based experiments in contrast ran TeraSort, but not Bayesian classification. The encountered errors have been reported as issues to the developers of HiBench via GitHub. Once these issues get resolved, we will update the results with the missing experiments.

Workload	Input Data Size	Duration (s)	Throughput (B/s)	Throughput per node (B/s)
HadoopAggregation	37276711	35.413	1052627	350875
JavaSparkAggregation	37276833	55.806	667971	222657
ScalaSparkAggregation	37276833	55.769	668414	222804
PythonSparkAggregation	37276833	56.960	654438	218146
HadoopJoin	1000350	63.467	15761	5253
JavaSparkJoin	190683757	64.727	2945969	981989
ScalaSparkJoin	195929471	64.204	3051670	1017223

PythonSparkJoin	195929471	65.539	2989509	996503
HadoopKmeans	4016371691	216.879	18518951	6172983
JavaSparkKmeans	4016371691	175.469	22889351	7629783
ScalaSparkKmeans	4016371691	119.113	33719003	11239667
PythonSparkKmeans	4016371691	225.720	17793601	5931200
HadoopPagerank	259928115	214.100	1214050	404683
JavaSparkPagerank	259928115	97.275	2672095	890698
ScalaSparkPagerank	259928115	93.249	2787462	929154
PythonSparkPagerank	259928115	108.506	2395518	798506
HadoopScan	186646828	44.345	4208971	1402990
ScalaSparkScan	186647745	54.391	3431592	1143864
PythonSparkScan	186647745	57.390	3252269	1084089
HadoopSleep	0	18.149	0	0
JavaSparkSleep	0	124.422	0	0
ScalaSparkSleep	0	124.418	0	0
PythonSparkSleep	0	126.125	0	0
HadoopSort	328492566	20.117	16329103	5443034
JavaSparkSort	328492566	38.614	8507084	2835694

ScalaSparkSort	328492566	38.899	8444756	2814918
PythonSparkSort	328492566	39.589	8297571	2765857
HadoopWordcount	2204463831	36.156	60970899	20323633
JavaSparkWordcount	2204463831	45.904	48023349	16007783
ScalaSparkWordcount	2204463831	42.498	51872178	17290726
PythonSparkWordcount	2204463831	55.586	39658616	13219538
HadoopBayes	575056284	43.703	13158279	4386093
JavaSparkBayes	575056284	153.322	3750644	1250214
ScalaSparkBayes	575056284	51.662	11131127	3710375

Table 4: Numerical results from host-based benchmarking.

Workload	Input Data Size	Duration (s)	Throughput (B/s)	Throughput per node (B/s)
HadoopAggregation	37276711	78.142	477038	59629
JavaSparkAggregation	37276711	60.269	618507	77313
ScalaSparkAggregation	37276711	63.109	590674	73834
PythonSparkAggregation	37276711	60.709	614024	76753
HadoopJoin	1000350	110.542	9049	1131
JavaSparkJoin	188832410	74.725	2527031	315878
ScalaSparkJoin	194078124	74.137	2617830	327228

PythonSparkJoin	194078124	74.402	2608506	326063
HadoopKmeans	4016371702	617.808	6501003	928714
JavaSparkKmeans	4016371702	323.785	12404440	1550555
ScalaSparkKmeans	4016371702	107.479	37368897	4671112
PythonSparkKmeans	4016371702	507.817	7909092	988636
HadoopPagerank	259928115	378.638	686481	85810
JavaSparkPagerank	259928115	160.677	1617705	202213
ScalaSparkPagerank	259928115	202.571	1283145	160393
PythonSparkPagerank	259928115	251.692	1032722	147531
HadoopScan	184795601	93.462	1977227	247153
ScalaSparkScan	184796438	58.220	3386534	423316
PythonSparkScan	184796438	57.726	3201268	400158
HadoopSleep	0	24.694	0	0
JavaSparkSleep	0	110.888	0	0
ScalaSparkSleep	0	111.025	0	0
PythonSparkSleep	0	127.452	0	0
HadoopSort	328493084	36.558	8985532	1497588
JavaSparkSort	328493084	30.920	10623967	1770661

ScalaSparkSort	328493084	38.008	8642735	1440455
PythonSparkSort	328493084	32.069	10243321	2560830
HadoopWordcount	2204473443	105.248	20945513	2618189
JavaSparkWordcount	2204473443	52.221	42214309	5276788
ScalaSparkWordcount	2204473443	39.189	56252352	7031544
PythonSparkWordcount	2204473443	77.069	28603893	3575486
HadoopTerasort	3200000000	127.281	25141222	3142652
JavaSparkTerasort	3200000000	64.074	49942254	6242781
ScalaSparkTerasort	3200000000	62.601	51117394	6389674

Table 5: Numerical results from VM-based benchmarking.

Figure 3 and Figure 4 visualise the duration and aggregate I/O throughput of each micro workload. These bar charts compare the durations for the same experiment as run on the directly on the host and in VMs. Almost all experiments take a longer time when running in VMs. Most micro workloads, show a relatively small overhead with VMs. We assume that these workloads are CPU-bound. The performance impact of running a VM is negligible in these cases, since computation in VMs is hardware-assisted and highly optimized. Some experiments take slightly less time when using VMs. This effect presumably stems from compute-bound micro-workloads and from an empirical variance. To clarify this initially counterintuitive results we will analyse the results in more detail in future. For this analysis, we will re-run the experiments a number of times and perform a statistical analysis. Furthermore, we will augment the results by providing detailed performance data on the running VMs and hosts. The statistical analysis requires additional time to perform more experimental runs and to perform the analysis. Getting performance data requires the monitoring integration to be set up.

The clearly increased durations for running k-means and computing the PageRank are striking in the results. We assume that these two algorithms are I/O-bound and that they can benefit clearly from I/O optimisation. The scattered resources for the VMs is another features that may lead to the increased durations. Whereas in the host-based experiments the system loads the data all in a huge shared memory space, in the VM-based scenario there are multiple smaller memory spaces. Thus on the same host sharing data between VMs may incur an additional overhead, which may be overcome by using RDMA for inter-VM communication with host-based shared memory.

Finally one striking result for the duration metric can be seen in the Scala-based implementation of k-means. Whereas the other implementations of k-means take a significantly longer time to finish, the Scala-based implementation is done even quicker on VMs than on the host. This result is surprising and warrants additional investigation.

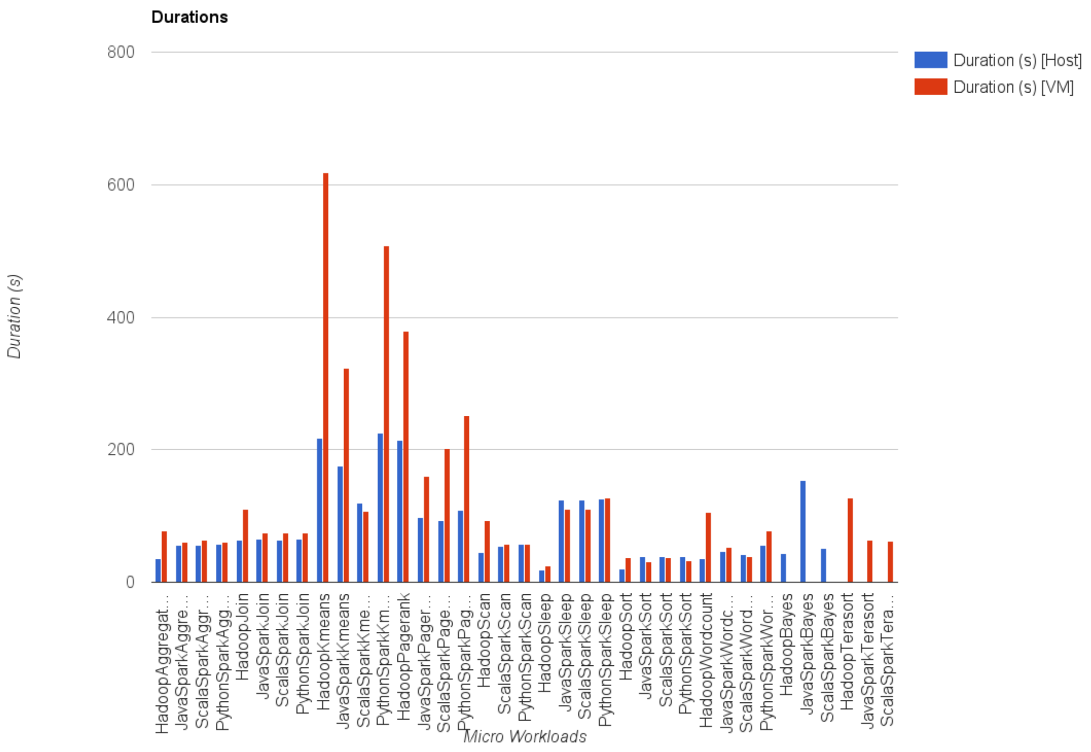

Figure 3: Comparison of benchmarking duration between host-based and VM-based runs.

The results for aggregate throughput in Figure 4 support the hypothesis that the implementations of k-means are I/O-bound. The results show a high aggregate throughput for all implementations of k-means. The Scala-based implementation has the highest throughput, which is higher in the VM-based experiment. For k-means the results for durations correlate with the results for aggregate throughput. This leads us to believe that the algorithm is I/O-bound. PageRank, which has significantly higher durations for the VM-based experiments, has relatively low aggregate I/O throughput. This may be due to a high frequency of small transfers or due to the algorithm not being I/O-bound. However, I/O latencies in VMs may lead to an increase of duration even if not much data is transferred.

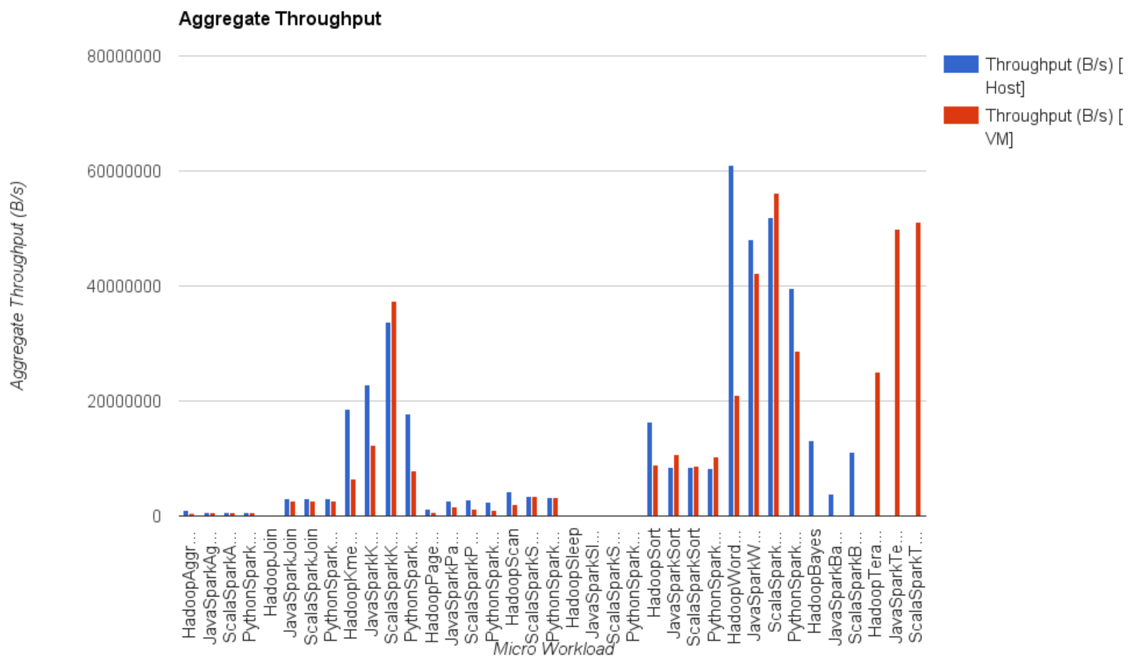

Figure 4: Comparison of aggregate throughput between host-based and VM-based runs.

The aggregate throughput of WordCount strikes as the workload with highest throughput. Again the throughput correlates nicely with the durations. The higher the throughput, the lower the duration. Similar to the k-means experiments the Scala-based implementation seems atypical, since it achieves a higher aggregate throughput and lower duration than the host-based version. Finally Bayesian classification and TeraSort show a negative correlation between duration and aggregate throughput as well. Thus, we assume that they may offer interesting results, as in decreased VM-based performance, once we get them to run fully.

Conclusively our experiments hint at several workloads, which may be I/O-bound and thus a target for the optimisations in sKVM and OSv. In the next setups, we will investigate the execution of the micro-workloads further, execute them on sKVM and OSv, and compare the results to the obtained baseline.

6.3 Evaluation Goals of Future Experiments

Missing KPIs will be evaluated in later project stages since they require further progress first. The current state of the use case only covers core KPIs that relate mostly to I/O efficiency. The initial experiments only cover baseline measurements and thus only the first part of the core KPI values. In this section we briefly discuss how we envision to monitor the progress on those metrics. The work to be carried out in future will have an impact on all fourteen of the project's KPIs. We briefly list how future extensions for the benchmarking part and for the mini use-cases will augment the measurement of the KPIs.

6.3.1 Extensions to the Benchmarks

The benchmarking part will be extended to include the MIKELANGELO stack, and potentially benchmarks for cloud bursting. The integration of MIKELANGELO components will allow us to evaluate those components with regards to I/O efficiency and complete the baseline of related KPIs (***KPI2.1, KPI4.2***). For extension the existing benchmarks can be re-used the way that they are. They just have to be installed in OSv as far as possible. Once the installation of big data services in OSv has been tested, ***KPI3.3*** can be evaluated by the compatibility of OSv with the big data services. ***KPI5.1*** will be tested automatically in future by running big data experiments on more machines, which will include heterogeneous hardware. Furthermore, in case that we will extend the benchmarks to cover bursting scenarios, the experiments would also cover ***KPI1.1, KPI3.2.***

6.3.2 Implementation of the Mini Use-Cases

The mini use-cases will cover all of the KPIs like the benchmarks and even extend the results for some KPIs. To cover bursting, additional software components and application packaging will allow the mini use-cases to provide deeper insights with respect to the KPIs.

Covering mini uses-cases with explicit bursting experiments will allow the use case to monitor ***KPI1.1, KPI3.2, KPI4.2.*** To facilitate the bursting experiments two extensions to the normal model of operation of the mini use-cases will be required. First, the time to set up the big data clusters needs to be measured. Second, there needs to be a parameter sweep across different cluster configurations and infrastructure load scenarios. In the mini use-cases we expect to cover a wealth of big data services that goes beyond the components tested with the benchmarks. This will allow us to give a better informed estimation for ***KPI3.3.*** Finally, for the mini use-cases there needs to be a high degree of automation that will require application management to roll out clusters on demand. Thus, the mini-use cases will allow to measure ***KPI4.1*** and ***KPI4.2.***

Conclusively, the foundations to measure KPIs have been laid out by the benchmarking part of the use case. Currently, the benchmarking part provides the baseline measurements for the core KPIs that relate to I/O efficiency. The benchmarks will be extended in future to cover most KPIs, probably with the exception of KPIs related to application packaging and bursting. Finally, the mini use-cases will be used to provide richer information to extend some KPIs and provide all measurements for others.

7 Conclusions

This report described the initial implementation strategy for the virtualised big data use case of the MIKELANGELO project. This use case will showcase the advancements achieved by MIKELANGELO for big data applications.

The use case aims to leverage MIKELANGELO's advancements in I/O efficiency of VMs and application management to run big data applications on virtual infrastructures. Besides integrating components from other MIKELANGELO work packages, this use case will extend existing big data middleware to run big data clusters on demand in a cloud. The goal of this use case is to reach a state at the end of the project that will allow other resource providers to deploy a self-service big data service on their cloud as well.

This report describes the use case itself according to its two main stages. These stages are the benchmarking stage and the mini use-cases stage. The description of the use case consists of a description of hardware and software infrastructure, a description of data sets, a description of mandatory requirements, and a description of how the use case relates to the project's key performance indicators. Furthermore, we provide an implementation plan for the benchmarks and the mini use-cases. For the benchmarks this report offers some more detailed descriptions as they will serve as initial baseline measurements.

The already performed work on the big data use case has led to a number of tangible results. First, this report presents an overview of the state of the art with regards to big data benchmarks. These form a basis for our first choice, HiBench, as big data benchmark. Second, we have run HiBench on a test bed to establish a baseline to be used for evaluation of MIKELANGELO's new developments. The data generated in these experiments can be reviewed in MIKELANGELO's open data repository. Third, we have formed an implementation plan for the period M9-M36 for the big data use case. This implementation plan relies on a requirements analysis and on the analysis of the project's KPIs. The implementation plan specifies the required features for the benchmarking part and for the mini use-cases.

This deliverable provides the most detailed description of the virtualised big data use case so far. This description recaps the current state of the implementation and an implementation plan to be executed during the next project stages. Conclusively, the use case has seen most work on the conceptual side and progresses smoothly into the implementation phases.

A References and Applicable Documents

[1] https://github.com/intel-hadoop/HiBench (retrieved on 25.8.2015)
[2] https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench
[3] http://prof.ict.ac.cn/BigDataBench/
[4] https://amplab.cs.berkeley.edu/benchmark/
[5] http://www.tpc.org/tpcx-hs/
[6] https://github.com/brianfrankcooper/YCSB/wiki
[7] http://parsa.epfl.ch/cloudsuite/cloudsuite.html
[8] https://github.com/facebook/linkbench
[9] https://cwiki.apache.org/confluence/display/PIG/PigMix
[10] https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[11] https://mesos.apache.org/
[12] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html
[13] http://ceph.com/
[14] https://cassandra.apache.org/
[15] https://hadoop.apache.org/
[16] https://spark.apache.org/
[17] https://www.elastic.co/
[18] https://lucene.apache.org/solr/
[19] https://wiki.debian.org/Apt
[20] https://puppetlabs.com/
[21] http://www.ansible.com/home
[22] https://hive.apache.org/
[23] https://pig.apache.org/
[24] https://www.openstack.org/
[25] https://wiki.openstack.org/wiki/Sahara
[26] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011).
[27] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3456
[28] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0983
[29] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0217
[30] http://www.linux-kvm.org/
[31] http://releases.ubuntu.com/14.04/
[32] D2.19 – The first MIKELANGELO architecture
[33] https://jenkins-ci.org/
[34] https://github.com/cloudius-systems/osv-apps/