MIKELANGELO

D6.1 First report on the Architecture and Implementation Evaluation

Workpackage	6	Infrastructure Integration	
Author(s)	John Kennedy, Marcin Spoczynski		INTEL
	Gregor Berginc, Daniel Vladušič		XLAB
	Uwe Schilling, Carlos Diaz, Nico Struckmann		USTUTT
	Peter Chronz, Maik Srba		GWDG
	Shiqing Fan		HUA
	Benoît Canet, Nadav Har'El		SCYLLA
	Niv Gilboa, Gabriel Scalosub		BGU
	Matej Andrejašič		PIPISTREL
Reviewer	Nico Struckmann		USTUTT
Reviewer	Gregor Berginc		XLAB
Dissemination Level	Public		

Date	Author	Comments	Version	Status
7 Mar 2016	John Kennedy	Initial structure	V0.0	Draft
27 June 2016	All WP6 Partners	Initial draft for review	V0.1	Review
30 June 2016	All WP6 Partners	Final version	V1.0	Final

Executive Summary

The MIKELANGELO project [1] seeks to improve the I/O performance and security of Cloud and HPC deployments running on the OSv [2] and sKVM [3] software stack. The project has now reached its halfway point. Architectures for all components have been prepared, and implementations for most are now available.

An evaluation of the individual components, integrated stacks, use cases and the development workflow of MIKELANGELO is presented in this document. The emphasis is on reviewing their architecture and implementation.

No unexpected architectural issues have been identified. Standalone testing of implementations of individual components has revealed that they deliver the performance improvements expected.

The integration of the Cloud and HPC stacks, and evaluation through the Cloud Bursting, HPC OpenFOAM [4] and Big Data use cases, has also revealed improvements in performance in various scenarios. However, these efforts have identified various gaps in functionality, and a number of locations and configurations of the MIKELANGELO stack that do not yet deliver the expected performance. Missing functionality has already been added. Detailed analysis of performance issues has begun and has already helped identify bugs and bottlenecks that, once addressed, are transforming the initial results. Several of the MIKELANGELO components are already live open-source projects and MIKELANGELO-developed enhancements are being continuously upstreamed and released publically.

The evaluation activities outlined in this deliverable have helped confirm, and drive, significant progress to-date. The ongoing evaluation efforts will continue to play a key role in identifying new requirements and opportunities for enhancements that will help maximise the impact of the project.

Acknowledgement

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

Table of contents

Table of Figures

Table of Tables

1 Introduction

The MIKELANGELO project seeks to improve the I/O performance and security of Cloud and HPC software running on top of the OSv and sKVM software stack.

The architecture of the MIKELANGELO project at Month 18 of the project is documented in Deliverable D2.20, The Intermediate MIKELANGELO Architecture [5].

This document presents an evaluation of this architecture and its implementation to date.

The technical architecture and the current implementation of all components of the MIKELANGELO stack are evaluated in Chapter 2. The architectural approach of the MIKELANGELO components are contrasted with alternatives in the marketplace. Both benefits and limitations of the MIKELANGELO approaches are discussed. Where relevant, Best Known Methods are described. This chapter also describes the results of functional testing and benchmarking where possible, and considers planned or potential enhancements and their expected significance.

The individual components evaluated in Chapter 2 are designed to complement each other when combined in full stacks for Cloud and HPC deployments. These full stacks are considered and evaluated in Chapter 3.

Putting it all to work, the overall architecture is evaluated in Chapter 4 by examining several Case Studies that have each adopted and exercised a selection of MIKELANGELO components. The experiences from three deployments are reviewed. They cover Cloud Bursting, an OpenFOAM Cloud HPC scenario, and Big Data leveraging Hadoop HDFS [6] and Apache Storm [7].

Whilst the previous chapters evaluate the technology developed by MIKELANGELO, the development workflow designed and adopted by the project is itself examined in Chapter 5.

Chapter 6 gathers the overall observations on the architecture and implementation evaluation at this stage of the project, and also describes current priorities.

Chapter 7 provides some concluding remarks, and references are provided in Chapter 8.

2 Component Evaluation

2.1 Introduction

This chapter evaluates the technical architecture, performance and the current implementation of individual components of the MIKELANGELO stack. The components considered are:

- Linux Hypervisor IO Core Management - sKVM's IOcm
- Virtual RDMA - sKVM's virtual RDMA
- Unikernel Guest Operation System - OSv
- Application Package Management - MPM
- Monitoring - snap [8]
- Hosted Application Acceleration - Seastar [9]
- Side Channel Attack Mitigation - sKVM's SCAM

2.2 Linux Hypervisor IO Core Management

2.2.1 Architectural Evaluation

In the current implementation of KVM, each virtual device gets its own vhost thread. This is a very simple programming model since threads are a convenient abstraction, but not necessarily the most efficient. In essence, as the number of virtual machines increases, so does the number of virtual devices, and in turn the number of vhost threads. At some point, all of these threads start to affect each other, and the overhead of switching between them gets in the way of the threads doing useful work.

One idea that has been proposed to address this issue from an architectural point of view is to use shared vhost threads. It turns out that sharing a vhost thread among multiple devices can reduce overhead, and improve efficiency. Moreover, each shared vhost thread occupies a core for the sole purpose of processing I/O. To further reduce the contention between the threads, we disallow the virtual machines to share the cores with vhost threads. This approach is described and evaluated in the ELVIS paper [10]. One major drawback of ELVIS is its inability to dynamically adjust the number of cores according to the current workload.

We took upon ourselves to enhance ELVIS with a mechanism to modify the number of shared vhost threads at run-time. Determining the optimal number of shared vhost threads is done automatically at run-time by the I/O manager. The I/O manager is a user-space application which continuously monitors the system, and adapts the number of shared threads according to the current CPU load.

Next we evaluate our prototype with the aforementioned architectural enhancement.

2.2.2 Performance Evaluation

Our test system is comprised of two physical machines: a load generator and a machine that hosts the Virtual Machines (VMs). Both machines are identical and of type IBM System x3550 M4, equipped with two 8-core sockets of Intel Xeon E5-2660 CPU running at 2.2 GHz, 56GB of memory and two Intel x520 dual port 10Gbps NICs. All machines run Ubuntu 14.04 with Linux 2.18 (guests, host, and load generator) [11]. The host's hypervisor is KVM [12] with QEMU 2.2 [13]. To minimize the benchmarking noise, hyperthreading and all power management features are disabled in the BIOS.

The machines are connected in a point-to-point fashion as depicted in Figure 1.

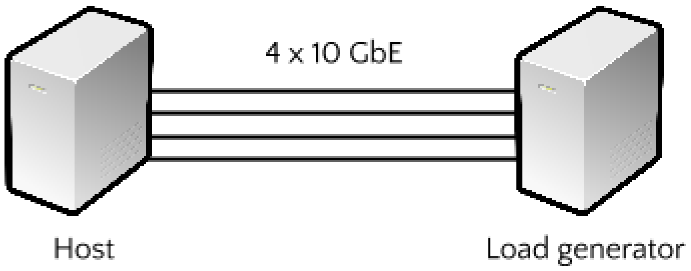


Figure 1. IO Core Management test system setup.

Each experiment is executed 5 times, for 60 seconds each. We make sure the variance across the results (both performance and CPU utilization) is negligible, and present their average. Benchmark parameters were meticulously chosen in order to saturate the vCPU of each VM.

The experiment evaluates the performance of three basic configurations:

- **baseline** We use KVM virtio as the state-of-practice representative of paravirtualization. We denote it as the baseline configuration
- **elvis-X** Our modified version of vhost, with a different number of dedicated I/O cores (1-4), denoted by X
- **io-manager** Our modified version of vhost driven by the I/O manager which automatically adjusts the number of I/O cores in response to the current load

With all configurations, we set the number of VMs to be 12[1] (overcommit) throughout all the benchmarks, utilizing only one 8-core socket. Each VM is configured with 1 vCPU, 2GB of

[1] We are mostly interested in the first four elvis-X configurations. To achieve balanced results, each I/O core is assigned equal number of VMs. 12 is evenly divided by 1, 2, 3 and 4.

memory and a virtual network interface. All four physical ports are in use and assigned evenly between VMs. The NICs are connected to the VMs using the standard Linux Bridge.

With the "elvis-X" configuration, we vary the number of I/O cores from 1 to 4 at the expense of available VM cores. Given a number of I/O cores, the VMs are assigned in a cyclic fashion to the remaining cores. For the "baseline" setup, there is no affinity between activities and cores, namely, interrupts of the physical I/O devices, I/O threads (vhost), and vCPUs.

The experiment is executed using two workloads: Netperf [14] and Apache HTTP Server [15].

Netperf Our first experiment evaluates a throughput-oriented application. We use the Netperf TCP stream for this purpose, which measures network performance by maximizing the amount of data sent over a single TCP connection, simulating an I/O-intensive workload. We vary the message size between 64 and 16384 bytes. Similar results are obtained for messages larger than 16KB.

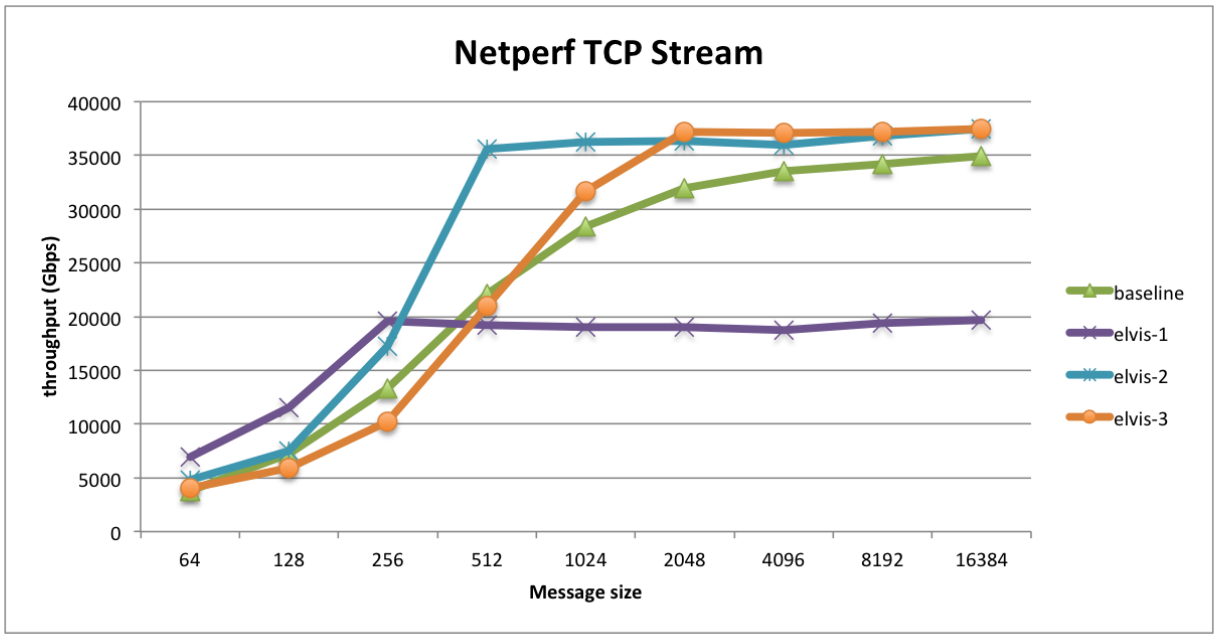


Figure 2. Performance evaluation of elvis-X for netperf TCP stream.

With elvis-X configurations, each additional I/O core comes at the expense of the cores that are available for running VMs. For example, elvis-4 dedicates 4 I/O cores and only 4 cores are shared among the 12 VMs. In the graph above we can see that elvis-3 underperforms elvis-1 for messages smaller than 1024 bytes, as the latter configuration allows 7 cores for the virtual machines while the I/O core is not saturated.

Naturally, an I/O core has a limit to the amount of traffic it can handle in a given period. For elvis-X, we can see the throughput curves become flatter at a certain point as message size increases. In elvis-1, the I/O core is saturated with the smallest message size, while for elvis-2 both I/O cores reached their maximum capacity with a message size of 512 bytes.

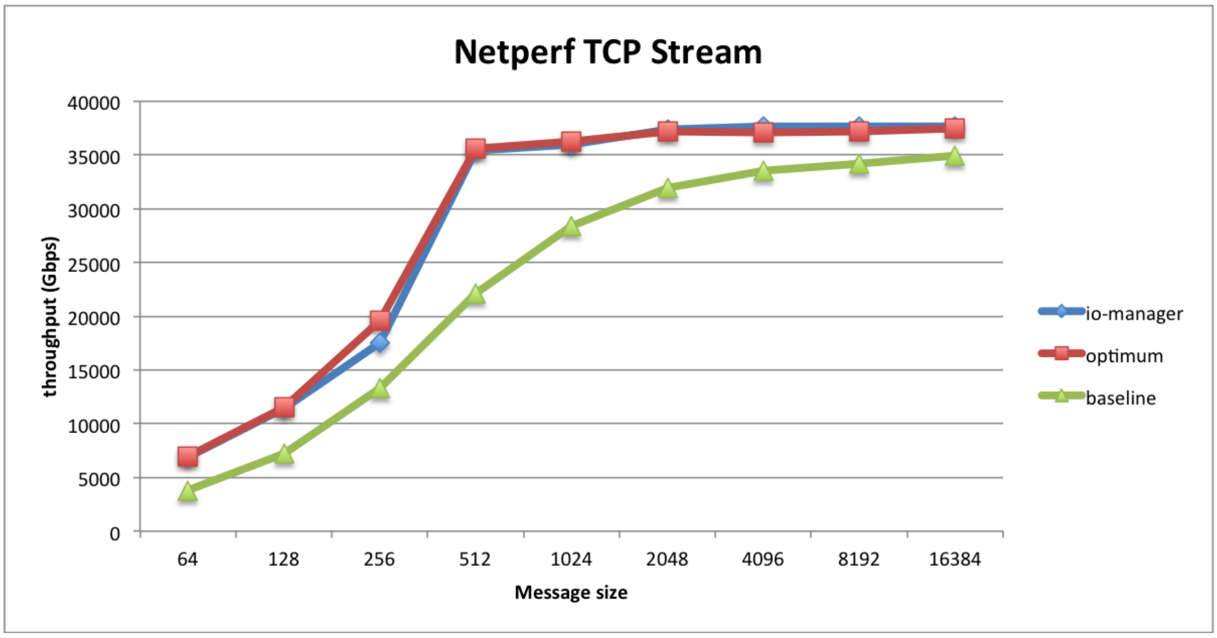


Figure 3. Performance evaluation of io-manager for netperf TCP stream.

The figure above presents the baseline result alongside the best of elvis-X configurations, depicted as “optimum”. Additionally, we present our automatic I/O manager (denoted by io-manager) which switches between elvis-X configurations based on the current state of the system.

Apache HTTP Server To evaluate the performance on a real application, we use the Apache HTTP Server. We drive it using the ApacheBench [16] (also called “ab”) which is distributed with Apache. It assesses the number of concurrent requests per second that the web server is capable of handling. We use 16 concurrent requests per VM for different file sizes, ranging from 64 bytes to 1 MB. The results are shown in the following figure.

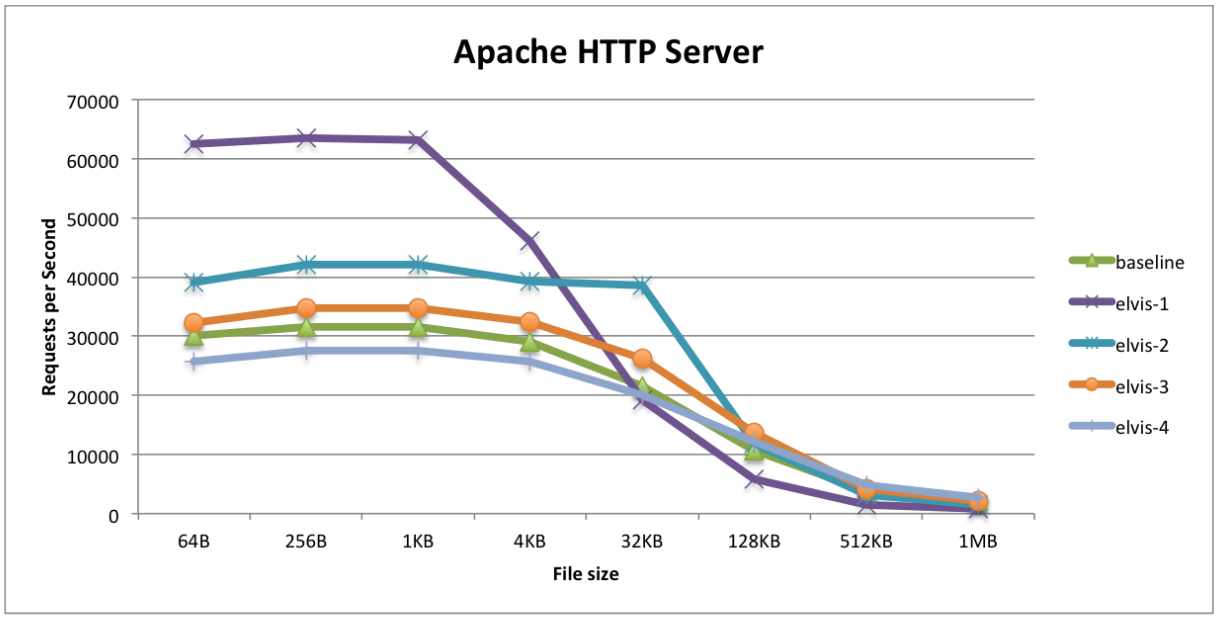


Figure 4. Performance evaluation of elvis-X for Apache HTTP server.

In baseline, KVM allocates one I/O thread per virtual device and one thread per VCPU. Thus, 24 threads compete for the available CPU cores. This contention increases the latency and is most acute when using small files as there are more requests per second.

For elvis-X configurations, instead of 12 I/O threads, only X threads are allocated and run on separate cores. This reduces the contention and improves the latency. From the above graph it is clear that elvis-1 outperforms the baseline for smaller requests, as latency is more dominant when requesting small files. However, all configurations converge as we increase the request size as now it becomes more stream oriented, thus hiding the latency with concurrent requests.

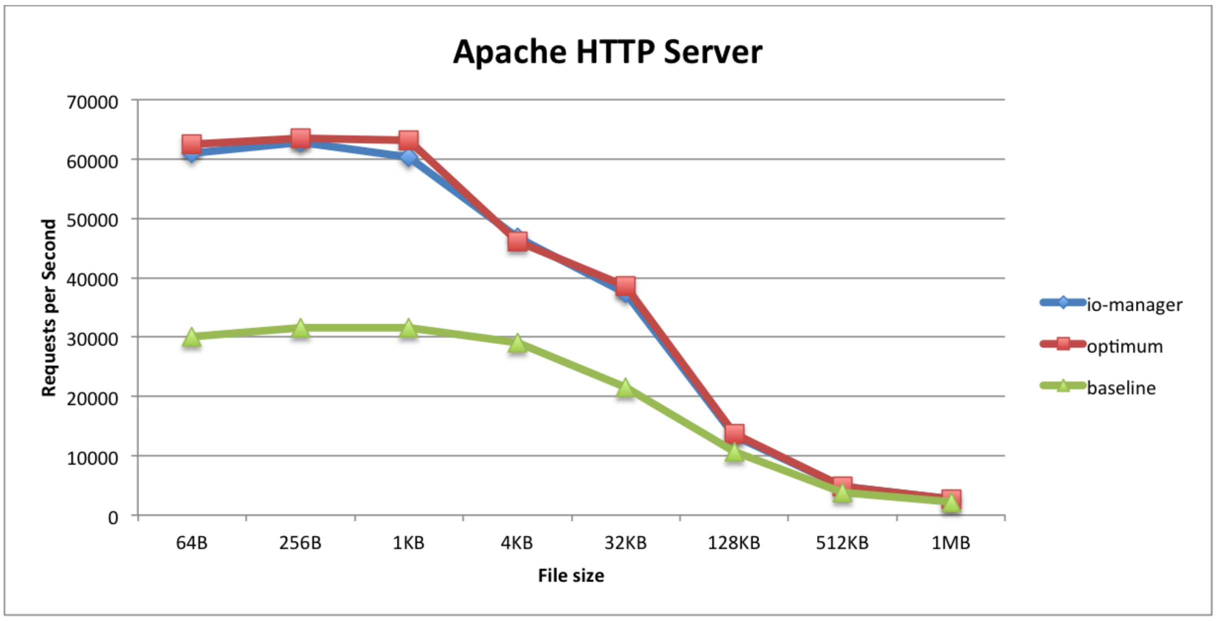


Figure 5. Performance evaluation of io-manager for Apache HTTP server.

Similar to netperf, we present the baseline result for Apache compared to the io-manager, bounded by the optimum (Figure 5). This again shows the need for dynamic monitoring and management of cores reserved for I/O operations based on the current workload.

2.2.3 Implementation Evaluation

The implementation is available from the MIKELANGELO git repository [17], and the latest code will be published as part of the M18 deliverable. The kernel portion does not have a unit test per se, since the normal procedure for the Linux kernel is to test once packages are integrated, and not test them standalone. We test the code through a series of benchmarks, which exercises various code paths for different packet types (virtio-net and virtio-scsi devices). The user space portion is a set of scripts that monitor the resource utilization and configure the cores through sysfs and will also be made available as part of the M18 source code release. We can show that these scripts are working as expected by comparing the throughput of the system at a given moment to what can be attained with a static ELVIS configuration with the same workload.

2.3 Virtual RDMA

2.3.1 Architectural Evaluation

Virtual RDMA prototype I is targeted at supporting a socket API for guest applications and uses the DPDK RDMA Poll Mode Driver (PMD) [18] directly on the host, which is integrated with Open vSwitch [19]. It has the simplest implementation of the three Virtual RDMA prototypes (details in Deliverable D2.13, The first sKVM hypervisor architecture [5]), as most of the modules in this case are offered as open source projects that can be directly used and integrated.

Open vSwitch uses DPDK to create a central bridge, which then connects the physical port on the host to a virtual port for the guest. No modification for the guest is necessary, and the user application will be able to use the virtio-net device to communicate to the virtual port on the host. Open vSwitch runs a daemon in the background to poll the virtual port and convert communications into the format that can be processed by the physical NIC driver. In the case of prototype I, the communication starts from the guest socket interface, and then it is converted to the RDMA format by the DPDK Poll Mode Driver. Finally, the converted communication is processed by the physical NIC driver.

The advantages of prototype I are: the guest OS and the guest application can be directly used without modification; communication through the RDMA channel is fast with lower latencies; implementation and integration are straightforward; with the setup of hugepage, shared memory communication is enabled by default for inter-VM communication on the same host; communication performance is better than the traditional virtio-net interface.

The disadvantages of prototype I are: at least one extra CPU core has to be occupied by the Open vSwitch daemon (DPDK Poll Mode Driver); the user may not be able to tune the communication easily, as the actual communication is handled by the DPDK Poll Mode Driver, which provides only very basic tuning options; Configuration of the environment for different systems may differ, this involves additional effort for integration; any misconfiguration or wrong deployment in each layer of the architecture by the system administrator may cause the entire environment to collapse.

For further implementation, we will work mainly on prototype II. Based on the results of prototype II, we will be able to compare and evaluate these two prototypes.

2.3.2 Performance Evaluation

Performance has been reviewed using OpenFOAM on the USTUTT testbed and an internal testbed. The infrastructure details of the USTUTT testbed has been described in Deliverable D2.19, The first MIKELANGELO architecture [5]. The host CPU is a Intel® Xeon® Processor CPU X5560 running at 2.80GHz. It has 8 CPU cores on two NUMA nodes. Because this particular CPU does not support 1GB hugepage, required by Open vSwitch and the guest OS, a series of contiguous 2MB (maximum allowed size) pages are used as an alternative solution.

Figure 6 shows the OpenFOAM test case with particular duration of around 15 minutes on the USTUTT testbed using 2, 4 and 8 parallel processes on two VMs. The VMs are on two different hosts connected with Ethernet and InfiniBand [20] ConnectX-3 cards. Each VM is pinned to 4 physical CPU cores. The performance results show that on Ubuntu guests, using virtual RDMA, prototype I is about %3 to 4% faster than the normal bridge interface. For OSv guests, all VMs are configured to use memory on the same NUMA core as they are pinned to, in order to avoid performance impacts due to the cross NUMA node constraints of OSv as described in Deliverable D2.20 The Intermediate MIKELANGELO Architecture [5]. But for virtual RDMA prototype I, an additional Open vSwitch daemon is required to be correctly assigned on a single CPU core that has fast local memory access with the NIC for better performance. Under this circumstance, it is not possible to avoid accessing memory across the NUMA nodes. If we put both Open vSwitch daemon and VM on the same NUMA core, then the entire NUMA node will be overburdened and the performance cannot be improved in such a case. With the same configuration as Ubuntu guest, tests on OSv have to always access the memory on both NUMA cores (Open vSwitch on NUMA node 0, and OSv on node 1). This remains a further focus to be evaluated and resolved in OSv, i.e. to break the limit of using memory from different NUMA core as efficiently as on Ubuntu.

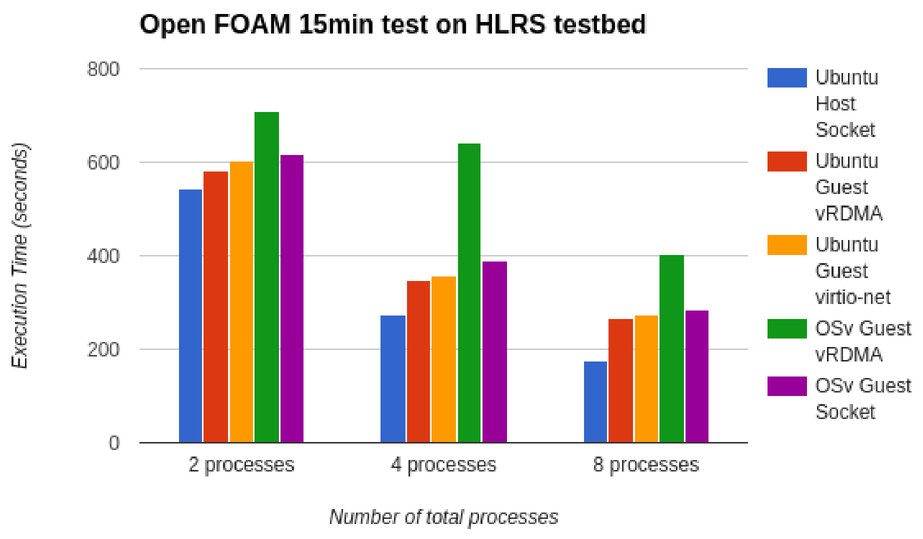


Figure 6. OpenFOAM 15 minute test with 2, 4 and 8 processes on Ubuntu and OSv.

In Figure 7, we show the results of the OpenFOAM test case with particular duration of around one hour on the USTUTT testbed, where using virtual RDMA prototype I has a 3% to 7% improvement of the overall execution time. As OpenFOAM is not an application that generates heavy communication between the workers, the performance results may not show the real improvements that it benefits from virtual RDMA. More benchmarking results were presented in Deliverable D4.1 The First Report on I/O Aspects [5], where it shows about 25% improvement.

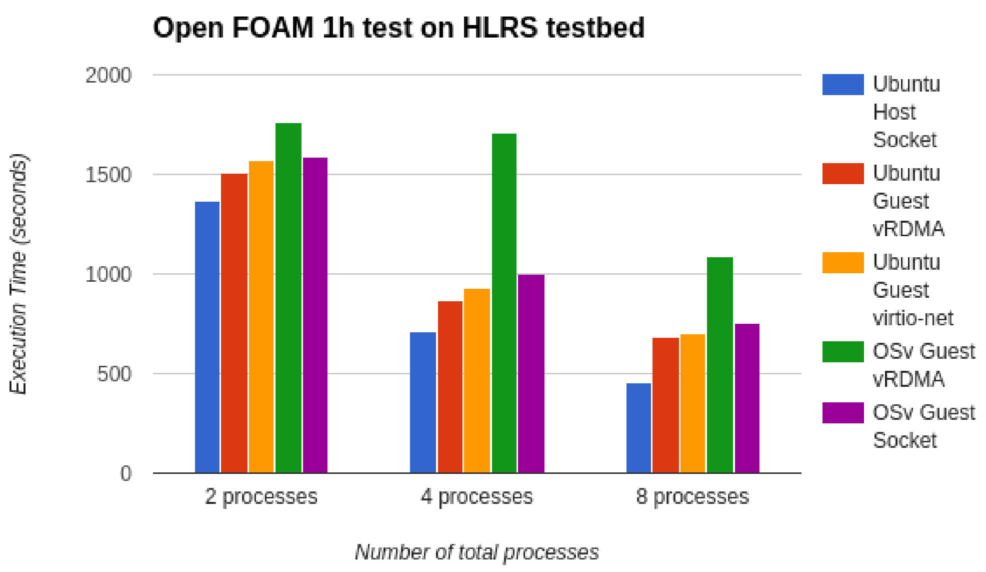


Figure 7. OpenFOAM 1 hour test with 2, 4 and 8 processes on Ubuntu and OSv.

Similar tests with OpenFOAM have been done on a local testbed at HUAWEI. The hosts are HP ProDesk 600 each with a 4-core Intel® Core™ i7-4790 Processor, and 16GB RAM memory with the host OS being Ubuntu 14.04 LTS server edition.

The local testbed system consists of two servers, each running 2 virtual machines. These two virtual machines are able to communicate through Ethernet or InfiniBand network interconnects between the hosts, or through shared memory inside the host. CPU core 2 and 3 are configured in Grub parameters to be isolated when the system boots, in order to make sure that only assigned VM processes will be able to run on the isolated cores. Eight 1GB hugepages are created and mounted for running the vhost-user configuration within Open vSwitch and DPDK Poll Mode Driver.

Figures Figure 8 and Figure 9 show the test results with two and four processes for OpenFOAM with particular durations around 15 minutes and one hour respectively. The performance of these tests show that using virtual RDMA prototype I will improve the performance by 7% to 10% compared to using virtio-net.

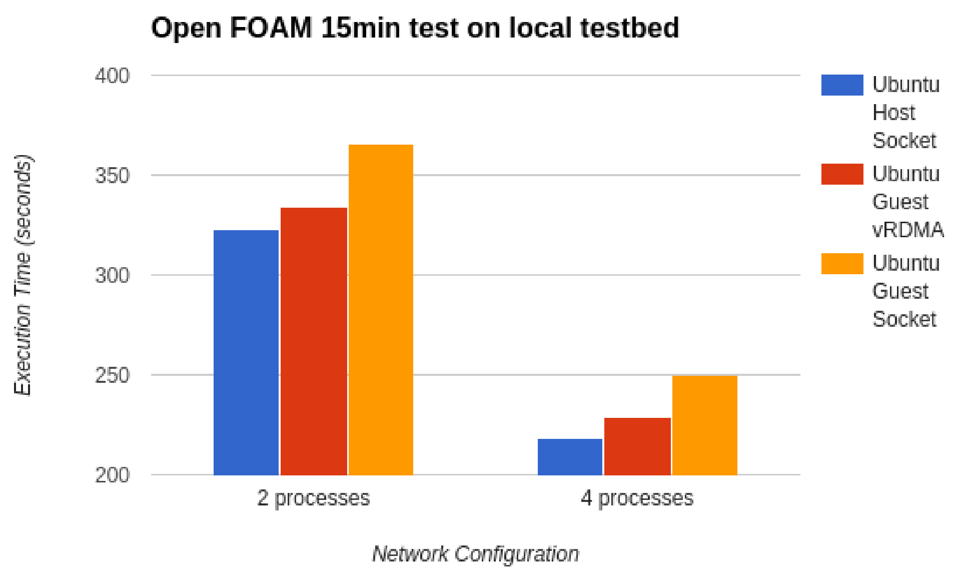


Figure 8. OpenFOAM 15 minute test with 2 and 4 processes on Ubuntu (local testbed).

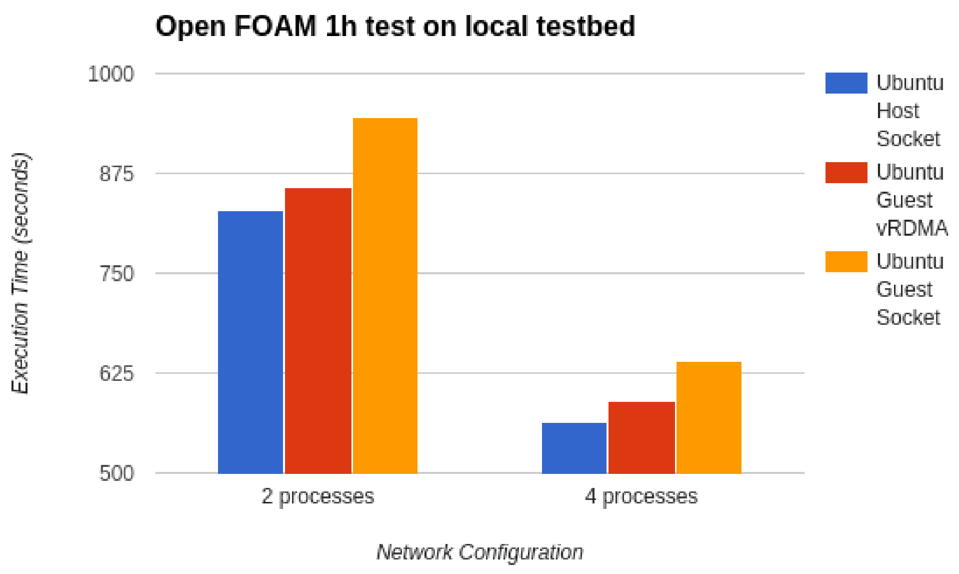


Figure 9. OpenFOAM 1 hour test with 2 and 4 processes on Ubuntu (local testbed).

2.3.3 Implementation Evaluation

Virtual RDMA prototype I has been implemented and shell scripts for configuring it on the host have been integrated with Torque on the HPC infrastructure at USTUTT. When the user submits or terminates a job, the corresponding scripts will be automatically started and the environment for the job will be created or destroyed. This integration has been pushed to the

MIKELANGELO project repository and will be published in the project's M18 release. The fundamental code that is prepared for further prototype II implementation has been pushed to the MIKELANGELO project repository.

Performance of the first prototype has been tested and evaluated with several benchmarks and use cases. Comparing with the traditional virtio-net, using prototype I will gain noticeable performance improvements in most of the tests.

The implementation of virtual RDMA prototype II has started. The implementation is based on Hyv [21], which was based on Linux kernel 3.13. The gap between kernel 3.13 and the targeting kernel 3.18 is huge, which makes it difficult and time consuming to get the initial version ready. Most of this work has been accomplished, and the rest of the implementation work will be finished shortly.

2.4 Unikernel Guest Operating Systems

2.4.1 Architectural Evaluation

The Guest Operating System for the MIKELANGELO architecture is unikernel-based [22]: the application runs in a virtual machine on top of a small and efficient Linux (POSIX) compliant kernel (OSv). This contrasts with recent market trends where containers are receiving a lot of attention. In containers, the host kernel resources are segmented in order to isolate each hosted application. The obvious drawback of containers compared to virtual machines and unikernels is the huge attack surface of the host kernel: a unikernel being run in a well audited virtual machine offers less holes for an attacker to leverage because the virtual machine hardware is small and scrutinized.

OSv is the C++11 unikernel used in the MIKELANGELO project. The main OSv differentiator compared to alternatives (such as ClickOS, Clive, Drawbridge, JaLVM, IncludeOS, LING, Mirage, Runtime.js and Rump Run [22]) is that OSv is intended to be a comprehensive alternative to the Linux kernel, supporting all existing Linux applications and multiple hypervisors. OSv acts as an almost drop-in GNU/Linux replacement optimized for virtual machine hardware. The unikernel closest to OSv is Rump Run due to its NetBSD-based sources that make it akin to a real operating system and not a simple library. IncludeOS is another example of a unikernel that claims to run existing Linux software, but it actually only implements a very limited subset of Linux, reducing its compatibility significantly.

The main benefits of a unikernel are revealed in a context where the dev-op team industrialize and streamline their virtual machine image construction. A combined unikernel and application resulting in a virtual machine artifact must be considered as a cloud-like process once executed in a virtual machine. The initial industrialization steps of using a unikernel are steeper than using a container because it forces the dev-op team to rethink their virtual machine construction processes but in the end the result is much cleaner than the traditional approach of relying on an unwieldy, difficult to maintain, collection of shell scripts.

Users of a unikernel like OSv cannot make use of multiple processes in a single virtual machine instance. One of the consequences is that shell scripts that inherently use process forking are unavailable in OSv. A high level REST API is thus provided by OSv allowing management of the entire lifecycle of the OSv instance and the application. One of the observations has been that this slightly limits the adoption of OSv: existing tools typically need to be reconsidered and rewired to the alternative interface and, consequently, an investment must be made into adapting existing processes.

Some challenges were encountered when employing OSv in MIKELANGELO, but these have all been addressed by the OSv team:

NFS [23] is currently used in a HPC context to get data in and out of the compute node. To support this MIKELANGELO has ported a basic NFS client to the OSv kernel. It builds as an additional option and is released under an LGPL license. Two additional OSv commands mount-nfs.so and unmount.so were added so the user can easilly mount their share. This NFS client implements NFS V3. The OSv architecture allowed the porting of NFS to be performed quickly. A description of the porting process has been documented online [24].

To simulate UNIX processes in Open MPI [25], an additional mechanism to isolate the memory space of a thread has been added to OSv. The isolation characteristics of these thread namespaces are somewhat weaker than UNIX processes, but they allow users to successfully run Open MPI payloads like OpenFOAM.

Some HPC payloads (such as provided by USTUTT) need to be linked with a configuration file generated by the rest of the infrastructure, without using NFS. With a Linux guest the classical method is to create an ISO9660 image [26] and pass it to the guest. Since only one configuration file is needed, tools have been created in MIKELANGELO to embed this configuration file in a raw image thus saving MIKELANGELO the overhead of porting a complete CDROM file system to OSv. The first tool is used to bake the configuration file in a raw image on the host and a second tool allows the extraction of the configuration file in the OSv guest.

2.4.2 Performance Evaluation

As OSv cannot optimize physical CPU or memory speed, our work is focusing on the optimization of virtual IO in OSv. This includes both disk and network virtual IO. Two specific benchmarks are presented in the following subsections.

2.4.2.1 Redis

Redis [27] is a simple service that fills a valuable niche between a key-value data store and a full-scale NoSQL database such as Cassandra [28]. Redis supports not just key-value items, but also more advanced data structures such as sets and queues.

The following benchmark compares Redis on OSv and Ubuntu 14.04 AMI. To do that, we have just launched a new AMI on Amazon EC2 [29] with Ubuntu 14.04. We use the configuration file shipped with Redis by default on one CPU core, with one change: we disable disk activity.

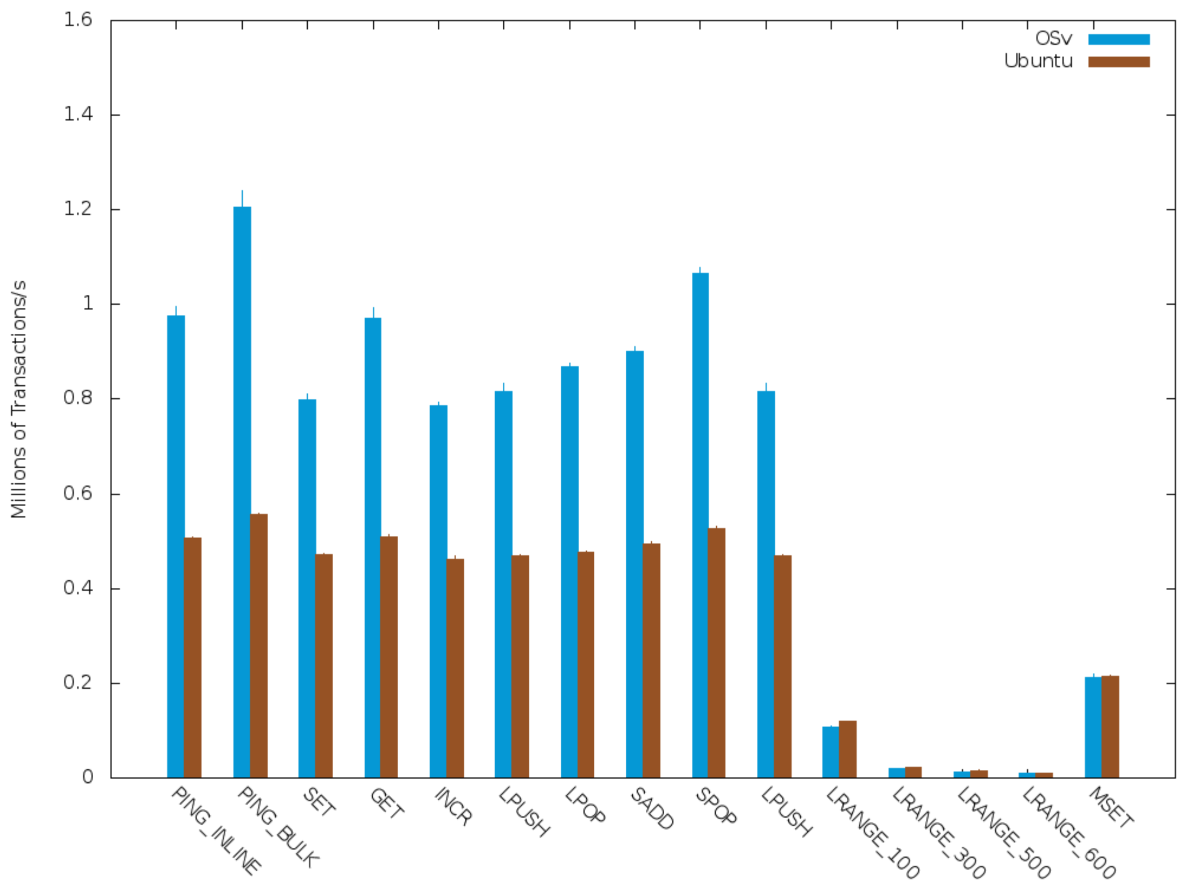


Figure 10. Redis performance evaluation.

On Ubuntu, Redis was run with:

```
numactl --physcpubind=1 redis-server ~/redis.conf
```

Using `numactl` considerably reduces the standard deviation as a result of Linux scheduling. The redis-benchmark command was run from another machine of the same type, running in the same zone and placement group.

We see that the advantage of OSv is clear on non-range queries because the redesigned TCP/IP stack of OSv allows to do smaller queries more efficiently. Range queries don't see any improvement because they generate bigger results.

2.4.2.2 memcached

Memcached [30] is a popular in-memory key-value store. It is used by many high-profile Web sites to cache results of database queries and prepared page sections, to significantly boost site performance.

An unmodified memcached running on OSv was able to handle about 20% more requests per second than the same memcached version on Linux. A modified memcached, designed to use OSv-specific network APIs, had nearly four times the throughput. These numbers are for one core and UDP request.

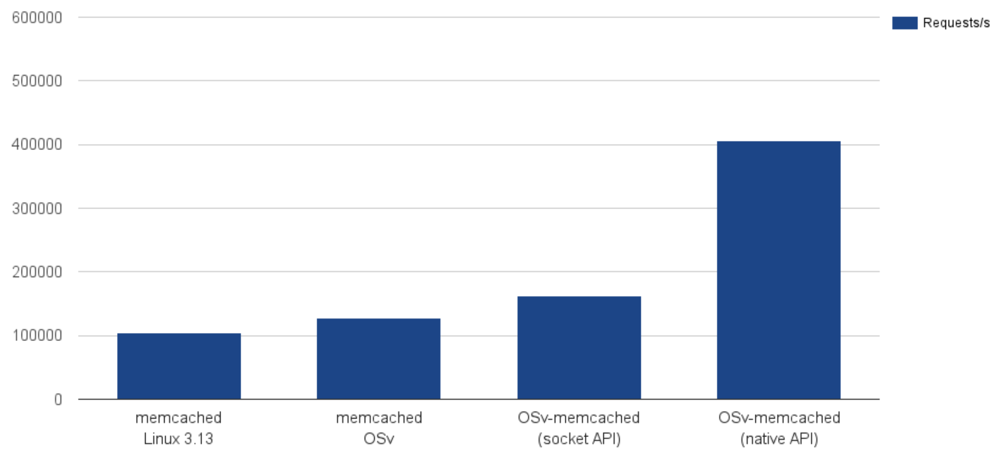


Figure 11. memchached performance evaluation.

2.4.3 Implementation Evaluation

Quality assurance at the Linux compatibility layer is ensured with an extensive set of functional tests OSv provides. All changes to the API start with a test validating the behaviour from the perspective of Linux or the POSIX standard.

OSv possesses its own functional tests in the /tests directory [31] which ensure its API is as compatible as possible with a Linux system. These scripts can be quickly invoked by running the command "make check".

2.5 Application Package Management

2.5.1 Architecture Evaluation

Building on top of the evaluation of the OSv unikernel from the previous section, this section focuses on a different aspect of any new operating system or platform: management of application packages. Even though it is clear from the previous evaluation that OSv provides a reasonably high level of compatibility with general purpose operating systems, such as Linux, the way package management is handled differs significantly.

Deliverable D2.16 The First OSv Guest Operating System MIKELANGELO Architecture [5] already described the essential value of an appropriate package management system supporting user adoption. In this deliverable we have focused on two approaches that the OSv community has already built for the purpose of building OSv-compliant virtual machine images, namely the developer scripts [32] and a tool called Capstan [33].

Developer scripts are provided by the OSv kernel source tree. They include a number of BASH and Python scripts that help OSv developers to build, test and validate their changes to the kernel using real applications. The main script is called *scripts/build*, which in turn consults the OSv kernel Makefile to build the kernel and the requested modules, as well as other scripts that will ensure that results of the compilation process (libraries, applications, supporting files, etc) are uploaded into the target QEMU [13] virtual machine image.

Existing applications are maintained in a central Github repository (osv-apps [34]) which is also linked in the main OSv kernel repository as a submodule. An important design decision of this approach was to maintain applications as some sort of recipes: instead of having prepackaged binary application packages, each application provides a set of scripts that builds the target application in a suitable way. The benefit of this is that it empowers end users to alter applications manually or update them to the newer versions. However, this also has several drawbacks, some of which are described next:

- The end user is required to have a full development environment to use applications.
- The user must build the entire application which, as it will be seen in the next sections, might take longer than expected.
- It is up to the application maintainer to ensure cross-compilation in case system-wide libraries are required for building the target application (for example, if a system

package is required, apt-get would have to be used for Ubuntu/Debian and yum for Fedora/CentOS).
- Very limited possibilities for validation/verification (no validation of packages, no dependency management, resolution, etc.).
- Lack of formalized structure and processes to be followed in building a complete OSv virtual machine image.

Capstan is a specialised tool for building virtual machine images for applications running on top of the OSv kernel. Capstan does not require recompilation of the OSv kernel when the application is being built for an OSv image. Instead, it is using the notion of a base image containing the kernel and one or more additional packages (modules) built into the image. Capstan further formalizes the structure of the application description. However, its reliance on the base images is also its biggest drawback as it only augments the base image by uploading additional files. Base images are of fixed size (10 GB) which is sufficient in most cases but does not provide the flexibility often required in the cloud. Resizing the base image would require the use of the OSv developer scripts discussed previously.

Based on these findings, the requirements collected from other partners and the analysis of frequently used package management systems in other systems, Deliverable D2.16 The First OSv Guest Operating System MIKELANGELO Architecture [5] proposed the following architecture (all components are described in detail in Deliverable D2.16).

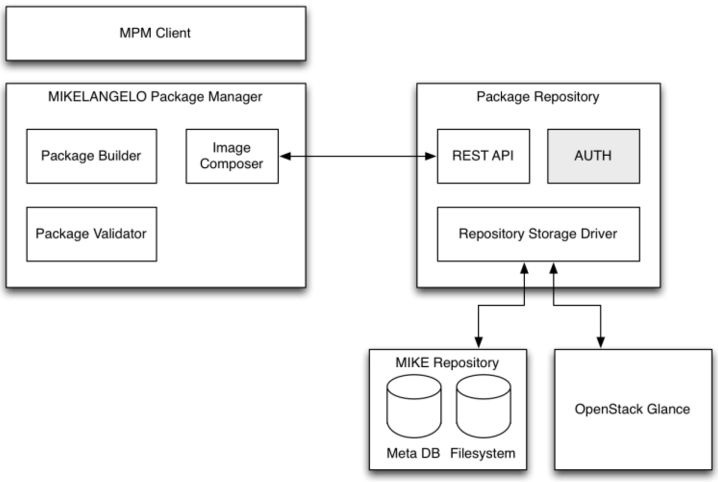


Figure 12. Architecture of MIKELANGELO's application package management.

The focus of the first released version of the application management tool was on the Image Composer and the Package Builder components. The updated tool also supports a preliminary integration with the OpenStack (Image and Compute) services [35]. The following subsections provide the evaluation of the MIKELANGELO Package Manager tool which extends the Capstan tool.

2.5.2 Performance Evaluation

Deliverable D4.7 First version of the application packages [5] introduced the initial performance evaluation comparing developer scripts, the original version of Capstan and the MIKELANGELO Package Manager (MPM) tool which is summarised in the following table.

Table 1. MIKELANGELO Package Management performance evaluation.

	Developer scripts	**Capstan**	**MPM (ratio to dev scripts, ratio to Capstan)**
HTTP Server	3.15 s	3.44 s	6.93 s (2.2, 2.0)
CLI	3.30 s	3.59	6.84 s (2.1, 1.9)
OpenFOAM	30.93 s	7.97 s	8.44 s (0.3, 1,1)

The table shows average times for building target virtual machine images for three different applications. The MPM column also shows relative comparison to developer scripts as ratio between time-spent in MPM vs. developer scripts or Capstan, respectively.

Developer scripts invoke the application's Makefile on every virtual machine image build. Simple applications like built-in HTTP Server and CLI (Command Line Interface) can be built efficiently because their Makefiles are simple. However, a more complex application (OpenFOAM) takes significantly more time just to check whether there are any changes in the OpenFOAM application code (the time required by developer scripts for OpenFOAM in the table above do not rebuild OpenFOAM, just check for changes and recompose the VM image). On the other hand Capstan and MPM only need to upload the resulting application onto the target VM, without checking the OpenFOAM source tree for changes. This makes both alternatives significantly faster.

When comparing Capstan and MPM one should be aware of the following two differences. First, HTTP Server and CLI are applications (modules) that are included in pre-built images. This means that creating an image containing HTTP Server or CLI reduces to making a copy of the base image. On the other hand, slight performance degradation in the case of OpenFOAM is due to the fact that MPM builds images out of packages. This means that even the base OSv has to be uploaded onto the target image resulting in approximately 10% performance loss. Even this degradation is certainly worth the flexibility offered by MPM. However, for the purposes of thorough evaluation we conducted additional tests offering more insights into different approaches to application packaging. These are presented in the following subsections.

2.5.2.1 First Time OSv and Application Users

As we have already described, developer scripts can only be used in conjunction with the entire OSv kernel source tree. Consequently, prior to building an application image, the OSv kernel must be compiled. Besides the time required to setup the development environment this results in additional over 10 minutes required to build the first application image. After the kernel and the application are fully compiled, the times from the aforementioned table are applicable.

This difference is even more significant for large applications with complex compilation logic, such as OpenFOAM. Because OpenFOAM is used in the Aerodynamics use case (as detailed in Deliverable D2.10 The First Aerodynamic Map Use Case Implementation Strategy [5]) the MIKELANGELO consortium provides the OpenFOAM application compatible with other OSv applications from the OSv-apps repository [34]. However, using developer scripts, this still takes several hours to compile before the application image can actually be used, not including the contextualisation of the image. Because Capstan uses the same build command, it would suffer from the same problem. Contrary to this MPM is not affected in any way because a pre-built package is available. The user is immediately allowed to download and compose their OpenFOAM simulation into an executable image.

2.5.2.2 Application Package Authors

The following comparison table focuses on evaluating the three tools from the perspective of application authors interested in sharing their applications as OSv-compliant application images or packages. The evaluation is based on the typical workflow.

Table 2. Application Package Management tool comparison.

Developer scripts	**Capstan**	**MPM**
Preparation of the application content		
Developer is supposed to prepare a script that ensures the application is downloaded, patched and compiled automatically.	Besides the build script which is the same as in the case of developer scripts, a dedicated Capstan image specification is also required (Capstanfile).	Application authors are encouraged to create a verbatim structure of the package. This can be done in an way. The tool supports the author with the creation of an

Developer scripts	Capstan	MPM
		application manifest file which describes the application and its dependencies.
Dependency management		
A Python module is available for OSv supporting the specification of dependent modules. These must be specified in a special Python script (module.py)	Capstan only supports the notion of a base image. This may contain arbitrary modules, however there is no way of composing several modules into a single application image (apart from iteratively building images until all modules are uploaded).	Required packages (modules) are specified in the application manifest file. Application packages are collected by the tool and uploaded onto the target application image.
Image building workflow		
The user needs to invoke the main OSv build script and specify the list of required modules. This script will in turn invoke scripts from required modules and eventually include all of them in the target image.	Capstanfile is consulted for information about the build process and the file structure of the application.	The application manifest is used for basic metadata and other required packages. Content is retrieved directly from the application's root directory.
Performance		
The main build script is highly optimised. However, applications' build scripts may not be efficient as they are provided by third-party application providers. Since the application user is required to recompile the application, this could affect their performance.	Same as in the case of build scripts because Capstanfile just references the build script.	The application author prepares the package in a form suitable for execution on top of OSv, consequently the end user never needs to rebuild the package manually. Additionally, the tool supports efficient incremental updates to

Developer scripts	Capstan	MPM
		target images by uploading only the content that has been changed.
Application execution		
The image must be built before it is used. A script for running OSv-based images is included in the OSv source tree. Being developer-oriented, it provides the most configuration options.	The image must be built before it can be used. Capstan only supports a subset of configuration options of the developer script.	Application may be launched immediately (the tool will ensure the image is updated with the latest content). The same set of configuration options as in the case of Capstan is available to date, apart from the ability to run applications on OpenStack directly.
Package and application repository		
Applications and packages are available in a Github repository [34]. This repository is already referenced as a submodule in the OSv kernel source. Additional local repositories may be added and configured in OSv (config.json file). At the time of this report, 73 different applications are available.	Application images are available on a repository hosted on Amazon S3. 11 base images are available.	MIKELANGELO currently does not provide a central repository from where required packages are downloaded on demand. Packages, provided by the MIKELANGELO project, can be downloaded and integrated into the local repository manually, though.

2.5.3 Implementation Evaluation

Application packaging is addressing two audiences that are important for the uptake of the OSv unikernel and the MIKELANGELO technology stack as a whole. First, application providers and integrators interested in providing self-contained application packages, and second, end-users interested in running their workloads on top of a lightweight Linux-compatible operating system for the cloud.

The following subsections will provide an objective evaluation of different aspects of the current version of the MIKELANGELO Package Management (MPM).

2.5.3.1 Package Metadata

MPM was built on top of the previously developed Capstan tool. Capstan distinguished the image repository from the running instances allowing execution of several instances from the same base image. MPM added another layer on top of this by introducing the notion of an application package. An application package is a compressed archive with additional metadata information used when composing a set of packages into application virtual machine images.

Package metadata is currently used only partially. This information is displayed to the user in the package listing (“capstan package list” command) allowing them to analyse the installed packages. The name of the package is used as a reference for specifying the required packages for the target application image. The requirements are always considered recursively allowing an application image to require packages that consequently rely on other packages.

One of the most important pieces of the metadata information that is currently being ignored is the version information. Even though the version is stored in the package repository and displayed to the end user when querying package information, it is not used by the dependency manager in any way. Consequently, it is currently not possible to have two or more versions of the same application package. This has not been an issue at this stage of the project as we are building applications simultaneously and are always interested in using only the latest (and most stable) version of the application package. However, in order to allow users to choose from different versions of the same package, MPM should be extended to use the version information as much as possible. This will, for example, allow users to choose from OpenFOAM 2.4.0 and 3.0.0 or even one of the legacy versions that they have been using in the past.

MPM metadata should also be extended with information on the capabilities (functionalities) a package provides to an end user. This may be provided in the form of package documentation (for example usage explanation and possible commands) or by providing specific command lines the user is able to reuse. We are currently estimating that a combination of both of these approaches should be employed for best flexibility.

2.5.3.2 Application Packages

Along with the updated tool for managing application packages and virtual machine images, the MIKELANGELO consortium maintains several pre-built application packages. Current packages are mainly required by internal use cases.

The OSv kernel has been compiled using the developer scripts. Two core artefacts have been used and integrated into MPM:

- Kernel loader: the OSv loader is responsible for loading the kernel when the virtual machine is instantiated
- Bootstrap package: a set of libraries and tools that are mandatory for any kind of application running on top of OSv. These include a tool for formatting the target partition (`mkfs`) and the tool to upload application content onto the OSv image. Libraries include the ZFS filesystem support and some of the libraries used by the kernel itself.

Both of these are available in the MPM package repository and are included into the target application image automatically without the user having to specify any explicit reference. Similar to these two components, MPM also provides several of the widely used OSv core modules (HTTP REST server, Command Line Interface, Java and cloud-init to name a few). These modules can easily be incorporated into any application image simply by specifying them as a required package. The benefit of providing these modules as MPM packages is that end users are not required to build them from source nor are they limited to the pre-built virtual machine images containing a subset of these modules.

OpenFOAM [4] provides a set libraries (framework) and applications (solvers) supporting intensive computational fluid dynamics. MPM currently provides two packages:

- OpenFOAM Core: a set of core libraries that are used by every OpenFOAM application.
- OpenFOAM simpleFoam application: a specific application that has been used by the Aerodynamics use case. It depends on the OpenFOAM Core package and only provides libraries and files not included in the core package.

The reason for separating OpenFOAM into two packages is because the use case will employ other applications in the future, each performing additional analyses. Furthermore, having the core separated from the actual application logic allows the end user to integrate their own application (solver) and compose an application image on top of the core.

Finally, the MIKELANGELO project also integrated Hadoop’s distributed filesystem (HDFS [6]) into OSv. A special MPM package is available with the basic mandatory configuration options as well as the configuration of the Java VM. Users can use this base package to create HDFS deployments with specific configuration options and additional modules. Preliminary experiments have been made with Apache Storm [7], however a package has not been provided yet due to some limitation of running Storm in OSv (presented in later sections).

Building all of the above MPM packages proved a trivial task once Capstan was updated with the additional functionalities. The common approach was to build the target application from source, extract relevant binaries and build the application image for testing. The most time consuming task of this approach is the actual validation procedure guaranteeing that the application and all of its required libraries are properly integrated into the package.

After the application image was thoroughly tested, the package was created and stored into MPM's central repository from where it can be used to compose other, more specialised application images.

2.5.3.3 Application Image Composition Caveats

The first release of MIKELANGELO extensions to the Capstan tool already promises significant simplification over existing approaches to building OSv-compliant virtual machine instances. However, in this section we focus on the current limitations as seen by the end user. These limitation will drive the provision of new requirements for the packaging tool.

Packages vs. Applications. Currently the tool does not distinguish between packages and applications - everything is a package and has to have a package manifest. However, frequently we discovered in our experiments that it is cumbersome to have to initialise the application package just to compose the application image. It would be preferable if it were possible to compose application images directly from the packages and the underlying directory structure.

Default command management. Base application packages should be allowed to specify one or more default command lines that would allow the user of the package to run them without knowing the actual command line. For example, the OpenFOAM simpleFoam package could specify its default command as:

```
$ --env=WM_PROJECT_DIR=/openfoam /usr/bin/simpleFoam.so -case /case
```

This tells OSv to set the environment variable and then launch the simpleFoam application with the input case at the specified (/case) location. By setting this command in the base package and providing the user with the ability to check it prior to using it, this package will allow end users to prepare the application package in a suitable way.

Furthermore, such an approach would allow the users to understand options that are available to them. For example, Hadoop HDFS may be used as a *namenode* (main node) or a *datanode* (node storing the data). If both of them would be provided as predefined commands, the user would be able to choose between them according to their needs.

This information could furthermore be propagated into the HTTP REST management API allowing users to launch the commands dynamically through the API. Instead of starting apps immediately upon OSv boot is finished, users would be allowed to launch one or more instances and then communicate to them the roles they should take in a cluster.

Java extensibility. Hadoop HDFS is an example of a complex Java [36] application that contains all required libraries and configuration files. It also sets the default command line to be used when running HDFS application However, prior to using HDFS, users are required to configure it (for example, to specify the location of the main node - namenode). When HDFS is integrated into a target application, the complex Java command line is not populated into the target VM. The tool should support such cases by using base package configuration if it is not provided by the application itself.

Runtime environments. Besides Java we are also investigating other popular environments that are going to bridge the gap between the way developers, system administrators or devops are deploying certain apps. Node.js [37], Python [38] and Go [39] are being investigated for now.

2.5.4 Additional Observations

In April 2016 a new project by EMC [40] called UniK [41] has emerged. The project is targeting compilation of custom applications for unikernel platforms. OSv [2] and rump kernel [42] are currently supported with MirageOS [43] support under way. It supports deployment of unikernel-based applications onto VirtualBox, Amazon and vSphere (OpenStack being one of the next enhancements). The project seems active as changes are added on a daily basis.

OSv support is somewhat limited for the time being as only simple Java applications can be used. Based on current activities and documentation rump kernel seems to be the preferred unikernel. The project integrates the original version of the Capstan tool for building application images resulting in the same limitations as described in previous subsections.

The UniK project does not directly pose a direct competitor to the MIKELANGELO Package Management being developed in the MIKELANGELO project. The project could be seen as a potential counterpart to our contributions focusing on the application runtime lifecycle as opposed to the packaging lifecycle of the MIKELANGELO Package Management.

Similar to UniK, but somewhat less generic, is a set of open source tools developed by Defer Panic [44]. Since these are primarily focused on Go application support and the rump kernel, they are only mentioned for completeness.

2.6 Monitoring

2.6.1 Architecture Evaluation

Cloud and HPC providers have a wide range of tools to choose from to meet their monitoring and instrumentation needs. The most popular options at the moment include collectd [45], Ganglia [46], Telegraf [47], and OpenStack's Ceilometer [48]. Each of these systems have a different architecture and capabilities, and each is written to address specific goals. All of them have the ability to integrate with Cloud and HPC deployments.

Deliverable 5.1 First Report on the Integration of sKVM and OSv with Cloud Computing [5] describes the requirements that drove the telemetry solution selected and developed for MIKELANGELO. It was decided to enhance a new telemetry framework written from the ground up for scalable, flexible, full-stack data-centre instrumentation, snap [8], rather than work with any of the existing platforms. The most important MIKELANGELO requirements had an important impact on the architecture of the system and its design.

Regarding the collection of **hypervisor metrics**, some of the outlined solutions like collectd and Ganglia allow metrics collection from the kvm hypervisor [12], mostly through the libvirt package [49]. But none of them has the possibility of tagging the data with the names of the virtual machines for OpenStack or the job IDs for HPC deployments. This functionality has been implemented in MIKELANGELO's snap: version 2 of the libvirt collector and the snap tag processor allows the virtual machine and HPC task identities to be captured. Indeed changes to the virtual machine such as the addition of a network card or memory can also be tagged, as can the migration of a machine from one host to another. This type of flexibility is not readily achievable with the alternative telemetry system implementations.

Regarding **guest OS metrics**, none of the alternatives considered allow the collection of metrics from the OSv operating system. Snap now allows users to monitor over 260 OSv metrics from the CPU, memory, and IO subsystems amongst others, with web server traces also available, Specific metrics can be turned on or off as required.

Regarding hosted application and service metrics, snap at this moment has 51 plugins released and at least 8 in development: the list of plugins that allows collection of data from various software stacks continues to grow. The extensibility of the architecture to capture data from arbitrary hosted applications and services has been validated by MIKELANGELO: custom plugins were successfully developed (and open-sourced) by the consortium to collect data from libvirt, OpenFOAM and OSv. Additional hosted application and service metrics will be instrumented as the use-cases mature and more specific requirements materialise.

Regarding more general requirements, the snap framework includes many architectural features not available in any of the other platforms. For example:

- **Distributed telemetry gathering workflows** are supported by allowing remote hosts to be specified to execute parts of a telemetry task workflow. A gRPC [50] server runs on each host so that actions can be received and handled by the scheduler. On task creation the workflow is walked and the appropriate remote host is selected or created for each step in the workflow. This allows the user to lower consumption of the local CPU and other resources by offloading any intense analytical steps they may require to remote systems. Distributing a telemetry workflow increases network traffic, so users should only distribute workflows when there is sufficient value in reducing the local overhead.
- **Dynamic metrics reconfiguration** is supported by snap without requiring an application restart. With other telemetry platforms the reconfiguration of metrics collection typically requires the user to stop the application, reconfigure, and then restart. This can demand frequent interventions by the system administrators. In the snap framework the collection of each metric is defined in a task, and tasks can be started, stopped and reconfigured without having to restart the daemon processes.
- **Tribe** is the name of the clustering feature in snap designed to greatly simplify management of large numbers of nodes. When it is enabled, snapd instances can join one another through an agreement, thus forming a tribe. When an action is taken by one snapd instance that is a member of an agreement, that action will be carried out by all other members of the agreement. When a new snap daemon joins an existing agreement it will retrieve plugins and tasks from the members of the agreement. Tribe can be turned on by passing "--tribe 1" argument to the daemon. In the near future INTEL is planning to extend Tribe agreements to support configuration and logging agreements.

2.6.2 Performance Evaluation

To evaluate the implementation of snap and compare it to other telemetry systems, automatic deployment scripts were written to gather equivalent sets of metrics from a local node using the following systems:

- Snap version v0.14 with the snap plugin pack [8]
- Collectd version 5.5.0 [45]
- Ganglia 3.6.0-1ubuntu2 distributed with Ubuntu 14.04 [46]
- Telegraf 1.0.0 available [47]

The system hosting the tests had an Intel® Xeon® E5320 CPU, 1.8g6 GHz, with the Virtual Machine allocated 4GB RAM, 2 VCPUs , 40GB disk space, and running Ubuntu 14.04.

An in-house, low-overhead telemetry system known as "Cimmaron" [51] was used to measure the performance of each of the telemetry platforms. Cimmaron gathered the following data for each test:

- cpu utilization / cpu saturation
- disk utilization / disk saturation
- used memory
- disk usage

The experiment was executed three times for each telemetry system: collecting Linux proc file system or sys filesystem data from 10, 50 and then 100 probes. This would allow the scalability of each telemetry gathering system to be observed. Each experimental run lasted 5 minutes, data was collected with a 1 second resolution, and all data gathered was published into a local comma-separated-variable file.

Unfortunately the 50 and 100 probe runs could not be performed using Ganglia as it does not allow the specification of individual metrics and only 10 metrics were available for the procfs filesystem.

The performance data gathered is summarised in the following tables.

Table 3. Comparing telemetry frameworks collecting 10 probes.

Telemetry Platform	**Idle**		**Runtime**		
	CPU Util	**Memory**	**CPU Util**	**Memory**	**Disk Util**
Collectd	0.3 %	50 MB	1-4%	50 MB	0%
Telegraf	0.0 %	10 MB	0-0.25 %	30 MB	0%
Ganglia	0.1 %	14 MB	0-0.25%	14 MB	0 %
Snap	0.1 %	140 MB	2-3%	160 MB	0%

Table 4. Comparing telemetry frameworks collecting 50 probes.

Telemetry Platform	Idle		Runtime		
	CPU Util	Memory	CPU Util	Memory	Disk Util
Collectd	0.3 %	50 MB	3-4%	80 MB	1%
Telegraf	0.0 %	10 MB	3-4 %	38 MB	0%
Ganglia	-	14 MB	-	-	-
Snap	0.1 %	140 MB	2-3%	150 MB	0%

Table 5. Comparing telemetry frameworks collecting 100 probes.

Telemetry Platform	Idle		Runtime		
	CPU Util	Memory	CPU Util	Memory	Disk Util
Collectd	0.3 %	50 MB	3-5%	110 MB	2-3%
Telegraf	0.0 %	10 MB	3-30 %	120-240 MB	0%
Ganglia	-	14 MB	-	-	-
Snap	0.1 %	140 MB	2-3%	150 MB	0%

Exploring this data it can be seen that all Telemetry Platforms except snap have a noticeable increase in demand for local memory and CPU resources as the number of probes being gathered increased. Snap's relatively static overhead is possible due an optimised allocation of memory and the very efficient scheduler enabled by the Go language, compiler and runtime [39].

To validate the implementation of snap clustering on a large-scale deployment, an ansible script was written to deploy snap on 500 compute nodes that were configured to continuously run a monte carlo simulation under various conditions. The snap daemon was configured using tribe agreements to

- collect 8 metrics:

  - compute utilization / saturation
  - memory capacity utilization / saturation
  - network card utilization / saturation
  - storage utilization / saturation,
- process data using automatic anomaly detection via the Tukey method [52]
- send processed data to an InfluxDB [53] database.

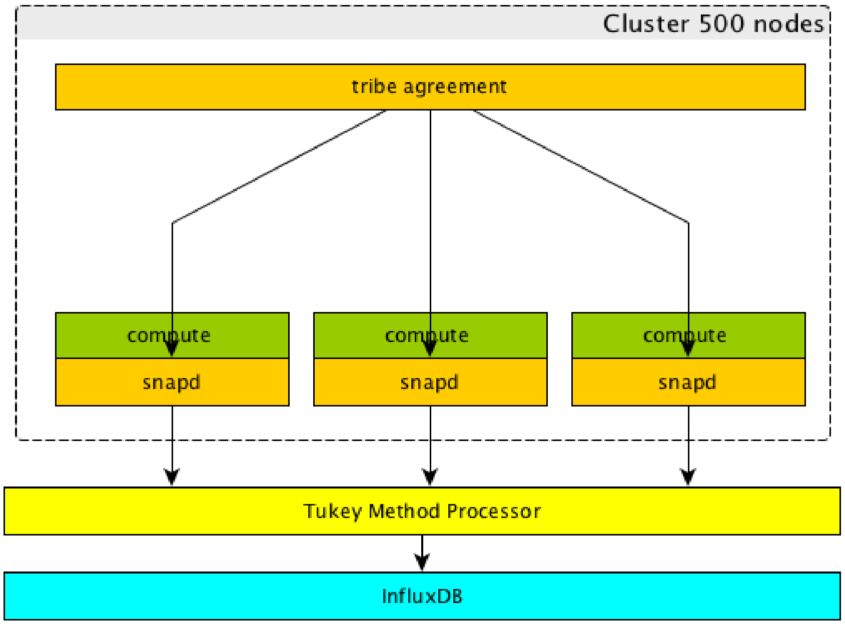


Figure 13. Snap task workflow for 500-node test.

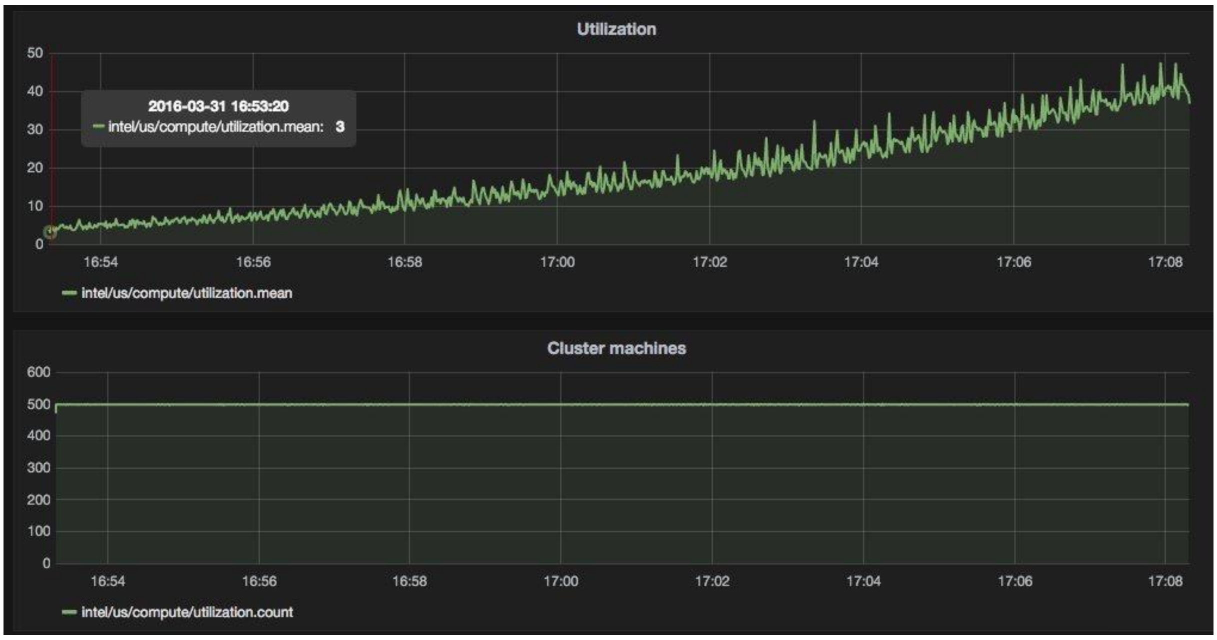


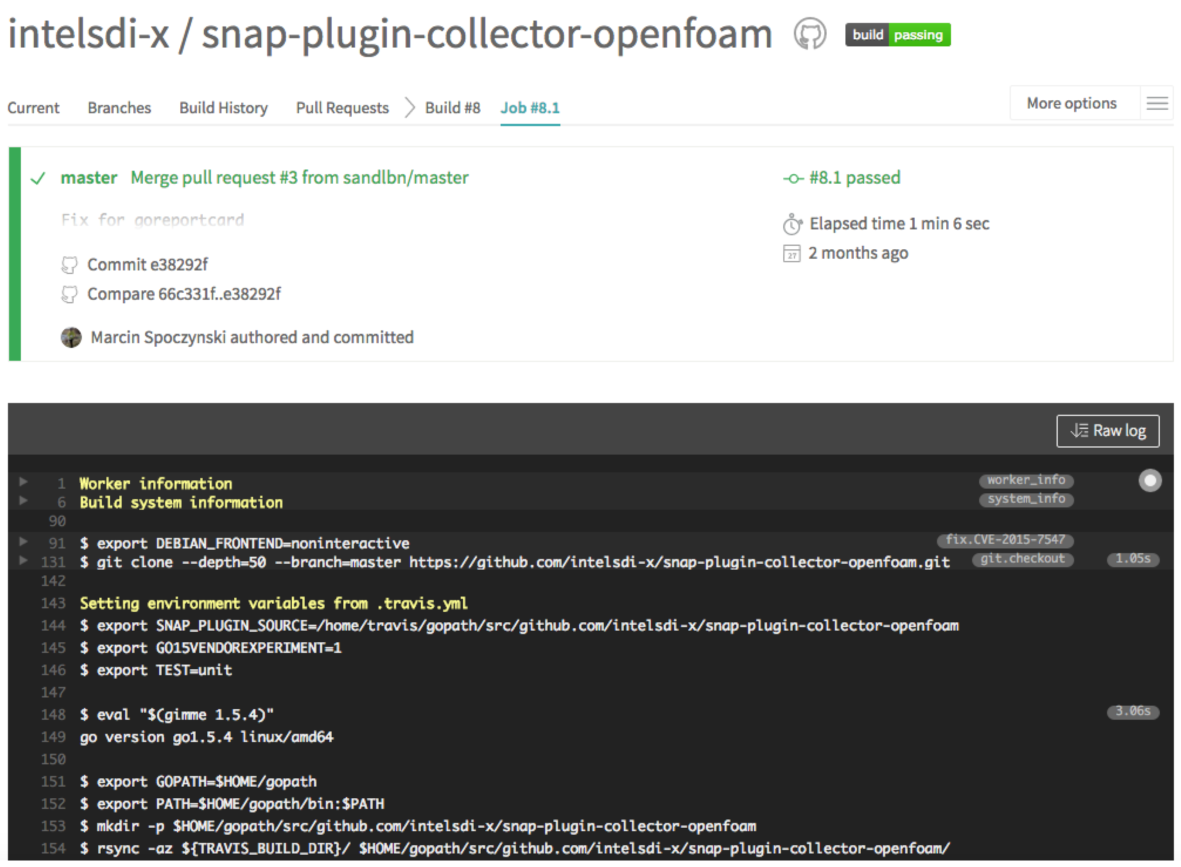
Figure 14. Snap Grafana dashboard illustrating mean CPU utilisation across 500 nodes.

Results showed that the clustering features built into the snap framework were able to manage a 500 node monte-carlo simulation with no issues. Telemetry was successfully captured from all nodes, processed locally, published to a scalable InfluxDB backend and available for review via the grafana dashboard graphical user interface.

2.6.3 Implementation Evaluation

Snap was open sourced in December 2015 and the open-source community are actively encouraged to contribute via the project facilities. Best-in-class development practices and tools such as GitHub, Travis CI [54] and Jenkins [55] have been adopted to maximise the quality and robustness of the code.

At the core of continuous integration of snap lies the ability to automate the tests that are being developed in parallel with the code. Code is tested for quality, functionality, integration and performance, Every plugin repository contains its own test files and configurations for Travis CI - the continuous integration tool employed by snap. Every change of the code in the repository automatically triggers testing to be carried out on both the Linux Travis environment and on the local Jenkins server.


intelsdi-x / snap-plugin-collector-openfoam

✓ master Merge pull request #3 from sandlbn/master

Fix for goreportcard

Commit e38292f

Compare 66c331f..e38292f

Marcin Spoczynski authored and committed

#8.1 passed

Elapsed time 1 min 6 sec

2 months ago

```
Worker information
Build system information

$ export DEBIAN_FRONTEND=noninteractive
$ git clone --depth=50 --branch=master https://github.com/intelsdi-x/snap-plugin-collector-openfoam.git

Setting environment variables from .travis.yml
$ export SNAP_PLUGIN_SOURCE=/home/travis/gopath/src/github.com/intelsdi-x/snap-plugin-collector-openfoam
$ export GO15VENDOREXPERIMENT=1
$ export TEST=unit

$ eval "$(gimme 1.5.4)"
go version go1.5.4 linux/amd64

$ export GOPATH=$HOME/gopath
$ export PATH=$HOME/gopath/bin:$PATH
$ mkdir -p $HOME/gopath/src/github.com/intelsdi-x/snap-plugin-collector-openfoam
$ rsync -az ${TRAVIS_BUILD_DIR}/ $HOME/gopath/src/github.com/intelsdi-x/snap-plugin-collector-openfoam/
```

Figure 15. Automated Continuous Integration tests managed by Travis CI.

Average coverage tests for snap repositories are about 85% . Some of the repositories like libvirt contain integration tests, performed on the real libvirt package, against a fully loaded plugin. This kind of test allows tests to be performed in an environment almost identical to production. To have a clear view of the state of a test, appropriate badges have been implemented and assigned to every developed plugin, allowing the developers and integrators to quickly check the status of a test suite and static code analysis check.

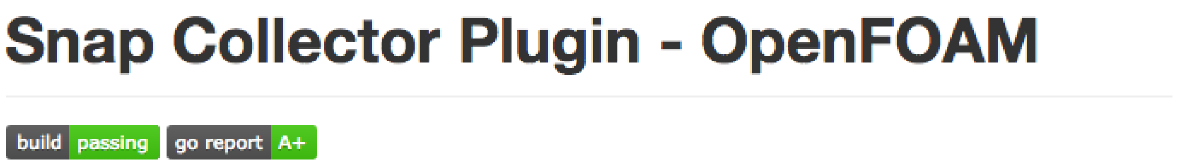

Snap Collector Plugin - OpenFOAM

Figure 16. Badges assigned to the MIKELANGELO developed OpenFOAM collector plugin.

2.6.4 Additional Observations

Being able to use the snap open source telemetry framework and integrate it with the Cloud and HPC environment allows MIKELANGELO resources to focus on implementing the precise functionality that the project requires, rather than attempt to construct and maintain an independent flexible framework, or be forced to work within the limitations of existing telemetry systems.

Snap is suitably flexible, extensible, scalable, and performant for MIKELANGELO purposes, and is gathering full stack data on both the project's Cloud and HPC environments.

Snap is new, and does not yet have a substantial open-source community, but significant facilities and resources have been put into making this an open platform, and the first plugins from the open-source community have already been accepted and published.

Regarding next steps, INTEL has plans to develop additional functionality as requested and prioritised by MIKELANGELO. An Open vSwitch plugin is envisaged to allow detailed virtual network statistics to be captured. Anomaly Detection and Utilisation/Saturation metrics will soon be available to automatically reduce data resolution when feeds are static, and to summarise high level metrics. A request has also been received to develop a snap controller for the Cloud which can automatically start metric collection from Guest operating systems like OSv or specific applications, based on information provided by the orchestration engine. There are also opportunities to further automate the analysis of data captured by snap, possibly leveraging the open source Trusted Analytics Platform [56]. Such a toolkit could automate the comparison of large volumes of data captured across multiple runs of an experiment, allowing key changes in performance and correlations to be discovered, and optimisations to be identified.

2.7 Hosted Application Acceleration

2.7.1 Architectural Evaluation

Seastar [9] is a novel asynchronous C++14 programing framework created by the ScyllaDB team in order to be able to write the ScyllaDB database [57]. Seastar drew inspiration from existing concurrent frameworks including:

- Vertx: [58] (Java)
- Nodejs: [37] (Javascript)
- Twisted: [59] (Python)
- Libevent: [60] (C)
- EventMachine: [61] (Ruby)

Important points distinguishing Seastar from these include the implementation language: Seastar is written in C++ whereas all of the other frameworks apart from Libevent are implemented in slower languages. Seastar has a sharded design, in contrast to NodeJS and VertX where threads are heavily used. Finally, Seastar features many low level optimizations due to the kernel developer background of the ScyllaDB team.

Compared to Libevent, Seastar provides a more complete programing model. Once a programmer overcomes the initial challenge of learning Seastar, they can consistently write huge asynchronous applications while keeping complexity at a minimum.

In contrast to other frameworks like NodeJS and Twisted, modern C++ helps avoid typical callback confusion by making heavy usage of lambdas. There is also a coroutine like programing model that has been recently added.

Another unique aspect of Seastar is the DPDK integration which allows Seastar to drive the network card directly by providing its own optimized poll mode TCP/IP stack.

2.7.2 Performance Evaluation

This section presents a benchmark done with the custom Seawreak HTTP load generator which was written by ScyllaDB in order to keep up with Seastar’s pace on a manycore machine. Tests of a new Seastar-based HTTP server show that it is capable of ~7M requests/second on a single node. Details of the benchmark are presented next.

This benchmark uses two identical Intel® Server System R2000WT servers. These servers are configured as follows:

- 2x Intel® Xeon® Processor E5-2695 v3: 2.3GHz base, 35M cache, 14 core (28 cores per host, with HyperThreading to 56 cores per host)

- 8x 8GB DDR4 Micron memory
- 12x 300GB Intel S3500 SSD (in RAID5 configuration, with 3TB of storage for OS)
- 2x 400GB Intel NVMe P3700 SSD (not mounted for this benchmark)
- 2x Intel Ethernet CNA XL710-QDA1 (two cards per server, cards are separated by CPUs. card1: CPU1, card2: CPU2)
- OS info: Fedora Server 21, update with the latest updates as of February 19, 2015.
- Kernel: Linux dpdk1 3.17.8-300.fc21.x86_64
- Default BIOS settings (TurboBoost enabled, HyperThreading enabled)

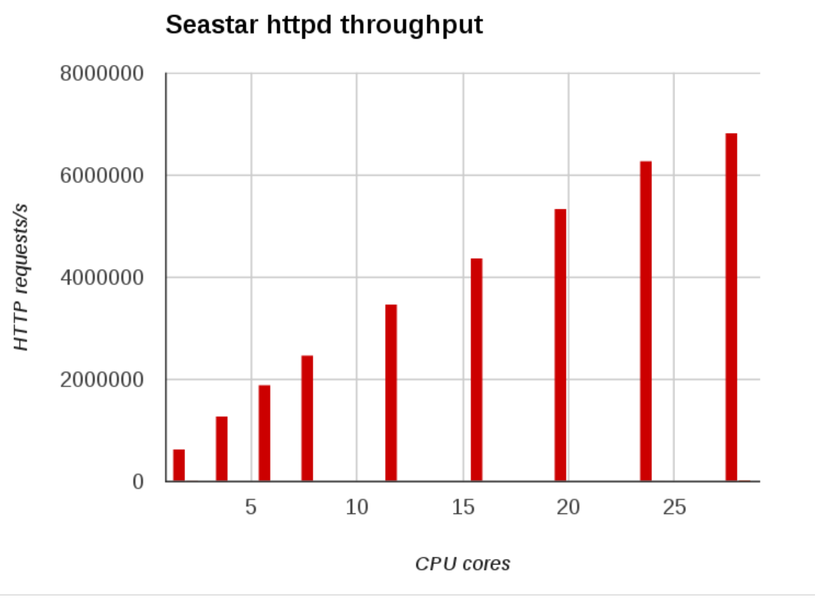


Figure 17. Seastar httpd performance evaluation.

The most important observation from the above Figure 17 is the linear scalability of the Seastar programming model. This is mainly due to the sharded design of Seastar application. Each request is assigned to a specific shard and all its processing and the response generation remains within that shard. No locks are present in the code so no contention will slow down the application.

To summarize Seastar allows to combine raw I/O hardware capacity with manycore machines to write high IOPS server side applications.

An independent benchmark [62] comparing how Seastar performs against the competition on a small machine with an HTTP workload has shown a two-fold performance improvement on a small machine with a few cores and direct access to the network card enabled. The main differentiator is the fact that the Linux kernel does not get in the way of accessing the network card removing the underlying inefficiencies. On a many core machine the lack of lock contention (two threads battling for a lock) inside Seastar would result in even greater margin compared to other concurrent frameworks.

2.7.3 Implementation Evaluation

Seastar is written in about 50k lines of modern C++14 code. Unit tests represent approximately 10% of the entire codebase. On top of Seastar and ScyllaDB there are multiple test suites:

- C+14 unit tests with one suite for Seastar and one for Scylla
- A Fork of Cassandra dtest which does functional testing of the database using python.
- A Small artifact result test suite to check that the various distributions (dpkg/rpm) of the database are well built.
- The scylla-cluster python test suite which test live clusters running on top of AWS EC2

Jenkins is used as a continuous integration tool for all the ScyllaDB projects running test suites automatically for every commit. The development process of Seastar and Scylla is modelled after the Linux kernel development process by using a public Google group as a mailing list. Other sub-projects of lesser importance are developed using GitHub.

A general thought about the Seastar and Scylla code base is that it would fare well as a showcase of what a modern C++14 code base should look like.

2.8 Side Channel Attack Mitigation

2.8.1 Architectural Evaluation

The purpose of Side Channel Attack Mitigation - SCAM - is to monitor cache activity to identify potential cache-based side channel attacks and to mitigate against these attacks. Architectures for mitigating the effects of such attacks are of three types: application-specific software measures, application agnostic software measures and hardware changes to the cache. SCAM is software only and is application agnostic.

Two notable examples of alternatives to SCAM are the specific protection that is part of the OpenSSL [63] implementation of modular exponentiation (which is the critical component of computing RSA signatures and DIffie-Hellman key exchanges) and Intel's Cache Allocation Technology (CAT) [64]. To the best of our knowledge, SCAM is the only architecture of the second type that includes both monitoring and mitigation. As such it can initiate the mitigation module only when the results of monitoring point to a possible attack taking place. SCAM will include novel features compared to earlier proposals including fine-grained monitoring of cache-sets and mitigation by adding noise to the cache.

It is safe to assume that application specific software measures have better performance than the SCAM approach. In the case of OpenSSL, the mitigation measure increases the time required for modular exponentiation by about 20%-30%, but since these operations occur only during the handshake at the beginning of a session, and constitute only a part of the handshake, the overall effect on performance is very small.

Of course, the main advantage of SCAM over application specific measures is that it is general and can protect a large range of applications.

Intel's CAT enables configurable separation of the cache into different regions such that each VM (or process) can be assigned an exclusive region. Since cache-based side-channel attacks rely on the shared nature of the cache, CAT may neutralize these attacks. However, CAT is being marketed as a performance enhancement for specific scenarios and will probably be used for security only as an afterthought. If it becomes widely used then it is quite possible that the mitigation module of SCAM will be necessary only in niche markets. However, the monitoring module of SCAM will still be needed, unless separating the cache at all times becomes the absolute norm in the market.

2.8.2 Performance Evaluation

The SCAM component is currently under development and it is too early to evaluate its performance at the time of writing.

2.8.3 Implementation Evaluation

The main method for evaluating the quality of SCAM is testing it against the side-channel attack that was developed in the first year of the project. To avoid over-fitting SCAM to the specific implementation of the attack (rather than inherent characteristics of such attacks) we continue developing the attack in parallel to the development of SCAM.

The main effort in developing the monitoring module has been devoted (so far) to understanding the value of cache hit/miss counters that are provided in many modern chipsets. These counters can be read in user space using the PAPI software library [65] and provide information on the total number of cache accesses, hits and misses for various cache levels over a user-defined time period. The advantage of this procedure is that it is relatively cheap in terms of performance. Initial tests showed that the ratio of L2 misses to total accesses easily distinguishes between our attack and a standard application (we used a web-server under various scenarios to simulate an application). However, it turned out that it was possible to tweak the attack in a way that essentially removes this distinction. The root cause of the problem is that the attack focuses on one cache set at a time and does not necessarily cause an unreasonable number of misses over the whole cache. Our next step will be to gather information on specific cache sets. This step will require different tools, such as prime and probe, and will be more performance intensive.

The mitigation module of SCAM is expected to comprise a noise sub-module and a page manipulation sub-module. A prototype of the noise sub-module has been developed and seems promising so far in that it completely prevents our attack and seems quite robust. It works by testing the cache sets and adding noise, i.e. reading data, to the cache sets that are most promising from an attacker's point of view. The noise sub-module works across many more sets than the attacker actually needs, but adding relatively little noise to a set seems sufficient to ruin the attacker's measurements.

3 Full Stack Evaluation

3.1 Introduction

A separate evaluation has been presented in the previous chapter for each of the MIKELANGELO components that have been implemented to date. The project has also dedicated significant resources into creating complete software stacks for both Cloud and HPC deployments. An evaluation of the architecture and implementation of these full stacks is presented in this section.

3.2 Full Stack for Cloud

For the full stack cloud deployment we evaluate the benefits of an integration of MIKELANGELO components in a full cloud stack. Our choice for the cloud stack is OpenStack for considerations laid out in Deliverable D5.1 First Report on the Integration of sKVM and OSv with Cloud Computing [5]. While D5.1 presented an overview and comparison of various cloud stacks to consider as the basis for MIKELANGELO's cloud stack, this deliverable considers the benefits of a full integration of the MIKELANGELO components. Each component is evaluated individually with regards to the added value it brings to the cloud.

We move through our architecture bottom up, starting with the hypervisor sKVM, and ending up with enhancements to OSv. Each component is evaluated with regards to criteria which are of special importance to cloud providers, and by extension to cloud users. The criteria are the difficulty of installation, the benefits gained from the component, potential for cross-layer optimization, and our future work on the component from the perspective of the cloud.

3.2.1 sKVM with OpenStack

Although we refer to sKVM as one component, there are actually three innovations contributed to KVM by MIKELANGELO. The first innovation is IOcm, which improves IO performance. The second innovation is SCAM, which improves the privacy of virtual machines running via sKVM. The third is Virtual RDMA, which allows for efficient inter-VM communication. Although all three components are integrated in sKVM their benefits can be reviewed independently. A holistic view of sKVM and further MIKELANGELO components from the cloud perspective is described briefly in the following sections and more detail regarding cross-layer optimization is available in deliverable D2.20.

3.2.1.1 IOcm

Installation. The installation of IOcm requires the use of a patched Linux kernel that contains our modified version of KVM. In principle, the installation is straightforward for any administrator. However, the requirement for a custom kernel raises trust issues and potentially adds an additional step in the already complicated process of setting up large infrastructures. Additionally the normal path to kernel updates, including security updates would be broken. We are working to overcome this issue by disseminating our results to push for an upstreaming of IOcm into the mainline Linux kernel. Currently, the patched kernel has been evaluated with Ubuntu running on cloud hosts at GWDG.

Benefits. The benefits of IOcm lie in its ease-of-use and potential gains in IO performance. A current evaluation on the Cloud stack focused on the functional aspects of IOcm. Thus, so far no gains in IO performance have been pursued or measured. From a functional view it was possible to run virtual machines with IOcm and dedicated IO cores. The dedication of cores to IO on the testbed has been controlled manually to date. The new IOcm functionality automatically dedicating cores to IO will be integrated when testing is complete.

Cross-layer optimization. Due to the early stage of integration there are no cross-layer optimizations yet. However, IOcm offers great potential for cross-layer optimization. The extra knowledge that a holistic view of the cloud provides can be leveraged to manage IO resources globally throughout the data center. Furthermore, when including even the application layer, such as in the case of big data applications, it will become possible to provide even more targeted optimizations. IO capacity of individual hosts requires online assessment and matching with cloud workload, especially big data workloads. IO-core configuration presents a valuable opportunity for infrastructure optimization.

Future work. In future we want to provide integrated packaging which will allow a simpler deployment of IOcm together with the whole cloud stack. This should happen before IOcm finds its way into the mainline Linux kernel. In addition to packaging, we will work on the optimization of IO resources within a single cloud host. Beyond a single host, work will commence to optimize IO resources across the whole cloud deployment.

3.2.1.2 SCAM

Installation. The installation of SCAM cannot be evaluated easily currently, since no generally applicable version is yet available. In general, we expect that the installation of SCAM will face similar hurdles to that of IOcm, as both are architected as kernel-extensions. With SCAM there may be the additional obstacle that the implementation targets a specific computer architecture. Thus, in comparison with IOcm, SCAM may face the additional issue that some tweaking of code may be necessary to get it to work on a new processor. A detailed analysis of the installation issues of SCAM, however, can only be provided once SCAM is ready for installation in the cloud.

Benefits. The potential benefits of SCAM are increased security in the cloud. SCAM would offer an increased level of privacy, particularly in public cloud offerings. Using SCAM, attacks on the other VMs can be monitored and then mitigated against. Mitigation can take the form of isolation of suspicious VMs or a general shutdown of those VMs and other VMs of that tenant. Since we expect this extra security to require additional computing resources, such a service can be offered as an additional option as part of an SLA (Service Level Agreement).

Cross-layer optimization. SCAM offers the potential for holistic monitoring of the cloud with global regions with differentiated security levels. For example, the security and mitigation feature can be combined with VM placement in the OpenStack Nova scheduler and in MIKELANGELO's extended online scheduler. With a holistic view the cloud layer can manage when and where the SCAM component should monitor for suspicious VMs. A differentiator between security levels may define the extent to which SCAM monitors for suspicious activity. High-frequency monitoring will be more costly in terms of computation, and thus pricing for the customer, but it will also offer increased security.

Future work. Currently, SCAM is being ported to hardware that is similar to that found in GWDG's cloud testbed. The next step after this porting process will be the rollout of SCAM at GWDG's testbed. The module will then be tested functionally. In addition, the performance penalty of using SCAM will be assessed and trade-offs will be evaluated.

3.2.1.3 Virtual RDMA

Installation. The installation of Virtual RDMA has not been tried yet, since GWDG's NICs are not supported by the current prototype of the Virtual RDMA module. In general, we expect to face mostly the same challenges and opportunities as with IOcm for Virtual RDMA. In contrast to IOcm, Virtual RDMA contains additional code that runs in user space. This code will need to be packaged and deployed separately. The deployment however can be aligned with the deployment of the whole cloud via automated puppet scripts.

Benefits. Virtual RDMA offers benefits for inter-VM communication within a host and between hosts. The communication within a host will provide improved efficiencies due to zero-copy communication via IVSHMEM. Communication between hosts will use RDMA to provide low overhead communication. Whether the latter will be available in a cloud setting is unclear since the current prototype only works with specific RDMA-capable NICs. The use of Virtual RDMA within hosts and between hosts can offer great benefits for a cloud setting, especially considering that OSv is going to be used. With OSv, we expect to run many more virtual machines, since each process requires its own instance. Thus, there will be more communication between VMs, which in turn calls for a more efficient way of communication.

Cross-layer optimization. Virtual RDMA offers great potential for cross-layer optimization due to its capability to provide efficient communication within a host and between hosts. Cross-layer optimization using cloud-layer information can co-locate VMs or spread VMs out across the data center to optimize based on communication patterns. In particular, a traffic matrix can be generated to assess communication patterns. Based on the traffic matrix and OpenStack Nova's admission control it becomes possible to speed up communication by colocation and activation of Virtual RDMA.

Future work. As future work of the integration of Virtual RDMA with the cloud, we will first deploy the prototype in the cloud testbed and test performance for intra-host communication. Further work will contain the implementation of cross-layer optimization and a potential integration for inter-host communication.

3.2.2 Snap in the Cloud

Installation. Snap offers simple installation via binary packages. For installation from source documentation is available as well. One of the major benefits of snap is its seamless integration with typical monitoring environments. For example, the GWDG cloud test bed uses collectd and InfluxDB for monitoring. Thanks to snap's integration with collectd as collector and InfluxDB as publisher snap has been deployed easily.

Benefits. The benefits of using snap in the cloud include flexibility due to its task model, and its scalability, as well as those benefits of the seamless integration mentioned above. The task model in particular allows improved integration with the cloud since it can monitor VMs dynamically. Parameters such as resolution and types of monitored metrics can be adjusted as needed using tasks. The scalability of snap is ensured by its distributed model. Another benefit lies in the large number of supported metrics even though the project is relatively young.

Cross-layer optimization. Snap enables cross-layer optimization as the central monitoring tool. All cross-layer optimization in MIKELANGELO hinges on the swift availability of monitoring data across the whole infrastructure stack.

Future work. In future snap's integration with the cloud will be extended by a tighter integration with OpenStack and by providing additional metrics.

3.2.3 OSv

Installation. The installation of OSv in a cloud environment requires making an OSv instance available as an image in OpenStack, or equivalent. In contrast to generic operating systems such as full Linux distributions, OSv has the disadvantage that an instance that is bundled with the application needs to be created. This approach is analogous to the creation of

Docker images. In the context of IaaS clouds, this concept is not yet widespread. However, using the MIKELANGELO Package Manager (MPM) it becomes easy to deploy applications bundled with OSv.

Benefits. OSv offers fast startup times, a small memory footprint, and good IO performance. Many applications, as shown by our use cases, benefit from those features directly.

Cross-layer optimization. OSv offers multiple opportunities for cross-layer optimization. First, OSv will integrate with MIKELANGELO components to offer a tight integration. Second, since OSv only runs one service per VM instance, larger services are split across multiple VMs. Thanks to OSv's small footprint the increased number of VMs is not an additional burden to the infrastructure. However, the distribution of complex services across multiple OSv instances allows a more fine-grained control of service allocation, which in turn allows for better control of sophisticated Quality of Service (QoS) features.

Future work. Future work on OSv includes improving IO throughput, compatibility with applications, and ease of use. To reach success in the cloud OSv will need to provide a very simple way of deploying new applications. Thus, one of the more important aspects of OSv's integration with the cloud is further development of MPM.

3.3 Full Stack for HPC

Nowadays, in the design of big data and HPC scientific applications the target infrastructure where the application will run is decisive. Two alternatives are commonly used: an HPC cluster, i.e. a supercomputer, or a Cloud system. There are pros and cons to both approaches.

In Cloud systems the programmer can easily put entire software stack into a VM image and the application will be executed unaware of the underlying hardware. On the other side, cloud systems have a lower I/O performance. That is, virtualization offers near-zero overhead for computation but can significantly deteriorate efficiency of I/O operations [66].

Virtual Machine images are typically based on a well-known operating system distribution of Linux such as Ubuntu [11], Debian [67], or CentOS [68]. Linux has not been developed specifically to be run as a guest OS in the cloud. Thus, it carries a lot of unnecessary baggage in the form of legacy code that was intended for other purposes. This legacy code leads to inefficiencies that result in different scopes like start-up times, computational throughput, I/O performance, and disk image size. Most Linux distributions currently provide VM images that are reducing these overheads in a form of cloud images. These images are still significantly larger (300+MB Ubuntu cloud image [68] to about 50MB OSv) than comparable images containing simple unikernels.

Scientific applications, like big data or HPC ones, still rely heavily on HPC clusters because the application will benefit from higher computational and communication performance. On the other side the programmer must adapt the application software stack to the specific hardware, operating system and storage platform used in the cluster. Furthermore, cluster administrators have to install software components like compilers or libraries as part of the software available in the cluster for each user application to build it and execute it properly. All of this typically leads to legacy and security issues, because the software cannot be updated on a regular basis.

MIKELANGELO project aims to apply the flexibility of Cloud systems in HPC systems for big data and HPC applications focusing in the improvement of the I/O performance in virtualization mentioned before.

3.3.1 sKVM with HPC

sKVM will provide better IO performance and additional security components than the default KVM. The integration of sKVM into the testbed involves IOcm, with new kernel functions to allow better control over the I/O, a configuration tool for managing dedicated I/O cores, virtual RDMA components for efficient communication and SCAM, the new security components.

3.3.1.1 IOcm

The Linux kernel component which is developed by IBM is integrated by building a new kernel. Ansible is used to roll out this newly built kernel across the HPC infrastructure. This simplifies the installation process and ensures that nodes have the same kernel and the same boot settings.

The second component is the static I/O core manager. This tool allows us to predefine the cores of the host CPU that are dedicated I/O cores. These cores will handle all the VM I/O and speed up the VMs input/output operations. IOcm is deployed in our shared workspace (/opt), and configured through Torque [69].

The integration of the newly developed dynamic core manager, which can handle IO core allocation automatically is under development. This will replace the static manager and will be able to react to the load changes inside the VM and allocate cores for IO dynamically. Torque integration will still allow users to specify expectations about the I/O usage (for example, in terms of minimum and maximum number of I/O cores).

3.3.1.2 Virtual RDMA

The goal of MIKELANGELO's new design of a virtual driver based on Remote Direct Memory Access (RDMA) is to provide more efficient I/O, zero-copy, and improved cache-efficiency for Inter-Process Communication (IPC) between virtual machines, running either on the same or on different hosts.

In order to use the implemented virtual RDMA prototype I on HPC infrastructure, additional integration was performed on the USTUTT testbed. The integration does not change any logic of requesting HPC resources for the user. Internally the integration prepares all the necessary services, daemons and environmental settings before the resources are assigned to the user. Several shell scripts have been integrated with the Torque extensions, which hides the complexity of the whole setup, eases the process of initializing the virtual RDMA environment, and then provides the user with necessary connection interfaces to be used in the configuration of the virtual machines. More details on how they are integrated into the HPC infrastructure are described in Deliverable D2.20 The Intermediate MIKELANGELO Architecture [5].

3.3.1.3 SCAM

The SCAM module is still in development, and at the time of writing is not ready for integration. Because the current scheduler strategy dedicates each node to a particular job, SCAM is not yet relevant to the envisioned full stack HPC deployment at USTUTT. In future versions of the HPC integration we might be supporting multi-tenancy on a node level. However, SCAM will be integrated to evaluate the possible performance impact.

3.3.2 Snap

Snap has multiple functions in our testbed. One is to monitor the overall health of the cluster. It is installed via Ansible on all nodes and is autostarted with its own startup script. All metrics are published to an InfluxDB database located in a VM on the frontend. In a larger environment this could be a distributed database, depending on the size of the cluster. We have decided to host the entire monitoring logic in a dedicated virtual machine for additional security and better flexibility. This VM also hosts the Grafana graphical interface, allowing dashboards to be easily generated from the readings stored inside the database.

The other use of snap is to monitor job data itself. Each job submitted via Torque is automatically tagged with the job id and user id. It allows the ready identification of related monitoring data for users and jobs in the database, facilitating developers concerned with the performance of their software, and administrators curious about their infrastructure. A particularly interesting feature of snap is the ability for custom data collectors, processors and publishers to be developed. This allows application specific metrics to be monitored in parallel with data about the underlying infrastructure - both physical and virtual.

The integration of snap on the HPC test bed is largely complete at the time of writing. Metrics have been successfully gathered, processed and published. However, an issue with the InfluxDB client library is currently being investigated that prevents a stable continuous monitoring.

3.3.3 OSv

The target guest operating system used in MIKELANGELO is OSv. Contrary to traditional linux-based OSs it has been specifically built for Cloud computing from scratch. Some of the most important features are:

- Optimized for running on top of a hypervisor: Xen, KVM or VMware.
- Single application, Single address space. The user application runs in the same context as the kernel so context switches become light-weight.
- JVM Integration, Java programs can also run in OSv.
- REST API through HTTP for VM management.
- Virtual network based on channels with low latency and high throughput.
- ZFS file system that includes support for high storage capacities and efficient data compression.
- Allows working with continuous integration systems and IDEs
- Easy deployment of VM images via the Capstan tool, also extended within MIKELANGELO project.

Two major advantages of using OSv are transfer times for VM images and boot time for VMs. OSv-VM images require only 12-20MB more than the application itself. Start-up times of OSv usually lie under a second. Consequently, running applications (processes) inside OSv-based images is almost the same as running these processes directly on host machines. This model has further been extended with the changes made to the Capstan tool. Capstan is a system for application deployment, and resembles Docker. In contrast to Docker, Capstan builds complete virtual machine images with the OSv unikernel. MIKELANGELO integrates Capstan to easily deploy applications with convenient interfaces.

3.3.4 Evaluation of the HPC-Stack

From the point of view of users, developers, administrators and infrastructure owners, the benefits of this architecture are manyfold.

An important implication of the MIKELANGELO software stack is that applications are provided as packages coming with their own custom, predefined environment. These package images are targeted to support both Cloud infrastructures and HPC infrastructures.

Depending on the kind of scenario and software - whether open source or proprietary - the users, administrators and software vendors need to compile applications for each HPC environment with different compiler flags, and deal with different CPUs to get the best performance for their applications. Operating system, kernel version, kernel configuration, as well as libraries available in a certain HPC environment may differ vastly from application requirements to deliver, for example, the most accurate results or the best performance.

MIKELANGELO offers developers, as well as HPC users/admins, the flexibility to prepare, test, debug and optimize applications on cheap commodity hardware, and then use them without any change in HPC environments, as well as Clouds.

Users with running jobs on failing systems as well as administrators will benefit significantly from the virtualization layer, as it will enable them to migrate or suspend and resume running applications in cases where maintenance is urgently required. Furthermore, checkpointing and restarting applications that require a longer runtime than available in certain HPC environments is also supported. The so-called wall-time for a job which may be a matter of hours or days can thus be accommodated.

Despite all these advantages, running HPC applications on Cloud is typically avoided due to the significant overhead costs that virtualization often implies. HPC applications, as the name indicates, are applications that require high performance, thus noticeable overhead is generally not tolerated, and not profitable. Even minimal reductions in performance can lead to significant additional expense.

Some features mentioned above, like checkpoint and restart, and live migration to spare nodes, are subject to future work. Other limitations also exist, such as that interactive jobs are currently not possible with virtual guests, since that requires modification of Torque's source code. Another limitation is the circumstance that even if the job actually runs in an HPC environment and the PBS variables are set in the guest, too, they will not be available to MPI applications. The reason for this limitation is the fact that whenever MPI detects a PBS environment, it expects a daemon running in the other guests.

This section describes which components are integrated and points out with functional tests what is already working. In addition, the initial performance data are provided to illustrate the progress of the integration of the different components developed inside MIKELANGELO.

3.3.4.1 Functional Tests

The following seven functional tests cover the integration at a high level. They test a certain feature that has been implemented in the Torque extensions and verify and validate its functionality.

1) **Test Case**: Test VM Instantiation
   Description: Submit a job with VM resource requests to Torque, check if VM(s) get booted before the actual job starts, and destroyed after the job has finished.
   Rationale: Test the functionality of the VM submission
2) **Test Case**: CPU count; CPU Pinning
   Description: Submit a Job with a lower CPU count than the physical machine. See that the calculating thread is not moved to another core.
   Rationale: Test the functionality of CPU Pinning - moving threads can harm the performance.
3) **Test Case**: Multi VMs per Node
   Description: Submit a job with more than one VM per node.
   Rationale: Test multiple VMs can run on one node.
4) **Test Case**: Interactive non-VM Jobs
   Description: Submit an interactive job, without a VM.
   Rationale: Test if the original functionality is unharmed.
5) **Test Case**: Qsub Submitted Jobs
   Description: Submit without new functionality.
   Rationale: Test if the original functionality is unharmed.

The results of these tests are provided in Appendix A.

3.3.4.2 Performance Measurements

To validate functionality and evaluate the performance of the HPC part of the MIKELANGELO software stack, the following measurements were taken with USTUTT’s use case, the cancellous bones HPC application.

A list of metrics that covers most of the implementation has been compiled to get deep insights into the HPC stack. These metrics cover aspects like I/O, network and CPU behaviors, to compare the physical machine with virtual guests. For more accurate results each measurement was taken several times and is taken on different physical machines with the same configuration. This setup allows minor performance variations due to minor physical hardware differences to be minimized and dropped out of the comparison. A key indicator is the plain runtime of the application: the faster it finishes, the more performant the MIKELANGELO software-stack works.

In order to run at small scale, the Cancellous Bones Simulation (as described in Deliverable D2.1 First Cancellous bone simulation Use Case Implementation strategy [5]) is set up in such a way that it can run on between 3 and 16 cores. To fully exercise these cores, and taking the actual cancellous bones dataset that is being analysed into consideration, the number of HPC domains to be calculated was set to be 28. This means that each core calculates at least one domain, but a few cores have to calculate two or more. The input data is always the same subset of the big domain cube to allow comparable results in each run. The simulation was submitted to Torque five times in a row and the simulation setup stayed the same for the domains.

Table 6. Cancellous Bones environment setup.

	Host	**VM**
Operating System	Ubuntu 14.04	
MPI Version	(Open MPI) 1.6.5	1.0.2ubuntu1 (openmpi_1.6.5)
Runtime library for GNU Fortran applications	4.8.4-2ubuntu1~14.04.3	

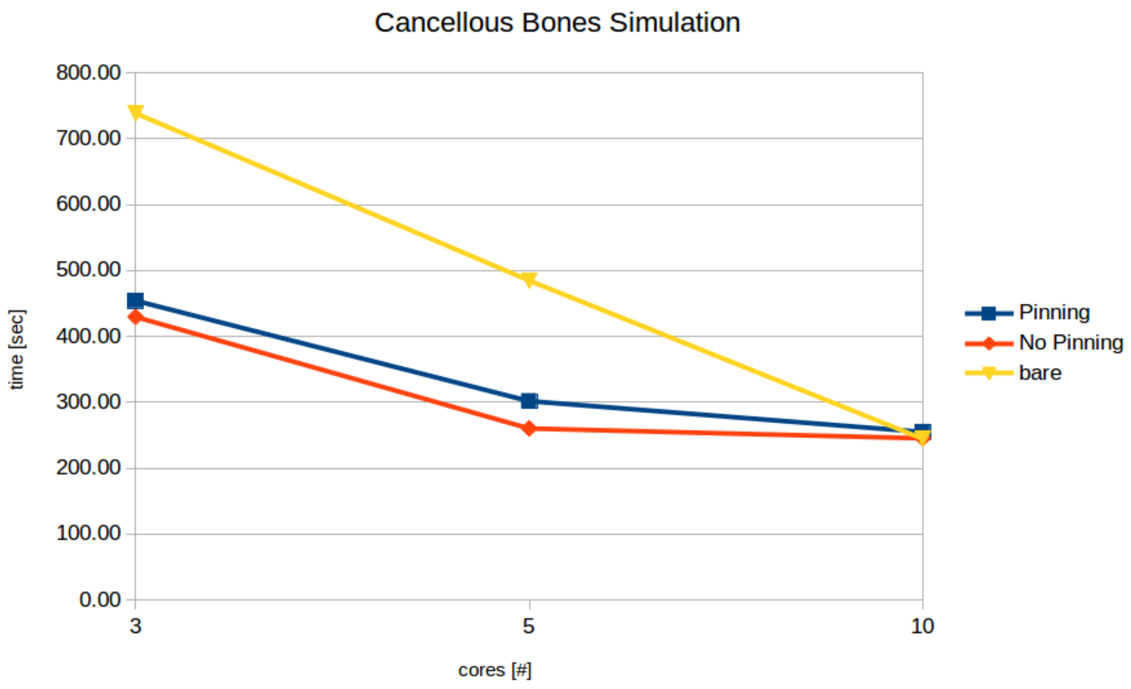


Figure 18. Runtime of the Cancellous Bones Simulation over different numbers of Cores.

Due to the scale down regarding total CPUs and the reduced data set, the behavior of the simulation at small scale is counterintuitive. The bare metal measurements are way slower than the VM counterparts, until 10 cores are assigned. Also, the pinned CPUs are slower. This behavior is not expected and will be further investigated and analysed.

Table 7. Runtime measurements of VM in seconds. Shaded cells are considered outliers.

# of cores	**Pinning**	**Run 1**	**2**	**3**	**4**	**5**	**Average**
3	Yes	450.182	7657.472	462.905	451.429	452.738	454.313
	No	435.878	430.644	437.273	7632.577	414.950	429.686
5	Yes	355.609	286.970	280.625	283.963	7490.347	301.7991
	No	248.824	274.856	281.903	247.289	248.761	260.326
10	Yes	277.843	236.386	252.693	278.626	229.831	255.075
	No	241.514	237.140	247.203	242.982	258.187	245.565

Table 8. Runtime measurements without VM (bare metal) in seconds.

# of cores	**Run 1**	**2**	**3**	**4**	**5**	**Average**
3	441.297	811.929	800.799	813.715	826.260	738.800
5	569.638	544.075	576.595	367.941	366.035	484.856
10	238.477	254.754	244.834	249.057	236.006	244.625
16	251.883	241.773	258.975	240.374	249.820	248.565

The three values for VMs with 3 cores and VMs with 5 cores that caused values bigger than 7000 seconds are treated as outliers and have not been considered for the calculation of the average runtime.

For the next period we need a more fine-grained view on the use case as well as the HPC part of the MIKELANGELO software-stack. Therefore more metrics are needed, such as those listed below.

List of metrics:

- Network - measure connections and communication between nodes
    - `Bytes_recv`
    - `Bytes_sent`
    - `Packets_recv`
    - `Packets_sent`

- CPU - measure usage of the CPU resources
    - `avg-cpu/%user`
    - `avg-cpu/%idel`
    - `avg-cpu/%system`

- Disc - measure File I/O behavior
    - `rkB_per_sec`
    - `wkB_per_sec`
    - `Avgrq-sz`
    - `Avgqu-sz`

- RAM - measure usage and distribution of the memory resources
    - `Available`
    - `Buffers`
    - `Free`
    - `Cached`
    - `used`

These metrics will be collected on the host system, as well as inside the VMs. Intel's snap monitoring framework is capable of collecting these metrics and will enable a deeper insight to be gained into the infrastructure and hosted applications.

3.3.4.3 Evaluation of MPI and NFS inside OSv

To evaluate the implementation of the full stack of HPC software an MPI+NFS OSv image has been built. The build command used was the following one:

```
scripts/build image=OpenMPI,openmpi-hello,hosts,cli,httpserver \
             -j16 nfs=true
```

The test code is based on the MPI Hello World application that can be found under the `/osv/mike-apps/openmpi-hello` directory in the OSv source code repository. This MPI Hello World has been modified for the MPI processes to write into a shared file instead of the standard output.

Four OSv-VMs were created and allocated across two physical nodes, with two VMs per node. In this way intra-node and inter-node MPI communication is tested. This particular test doesn't use Torque to build the execution environment in order to simplify the isolation of possible errors. Below is the output file of the test execution:

```
user@node0101:/scratch$ cat ompi-hello.out
Hello world! I am process number 0 of 4 on VM: 172.18.2.247
Hello world! I am process number 1 of 4 on VM: 172.18.2.239
Hello world! I am process number 2 of 4 on VM: 172.18.2.231
Hello world! I am process number 3 of 4 on VM: 172.18.2.223
```

Each of the four VMs mount a shared file-system in the directory `/scratch` using NFS at boot time. Then the MPI program is launched in the first VM and spawns automatically to the other virtual nodes. Once all the MPI processes are running, that is, one MPI process per VM, the MPI processes wait in a synchronization barrier before printing their output into the shared file as shown above.

A more complex scenario with more VMs and physical nodes could help to reveal possible scaling issues related to MPI or NFS on OSv that are currently hidden. However, when repeating the described test, failures have sometimes been observed. The error reported then is:

```
----------------------------------------------------------------------
ORTE has lost communication with its daemon located on node:

  hostname:  172.18.2.247

This is usually due to either a failure of the TCP network
connection to the node, or possibly an internal failure of
the daemon itself. We cannot recover from this failure, and
therefore will terminate the job.
----------------------------------------------------------------------
program exited with status -108
```

Successive runs of the test also fail with the following error, a result of the fact that the shared directory can no longer be mounted:

```
Failed to load object: tools/mount-nfs.so. Powering off.
```

We are continuing with the investigation of these issues to stabilize integration of MPI and NFS into OSv. However, preliminary tests already look encouraging.

3.3.5 Conclusions and Next Steps

Most of the components for the HPC-Stack are integrated in a first version. In the following months these components will be refined and finalized, and their integration completed. The process of installing the Torque modifications will be further automated and documented.

More fine-grained data will be collected to get a deeper understanding over the performance bottlenecks of VMs compared to bare-metal execution, but already almost 3.500 jobs have been submitted via Torque in the process of developing, testing and evaluating this emerging HPC stack.

4 Use Case Architecture and Implementation Evaluation

4.1 Introduction

Four use cases that span cloud computing and HPC drive the requirements, evaluation, and verification of MIKELANGELO's overall architecture. This chapter describes these case studies, highlighting advantages, considerations and limitations discovered or confirmed when putting the MIKELANGELO stack to work.

The cloud bursting use case targets improvements in dealing with bursts of requests for internet services. There are two important metrics that drive how well a cloud handles cloud bursts: transfer times for VM images and boot times for VMs.

The first HPC use case deals with the simulation of cancellous bones on a virtualised environment and has already been described in Section 3.3.5 as part of the HPC Stack evaluation.

The second HPC use case runs simulations in computational fluid dynamics with OpenFOAM. In this use case OpenFOAM is ported to OSv and combined with sKVM and RDMA.

The Big Data use case primarily targets Apache's big data stack including Hadoop [6], which will be managed using OpenStack Sahara [70].

These use cases test how effectively the emerging MIKELANGELO architecture is providing a powerful framework to increase the performance, security, and flexibility available to users and administrators in the world of Cloud and HPC.

4.2 Case Study: Cloud Bursting

4.2.1 ScyllaDB

ScyllaDB is the first big and real-world application written using Seastar. It acts as a Seastar showcase allowing the ScyllaDB team to exercise and enrich the framework. It also brings a viable revenue stream to the company behind the framework and the database.

In essence ScyllaDB is a faster Cassandra drop-in replacement.

The following benchmark compares Scylla and Cassandra on a small cluster with replication factor 3 and statement consistency level QUORUM [71].

4.2.1.1 Test Bed

The test was executed on physical machines. The following configurations were used:

3 DB servers (Scylla / Cassandra):

- Bare metal server
- CPU: 2x 12-core Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz, with hyperthreading
- RAM: 128 GB
- Networking: 10 Gbps
- Disk: MegaRAID SAS 9361-8i, 4x 960 GB SSD
- OS: Fedora 22 chroot running in CentOS 7, Linux 3.10.0-229.11.1.el7.x86_64
- Java: Oracle JDK 1.8.0_60-b27
- Scylla version: 0498cebc58b9fbadb25a7b018cebf95d965d88da
- Cassandra version: 2.1.19

22 load servers (cassandra-stress):

- VM server
- XEN hypervisor
- CPU: Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz, 16 logical cores
- RAM: 16G
- Networking: 1 Gbps
- OS: CentOS 7, Linux 3.10.0-229.7.2.el7.x86_64

All machines were located in the same data center.

4.2.1.2 Workloads

Three workloads were tested:

- Write only:

```
cassandra-stress write cl=QUORUM duration=15min \
                 -mode native cql3 \
                 -rate threads=700 -node $SERVERS
```

- Read Only:

```
cassandra-stress mixed cl=QUORUM \
           'ratio(read=1)' duration=15min \
           -pop 'dist=gauss(1..10000000,5000000,500000)' \
           -mode native cql3 -rate threads=700 -node $SERVERS
```

- Mixed: 50/50 Read/Write:

```
cassandra-stress mixed cl=QUORUM \
           'ratio(read=1,write=1)' duration=15min \
           -pop 'dist=gauss(1..10000000,5000000,500000)' \
           -mode native cql3 -rate threads=700 -node $SERVERS
```

4.2.1.3 Cassandra results

The following table contains statistics of total operations per second by workload in the whole cluster:

Table 9. Queries metrics for Cassandra cluster.

Workload	**Average**	**Stdev**	**Min**	**Max**
write	125,224	12,382	105,436	166,314
read	48,291	19,238	26,631	98,353
mixed	65,950	20,179	1,592	88,847

4.2.1.4 Scylla results

The following table contains statistics of total operations per second by workload in the whole cluster:

Table 10. Queries metrics for ScyllaDB cluster.

Workload	**Average**	**Stdev**	**Min**	**Max**
write	1,930,833	3,190	1,650,625	2,010,829
read	1,951,835	1,873	1,943,209	1,955,225
mixed	1,552,604	68,185	1,094,988	1,651,162

The following figure shows average results from the above two tables in a chart to show the performance improvement of this benchmark.

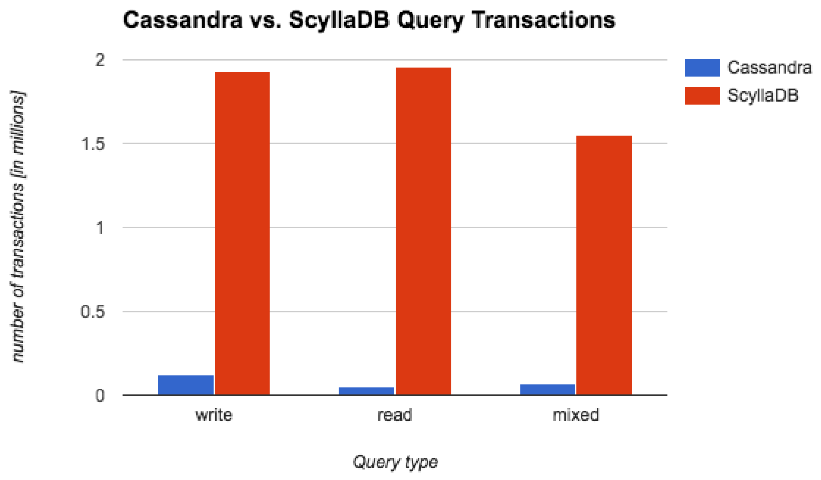


Figure 19. Comparison of the number of transactions in Cassandra and ScyllaDB cluster.

An external and independent benchmark comparing ScyllaDB against the Cassandra is presented in [72]. Relatively small machines have been used for the test, however even this small cluster shows that ScyllaDB outperforms Cassandra by a factor of two for write operations. As suggested above and also in the comments to this benchmark, even more benefits are expected in large clusters with powerful CPUs.

4.2.2 Evaluation and Validation of the Cloud bursting use case

One of the key concepts of cloud services is elasticity. Elasticity is the capacity to quickly grow or shrink the size of a cloud computing resource allocation. When the workload suddenly increases the computing resource allocation can grow, consuming more hardware, and when the workload decreases the computing resource can shrink, consuming less hardware. Elasticity is in essence the core of the cloud computing promise. To do so the computing resource (in the scylla case a database) must be able to evolve in size very dynamically without altering its nominal behavior. The extreme case of elasticity is given by Amazon Lambda [73] because it allocates computing resources to execute code snippets on the fly and the granularity of the allocation is very small (a code function). Here in the cloud bursting use case elasticity must be a property of a ScyllaDB [57] cluster (A group of machines working together to serve a ScyllaDB database). The performance of the regular usage of the resource must stay the same during the growth.

The scylla-cluster-test [74] tests available on GitHub have been enhanced with a custom script grow_cluster_test.py that triggers a cloud bursting use case in the EC2 infrastructure.

A minimal 3-node cluster will be started and then grown to either 4, 5 or 30 nodes in a minimal amount of time. This unit test execution time is measured and compared against Cassandra before being graphed out.

The ScyllaDB cluster will need to grow quicker than Cassandra while keeping its regular use case performance. The first point will be addressed trivially by the fact that Scylla is faster than cassandra; the second point is to be addressed.

The following evaluation shows an intermediate result of the work on the cloud bursting use case (full comparison is not yet possible at this point). The two graphs compare the behaviour of two different versions of ScyllaDB while adding a new database node to the cluster. The first graph (Figure 20) shows a huge decrease in the number of operations when streaming occurs, while the second one shows a smoother transition as a result of our improved implementation of the Seastar framework and ScyllaDB database.

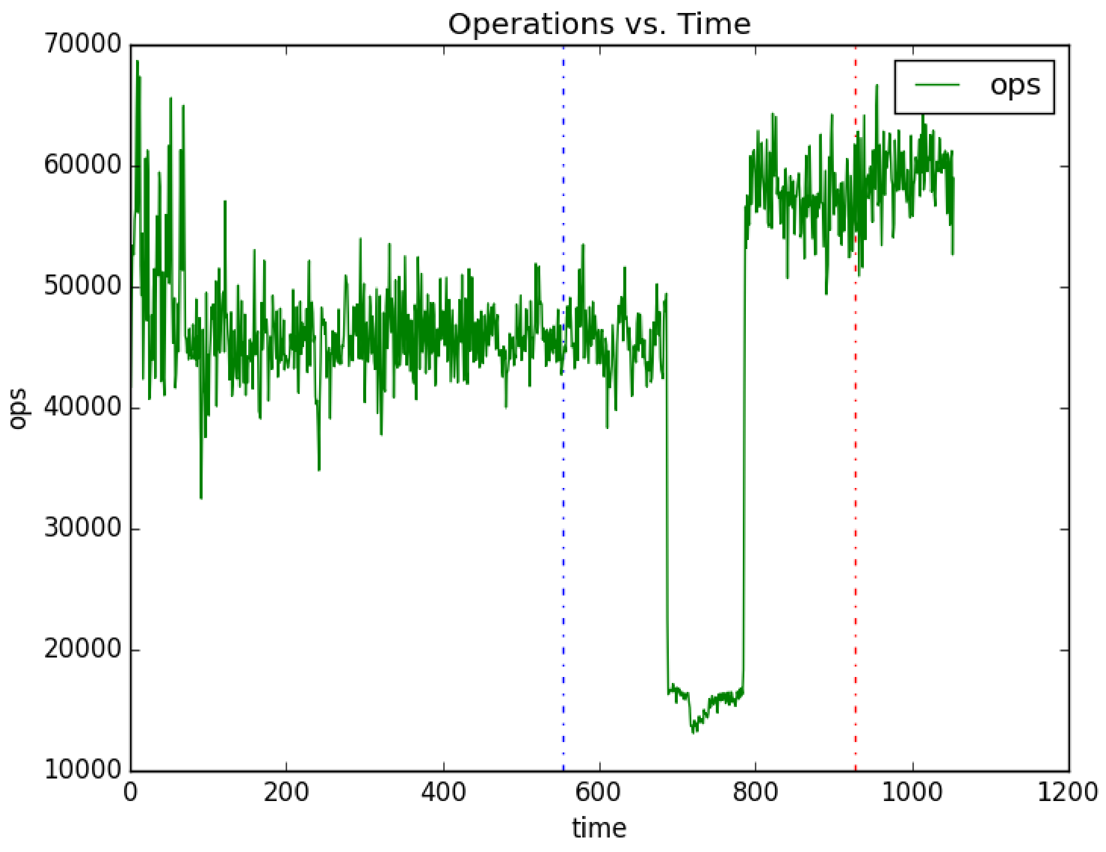


Figure 20. Reduced performance during expansion process with ScyllaDB 1.1.

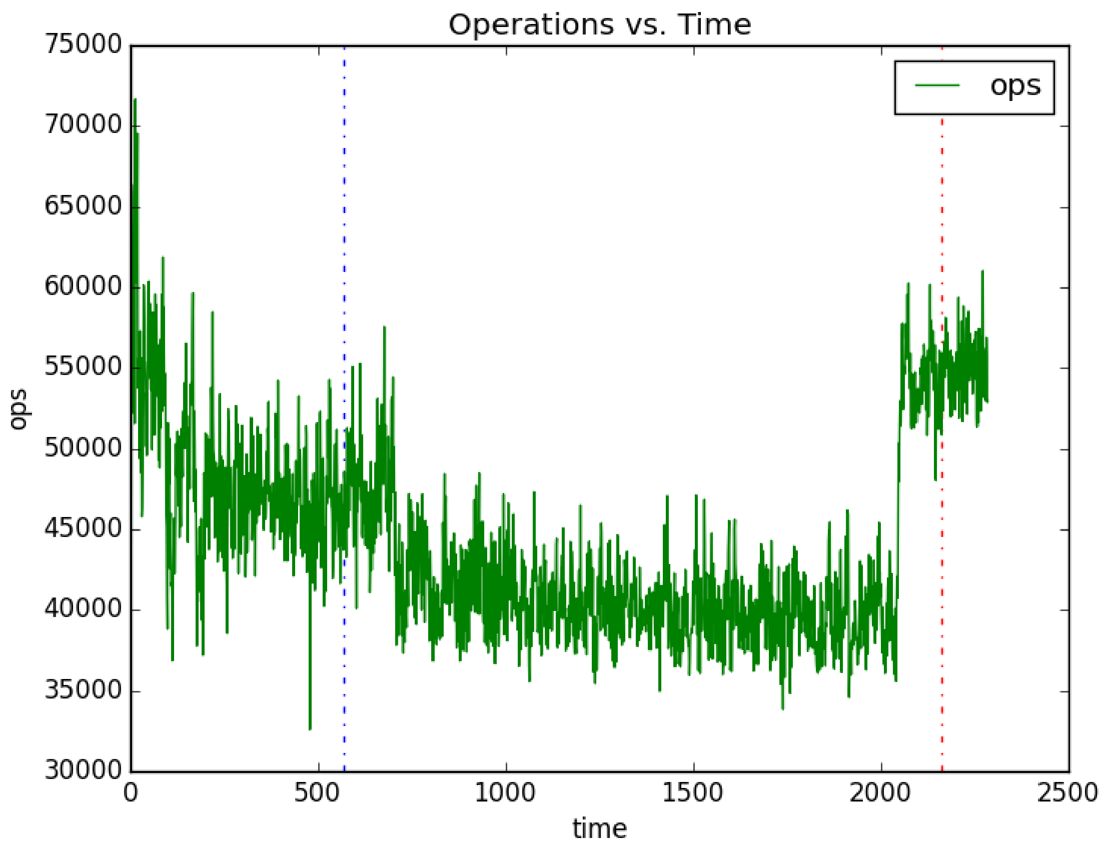


Figure 21. Almost no performance loss during expansion process with ScyllaDB 1.2.

More work is being done to improve both Seastar and ScyllaDB at the time of writing to reduce these op/s cliffs. Further progress in this cloud-bursting use case will be documented in future deliverables.

4.3 Case Study: OpenFOAM Cloud

One of the most significant benefits of the MIKELANGELO project lies in its thoroughness in addressing the entire technology stack. It starts by improving the I/O performance of KVM, one of the most widely used hypervisors, and empowering the effectiveness of a lightweight unikernel (OSv) suitable for running HPC and big data workloads. These underlying technologies are then integrated into a fully virtualized batch system (Torque [69]) and a cloud management system (OpenStack [35]). On top of that, MIKELANGELO integrates snap [8] telemetry into all layers of the architecture seamlessly offering insights into every aspect of the system.

The Aerodynamics Map use case has been introduced in Deliverable D2.10 The First Aerodynamic Map Use Case Implementation Strategy [5]. This use case, provided by MIKELANGELO partner Pipistrel, the manufacturer of light aircraft, has a goal to be among the first to evaluate the performance and manageability improvements of the MIKELANGELO project.

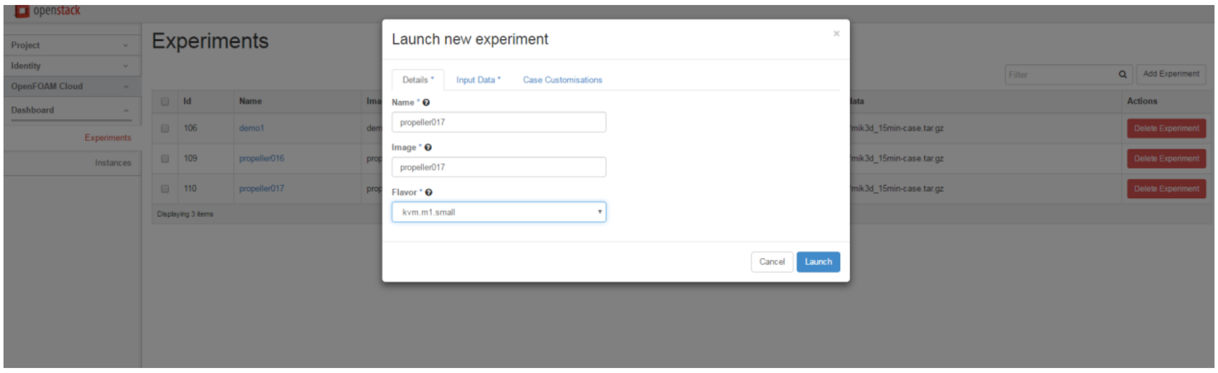
The use case is based on OpenFOAM [4]. OpenFOAM is a popular open source toolbox for Computational Fluid Dynamics (CFD), allowing complex simulation and evaluation of various flows (air, liquids, gases). OpenFOAM is ideal when it comes to designing new airplanes or even just improving parts of existing airplanes.

However, one of the challenges scientists are faced with is the way they are supposed to interact with the OpenFOAM applications. Traditionally, users interact with OpenFOAM - pre-process, process, post-process and visualising their scenarios - through terminal access. This can be provided by an internal (private) infrastructure or an HPC provider. In recent years this has changed with public cloud providers such as Amazon AWS and Microsoft Azure offering powerful, HPC-ready virtual machine instances. The advent of these machines allows end users to rent virtualised infrastructure and run their simulations whenever they are required - no more cumbersome contracts and waiting for core-hours to be allocated to the user, a frequent issue for smaller customers in particular.

This opened the door to new kind of a service offering: scientific computing in the cloud. The UberCloud [75] project is among the first ones that started offering specialised virtual machine images wrapping all required software components for specific public cloud providers. This literally allowed end users to launch their simulations within minutes, dramatically lowering the costs of renting the infrastructure. The UberCloud also offers expert services supporting their users with expertise in design and execution of OpenFOAM simulations. A similar approach is offered by CFD Direct Cloud [76].

SimScale [77] goes even further and provides the entire ecosystem for CFD computations (OpenFOAM) and some other types of simulations. SimScale is a web-based application supporting users throughout all phases of the scientific computation: design, pre-processing, processing, post-processing and visualisation.

Based on the analysis of the aforementioned services and the MIKELANGELO integration plan we have designed and implemented an initial version of a similar service - OpenFOAM Cloud. OpenFOAM Cloud is currently a prototype serving primarily testing purposes in the MIKELANGELO project. It exposes OpenFOAM functionalities through a lightweight OpenStack dashboard. The following figures present some of the functionalities of this dashboard.



Experiments

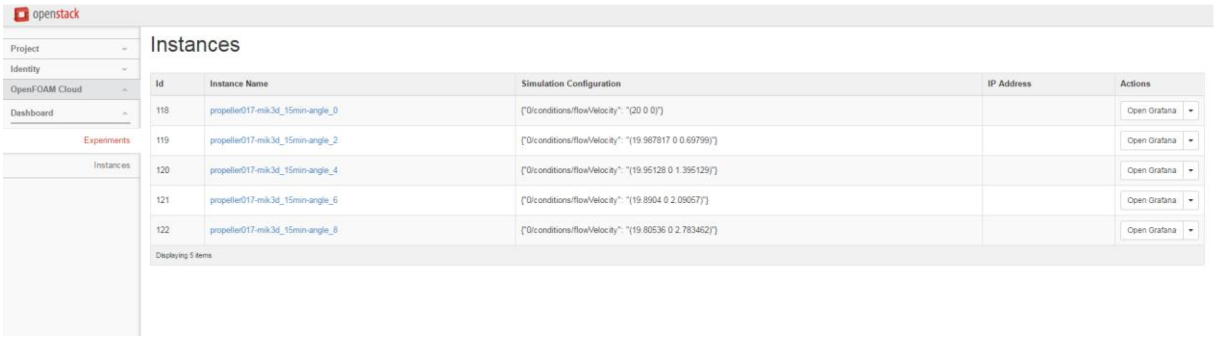
	Id	Name	Ima	ata	Actions
	106	demo1	dem	mik3d_15min-case.tar.gz	Delete Experiment
	109	propeller016	prop	mik3d_15min-case.tar.gz	Delete Experiment
	110	propeller017	prop	mik3d_15min-case.tar.gz	Delete Experiment

Displaying 3 items

Launch new experiment

Name * propeller017
Image * propeller017
Flavor * kvm.m1.small

Figure 22. Launching an experiment is integrated into OpenStack Horizon dashboard.



Instances

Id	Instance Name	Simulation Configuration	IP Address	Actions
118	propeller017-mik3d_15min-angle_0	{"0/conditions/flowVelocity": "(20 0 0)"}		Open Grafana
119	propeller017-mik3d_15min-angle_2	{"0/conditions/flowVelocity": "(19.987817 0 0.69799)"}		Open Grafana
120	propeller017-mik3d_15min-angle_4	{"0/conditions/flowVelocity": "(19.95128 0 1.395129)"}		Open Grafana
121	propeller017-mik3d_15min-angle_6	{"0/conditions/flowVelocity": "(19.8904 0 2.09057)"}		Open Grafana
122	propeller017-mik3d_15min-angle_8	{"0/conditions/flowVelocity": "(19.80536 0 2.783462)"}		Open Grafana

Displaying 5 items

Figure 23. OpenFOAM simulations dashboard.

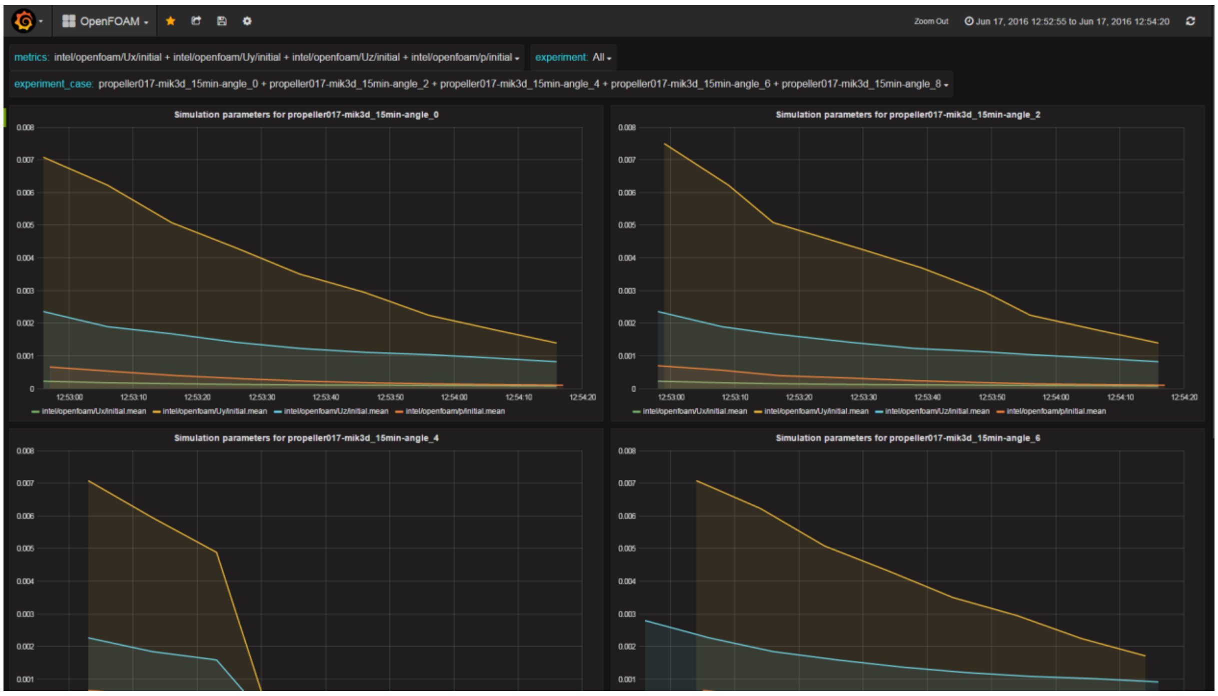


Figure 24. Visualisation of simulation parameters in Grafana, collected with snap.

The OpenFOAM Cloud application integrates several components of the MIKELANGELO stack. These are presented in the following subsections.

4.3.1 Integration Evaluation

The purpose of this evaluation is to assess the complexity of using different components developed by the MIKELANGELO consortium.

4.3.1.1 Running OpenFOAM in OSv

OSv executes applications by loading them dynamically and running them within the same process as the OSv kernel (OSv is a unikernel). Consequently applications must be provided as shared objects or libraries that can be reallocated by the kernel. Typically, this involves recompilation of the application sources by setting certain compiler and linker flags. Although OpenFOAM is a massive open source project, the same strategy has proved successful. Some minor issues were immediately detected due to missing dependency specification in OpenFOAM source code. Once the application was properly built, we were able to execute the simple solver (simpleFoam) on real input data.

All patches and build recipes are provided in the MIKELANGELO application repository [17]. The OpenFOAM core and simpleFoam solver are provided as an MPM package as well.

According to preliminary benchmarks, execution time of simple OpenFOAM solver results in up to 5% performance loss when compared to a Linux-based virtual machine. This performance loss is still to be analysed as similar results have been observed in some other cases.

4.3.1.2 Parallelisation of OpenFOAM in OSv

OpenFOAM relies heavily on Open MPI [25] for parallelisation of the workloads. Parallelisation is achieved by decomposing the entire simulation into several smaller ones. Each of these are processed in a separate Open MPI process sharing information with other processes.

The underlying execution model of Open MPI is in contradiction to the core concept of OSv unikernel, namely no support for multiple processes. As opposed to ensuring OpenFOAM runs on OSv, support for Open MPI in OSv was much more challenging. However, since a majority of widely used HPC workloads rely on Open MPI, an open source implementation of the MPI specification, the MIKELANGELO project addressed these issues. This resulted in several enhancements of the OSv kernel simulating processes with independent threads.

Further changes to Open MPI were required, primarily to integrate the starting of independent OSv threads instead of processes. These changes are provided as a dedicated module for Open MPI. This allows the use of an unmodified Open MPI with OSv or in traditional systems.

Preliminary results show that modified Open MPI for OSv does not represent any additional overhead compared to a default configuration of Open MPI in Linux-based virtual machines. However, a more detailed evaluation, considering various optimisation options of Open MPI and virtualisation, have shown significant improvement in execution in Linux while no improvement or even performance degradation was detected in OSv. The main reason for this is the fact that Open MPI is capable of optimising the process allocation (core and memory pinning) based on the information from the operating system. OSv, on the other hand, does not provide the required information to Open MPI preventing such optimisation.

Results are shown in figures Figure 25 and Figure 26. The tests are using virtio-net for communication between VMs on different hosts. Similarly to the cancellous bones, interesting observation is that for a single worker the guests perform better than the host even on multiple runs of the same test (different nodes were used to minimise the effect of hardware or software differences). OSv-based guests perform slightly slower than Ubuntu ones. Exact reasons for this are subject of further research. OpenFOAM and in particular the specific input cases are CPU intensive calculations with insignificant disk or network load.

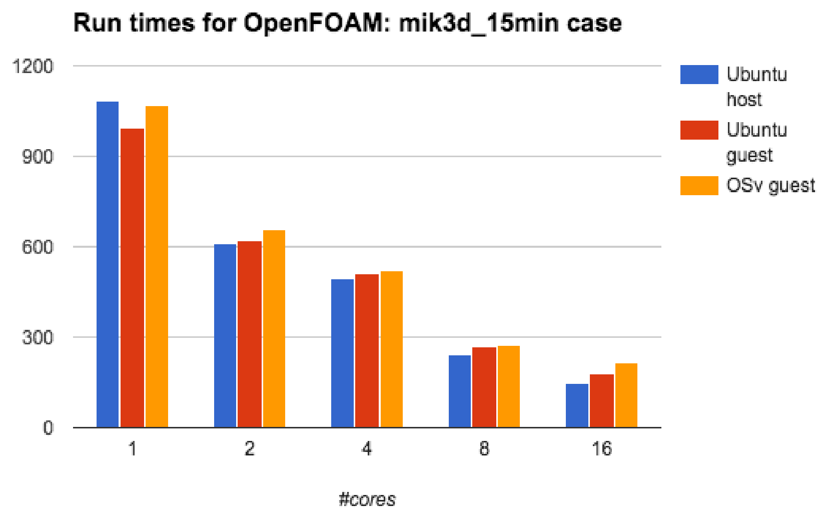


Figure 25. Comparison of run times required by OpenFOAM on a small input case.

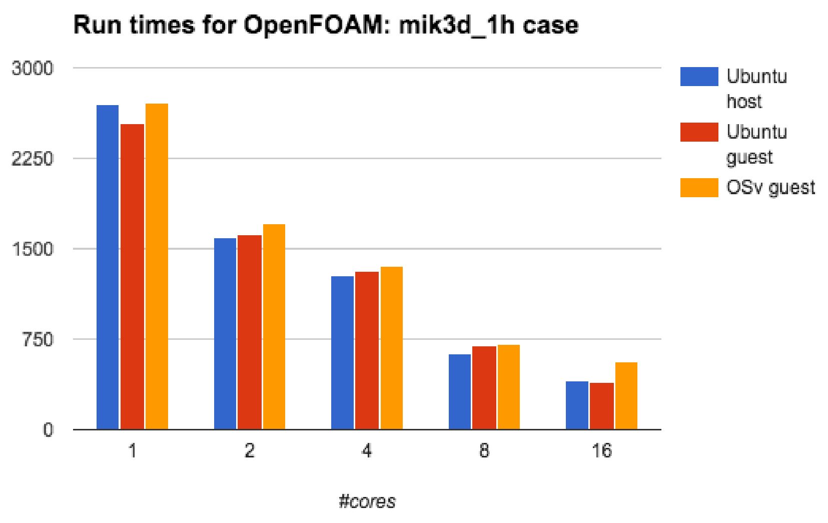


Figure 26. Comparison of run times required by OpenFOAM on a medium input case.

The most notable performance degradation can be spotted in case of 16 workers. In this case, 2 VMs were used on two different physical nodes. Each VM uses all 8 of the available physical cores. The problem of this configuration is that the OSv currently does not expose the information about the underlying CPU topology: in case of USTUTT testbed, processors are dual-socket with 2 NUMA nodes. Because the NUMA information is not exposed, Open MPI is not able to optimise the deployment of worker threads in OSv in the same way as in case of Ubuntu.

Despite the fact that Ubuntu currently outperforms OSv one major advancement of the MIKELANGELO project is the sheer fact that applications such as OpenFOAM relying on complex MPI-based middlewares are working almost unmodified. Future versions of Open MPI and OSv are going to address these issues based on a thorough analysis of the possible reasons for performance loss.

Additional evaluation comparing the two images (Ubuntu and OSv) has been conducted (Table 11). The first table row compares corresponding image files (both images in QCOW2 format [13] and containing only the required packages). The remaining rows show times required for different operations during the spawning of virtual jobs:

- Time to copy VM: time required to copy VM to target machines. In case shared storage is used, it is still required that the image is copied so that each instance uses its own copy.
- Time to start VM: VMs are started using libvirt. The time here represents the time required by the *virsh* tool to return from the *virsh start <vm-domain-file>*. The VM is not actually ready for execution at this point.

- Time to phone home: The amount of time that the VM takes to initialise and respond to the call back. In case of OSv, the time is measured as the time required to successfully connect to OSv's HTTP server, while in case of Ubuntu, this equals to time required for a successful connection via SSH.

Table 11. Comparing OSv-based image to equivalent Ubuntu-based image.

Metric name	OSv	Ubuntu
Image size [MB]	99	1100
Time to copy VM [s]	0.22	6.15
Time to start VM [s]	0.58	0.76
Time to phone home [s]	4.16	7.97

It is obvious that the times do not play significant role in large cases, however the table shows some of the benefits of using OSv as the underlying operating system.

4.3.1.3 sKVM

OpenFOAM does not require any specific integration with sKVM because the I/O optimised hypervisor does not change any application-facing interface. Because the OpenFOAM workload used in current experiments is not I/O bound, no performance impact has been measured depending on the underlying hypervisor.

Future reviews will include I/O intensive workloads allowing in-depth analysis of I/O optimisation on the hypervisor layer.

4.3.1.4 OpenFOAM Application Packages

Preparation of OpenFOAM application packages using MPM was trivial because an MPM package can be built directly from the content of the given directory. However, in order to reduce the size of the package, we have removed all intermediate files or files that are not used by specific application solvers. This resulted in two packages, one containing the core OpenFOAM framework libraries and configuration files and the other only the simpleFoam solver application.

MPM furthermore simplifies execution of test cases in a local environment. This is an extremely important feature of the tool allowing either the developer building the application package or the end user to test the behaviour of the package and input data. MPM is still missing integration with external services, such as OpenStack's image and compute service.

These are planned in future versions, as described in Deliverable D2.20 The Intermediate MIKELANGELO Architecture [5].

4.3.1.5 Telemetry with Snap

One of the main benefits of snap is the flexibility provided by its plugin API. For the purposes of the OpenFOAM application running inside OSv-based virtual machines an advanced data collector has been implemented. The collector is using the management API of the OSv kernel to gather the current state of the simulation. The state is analysed and the most relevant data pushed into a time series database.

The OpenFOAM web application integrates seamlessly with snap via its REST API. The snap collector is appointed to each submitted simulation. The data get collected as soon as the instance becomes active allowing users to review the progress over time. For simpler presentation, the popular Grafana real-time visualisation suite is integrated into the dashboard.

4.3.2 End-user Evaluation

From the end-user's point of view (Pipistrel) the prototype of OpenFOAM Cloud application implemented as an OpenStack dashboard represents an excellent starting point that has the potential to evolve into a very useful and user-friendly application in the remainder of the project. The foundations are set correctly, the main focus in the next phase must be directed towards proper plotting of intermediate simulation data and end results.

The application builds upon an existing template OpenFOAM case that has to be prepared in advance and uploaded to the OpenStack dashboard. Currently the case is uploaded to a dedicated storage service (for example OpenStack Swift or Amazon S3), but a more natural way of submitting the data would be to upload the data along with the submitted simulation request. The final version should also include a small number of verified test cases that would shorten the learning curve for the new user.

The next step is to choose the set of parameters to be varied in the parametric study. The current, rather cumbersome, procedure is to provide the specification of a set of parameters as a JavaScript Object Notation (JSON) object together with their location in the OpenFOAM case. The plan is to migrate to a new input document, with specified parameter values in a spreadsheet-like column fashion. Additional instructions will have to be provided for the end-user explaining how to properly set the input document. Another possibility would be to integrate the pyFoam utility [78], a Python script that is used to preprocess an OpenFOAM case template and deploy multiple instances with different configurations (parameters). The latter option needs to be explored. Configuration of required resources should also be simplified allowing end users to focus only on their computation (without actually understanding the underlying infrastructure).

After all the OpenFOAM cases are deployed to the cloud, the user can check the status of each case (instance) in the “Instances” tab. Here, an Id number is appointed to each case. A name and parameter set-up is also presented for each instance. The user has a possibility to check the log file of each instance with a unix “tail” like command. Another possibility to follow the simulation progress is through a Grafana interface, which displays data collected by snap collectors. Currently it can display residuals (numbers that indicate convergence of a simulation; refer to Deliverable D2.2 Intermediate Use Cases Implementation Strategy [5] for more details) of several physical quantities for all instances. It is a great tool for a researcher to follow the convergence of each simulation, but if a large number (hundreds) of simulations are deployed, it becomes impossible to check the convergence of all instances. A smart way of displaying the collection/aggregation of all residuals and also aerodynamic forces (they represent the most important end result of each simulation) needs to be implemented in order to simplify the researcher’s work. A collection of aerodynamic forces with respect to specific parameter values would also represent an end result of a complete analysis. More on this topic is available in D2.2.

Another important attribute of each instance is its status. The end user should be aware of the status of each instance already by checking the Instances tab, without having to display the log file or the data collected by the snap application. Each instance should therefore have a coloured status column. Possible status descriptions could be “running”, “finished” and “stopped”. “Finished” coloured in green would mean that the simulation converged, whereas “stopped” coloured in red would mean that the simulation was stopped prematurely for some reason. The latter could be the result of an invalid OpenFOAM case set-up or just because the simulation diverged.

The design of OpenFOAM Cloud will in the remainder of the project try to address just the presented improvements - these are from the end-user’s point of view the most important ones. Additional and less important improvements concerning transferring final results back to the end-user’s computer, handling new simulations, and descriptions and suggestions that would help the new end-user, will have to be addressed subsequently.

4.4 Case Study: Big Data Stack in OSv

Apache Big Data technologies are at the core of the Big Data use case. The initial evaluation has been done using Hadoop HDFS and Apache Storm projects. HDFS (Hadoop Distributed File System) sits at the core of any Hadoop based computation providing efficient and secure data storage. Apache Storm on the other hand provides a framework for stream processing of real-time data. Both projects are open source projects written in Java. Because OSv initially evolved around the Java Virtual Machine (JVM) and Cassandra application, these two projects are ideal for additional evaluation of the modified kernel.

4.4.1 Running with OSv

Similar to the OpenFOAM use case we are going to start the evaluation describing the effort necessary to support these technologies in OSv. The benefit of being built on Java is that no additional recompilation is needed. Both applications can be packaged into a target virtual machine image and executed using the same command line as used in the Linux operating system. The only notable differences when specifying the command to execute in OSv are the locations of various configuration and data files and directories. This observation is extremely valuable to users interested in running their Java workloads in lightweight OSv-enabled virtual instances.

Once the applications have been successfully started in OSv-based virtual machines, we were able to validate them using standard testing facilities. To do this, we have deployed two actual environments for Storm and HDFS:

- Apache Storm [7]: centralised Nimbus [79] and Zookeeper [80] were deployed and linked. To test this deployment we have first tested Storm supervisors running Linux virtual machines.
- Hadoop HDFS: we have deployed a central Namenode [81]. Data nodes running in Linux guest VMs have been used to validate the configuration.

Following the initial validation, we were ready to move to OSv-based applications. The following sections describe each application in more detail.

4.4.1.1 Apache Storm

Initially Storm supervisors launched without errors and properly joined the cluster through Zookeeper and Nimbus. However, as soon as the first Storm job was submitted, the OSv instance terminated. The supervisor tried to fork itself and run a shell command that would in turn run a new JVM (Java Virtual Machine) with the actual worker. Because multiple processes are not supported in a single OSv instance, we have started to explore two approaches:

- Instead of forking the JVM, it is possible to start a new thread in the same OSv instance. The thread must start a new JVM because the supervisor and the worker must not share the environment (for example, their JVMs have different properties). A stripped down demo application has been prepared specifically for this and a patch for OSv's Java interface created.
- The supervisor itself is merely a simple process listening for requests for executions and monitoring the underlying worker processes. Instead of creating a second JVM in

the same OSv instance, it could also launch a completely separated OSv instance with the new Java command.

At the time of writing this report, neither of the approaches have been realised in full. However, the two approaches already show possible routes for porting applications to OSv. The latter approach is much more in line with the unikernel philosophy where individual components are independent instances communicating over a common (network) channel.

4.4.1.2 Hadoop HDFS

Contrary to Storm, the default version of HDFS (2.7.2) [6] has failed immediately after the application has been started. A thorough analysis of the HDFS source code has revealed that it is trying to get information about the filesystem using two shell commands: *ls* to get directory permissions and *du* for estimation of data directory utilisation.

Permissions play no role in OSv because an instance is running a single application owned by a single user. There is therefore no way that a user running the application would not be granted access to a data directory. Thus the permission check was resolved by skipping it in the source code. Directory usage required reimplementation of the function to use a pure Java implementation. Instead of invoking a shell command in a different process (or thread) this new function recursively analyses the data directory tree and returns its size.

Both modifications are provided as separate patches. A pre-built HDFS application package is also provided [82].

4.4.2 Performance Evaluation

For the performance evaluation of big data tools the HDFS application has been used. The following table shows benchmark results for different configurations. In order to reduce external influences, a single data node was used with a single name (master) node. Linux and OSv virtual machines were all configured using exactly the same virtual hardware specification. Each run wrote 16 files (1GB in size) in parallel, using TestDFSIO test suite [83].

Table 12. TestDFSIO results in various guest operating systems.

Configuration	**Time**	**Throughput [mb/s]**	**Average I/O rate [mb/s]**	**I/O rate standard deviation [mb/s]**	**Time ratio**	**I/O rate ratio**
Linux Guest	190	95	115	61	1.00	1.00
Unmodified OSv	409	41	42	7	2.15	0.37

OSv with ZFS locking improvement patch	211	81	82	10	1.11	0.71
OSv with an LZ4 compressed ZFS	45	389	394	37	0.24	3.43

The Linux Guest is based on Ubuntu 14.04 cloud image with the kernel 3.19.0, while the unmodified OSv used the upstream OSv version 0.24 (GitHub commit 30fe998, June 7 2016). Since there were no performance improvements in OSv prior to this commit we refer to this as a baseline for OSv HDFS benchmark. All images used the RAW image format to prevent biased differences.

The last two columns show that the unmodified OSv produces significantly worse results than Linux which alerted the consortium to examine the software for possible bottlenecks. The first improvement includes a patch for the integration of ZFS with OSv's virtual file system layer. The patch removes several locks inside ZFS read and write operations. Because these locks were acquired for the entire duration of reads and writes, parallel operations were essentially serialised. With this patch, the overall running time has nearly reached the one achieved by the Linux guest.

Still, observing the ratio of writes made to the physical drive (using iostat tool) revealed that OSv is doing much more writes than Linux (nearly twice as many). We therefore enabled the LZ4 compression of the ZFS file system (last row in the table above). Enabling the compression creates smaller files requiring less actual writes to the physical storage device. However, the experiment and the results from the last row of the table are somewhat misleading. The TestDFSIO tool that comes with the Hadoop HDFS is storing data in multiple files containing nothing but zeros. Using LZ4 compression a 1 GB file is compressed to a file of size 4 MB.

Further investigation of the HDFS on top of OSv is required. The first important conclusion is that HDFS works with minor changes to the upstream source code and that an additional patch to the OSv kernel reached nearly the performance of the Linux virtual machine. Obviously there are differences in the underlying file system used that are to be further researched (for example, the I/O vector sizes of the two systems).

None of the two patches have been upstreamed yet. The patches could potentially cause issues in scenarios not covered by the HDFS experiment so they must undergo a thorough review process.

5 Development Workflow Evaluation

The importance of the proper development workflow has been recognised very early in the project. In the first all partner call the consortium held before the official kick-off meeting a preliminary proposal was already introduced by the management team. In defining the workflow we have followed various best practices from existing open source projects. Particular focus was on those projects to which MIKELANGELO is planning on contributing to, in particular the Linux kernel (sKVM), OpenStack, and OSv.

The workflow was documented in an internal report (Project Management Handbook) initially delivered in M2 and later extended with additional information based on the new requirements of the project.

So far the project has focused on the development and testing of individual components. Integration has happened mostly manually. The resulting workflow allowed a quick start in the beginning of the project and a swift development of individual components. However, now that the project matures special care needs to be taken to test the components in an integrated scenario regularly. Due to the complexity of MIKELANGELO's stack such tests pose a particular challenge. To allow quick feedback for developers and holistic management of the workflow we have adopted a continuous integration setup including Gerrit [84], Zuul [85], and Jenkins [55]. Gerrit has been used from the project's start for code review. Now the combination of Zuul and Jenkins allows us to also run complex system-level tests.

The CI infrastructure is capable of modifying the compute infrastructure itself. Thus MIKELANGELO's CI enables developers to change the kernel code of the host machines that run the virtual infrastructure, as required in the case of any modifications in sKVM. To ensure a smooth operation even in cases where the deployed infrastructure may break, we have developed an automated roll-out of this CI setup. As far as we can tell MIKELANGELO is the first project in the public domain that has delivered such advanced automation.

The CI installation works in a proof of concept deployment and, as of writing this deliverable, is being prepared for production use and as an open source release. The CI system follows the setup outlined and used by OpenStack for its continuous integration. The working system employed by the OpenStack project is available online. A similar setup will soon be in use within MIKELANGELO.

The CI setup works by integrating the workflow of Gerrit and Jenkins via Zuul. Gerrit is a general tool used for code review. The general outline of the gerrit development workflow [86] lets developers clone/pull their code via Gerrit and start a feature branch. Once they have developed a new feature they send it for review to Gerrit. In Gerrit either other developers need to acknowledge the patch and/or automated tests need to be passed. In our case, changes to code will trigger Zuul which acts as an intermediary between Gerrit and Jenkins. Zuul allows multiple projects to reference each other and manages inter-project dependencies. Furthermore, Zuul provides a pipeline system, which resembles scheduling queues with various properties. When Zuul jobs are scheduled for execution, Zuul dispatches those jobs to Jenkins, which actually performs the system tests. Results from Jenkins are then provided directly to Gerrit. The CI setup is shown in the following figure.

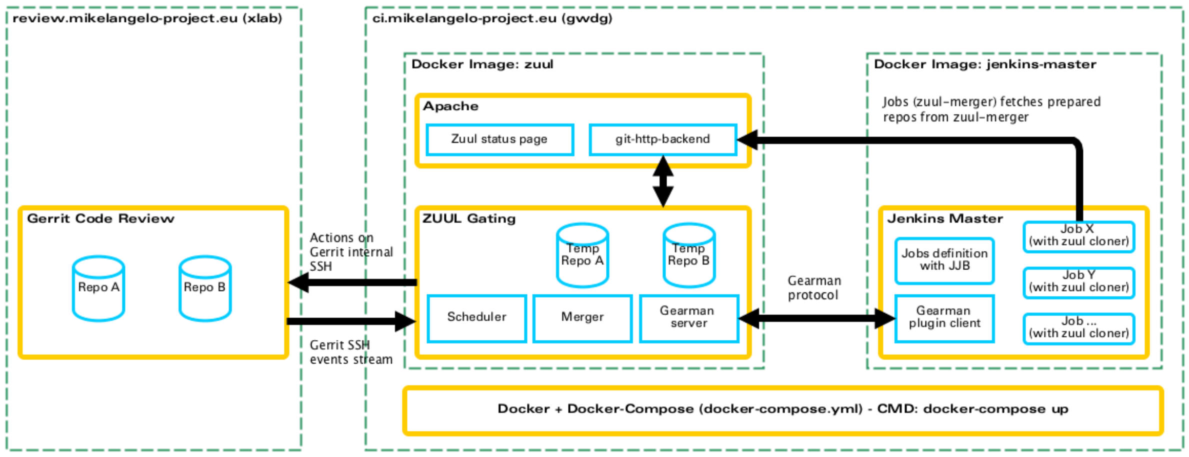


Figure 27. MIKELANGELO's Continuous Integration environment.

This setup is considerably more complex than a simple setup with only Jenkins would be. MIKELANGELO's complex system architecture, however, poses special requirements for the integration. As for the OpenStack project itself zuul provides the main benefit that dependencies of features changes can be tracked. In MIKELANGELO this allows us to test components in a dynamically integrated scenario. For example, a new kernel patch for IOcm that needs testing can reference a recent patch for virtual RDMA and another one for snap. Such dynamic tests in zuul furthermore allow the tracking of which component in a dependency tree breaks an integration test. In addition, zuul provides hierarchical queues, called gaters. These gates can be used to run different types of tests, such as component tests, lightweight integration tests, and full-blown, long-running integration tests.

6 Observations and Priorities

The architecture and implementation of the MIKELANGELO software stack has been evaluated in the previous chapters. This chapter gathers key observations, and summarises the current priorities of the consortium regarding future efforts. These priorities have been directly informed by the evaluation activities to date.

6.1 Individual Components

6.1.1 Linux Hypervisor IO Core Management- sKVM's IOcm

Key Observations

- Automated dynamic allocation of cores to IO has been successfully demonstrated.
- Performance improvements using both throughput intensive (netperf) and real world (HTTP server) workloads have been confirmed.
- The implementation has been developed using Linux kernel best practices to simplify upstreaming.

Current Priorities

- integration of the automated version of the IOcm into the use cases for both Cloud and HPC stacks.
- investigation of enhanced algorithms to tune the number of IO cores.
- pursuing upstreaming to widen the availability and impact of this work.

6.1.2 Virtual RDMA - sKVM's virtual RDMA

Key Observations

- Prototype I has delivered up to 25% performance improvements for OpenFOAM running on Linux hosts, but slower performance for OpenFOAM running on OSv hosts, due to memory access across NUMA nodes.
- Prototype I has been developed within architectural constraints that limit performance improvements. Future efforts will concentrate on Prototype II – the implementation is already well advanced.

Current Priorities

- Completion of Prototype II and integration and validation in both the cloud and HPC stacks

6.1.3 Unikernel Guest Operation System - OSv

Key Observations

- OSv has unique architectural optimisations that enable significant performance improvements on cloud-hosted applications. Memcached can deliver 20% more throughput on OSv compared to Linux – without rewriting memchached.
- OSv does not implement all Linux APIs, and so attempting to run applications on OSv may reveal missing functionality. OSv developers have quickly implemented missing functionality unearthed by MIKELANGELO use cases.
- OSv is developed and supported as a public offering by ScyllaDB, and is continuously being enhanced.

Current Priorities

- Continue to address missing functionality as it is revealed.
- Continue to investigate and address performance issues as they are discovered.

6.1.4 Application Package Management - MPM

Key Observations

- MPM provides a powerful tool for application package management, an essential requirement for integration with modern Cloud and HPC deployments.
- MIKELANGELO use cases have demonstrated that MPM has the flexibility to create and compose complex, nested, end-user images. The efficiencies of MPM allow particularly complex images to be constructed faster than with conventional scripts and tools.
- The development of MPM has driven complementary enhancements to Capstan, a tool that constructs images for OSv-hosted applications.

Current Priorities

- Continue to support use-case adoption.
- Add additional end-user friendly features including support for package versioning, composing application images directly from packages, extended package meta-data and default command management.

6.1.5 Monitoring - snap

Key Observations

- The flexible, extensible architecture of snap has been successfully demonstrated with numerous plugins already developed, published and deployed across the use-cases.
- The management and performance of snap have been explored and overhead is nearly always less than alternatives, including collectd, especially at scale.
- snap is developed and supported as a key public offering by INTEL, and is continuously being enhanced.

Current Priorities

- Continue to support use-case adoption, exposing additional metrics that may add value, and addressing any other requirements that may arise.
- Enhance and possibly extend snap to support more automated performance analysis.

6.1.6 Hosted Application Acceleration - Seastar

Key Observations

- Seastar is a powerful C++ library that has been architected to deliver significant performance improvements over standard Linux libraries and APIs for cloud-hosted applications.
- A re-implemented memcached running delivered approximately 3 times the throughput of standard memcached running on Linux. Other tests have demonstrated that performance scales linearly with resources.

Current Priorities

- Continue to investigate and address performance issues as they are discovered.
- Extend Seastar with support for additional optimisations as opportunities are revealed..

6.1.7 Side Channel Attack Mitigation - sKVM's SCAM

Key Observations

- A proof-of-concept attack has been developed that will help prove the effectiveness of the implementation.
- Various mitigations of side channel attacks have been architected and designed. Implementation is currently underway.

Current Priorities

- Complete the implementation and deploy and validate in both the Cloud and HPC stacks.

6.2 Integrated Stacks

6.2.1 Full Stack for Cloud

Key Observations

- sKVM's IOcm has been integrated successfully. Performance improvements have not yet been pursued or measured. Future integration with the Cloud scheduler will allow cross-cluster optimisations.

- Snap has been integrated and is robust enough to now drive future requirements for tighter snap integration with the platform, as well as optimisation of the Cloud stack.
- MPM is an essential tool to support deployments. As applications are ported to OSv they will be deployed.

Current Priorities

- Integrate improved components (sKVM vRDMA, SCAM, etc) as they become available.
- Work with component developers to investigate and address where possible any unexpected performance issues that arise.
- Pursue tighter integration with relevant OpenStack components, upstreaming enhancements where appropriate.

6.2.2 Full Stack for HPC

Key Observations

- Significant extensions to Torque have been made which now enable it to host virtual machines as opposed to bare metal jobs.
- sKVM’s IOcm has been successfully deployed and integrated into Torque extensions
- sKVM’s vRDMA was integrated with Torque and integrated into Torque extensions
- Snap has been integrated and is capturing full-stack telemetry, however an integration issue with the InfluxDB client is being investigated.
- OSv has been successfully used in a proof-of-concept deployment as a guest operating system on the HPC testbed.
- In general, enabling virtualisation for HPC applications offers significant new flexibility to end-users and system administrators.
- The performance optimisations developed by MIKELANGELO are designed to reduce the overhead of virtualisation significantly, but these improvements have not been materialised yet - a low-level analysis is underway.
- An intermittent MPI issue has been unearthed during testing, and is being debugged.

Current Priorities

- Integrate improved components (SCAM, etc) as they become available.
- Work with component developers to investigate and address where possible any unexpected performance issues that arise.
- Pursue sharing of relevant Torque enhancements, where appropriate.
- Improving the code base

6.3 Case Studies

6.3.1 Cloud Bursting

Key Observations

- Rewriting an application using Seastar can deliver ten times the performance, as is the case with the ScyllaDB rewrite of Cassandra.
- Ongoing optimisations have smoothed out performance spikes during the cloud-bursting process.

Current Priorities

- Integrate improved components (sKVM, OSv, Seastar, etc) as they become available.
- Work with component developers to investigate and address where possible any unexpected performance issues that arise.

6.3.2 Cancellous Bones Simulation

Key Observations

- This simulation has been successfully integrated into the virtualised HPC stack
- Preliminary performance analyses are now possible - and yielding unexpected results (very slow bare-metal baseline performance) that is being further investigated
- This use case has championed the migration of both MPI and NFS to OSv. Both migrations revealed shortcomings that were addressed and integrated into OSv.

Current Priorities

- Complete migration of the simulation to the full MIKELANGELO stack, leveraging OSv as the guest OS.
- Work with component developers to investigate and address where possible any unexpected performance issues that arise.

6.3.3 Aerodynamic Maps

Key Observations

- OpenFOAM has been successfully deployed inside OSv images, and integrated with snap. The integration process has revealed gaps in the OSv implementation which have being closed. Open MPI has been modified to run on OSv.
- Confirmed benefits of OSv over Linux as a guest OS have included ten times smaller image size, quicker boot time (sub-second) and quicker application startup (nearly 50% faster).
- In preliminary tests OpenFOAM is performing 5% slower on OSv than on Linux. Now that integration is complete, performance testing and troubleshooting has begun.

Adding NUMA support to OSv will help deliver improvements. I/O intensive workloads are expected to reveal significant performance gains.
- Snap can be integrated seamlessly, supporting rich dashboards and flexible offline analysis over all layers of the stack: infrastructure, virtual environment and application.

Current Priorities

- Integrate improved components (sKVM, OSv, MPM etc) as they become available.
- Work with component developers to investigate and address where possible any unexpected performance issues that arise.

6.3.4 Big Data

Key Observations

- Important Big Data tools are being integrated into the MIKELANGELO stack.
- Two workarounds have been identified for an issue discovered when integrating Apache Storm.
- An attempt to deploy HDFS revealed only two minor issues due to unnecessary use of shell utilities in the Java code. Both have been addressed and resolved.
- An initial performance evaluation using HDFS revealed poorer performance on OSv compared to Linux. Subsequent analysis has revealed several opportunities for improvements which are in the process of being implemented. In certain circumstances HDFS is now 4 times faster on OSv compared to Linux, but additional analysis is required.

Current Priorities

- Further analyse the performance of HDFS and in particular various parts of the OSv that might be responsible for variations in the performance
- Integrate additional big data tools and validate/evaluate concrete big data cluster deployments
- Integrate improved components (sKVM, OSv, etc) as they become available.
- Work with component developers to investigate and address where possible any unexpected performance issues that arise.

6.4 Development Workflow

Key Observations

- A powerful continuous integration environment has been successfully constructed and tested in a proof-of-concept. It includes integration of Gerrit, Zuul and Jenkins, and can modify the compute environment as part of the automated validation processes.

- MIKELANGELO is the first successful integration of these components into a development workflow that the consortium is aware of.

Current Priorities

- Trial the development workflow with appropriate selection of MIKELANGELO components.
- Document and share the learnings from this integration to enable replication of this development workflow outside the consortium.

7 Concluding Remarks

The MIKELANGELO project has now reached its halfway point. Architectures for all components have been prepared, and implementations for most are now available.

All individual components, integrated stacks, use cases and the development workflow have been examined in this deliverable.

No unexpected architectural issues have been discovered. All components have delivered the performance and functionality expected during individual analysis. Plans are in place to overcome some known architectural limitations (e.g. in vRDMA Prototype I).

Integration of both Cloud and HPC stacks continues to mature. As components are integrated issues are being revealed. Requirements for additional functionality and support are being discovered, and are being addressed as a priority by the consortium. Enhancements are also being upstreamed quickly to public open-source repositories where appropriate (e.g. for OSv, Seastar and snap), to maximise the benefits of this work.

The MIKELANGELO use cases are very representative of typical Cloud and HPC workloads, and are proving very valuable tools to evaluate the effectiveness and usability of various combinations of MIKELANGELO components. Significant improvements in performance have been observed in numerous areas, however the use cases have also revealed a number of locations and configurations of the MIKELANGELO stack that do not yet deliver the expected performance. Detailed analysis has begun and has already begun to identify bugs and bottlenecks that, once addressed, are transforming the initial results.

This evaluation of the architecture and implementation of MIKELANGELO has demonstrated significant progress to-date in delivering value-added enhancements for advanced Cloud and HPC environments. It has also helped prioritise future efforts to help maximise the impact of this project.

8 References and Applicable Documents

[1] The MIKELANGELO project, http://www.mikelangelo-project.eu/

[2] The OSv guest operating system for the cloud, http://osv.io/

[3] sKVM, The First Super KVM - Fast virtual I/O hypervisor, https://www.mikelangelo-project.eu/wp-content/uploads/2016/06/MIKELANGELO-WP3.1-IBM-v1.0.pdf

[4] OpenFOAM, The Open Source CFD Toolbox, http://www.openfoam.com/

[5] MIKELANGELO Public Deliverables, https://www.mikelangelo-project.eu/deliverables/

[6] Hadoop HDFS, http://hadoop.apache.org/

[7] Apache Storm, http://storm.apache.org/

[8] Snap, the open-source telemetry framework, http://intelsdi-x.github.io/snap/

[9] Seastar, the C++ framework for high-performance server applications, http://www.seastar-project.org/

[10] ELVIS, Efficient and Scalable Paravirtual I/O System, N. Har'El et al, available at http://www.hypervisorconsulting.com/pubs/eli/elvis-atc13.pdf

[11] Ubuntu 14.04, http://www.ubuntu.com/

[12] KVM, the Kernel Virtual Machine, http://www.linux-kvm.org/

[13] QEMU 2.2, http://wiki.qemu.org/

[14] Netperf, http://www.netperf.org/

[15] The Apache HTTP Server Project, https://httpd.apache.org/

[16] Ab, the Apache HTTP server benchmarking tool, https://httpd.apache.org/docs/2.2/programs/ab.html

[17] The MIKELANGELO project GitHub, https://github.com/mikelangelo-project

[18] DPDK, the Data Plane Development Kit, http://dpdk.org/

[19] Open vSwitch, http://www.openvswitch.org/

[20] Infiniband, http://www.infinibandta.org/content/pages.php?pg=about_us_infiniband

[21] Hyv, A hybrid I/O virtualization framework for RDMA-capable network interfaces, https://github.com/zrlio/hyv

[22] Unikernel, http://unikernel.org/

[23] NFS, http://nfs.sourceforge.net/

[24] The Porting of NFS to OSv, https://www.mikelangelo-project.eu/2016/04/nfs_on_osv/

[25] Open MPI, https://www.open-mpi.org/

[26] ISO 9660, http://www.iso.org/iso/catalogue_detail.htm?csnumber=17505

[27] Redis, http://redis.io

[28] Cassandra, http://cassandra.apache.org/

[29] Amazon ec2, https://aws.amazon.com/ec2/

[30] Memcached memory database https://memcached.org/

[31] OSv functional tests https://github.com/cloudius-systems/osv/tree/master/tests

[32] OSv developer scripts https://github.com/cloudius-systems/osv/tree/master/scripts

[33] Capstan, https://github.com/cloudius-systems/capstan
[34] OSv-apps, https://github.com/cloudius-systems/osv-apps
[35] OpenStack, https://www.openstack.org/
[36] Java, https://www.java.com/en/
[37] Node.js, https://nodejs.org/en/
[38] Python, https://www.python.org/
[39] Go, https://golang.org/
[40] EMC, http://www.emc.com/en-us/index.htm
[41] UniK, https://github.com/emc-advanced-dev/unik
[42] Rump Kernel, https://github.com/rumpkernel/rumprun
[43] MirageOS, https://mirage.io/
[44] Defer Panic, https://github.com/deferpanic
[45] Collectd, https://collectd.org/
[46] Ganglia, http://ganglia.info/
[47] Telegraf, https://github.com/influxdata/telegraf
[48] OpenStack Ceilometer, https://wiki.openstack.org/wiki/Telemetry
[49] Libvirt, http://libvirt.org/
[50] gRPC, http://www.gprc.io/
[51] Metsch, T., Ibidunmoye. O., Bayon-Molino, V., Butler, J.,Hernández-Rodriguez, F., Elmroth, E. Apex Lake: A Framework for Enabling Smart Orchestration, Article No.: 1, Proceedings of the Industrial Track of the 16th International Middleware Conference doi: http://dx.doi.org/10.1145/2830013.2830016
[52] Tukey Method, http://www.itl.nist.gov/div898/handbook/prc/section4/prc471.htm
[53] InfluxDB, https://influxdata.com/
[54] Travis CI, https://travis-ci.org/
[55] Jenkins, https://jenkins.io/
[56] Trusted Analytics Platform, http://trustedanalytics.org/
[57] ScyllaDB, http://www.scylladb.com
[58] Vertx, http://vertx.io/
[59] Twisted, http://twistedmatrix.com/trac/
[60] Libevent, http://libevent.org/
[61] EventMachine, http://rubyeventmachine.com/
[62] Seastar DPDK Web Framework Showdown, http://pseudo.co.de/seastar-dpdk-web-framework-showdown/
[63] OpenSSL, https://www.openssl.org/
[64] Intel's Cache Allocation Technology, http://www.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html
[65] PAPI, http://icl.cs.utk.edu/papi/

[66] Ubuntu 14.04.4 LTS (Trusty Tahr) Daily Build, https://cloud-images.ubuntu.com/trusty/current/
[67] Debian, https://www.debian.org/
[68] CentOS, https://www.centos.org/
[69] Torque, http://www.adaptivecomputing.com/products/open-source/torque/
[70] OpenStack Sahara, https://wiki.openstack.org/wiki/Sahara
[71] DataStax, Configuring data consistency, http://docs.datastax.com/en/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html
[72] ScyllaDB vs Cassandra: towards a new myth?, http://blog.octo.com/en/scylladb-vs-cassandra-towards-a-new-myth/
[73] Amazon Lambda, https://aws.amazon.com/lambda/details/
[74] Scylla-cluster-test, https://github.com/scylladb/scylla-cluster-tests
[75] UberCloud, https://www.theubercloud.com/
[76] CFD Direct Cloud, http://cfd.direct/cloud/
[77] SimScale, https://www.simscale.com/
[78] pyFoam, https://openfoamwiki.net/index.php/Contrib/PyFoam
[79] Nimbus, http://www.nimbusproject.org/
[80] Apache Zookeeper, https://zookeeper.apache.org/
[81] Apache Namenode, https://wiki.apache.org/hadoop/NameNode
[82] Pre-built HDFS application package, https://mikelangelo-pkg.s3.amazonaws.com/app-packages-160615.tar.gz
[83] TestDFSIO, https://discuss.zendesk.com/hc/en-us/articles/200864057-Running-DFSIO-MapReduce-benchmark-test
[84] Gerrit, https://www.gerritcodereview.com/
[85] Zuul, http://status.openstack.org/zuul/
[86] Gerrit Development Workflow, https://gerrit-review.googlesource.com/Documentation/intro-user.html

Appendix A. HPC Full Stack Functional Test Results

A.1 Test 1 - Test VM Instantiation

Output:

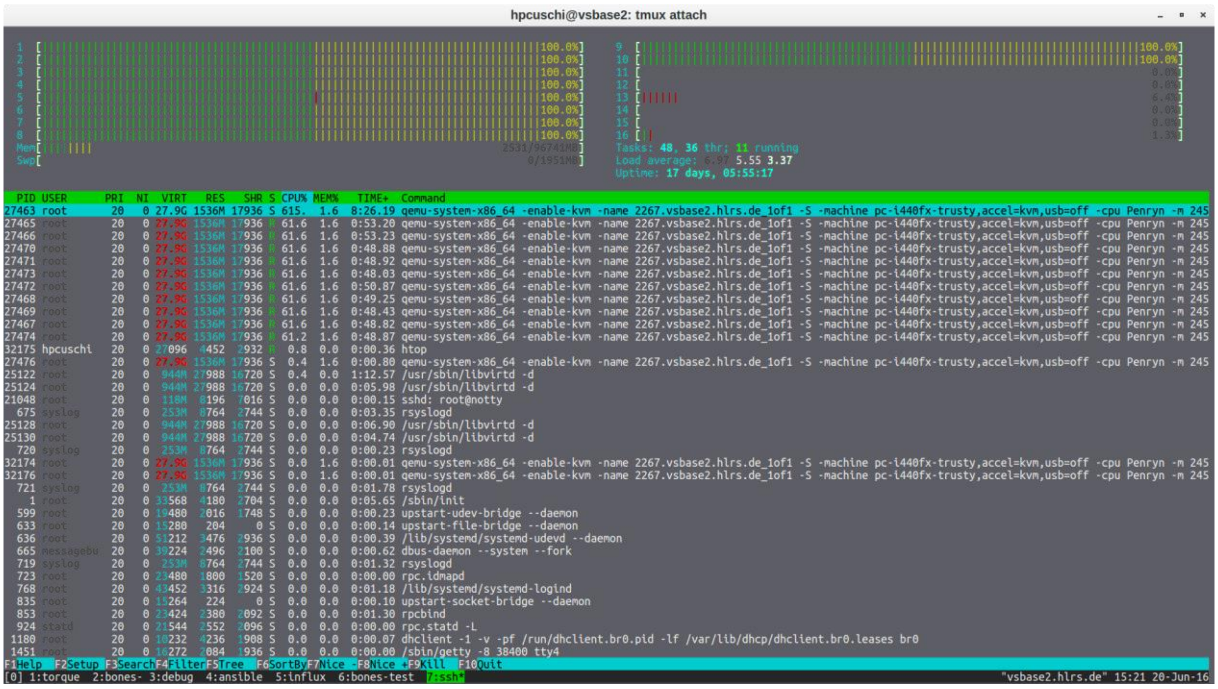
```
qsub -l nodes=1 \
      -vm \
        img=/images/pool/ubuntu_bonesV01.51c.img,iocm=false\
        /home/hpcuschi/projects/mikelangelo-bones/bones-jobscript.sh


[node0107|15:06:05|vmPrologue.parallel|DEBUG] Booting VM number '1/1' on
compute node 'node0107' from
domainXML='/home/hpcuschi/.pbs_vm_jobs/ac78afa3994f8bd/node0107/2265.vsbase
2.hlrs.de_1of1.xml'
[hpcuschi@node0107 ~]$ virsh list --all
Id     Name                          State
----------------------------------------------------
4      2265.vsbase2.hlrs.de_1of1     running
[node0107|15:06:54|vmEpilogue.parallel|DEBUG] Shutting down and destroying
all local VMs now.
[hpcuschi@node0107 ~]$ virsh list --all
 Id    Name                          State
----------------------------------------------------
```

Result: The Qsub wrapper generated the VM and qsub has placed it on node0107. Virsh on node0107 shows the vm (state=running) and shows that the VM is stopped and removed after the job has run.

A.2 Test 2 - CPU count; CPU Pinning

Output:

```
qsub -l nodes=1 -vm
img=/images/pool/ubuntu_bonesV01.51c.img,iocm=false,vcpus=10,vcpu_pinning=/
home/hpcuschi/projects/mikelangelo-bones/cpu_map.txt
/home/hpcuschi/projects/mikelangelo-bones/bones-jobscript.sh
```



```
hpcuschi@vsbase2: tmux attach
  1  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]     9  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  2  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]     10 [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]
  3  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]     11 [                                                                       0.0%]
  4  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]     12 [                                                                       0.0%]
  5  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]     13 [||||||                                                                 6.4%]
  6  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]     14 [                                                                       0.0%]
  7  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]     15 [                                                                       0.0%]
  8  [||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||100.0%]     16 [||                                                                     1.3%]
  Mem[|||||||||                                                      2531/96741MB]     Tasks: 48, 36 thr; 11 running
  Swp[                                                                  0/1951MB]     Load average: 6.97 5.55 3.37
                                                                                       Uptime: 17 days, 05:55:17

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
27463 root       20   0 27.9G 1536M 17936 S 615.  1.6  8:26.19 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27465 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:53.20 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27466 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:53.23 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27470 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:48.88 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27471 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:48.92 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27473 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:48.03 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27472 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:50.87 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27468 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:49.25 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27469 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:48.43 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27467 root       20   0 27.9G 1536M 17936 R 61.6  1.6  0:48.82 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
27474 root       20   0 27.9G 1536M 17936 R 61.2  1.6  0:48.87 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
32175 hpcuschi   20   0 27096  4452  2932 R  0.8  0.0  0:00.36 htop
27476 root       20   0 27.9G 1536M 17936 S  0.4  1.6  0:00.80 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
25122 root       20   0  944M 27988 16720 S  0.4  0.0  1:12.57 /usr/sbin/libvirtd -d
25124 root       20   0  944M 27988 16720 S  0.0  0.0  0:05.98 /usr/sbin/libvirtd -d
21048 root       20   0  118M  8196  7016 S  0.0  0.0  0:00.15 sshd: root@notty
  675 syslog     20   0  253M  8764  2744 S  0.0  0.0  0:03.35 rsyslogd
25128 root       20   0  944M 27988 16720 S  0.0  0.0  0:06.90 /usr/sbin/libvirtd -d
25130 root       20   0  944M 27988 16720 S  0.0  0.0  0:04.74 /usr/sbin/libvirtd -d
  720 syslog     20   0  253M  8764  2744 S  0.0  0.0  0:00.23 rsyslogd
32174 root       20   0 27.9G 1536M 17936 S  0.0  1.6  0:00.01 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
32176 root       20   0 27.9G 1536M 17936 S  0.0  1.6  0:00.01 qemu-system-x86_64 -enable-kvm -name 2267.vsbase2.hlrs.de_1of1 -S -machine pc-i440fx-trusty,accel=kvm,usb=off -cpu Penryn -m 245
  721 syslog     20   0  253M  8764  2744 S  0.0  0.0  0:01.78 rsyslogd
    1 root       20   0 33568  4180  2704 S  0.0  0.0  0:05.65 /sbin/init
  599 root       20   0 19480  2016  1748 S  0.0  0.0  0:00.23 upstart-udev-bridge --daemon
  633 root       20   0 15280   204     0 S  0.0  0.0  0:00.14 upstart-file-bridge --daemon
  636 root       20   0 51212  3476  2936 S  0.0  0.0  0:00.39 /lib/systemd/systemd-udevd --daemon
  665 messagebu  20   0 39224  2496  2100 S  0.0  0.0  0:00.62 dbus-daemon --system --fork
  719 syslog     20   0  253M  8764  2744 S  0.0  0.0  0:01.32 rsyslogd
  723 root       20   0 23480  1800  1520 S  0.0  0.0  0:00.00 rpc.idmapd
  768 root       20   0 43452  3316  2924 S  0.0  0.0  0:01.18 /lib/systemd/systemd-logind
  835 root       20   0 15264   224     0 S  0.0  0.0  0:00.10 upstart-socket-bridge --daemon
  853 root       20   0 23424  2380  2092 S  0.0  0.0  0:01.30 rpcbind
  924 statd      20   0 21544  2552  2096 S  0.0  0.0  0:00.00 rpc.statd -L
 1180 root       20   0 10232  4236  1908 S  0.0  0.0  0:00.07 dhclient -1 -v -pf /run/dhclient.br0.pid -lf /var/lib/dhcp/dhclient.br0.leases br0
 1451 root       20   0 16272  2084  1936 S  0.0  0.0  0:00.00 /sbin/getty -8 38400 tty4
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
[0] 1:torque  2:bones- 3:debug  4:ansible  5:influx  6:bones-test  7:ssh*                                        "vsbase2.hlrs.de" 15:21 20-Jun-16
```

Figure 28. Testing IOcm CPU pinning on the HPC Stack.

The first 10 cores are under load on the host system. Cores 11 to 16 are not used. The load did not move from core to core.

Result: hTop on the host node shows that the CPU load is not moving between different cores, as expected.

A.3 Test 3 - Multiple VMs per Node

Output:

```
qsub -l nodes=2 \
      -vm \
        img=/images/pool/ubuntu_bonesV01.51c.img,\
        iocm=false \
      /home/hpcuschi/projects/mikelangelo-bones/bones-jobscript.sh

[node0108|15:06:33|vmPrologue|DEBUG] Created/transfered files for VM (1/2)
of (1) on host 'node0108'.
[node0108|15:06:39|vmPrologue|DEBUG] Created/transfered files for VM (2/2)
of (1) on host 'node0107'.
```

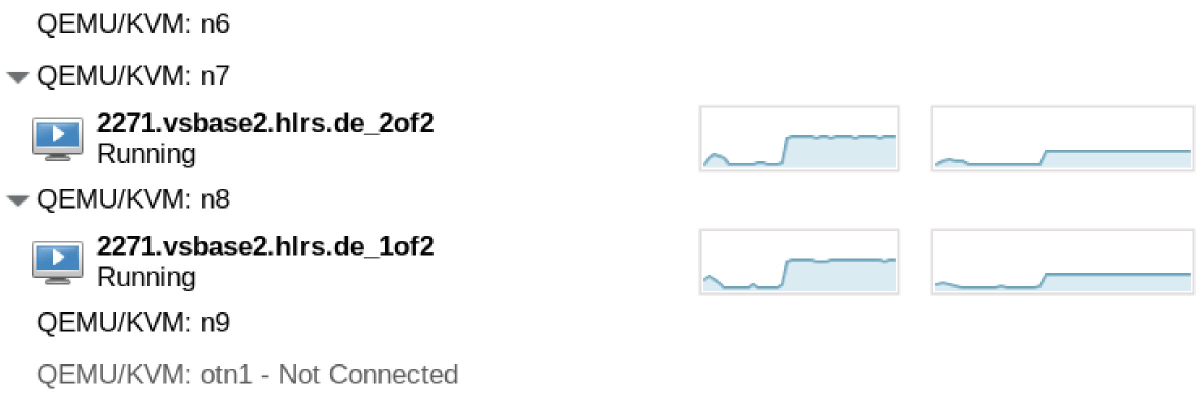

QEMU/KVM: n6
QEMU/KVM: n7
2271.vsbase2.hlrs.de_2of2
Running
QEMU/KVM: n8
2271.vsbase2.hlrs.de_1of2
Running
QEMU/KVM: n9
QEMU/KVM: otn1 - Not Connected

Figure 29. Two VMs generated on separate nodes and running the same job in parallel.

Result: Two VMs are placed on two different nodes for one Job, as expected.

A.4 Test 4 - Interactive non-VM Jobs

Output:

```
qsub -I -l nodes=1

[hpcuschi@node0107 ~]$
Job ID                  Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2272.vsbase2            STDIN            hpcuschi               0 R batch

[hpcuschi@node0107 ~]$echo $PBS_JOBID
2272.vsbase2.hlrs.de
```

Result: Submitted an interactive job to qsub. Got a node (node0107), job is listed in qstat as running and has the correct job ID.

A.5 Test 5 - Qsub Submitted Jobs

Output:

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

[vsbase2|15:06:29|qsub|DEBUG] **************** BEGIN OF QSUB
WRAPPER *******************

[vsbase2|15:06:29|qsub|DEBUG] cmd line: ' 2412
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 /bin/bash
/opt/dev/HPC-Integration/src/qsub -l nodes=1:ppn=16 ./bones-
jobscript-metal.sh'

[vsbase2|15:06:29|qsub|DEBUG] All parameter received:  '-l
nodes=1:ppn=16 ./bones-jobscript-metal.sh'
```

```
[vsbase2|15:06:29|qsub|DEBUG] Checking preconditions, whether
everything is in place we depend on.

[vsbase2|15:06:29|qsub|DEBUG] Preconditions check passed
successfully.

[vsbase2|15:06:29|qsub|DEBUG] Checking file for inline '#PBS -vm
key=value' in file './bones-jobscript-metal.sh'.

[vsbase2|15:06:29|qsub|DEBUG] No vm job.

2273.vsbase2.hlrs.de

qstat

Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2272.vsbase2              STDIN            hpcuschi        00:00:00 C batch
2273.vsbase2              ...ript-metal.sh hpcuschi     00:12:59 R batch
```

Result: The qsub wrapper noticed a non-VM job and placed it via qsub, as expected.