



ACTiCLOUD: ACTivating resource efficiency and large databases in the CLOUD

Project No: 732366

H2020-ICT-2016-1

D4.4: ACTiCLOUD Final Prototype

Due date of deliverable:	M34 (2019/10/31)
Actual submission date:	M36 (2019/12/09)

Executive summary:

Deliverable D4.4: “ACTiCLOUD Final Prototype” integrates the final (v2.0) versions of the components of the ACTiCLOUD architecture. More specifically, the deliverable focuses on various integration scenarios of the final ACTiCLOUD components, i.e., the two hardware platforms that are available within the project (Numascale and KMAX), the MicroVisor, the OpenStack cloud framework, the ACTiManager, the HyperscaleJVM, the system libraries, and the MonetDB and Neo4j databases. The purpose of the integration scenarios is to demonstrate the interoperability between the major ACTiCLOUD core components. To maximise the impact of the core components, the validity of the integration scenarios, and to solve other challenges that occurred during the course of the project, we also include commodity hardware platforms (Intel and AWS bare metal) and virtualization technologies (KVM and Xen) in our integration scenarios. The deliverable includes documentation together with the corresponding software packages. Finally, the deliverable concludes with the remaining open issues and draws directions for future work.

List of authors:

Author	Affiliation
Atle Vesterkjær	NSCALE
Georgios Goumas, Vasileios Karakostas, Dimitrios Siakavaras, Stefanos Gerangelos, Stratos Psomadakis	ICCS
Monica Vatteroni, Hazeef Mohammed	KALEAO
Michail Flouris, Stelios Louloudakis	ONAPP
Foivos Zakkak, Christos Kotselidis	UNIMAN
Jim Webber	NEO
Ewnetu Bayuh Lakew	UMU
Ying Zhang, Martin Kersten	MDBS

Dissemination Level	X	PU (Public)
		PP (Restricted to other programme participants)
		RE (Restricted to a group specified by the consortium)
		CO (Confidential, only for members of the consortium)
		Where restricted, access granted to:
Nature		R (Report)
		P (Prototype)
		D (Demonstrator)
	X	O (Other)

Review Status		Draft
		WP Leader accepted
		QA approved
	X	Coordinator accepted

Revision History:

Version	Author(s)	Notes
0.1	Atle Vesterkjær	TOC and initial content inserted
0.2	All authors	Content updated
0.3	Martin Kersten, Jim Webber	Document reviewed
0.4	All authors	Comments and suggestions addressed
0.5	Vasileios Karakostas	Editorial changes
1.0	Georgios Goumas, Vasileios Karakostas	Submitted version

ACTiCLOUD Consortium:

Participant No	Participant organisation name	Short name	Country
1 (Coordinator)	Institute of Communication and Computer Systems	ICCS	Greece
2	Numascale AS	NSCALE	Norway
3	Kaleao Limited	KALEAO	UK
4	OnApp Limited	ONAPP	Gibraltar
5	University of Manchester	UNIMAN	UK
6	MonetDB Solutions B.V.	MDBS	Netherlands
7	Neo Technology	NEO	Sweden
8	UMEA University	UMU	Sweden



NUMASCALE



Confidentiality:

This document contains proprietary and confidential material of certain ACTiCLOUD contractors, and may not be reproduced, copied, or disclosed without appropriate permission. The commercial use of any information contained in this document may require a license from the proprietor of that information.

THIS DOCUMENT IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENT, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table of Contents

1	About ACTiCLOUD	9
1.1	Purpose of this Document	9
1.2	Document Structure	10
2	Virtualizing scale-out resources: Microvisor on KMAX	11
2.1	Overview of the MicroVisor	11
2.2	KMAX Hardware support for the MicroVisor	12
2.3	Integration of the MicroVisor on KMAX.....	13
2.4	Summary.....	15
3	Virtualizing cloud resources: MicroVisor on AWS	15
3.1	Introduction	15
3.2	Deployment	16
3.3	Summary.....	17
4	Efficient management of scale-out heterogeneous resources: KMAX - KVM - OpenStack - ACTiManager	18
4.1	Introduction	18
4.2	KMAX support for KVM	18
4.3	Integration of KVM, OpenStack, and ACTiManager on KMAX.....	18
4.3.1	Integration with KVM	18
4.3.2	Integration with OpenStack	20
4.3.3	Integration with ACTiMamanger.....	21
4.3.4	Test and Validation	21
4.4	Summary.....	21
5	Generic cloud resource management the ACTiCLOUD way: MicroVisor - ACTiManager	22
5.1	Introduction	22
5.2	Integration of MicroVisor and OpenStack.....	22
5.3	Integration of ACTiManager and MicroVisor	23
5.3.1	ACTiManager.external	23
5.3.2	ACTiManager.internal.....	23
5.3.3	Test and Validation	23
5.4	Summary.....	23
6	Scaling Java workloads on low power processors: Hyperscale JVM on AArch64 & KMAX	24
6.1	DaCapo Benchmark suite	24
6.2	MonetDBLite-Java	24
6.3	Neo4J	24

7	Running very large databases in Memory	26
7.1	Introduction	26
7.2	MonetDB on Numascale Shared Memory System	26
7.3	Summary	27
8	Scaling-up cloud resources to support large JVM-based databases: Numascale - KVM - Hyperscale JVM - Neo4j - MonetDB Lite	28
8.1	Integration testing	28
9	Efficient management of scale-up resources: Numascale - ACTiManager	30
9.1	Introduction	30
9.2	Bringing up KVM and adding NUMA topology to the guest OS	30
9.2.1	Moving to CentOS 8	30
9.3	Configuring KVM PASSING BOOT PARAMETERS for NUMA-awareness	31
9.4	Extending ACTiManager.Internal to expose NUMA-awareness to Guest VMs	32
10	Interference-aware applications: MonetDB & ACTiManager	33
11	Big data analytics integrations in ACTiCLOUD	35
11.1	SQALPEL: a general benchmark platform	36
11.2	Summary	40
12	Conclusions and future work	41
Appendix A: Patches for Integrating ACTiManager on KMAX		42
Appendix B: Patches for Integrating ACTiManager with MicroVisor		44
Appendix C: Running MonetDB using more than 1 TB RAM		44
Appendix D: Example usage of MonetDB performance monitor script perf_monitor.py		47
Appendix E: Overview archived milestones of evolving MonetDB		48

Figures

Figure 2.1: Virtual device mappings.	14
Figure 6.1: Bug report.	25
Figure 6.2: Neo4j build error.	25
Figure 8.1: Integration stack.	29
Figure 10.1: Interaction between PerformanceAlerter and other components.	33
Figure 11.1: Integration stacks for scale-out and scale-up database applications.	35
Figure 11.2: SQALPEL grammar for the TPC-H query 1.	37
Figure 11.3: SQALPEL's query pool.	38
Figure 11.4: Experiments history.	38
Figure 11.5: Principle components.	39
Figure 11.6: Query speedup.	39
Figure 11.7: Query differentials.	40

List of Abbreviation

Abbreviation / Acronym	Meaning
AMI	Amazon Machine Image
AoES	ATA Over Ethernet Switch
API	Application Programming Interface
APIC	Advanced Programmable Interrupt Controller
AWS	Amazon Web Services
AXI	Advanced Extensible Interface
CPU	Central processing unit
DBMS	Database Management System
DHCP	Dynamic Host Configuration Protocol
DMA	Direct Memory Access
DNS	Domain Name System
FPGA	Field-programmable Gate Array
HMP	Heterogeneous Multi-Processing
HV	HyperVisor
HW	Hardware
IPC	Instructions per Cycle
ISA	Instruction Set Architecture
KVM	Kernel-based Virtual Machine
LUN	Logical Unit Number
MPKI	Misses per Kilo Instructions
MV	Micro Visor
NIC	Network Interface Card
NPU	Network Processing Unit
NUMA	Non-Uniform Memory Access
NVMe	Non-Volatile Memory express
PCIe	Peripheral Component Interconnect Express
PS	Processing System
QEMU	Quick Emulator
SD	Secure Digital
SDK	Software Development Kit
SMP	Symmetric Multi-Processing
SPU	Storage Processing Unit
SW	Software
UART	Universal Asynchronous Receiver-Transmitter
UI	User Interface
vCPU	Virtual CPU
VM	Virtual Machine
VMM	Virtual Machine Monitor

1 About ACTiCLOUD

ACTiCLOUD's vision is to develop novel cloud architecture components that will break existing scale-up and share-nothing barriers for cloud infrastructure, and enable holistic management of physical resources at the local cloud site and at multi-site levels. ACTiCLOUD targets significantly improved utilization and scalability of resources. This will ultimately translate to:

1. significant cost and performance improvements for Cloud Service Providers (CSPs),
2. higher performance, stability and lower pricing for cloud applications,
3. enhanced flexibility and scalability of cloud resources for intensive database applications that have until now faced tough challenges in covering their resource demands from existing cloud offerings.

ACTiCLOUD aims to enhance the viability of cloud deployment scenarios through enhancement of the various technology ingredients, i.e., the hypervisor, the cloud manager, system libraries, language runtimes, and database systems, with a novel and holistic set of mechanisms and policies built on top of these new-generation computing system architectures. Therefore, ACTiCLOUD will enable the creation of distributed, hyper-converged, "share-anything", resource scale-out cloud platforms to broaden the applicability of cloud technologies across more markets through richer and more cost effective application deployments.

1.1 Purpose of this Document

The purpose of this Deliverable D4.4 "ACTiCLOUD Intermediate Prototype" is to provide technical details regarding the integration of the final version 2.0 of the components of the ACTiCLOUD architecture, i.e., the MicroVisor, the OpenStack cloud framework, the ACTiManager, the HyperscaleJVM, the System Libraries, and the MonetDB and Neo4j databases, on top of the provided hardware platforms, KMAX and Numascale. Since ACTiCLOUD works on cutting edge technologies both at the software and hardware level, proceeding with full integration of all components has been challenging during the entire course of the project. More specifically, there were integration challenges regarding the integration of MicroVisor on top of Numascale, as documented in D4.2 "ACTiCLOUD Intermediate Prototype" that made us decide to bring forward as "mainline" setup the Linux/KVM hypervisor for the Numascale platform. To maximise further the impact of the core components and the validity of the integration scenarios, we also include in our integration scenarios commodity hardware platforms (Intel and AWS bare metal servers).

Hence, we proceeded with various integration scenarios that provide extensive testing of the capabilities of the core components and the integration challenges

- **Integration of MicroVisor on the KMAX platform:** This integration scenario provides a substrate to support the capabilities of the KMAX platform with lightweight virtualization.
- **Integration of MicroVisor on AWS bare metal servers:** This integration scenario targets to maximise the impact of the MicroVisor by enabling its execution on bare metal instances on AWS.
- **Integration of KVM, OpenStack, and ACTiManager on the KMAX platform:** The goal of this scenario is to demonstrate the ACTiManager capabilities on top of the KMAX platform. We decided to use KVM instead of OpenStack on the KMAX platform for the following reasons: (i) ACTiManager has been developed to be compatible with the OpenStack Pike version, (ii) the MicroVisor was originally developed to support an older version of OpenStack (Kilo), (iii) ONAPP completed the porting of MicroVisor to work with OpenStack Pike version at the end of the project but only with x86 platforms, and (iv) the KVM virtualization technology is supported by OpenStack Pike. KVM is a

dominant, widely-adopted, open-source, and stable virtualization technology for Linux-based systems. While virtualization is supported for the KMAX platform by MicroVisor, we focus here on KVM to broaden the impact of the project's work. In the future, we plan to integrate the MicroVisor with OpenStack Pike for the KMAX platform as well. Note that in Deliverable 2.4 "Distributed Cloud Resource Manager v2.0" we tested and integrated all components of the v2.0 of ACTiManager, with KVM and OpenStack Pike, on a cluster of commodity x86 Intel platforms to further test and demonstrate core ACTiManager capabilities.

- **Integration of MicroVisor, OpenStack, and ACTiManager on commodity x86 platforms:** The purpose for this integration scenario is to integrate, test, and demonstrate the components of the v2.0 of ACTiManager together with MicroVisor and Openstack Pike on top of commodity x86 platforms. We choose this setup for the integration of ACTiManager and MicroVisor for the same reasons as explained above in the "Integration of KVM, OpenStack, and ACTiManager on the KMAX platform" integration scenario.
- **Integration of HyperscaleJVM on KMAX:** The purpose for this integration scenario is to demonstrate the integration of the HyperscaleJVM with the AArch64 ISA of KMAX.
- **Integration of very large in-memory databases on the Numascale platform:** The purpose for this integration scenario is to demonstrate the integration and execution of a very large in-memory MonetDB database instance and a very large JVM-based Neo4j database on top of the Numascale platform.
- **Integration of KVM, OpenStack, and ACTiManager on the Numascale platform:** The purpose for this integration exercise is twofold. On one hand, this scenario supports the demonstration of ACTiManager capabilities on a large shared memory system (big SMP, scale-up system) with NUMA architecture, like the Numascale platform. On the other hand, this scenario provides a solid alternative software stack on top of the Numascale system.
- **Integration of MonetDB and ACTiManager.** The purpose of this integration scenario is to showcase (i) the capability of MonetDB to provide application-specific metrics and the (ii) capability of ACTiManager to receive and consider such metrics for even better and more sensible resource management.
- **Integration of big data analytics in ACTiCLOUD:** The purpose of this integration scenario is to showcase the ability of MonetDB to support big data analytical applications from micro-scale to macro-scale, from a single DBMS server to distributed DBMS clusters.

1.2 Document Structure

The document is structured in the following way. Sections 2 and 3 describe the integration scenarios of MicroVisor on top of the KMAX and the AWS bare metal platforms, respectively. Sections 4 and 5 describe the integration scenarios of ACTiManager with KVM and OpenStack on top of the KMAX platform, and with MicroVisor and OpenStack on top of generic x86 hardware platforms. Section 6 describes the integration of HyperscaleJVM on top of the KMAX platform. Section 7 describes the integration of a very large in-memory MonetDB database instance on top of the Numascale platform, while Section 8 focuses on very large JVM-based databases (i.e., Neo4j), again on top of the Numascale platform. Section 9 describes the integration of ACTiManager with KVM on top of the Numascale platform. Section 10 describes the integration of MonetDB with ACTiManager for providing application-specific metrics for even better and more sensible resource management. Section 11 describes further integration actions regarding the MonetDB database. Finally, Section 12 provides conclusions and summarizes challenges and future work.

2 Virtualizing scale-out resources: Microvisor on KMAX

The goal of this integration scenario was to port the OnApp MicroVisor directly on top of the KMAX hardware, to combine the underlying hardware capabilities of KMAX with the flexibility provided by the MicroVisor.

2.1 Overview of the MicroVisor

Virtualization of server hardware is a commonly used practice to provide efficient resource management and it is a key technology for cloud computing, since it allows isolation between workloads running on multi-tenant servers through the Virtual Machine (VM) abstraction. A Hypervisor (HV) or virtual machine monitor (VMM) is a piece of software that creates, manages, and runs Virtual Machines (VMs).

The Hypervisor presents the guest operating systems in the VMs with a virtual operating platform and manages the execution of the guest operating systems. The VMs created by the hypervisor provide a secure environment, isolating user tasks from other tasks, applications, and other systems on the network. Tasks in this case entail the computation that takes place within an application as well as within the system kernel¹, so the hypervisor ensures security at both the application and operating system² kernel levels.

In addition to commonly deployed commercial hypervisors such as VMware, there are two dominant open-source hypervisor platforms: Kernel-based Virtual Machine (KVM)³ and the Xen Hypervisor⁴. In contrast to the KVM Hypervisor platform, the Xen Hypervisor provides a true Type I Hypervisor, since it runs directly on the bare-metal hardware, managing guest OS instances directly above it. Since there is no host operating system required, the Type I architecture is considered to be a minimal, high performance shim.

Despite being minimal and providing high performance, the standard architecture of Xen is based on a centralized control domain (Dom0 in Xen terminology) model, where split driver requests have to pass through the control domain for each request. Thus, the Dom0 becomes a bottleneck when many requests are being served to multiple guest domains (DomUs) and does not scale well with the increasing number of network and IO requests.

The Dom0 bottleneck is the main reason behind a new optimized approach called the “MicroVisor”, that is currently under constant development by OnApp. The main difference of the MicroVisor from Xen and KVM is that there is no control domain or host OS, which results in better scaling of network and IO, as well as lower virtualization overhead and higher performance for VMs (as already described in Deliverable 2.1).

The MicroVisor platform is designed to be a lightweight hypervisor platform that is better suited for emerging hardware platforms (e.g., Xen or KVM) compared to traditional hypervisors. ARM based micro-server hardware platforms, such as KMAX, incorporate many cores but which have lower performance than the more widely used x86 (Intel/AMD) server platforms. However, they are more energy efficient and are therefore entering the data center server market as a cost-effective alternative. For maximum applicability, the MicroVisor has been designed to work on both x86 (Intel/AMD) and ARM hardware platforms but its performance and power advantages are more pronounced for ARM-based processors.

¹ <https://www.webopedia.com/TERM/K/kernel.html>

² https://www.webopedia.com/TERM/O/operating_system.html

³ <https://www.linux-kvm.org/>

⁴ <https://www.xenproject.org/>

2.2 KMAX Hardware support for the MicroVisor

KMAX provides multiple compute units based on ARM, 64-bit, big.LITTLE architecture⁵ that can be used in clusters for cloud computing. These computing units are “Exynos-7420” SoC’s that form clusters for cloud computing applications. KMAX also provides high network bandwidth between each compute unit and also a high bandwidth storage path, which makes it unique for cloud computing applications.

For the integration of MicroVisor with KMAX, the following actions were performed by Kaleao:

Bare metal Exynos: Exynos were first developed on SDK (Software Development Kit) to check the peripherals and IO’s. The code that was supplied by the manufacturer was for an older kernel and significant effort was put in to port the kernel and driver to a later linux version, so as to give the project the advantage of making use of other newer subsystems, such as KVM, cgroups, heterogeneous multi-processing (HMP), for application development. For better performance, the PCIe drivers were modified from the original source. The original PCIe drivers provided by the manufacturer were for Gen2 and modifications were necessary to move to Gen3 PCIe. Therefore, the development of new modules was required: Kgex for network controller, kmax_pmu for communication with FPGA HW. The Exynos software subsystem was also modified to be compatible with other IO’s and new carrier board (Compute Node).

FPGA Development: Exynos 7420 is an energy efficient SoC with small footprint. KMAX HW Architecture incorporates four of these SoCs on a single board, referred to as a Compute Node. To have seamless connectivity between these SoC’s and also to adjacent Compute Nodes, it was necessary to build the HW blocks on the FPGA. These FPGA HW blocks enable high speed network and storage connections in the cluster.

Network FPGA development: The Xilinx 7045 FPGA, along with its Processing System (PS), is used to manage the networking on a single compute node hosting four Exynos. This is the Network Processing Unit (NPU). The PS on this Xilinx also acts as Management agent for all four Exynos of the compute node. Significant effort was placed in the development of this management functionality, such as shared memory communication between the PS and the compute node's Exynos, UART to each Exynos, network configuration of each Exynos, shared SD emulated booting mechanism, etc. Apart from these management functions, a lot of effort was dedicated to provide stable PCIe link between the four Exynos and the FPGA. In addition, various functionalities were developed to manage and provide networking to another Xilinx 7045 FPGA that is used as a Storage Subsystem.

The hardware IP blocks that are listed below were implemented into the FPGA to achieve the full functionality of the platform:

- ATA over Ethernet Switch (AOES)
- Network Interface Card (NIC)
- SD Emulator
- UART Multiplexer
- MSIX-Interrupt Support
- PCI End Point
- AXI DMA
- AXI interconnect
- NPU control

⁵ <https://www.arm.com/why-arm/technologies/big-little>

Storage FPGA development: Another Xilinx 7045 FPGA along with its PS is used to manage the Storage on a single compute node, known as the Storage Processing Unit (SPU). The Storage Subsystem, which includes the SPU, is capable of hosting a NVMe disk, that is shared with the four Exynos of the compute node. The PS on this Xilinx manages the storage disk on the compute node, providing direct PCIe link to the four Exynos to access the disk. The booting and configuration of the storage subsystem is managed by the Network FPGA and its PS. Different blocks were developed on the Storage subsystem to enable communication and configuration between the Network FPGA, the four Exynos, and itself. These different blocks are listed below:

- ATA over Ethernet Switch (AOES)
- MSIX-Interrupt Support
- PCI End Point
- AXI DMA
- AXI interconnect
- NPU control
- Chip-to-Chip
- AXI Memory Map to PCI Express (Root Complex for SSD)

Enabling High Speed Networking: A high speed 16 port switch “Marvel Prestera 98DX8216” is used to interface the compute nodes with the external world for application deployment and communication. Different set of activities were performed to configure and boot these high speed switches to keep the QoS high.

Delivery of HW: The integration of all of the above subsystems resulted in the delivery of the HW, made available for the consortium's partners to use. Full board management control was also developed to manage the various hardware resources. Two systems were eventually installed as part of ACTiCLOUD:

- Manchester University: 6 Blades (96 Exynos)
- OnApp Datacenter: 6 Blades (96 Exynos)

2.3 Integration of the MicroVisor on KMAX

For the KMAX integration with the MicroVisor platform, OnApp has worked closely with Kaleao to port necessary drivers, bootloaders and configuration scripts.

All the required components for the MicroVisor platform, including OpenStack with customized drivers, the monitoring statd service, the virxtd agent and Go API services, are packaged into a virtual machine filesystem by a set of build scripts that automate the final build package. These scripts were developed by the OnApp MicroVisor team to create a VM image (in binary form) that contains all that is needed for a complete MicroVisor Controller node. The scripts automate the build process and eliminate any human intervention, so that automated build servers (e.g., a Jenkins service) can build and/or deploy the controller image, depending on rules set by the team, such as daily builds, deployment to test clusters and so on. The diagram displayed below depicts the virtual device mappings between the KMAX hardware and the MicroVisor nodes.

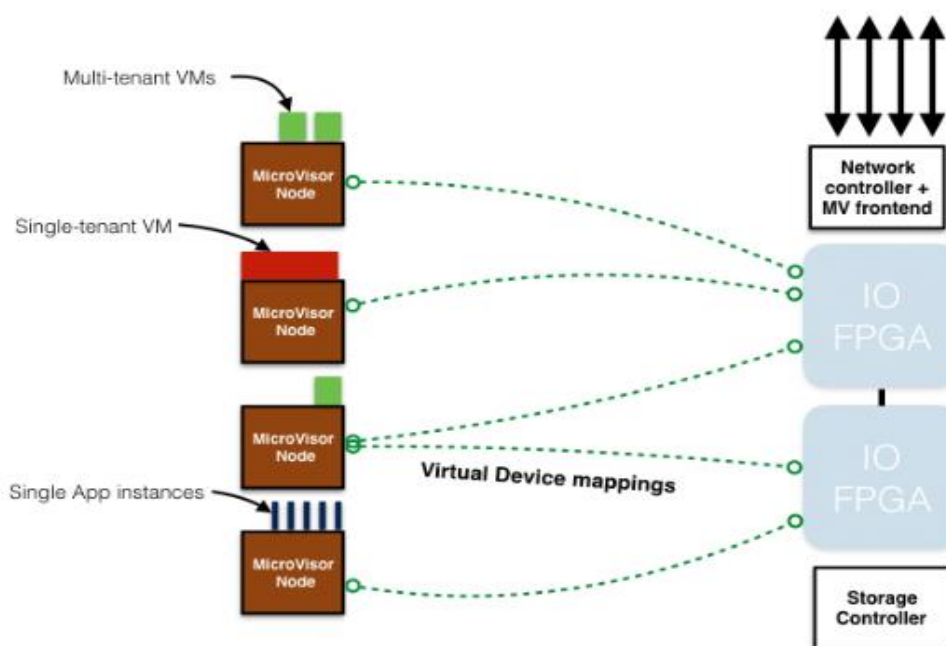


Figure 2.1: Virtual device mappings.

The installation scripts provide repeatable automation for several software components:

- For the deployment of the OpenStack controller VM, it was necessary to create an OpenStack image LUN that can be bootstrapped on top of the MicroVisor once it is running. On KMAX, this LUN can be accessed and installed/configured by directly opening an ssh shell session to the Storage Processing Unit (SPU). Once the SPU boots, it is possible to open an ssh shell to the storage node and configure the LUN.
- The installation of the MicroVisor on KMAX required the setup of the binary images and the proper boot parameters, so that compute nodes can boot the hypervisor and OpenStack controller VM. In the case of the KMAX system, it was necessary to bootstrap first each Network Processing Unit (NPU) on every compute node, which was performed using the base boot images provided by Kaleao. This was followed by loading and bringing up the Storage Processing Unit (SPU) on the Compute node by inserting the correct images (firmware, kernel, boot ramdisk, i.e., initrd) in the emulated SD card location, mounted on the NPU.
- In order to use the network on KMAX, it was also necessary to install and execute scripts for setting up the predefined MAC addresses and MAC address masks for all components (NPU, SPU, and Exynos SoCs). If the MAC addresses and masks are not properly configured, no Ethernet network traffic will be able to go through the KMAX network switches, to the storage node or externally. Supporting proper network operation for the hypervisor and VMs required significant changes to the MAC address block in the NPU firmware, as well as the software network switch in the MicroVisor.
- Next, the Exynos SoC compute nodes were booted with the correct MicroVisor image. That image contains the logic to enable the storage and network Processing unit NICs presented by the FPGAs over the PCI bus.
- Finally, the network processing unit (NPU) has to determine whether any of the local Exynos CPUs is responsible for running the OpenStack control node. If this is the case,

then the MicroVisor boot logic enables the virtual device on the SPU, sends out a broadcast message to the local Exynos MicroVisor nodes to announce the virtual device availability, and then instructs the proper MicroVisor node to map the ATA-over-Ethernet (AoE) virtual LUN and boot the OpenStack controller node using that LUN and the kernel image installed on the Exynos SoCs. All of this boot logic is triggered from the KMAX chassis controller, which ensures that the correct bootstrap logic is triggered from the initial NPU boot sequence.

2.4 Summary

The task of commissioning the KMAX hardware was completed successfully. Furthermore, new drivers and utilities for interaction between the FPGAs and processors were developed. The bare metal system was also optimized to the initial benchmark expectations. These drivers and utilities developed by Kaleao were passed over to OnApp to be integrated with the MicroVisor solution.

OnApp has integrated its MicroVisor on the KMAX hardware for the final prototype (M34), dealt with support and resolved incompatibility issues during the course of the project, in order to deliver a stable system. Many challenges have been addressed, in order for the prototype to be fully operational.

The challenges addressed on the software platform side, in order to port the MicroVisor to the Kaleao ARM 64-bit architecture, relate to:

- The isolated storage and network control points (FPGA accelerator units);
- Addressing hardware switch issues (e.g., support for broadcast/multicast Ethernet packets, and allowing MAC addressed for many virtual network interfaces); and
- Network device instability (e.g., packet loss or link shutdown).

The efforts to improve stability towards the final prototype were continuous and the final goal for a stable prototype that performs well within the ACTiCLOUD applications (JVM and column database) and overall framework was successful. A significant number of benchmarks have been acquired throughout the duration of the project, both for the intermediate as well as for the final evaluation, in order to verify the performance of the MicroVisor platform over KMAX, as will be reported in Deliverable 4.5.

3 Virtualizing cloud resources: MicroVisor on AWS

3.1 Introduction

One of the risk mitigation integration paths of ACTiCLOUD is the MicroVisor integration on Amazon Web Services (AWS) for running large datasets provided by MonetDB. AWS provides on-demand cloud computing platforms and APIs on a metered pay-as-you-go basis. Using a public cloud mitigates the risk of other ACTiCLOUD infrastructure failing to meet project deadlines.

In aggregate, the AWS cloud computing web services provide a set of primitive abstract technical infrastructure and distributed computing building blocks and tools. One of these services is Amazon Elastic Compute Cloud (EC2), which allows users to utilize a virtual cluster of computers via the Internet. AWS's version of virtual computers emulate most of the attributes of a real

computer including, hardware central processing units (CPUs) and graphics processing units (GPUs) for processing, local/RAM memory and hard-disk/SSD storage⁶.

The MicroVisor can be deployed on AWS bare metal servers on-demand as a direct replacement for the native AWS hypervisor, providing performance gains and lower costs compared to high-performance instances offered by AWS. This product is currently being developed and previewed by Sunlight.io, a spin-off company of OnApp, which is working on the commercial version of the Microvisor technology, as described in Deliverable 5.7: "Final exploitation plans". The commercial hypervisor product will be named NexVisor and the platform on AWS will be referred to as "Sunlight on AWS". A MicroVisor deployment on the AWS bare metal servers aggregates the direct attached NVMe flash drives together, across a cluster, into a massive pool of high performance storage with redundancy for availability and failover features⁷.

3.2 Deployment

The MicroVisor can be deployed as a fully managed platform on a cluster configured on the AWS bare metal infrastructure. The currently supported infrastructure option is AWS i3.metal servers. Amazon i3 instances are Storage Optimized instances for high transaction, low latency workloads. On the AWS EC2 environment, the MicroVisor platform is installed in a multi-node cluster, each MicroVisor node being installed and configured on a full bare metal server and multiple nodes are connected and communicate using the AWS networking infrastructure. Each node is equipped with a number of network interfaces used for both management and data, for which the Microvisor platform provides support through specialized NIC drivers. The total aggregate network bandwidth that can be achieved by a node is capped by the AWS EC2 instance capability. The storage capacity is also based on the bare-metal instance specification, as reported in section "Supported AWS bare-metal instances" below.

Deploying MicroVisor on AWS requires the following two major steps.

1. AWS Environment Preparation. This step is related to setting up the account for using AWS resources⁸, preparing a key pair to be used for the instances on AWS⁹ and creating an AWS t2.micro instance using any Linux image, which serves as a manager to manage the MicroVisor cluster on AWS¹⁰.
2. MicroVisor on AWS Dashboard Deployment. This step is analyzed below.

High Performance Storage

Amazon i3 instances include Non-Volatile Memory Express (NVMe) SSD-based instance storage optimized for low latency, very high random I/O performance, and high sequential read throughput, delivering high IOPS and up to 25 Gbps of network bandwidth.

Supported AWS Bare-metal Instances

AWS i3 instances offer bare metal size that provide applications with direct access to the compute and memory resources of the underlying next generation AWS hardware and software infrastructure. Bare metal instances allow running a variety of workloads on AWS, including workloads that benefit from direct access to physical resources. The following servers have been validated to be officially supported (by OnApp) for the MicroVisor platform on AWS.

⁶ https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud

⁷ <https://www.sunlight.io/sunlight-on-aws/>

⁸ <https://docs.aws.amazon.com/general/latest/gr/aws-security-credentials.html>

⁹ <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html#having-ec2-create-your-key-pair>

¹⁰ https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html

AWS Instance Type	CPU	Memory	Network	Storage
i3.metal	Intel Xeon E5-2686 v4	512 GB	25 Gbps	8 x 1.7TB NVMe
i3en.metal	Intel Xeon Scalable (skylake)	768 GB	100 Gbps	8 x 7.5TB NVMe
z1d.metal	Custom Intel Xeon 3	384 GB	25 Gbps	2 x 900GB NVMe

MicroVisor on AWS Dashboard Deployment

MicroVisor on AWS Dashboard is a MicroVisor predefined Amazon Machine Image (AMI). It includes all the necessary tools to run the Infrastructure Manager for the MicroVisor platform. The tools we have used and provide for the Infrastructure Manager include:

- Nginx (in order to serve the UI part of the MicroVisor)
- Mysql (stores user data)
- Nodejs (is used to run the server/client side of the UI)
- Pm2 (is used to manage the server side of the UI)
- Python (is used to run the API of the MicroVisor)
- Letsencrypt (is used in the case of securing a domain name)

Creating a MicroVisor AWS Dashboard VM

In order to deploy a Dashboard VM for configuring the Microvisor platform on AWS, it is essential to select the AMI from the AWS marketplace images (using either the AWS Console, or AWS cli/api), also choosing one of their AWS stored ssh keypairs, for ssh access to the VM. The user must make sure that the http, https, and ssh services are allowed in the Network Security Group configuration. In order to run the MicroVisor Infrastructure Manager, the following steps have been followed:

Step 1: As soon as the VM is built, it is necessary to assign an elastic IP to that VM. At this stage, the control panel UI via this IP can be viewed. The objective in this step is to assign/create a domain or a subdomain pointing to the elastic IP of this new VM, so that it will allow us to create an SSL (issued by Let's Encrypt ¹¹). In this way, a secured (https) connection to the UI of the control panel can be established, providing encrypted communications between the web client and the UI. Then, a domain/subdomain entry on the DNS service has to be created. A new (or existing) Elastic IP can be allocated through the AWS EC2 dashboard, which then can be associated to the MicroVisor_on_AWS_dashboard instance. Once the DNS change has been performed and the records have been propagated, the MicroVisor default UI login web page can be accessed (through the HTTP protocol), by visiting the domain/subdomain assigned to the elastic IP.

Step 2: This step refers to the configuration of the manager, in order to respond to url requests, under https, to the domain/subdomain assigned to the elastic IP. This step is performed under the setting options of the MicroVisor, by inserting the preferred domain/subdomain at the "secure your domain" section.

Step 3: This final step is to enter the AWS key credentials provided by OnApp, through a secure https connection via the MicroVisor UI.

3.3 Summary

The evaluation results of the implementation described above, using MonetDB datasets, are reported and analyzed in Deliverable 4.5 "ACTiCLOUD Final Evaluation".

¹¹ <https://letsencrypt.org/>

4 Efficient management of scale-out heterogeneous resources: KMAX - KVM - OpenStack - ACTiManager

4.1 Introduction

The goal of this integration scenario is to combine all components from ACTiManager v2.0 (described in Deliverable 2.4: "Distributed Cloud Resource Manager v2.0") on the KMAX platform. ACTiManager is an advanced resource manager that supports the core goals of ACTiCLOUD towards resource-efficient IaaS cloud offerings. In this integration scenario, we use the KVM virtualization technology instead of OnApp's MicroVisor because ACTiManager was developed for the OpenStack Pike version, while the MicroVisor was intended for use with the OpenStack Kilo version, and only later was ported to Pike. The porting was completed in the last phase of the project and targeted commodity x86 platforms. Hence, to avoid any delays on this integration scenario that builds on top of the KMAX platform, we decided to use the KVM hypervisor on top of the KMAX platform. This integration scenario involves the following components of the ACTiCLOUD architecture:

- KMAX platform (hardware)
- KVM-QEMU/Linux virtualization technology (hypervisor)
- OpenStack Pike (core cloud manager)
- ACTiManager v2.0 (resource manager)

4.2 KMAX support for KVM

Kaleao's contributions for supporting KVM on KMAX were:

- Bare metal Exynos bringup
- Network FPGA development
- Storage FPGA development
- Delivery of HW
- KVM-QEMU porting

The first four components have been described in detail in Section 2.

KVM-QEMU porting: Once the bare metal system was up and running and stable it was Kaleao's responsibility to port the KVM-QEMU subsystem. The initial porting and building of KVM-QEMU were completed successfully but there were issues getting a guest up and running. More detailed debugging showed that accessing one of the HW registers was not supported by the manufacturer. This HW issue took significant time and effort to resolve so that KVM can work on KMAX. This has enabled Kaleao to deliver the KVM-QEMU to the ACTiCLOUD project.

4.3 Integration of KVM, OpenStack, and ACTiManager on KMAX

For the KVM / OpenStack / ACTiManager integration setup on the KMAX system, we utilize a single compute node of a KMAX blade, consisting of four Exynos servers. We describe briefly the challenges that we encountered and the corresponding solutions that we applied.

4.3.1 Integration with KVM

QEMU on big.LITTLE cores

QEMU failed to start properly when instructed to boot on both big and little cores, preventing us from properly booting VMs with QEMU/KVM on KMAX. Our solution to this was to always set the

affinity of the QEMU command to one group of cores, e.g., the group of big cores. This only affects the initial VM spawning, since after that the VM's vCPUs can be re-pinned to both big and little cores.

Note that this was not initially an issue on KMAX, because KALEAO installs an additional init script that runs at boot time. The script works as follows: (i) it splits the two groups of cores at boot time in two separate cpuset cgroups (i.e., the big group and the little group, respectively), and (ii) it assigns the init / systemd process (with pid equal to 1) to the big group. The bash shell that runs QEMU is effectively spawned by systemd (throughagetty, login, etc.), so it is also in the same cgroup. The cpuset cgroup affinity overrides the one set by the user, using taskset, hence even 'tasksetting' QEMU to run on both big and little cores will effectively run QEMU only on big cores, hiding the original problem. The problem is exposed when removing the shell (or systemd) from the big cpuset group. We solve this by wrapping the QEMU/KVM binary around taskset.

QEMU EFI Firmware on Exynos

The Exynos firmware sets incorrectly the CNTFRQ register. The register is set to 0, instead of the actual Exynos system timer frequency that is 24MHz. Both QEMU and the QEMU-EFI firmware fail to work when this is not set properly. KALEAO had provided a patched QEMU binary for Exynos as mentioned above; we additionally patched EDK2 and built a new QEMU-EFI firmware (Appendix A).

QEMU-EFI is needed, because Openstack Pike only supports (U)EFI boot for the AARCH64 architecture, as is the case in Exynos. Note that for testing purposes, directly booting a kernel from QEMU is possible for non-Openstack VMs. For a manual EFI boot, QEMU needs to be provided with the EFI firmware and a (64M) RW NVRAM image, in order to store EFI variables.

Custom QEMU binary and QEMU-EFI firmware

As mentioned above, we used a patched QEMU binary and a patched QEMU-EFI firmware image. The patched QEMU binary is placed under /usr/local/bin and we also modified the distro-provided /usr/bin/kvm.

The QEMU-EFI firmware replaces the distro-provided one, under /usr/share/AAVMF/AAVMF_CODE.fd. These paths should be configurable in Openstack, but trying to set a custom path for /usr/bin/kvm did not initially work, so we fell back to just replacing the distro-provided files.

VM Networking

The L2 VM networking does not work out of the box due to limitations of the FPGA NPU on KMAX. The NPU switch does not implement MAC address learning. Instead, the NPU switch relies on forwarding 4 fixed MAC addresses per port.

To solve that issue and get L2 VM networking to work properly, we create (up to 4) MacVTap interfaces with specific MAC addresses and pass them to QEMU.

This should be the easiest way to get OpenStack VMs with working network interfaces. Alternatively, for testing purposes, one can manually spawn QEMU VMs through QEMU's SLIRP¹² userspace networking implementation, which has some limitations, e.g., lack of support for ping by default (although that could be possible by tweaking ping_group_range), and need for port forwarding as the VM network is effectively behind a NAT, among others.

¹² <https://wiki.qemu.org/Documentation/Networking>

Kernel and device tree issues

Earlier KMAX kernel images lacked support for some features needed by Openstack. More specifically:

- We enabled NFS¹³ (server) to get live instance storage for live migrations.
- We enabled ebttables¹⁴, that is needed by the neutron L2 bridge plugin for networking, although ebttables seem unnecessary when using the macvtap driver. Note that this could be bypassed with a small patch of ebttables and arp-spoofing-protection function under neutron/plugins/ml2/drivers/linuxbridge/agent/arp_protect.py.
- We modified the corresponding entries in the linux device tree and enabled both Exynos PMUs for big and little cores, to get access to the hardware performance counters with perf. Note that the perf binary was provided by KALEAO.

4.3.2 Integration with OpenStack

We setup the OpenStack Controller on one Exynos server, and the OpenStack Telemetry services on another one in order to split the memory load between the two servers. Finally, we setup the other two Exynos servers as the compute nodes of our OpenStack installation.

All of the Exynos servers run a modified version of Ubuntu 16.04 (Xenial), provided by KALEAO, which also includes a modified Linux kernel (v4.4.16). We also used a patched QEMU binary (v2.9.0), since the vanilla QEMU did not work on Exynos (Section 2).

Our setup is based on the OpenStack Pike version. We installed: (i) the OpenStack Identity (Keystone) and Image Registry (Glance) services on the first Exynos, (ii) the Metering (Ceilometer) and Metric (Gnocchi) services on the second Exynos, and (iii) the Compute (Nova) and Network (Neutron) services, along the necessary Metering agents, on the other two Exynos servers of the KMAX node. We followed the OpenStack installation guide¹⁵ to perform the setup.

Accommodating OpenStack services on KMAX resources

We had to fine-tune the resource usage of the various OpenStack services, especially on the Controller node, to accomodate for the limited memory of the KMAX Exynos servers.

Enabling Live Migration

We also set up live migrations between the Compute nodes, in order to enable the external component of ACTiManager to move VMs between the Compute nodes and apply prioritisation, consolidation, and load balance policies.

OpenStack modifications

We modified the Nova service to pass-through the information of the host CPU to the VMs, and extended the commands for creating new VMs. We also instructed OpenStack to set up an L2 provider network with a macvtap linux agent on the compute nodes. While the interface works with the macvtap driver, the neutron DHCP agent does not work properly, so the VM fails to get automatically an IP address at boot time. In addition, the neutron DHCP agent will not work probably during migrations. We are currently investigating whether it is possible to switch ports while migrating an instance. Otherwise, we will probably need to patch the neutron macvtap-agent.

¹³ https://en.wikipedia.org/wiki/Network_File_System

¹⁴ <https://ebtables.netfilter.org/>

¹⁵ <https://docs.openstack.org/install-guide/>

4.3.3 Integration with ACTiManager

Once the OpenStack installation was completed, we installed and set up ACTiManager v2.0. More specifically, we patched the OpenStack with the ACTiManager.external component to manage the placement of VMs at blade level (that consists of four Exynos), and we installed the ACTiManager.internal component on each Exynos to manage the pinning of VMs on big and LITTLE cores.

4.3.4 Test and Validation

To test the functionality of the integration scenario, we target the placement logic of ACTiManager in KMAX (regarding the external component) and the mapping of VMs to big/little cores in the Exynos servers (regarding the internal component). More specifically, we spawn several VMs, which we pre-label as gold or silver using the global anti-affinity prioritization groups. The ACTiManager.external takes special care when placing VMs on servers, e.g., it places gold VMs on separate physical servers, when they are created. Once the VM is placed on a server, the internal component of ACTiManager periodically checks the node state and remaps the instances, so that gold instances exclusively run on the big cores of the Exynos, while the silver instances run on the little cores. The details and the results of these experiments will be included in the Deliverable D4.5 "ACTiCLOUD Final Evaluation".

4.4 Summary

This integration scenario focused on integrating the OpenStack Pike version and ACTiManager v2.0 on the KMAX system using the KVM-QEMU virtualization technology. The initial task of bringing up the KMAX hardware was successful. New drivers and utilities for interaction between the FPGAs and processors were also implemented successfully. The bare metal system was also optimized to achieve benchmark expectations. The KVM-QEMU system was ported and a guest VM was booted on it. OpenStack Pike and ACTiManager were integrated on KMAX with several VMs running on the processors under the control of ACTiManager. The integration has been completed successfully, providing functionality for all components after addressing several challenges.

5 Generic cloud resource management the ACTiCLOUD way: MicroVisor - ACTiManager

5.1 Introduction

The goal of this integration scenario is to put together ACTiManager with MicroVisor. In this integration scenario, we use commodity x86 hardware platforms. We chose to focus on commodity x86 hardware platforms instead of Numascale's system, which is also based on x86 architecture, due to the integration challenges regarding the porting of MicroVisor on top of the Numascale platform, as documented in D4.2 "ACTiCLOUD Intermediate Prototype" that made us decide to bring forward as "mainline" setup the Linux/KVM hypervisor for the Numascale platform. It has to be noted that stability and performance issues of the XEN hypervisor have continued to exist throughout the second year of the project, despite the numerous efforts performed by OnApp and Numascale, in order to resolve them. The main issue encountered is related to the VM deployment and execution, which is performed only on the first server of the available resources. Accessing other servers in the integrated system resulted in hangs. The microkernel of Xen does not have all the features that a standard Linux OS kernel has (as used by KVM for host OS). Therefore, the Numascale bootloader required serious modifications to support the approach of Xen microkernel, including:

- A new optimal bootloader that provided the correct hardware configuration information, including specialized Numascale timers and the NUMA topology of the system.
- A Xen patch that implements the Numachip APIC driver. With this patch a user just has to boot a kernel with Xen enabled and it will use the hypervisor adjustments.
- Finally, we found that Xen hypervisor had a livelock race in serial port writing (including the current release 4.10.0) which was preventing debugging core booting; a workaround has been implemented to proceed with further debugging. This has been fixed with a patch from Numascale.

All these issues made it very time consuming in finding and developing the most proper way to resolve it (including debugging Xen issues), alternative actions had to be devised and the most prominent one is analyzed below. The high level of KVM support in the latest linux kernels that are so important for Numascale technology makes this a very good mitigation path.

Hence, this integration scenario involves the following components of the ACTiCLOUD architecture:

- Commodity x86 platform (hardware)
- MicroVisor virtualization technology (hypervisor)
- OpenStack Pike (core cloud manager)
- ACTiManager v2.0 (resource manager)

5.2 Integration of MicroVisor and OpenStack

The integration of MicroVisor with OpenStack Pike version has been described in detail in Section 4 of D2.3 "Rack scale MicroVisor v2.0".

5.3 Integration of ACTiManager and MicroVisor

For the integration setup of MicroVisor, OpenStack, and ACTiManager on commodity x86 platforms, we utilize a single controller node and two compute nodes. We describe briefly the challenges that we encountered and the corresponding solutions that we applied.

5.3.1 ACTiManager.external

ACTiManager.external is responsible for the placement of VMs across different compute nodes. ACTiManager.external is mostly agnostic to the underlying hypervisor technology as it communicates with OpenStack to apply decisions at site level. Hence, we setup the external component of ACTiManager in the controller node of the OpenStack installation. The main functionality of ACTiManager.external is located in new OpenStack filters and weights. As MicroVisor is integrated with OpenStack through a DevStack setup, we install the new filters and weights in the proper directories, as explained in Appendix B. Then, the ACTiManager.external component is responsible for placing the VMs across the different compute nodes taking into account the characteristics of the VMs for interference mitigation.

5.3.2 ACTiManager.internal

ACTiManager.internal is responsible for the placement of VMs within a compute node. In contrast to the external component, ACTiManager.internal relies heavily on the underlying hypervisor technology as it communicates with the hypervisor to collect metrics and apply decisions at node level. In a system with QEMU/KVM virtualization technology, ACTiManager.internal runs locally as a process/daemon interacting directly with the host of that node. However, the architecture of MicroVisor does not support that mode of execution for ACTiManager.internal as there is no actual control domain (Dom0 in Xen terminology) or host OS as with QEMU/KVM. To allow communication with the local instance of the MicroVisor that runs on a compute node and decision making, OnApp provides an API that includes a command that allows the pinning of a VM on specific cores.

To mitigate the limitation of no control domain with MicroVisor, we slightly modified ACTiManager.internal to run on the OpenStack controller node (instead of the compute node) and issue remote Microvisor-specific commands to the corresponding compute node. On the controller node the following components of ACTiManager are running: (i) a single External instance as described earlier, and (ii) multiple Internal instances with each one controlling the placement of VMs within a certain server. In addition, we modified ACTiManager.internal to use OnApp's API for pinning VMs to the server's physical cores.

5.3.3 Test and Validation

To test the functionality of the integration scenario, we target the placement logic of ACTiManager on a cluster of servers (regarding the external component) and the mapping of VMs to physical cores in those servers (regarding the internal component). More specifically, we spawn several VMs, which we pre-label as gold/silver, noisy/quiet, and sensitive/insensitive. The ACTiManager now takes special care when selecting servers and cores to place VMs.

5.4 Summary

This integration scenario focused on putting together the ACTiManager v2.0 with the MicroVisor and OpenStack Pike on top of commodity x86 hardware platforms.

6 Scaling Java workloads on low power processors: Hyperscale JVM on AArch64 & KMAX

The ACTiCLOUD platform features two different instruction set architectures (ISAs) at its hardware layer, x86-64 on the NUMASCALE platform and AArch64 on the KMAX platform. The core of the Hyperscale JVM at the beginning of the project did not support AArch64, and thus it could not run on KMAX. As described in D3.2 Hyperscale JVM v1.0 and D3.6 Hyperscale JVM v2.0, in the context of the ACTiCLOUD project the core of Hyperscale JVM has been ported and optimized to support the AArch64 ISA and thus KMAX.

To demonstrate the integration of the Hyperscale JVM with the AArch64 ISA and thus the KMAX system we run monetDBLite-Java and the DaCapo benchmark suite¹⁶ with the Hyperscale JVM on an Odroid-C2¹⁷ and on the KMAX platform. The results are satisfying showing 100% pass-rate for the TPC-H benchmark on monetDBLite-Java and 71-78% for the DaCapo benchmark suite. Unfortunately, we were not able to test Neo4J due to an OpenJDK bug that prevents us from building it on AArch64.

6.1 DaCapo Benchmark suite

Hyperscale JVM achieves 64.29% overall pass-rate for the DaCapo benchmark suite on the KMAX platform. In comparison, HotSpot JVM, the official OpenJDK JVM, achieves 71.43% overall pass-rate (supports one more benchmark). If we consider only the benchmarks working with HotSpot JVM, then **Hyperscale JVM achieves a 90% pass-rate**. Namely the benchmarks that fail in both VMs are batik, tomcat, tradebeans, and tradesoap, while the benchmark that fails only on the HyperscaleJVM is eclipse.

6.2 MonetDBLite-Java

For the integration testing of Hyperscale JVM on KMAX we use the TPC-H benchmark running on MonetDBLite-Java (<https://github.com/acticloud/tpch-monetdblite>) which completes successfully after executing all 22 queries on both the Odroid-C2 and the KMAX platform.

6.3 Neo4J

Neo4J fails to build on KMAX due to a bug in OpenJDK. The bug has been reported to Oracle and was assigned internally the ID 9062842. Figure 6.1 presents a screenshot of the bug report. At the time of writing we have not heard back from Oracle regarding the progress of this issue. This bug prevents Neo4J from being built and thus ran on the KMAX platform. This regression has a small impact on the integration of the ACTiCLOUD stack in that Neo4J is designed for shared memory systems while the KMAX platform is a distributed memory platform.

¹⁶ <http://dacapobench.org/>

¹⁷ <https://www.hardkernel.com/shop/odroid-c2/>

ORACLE Java Bug Database

Search

Oracle Technology Network > Java > Java SE > Community > Report a Bug

Bug Report Submission Complete

Thank You

Thank you for taking the time to provide us with your details.

What's Next

We will review your report and have assigned it an internal review ID : 9062842. Depending upon the completeness of the report and our ability to reproduce the problem, either a new bug will be posted, or we will contact you for further information.

We will try to process all newly posted bugs in a timely manner, but we make no promises about the amount of time in which a bug will be fixed. If you just reported a bug that could have a major impact on your project, consider using one of the technical support offerings available at [Oracle Support](#).

Thank you again for using this site.

Figure 6.1: Bug report.

```
[ERROR] # A fatal error has been detected by the Java Runtime Environment:
[INFO] #
[INFO] # SIGILL (0x4) at pc=0x0000ffff905fcfd0, pid=27583, tid=0x0000ffff9cc0b200
[INFO] #
[INFO] # JRE version: OpenJDK Runtime Environment (8.0_222-b10) (build 1.8.0_222-8u222-b10-lubuntu1-16.04.1-b10)
[INFO] # Java VM: OpenJDK 64-Bit Server VM (25.222-b10 mixed mode linux-aarch64 compressed oops)
[INFO] # Problematic frame:
[INFO] # J 8474 C2 scala.reflect.internal.Scopes$Scope.enterAllInHash(Lscala/reflect/internal/Scopes$ScopeEntry;I)V (107 bytes) @ 0x0000ffff905fcfd0 [0x0000ffff905fcfc0+0x10]
[INFO] #
[INFO] # Failed to write core dump. Core dumps have been disabled. To enable core dumping, try "ulimit -c unlimited" before starting Java again
[INFO] #
[ERROR] # An error report file with more information is saved as:
[INFO] # /home/kaleao/neo4j/hs_err_pid27583.log
[INFO] #
[INFO] # If you would like to submit a bug report, please visit:
[INFO] # http://bugreport.java.com/bugreport/crash.jsp
[INFO] #
```

Figure 6.2: Neo4j build error.

7 Running very large databases in Memory

7.1 Introduction

The most important applications running on next generation Numascale hardware platform¹⁸ are databases. Numascale wants to be able to run any large database and has been targeting this with its presence in ACTiCLOUD, with specific interest in MonetDB and Neo4j databases. Prior to ACTiCLOUD, Numascale had not been able to run database workloads of sizes close to one TB successfully. Given the integration and support from ACTiCLOUD partners MonetDB, Neo4j and UNIMAN, a database that holds more than 1TB in memory worked successfully for the first time. This has been validated with the MonetDB test in this section. This experience has been proven valuable towards Numascale's goal to support better the SAP HANA database that comprises the target workload for 80% of the Numascale systems that are sold. SAP HANA¹⁹ (high-performance analytic appliance) is an application that uses in-memory database technology that allows the processing of massive amounts of real-time data in a short time. The in-memory computing engine allows HANA to process data stored in RAM as opposed to reading it from a disk. Numascale is in the process of qualifying SAP HANA for systems with 16 sockets and 24 TB of RAM.

7.2 MonetDB on Numascale Shared Memory System

One of the major targets for ACTiCLOUD was to run databases on TBs of RAM. We have struggled to get the ACTiCLOUD systems used for prototyping to run successfully with databases when utilizing larger than 600 GB of RAM. It turned out that this was due to a BIOS bug in the AMD Opteron systems delivered by Supermicro, that were installed in Cambridge and Athens. However, we have succeeded running MonetDB on a large Numascale Shared Memory Server with Intel architecture. The results from experiments with using the SF1000 dataset (1TB) are available in a Google sheet produced by MonetDB²⁰.

Appendix C contains details on how to run a 1TB dataset in RAM on the latest Numascale Shared Memory system (a short version is presented here). The system is a BullSequana S8²¹, also known as the fastest SAP HANA system in the world²². We are able to use it as an ACTiCLOUD prototype, benefitting from all the system libraries and NUMA optimizations created in ACTiCLOUD. It exposes its NUMA topology in the same way as previous generations of Numascale systems, but using the Intel Xeon architecture instead of AMD Opteron. Numascale is inherently using Numascope (developed in ACTiCLOUD) to analyze the workloads on BullSequana systems too.

The results below show that for this benchmark (MonetDB with TPC-H and SF1000 dataset, i.e., 1TB), 56 cores are used for the shortest runtime.

56 cores

```
time ./horizontal_run.sh -d SF-1000 -n 4
```

..... *

¹⁸ Numascale node controllers enable the 16 socket and 32 socket version of the The BullSequana X800 series, <https://atos.net/en/products/high-performance-computing-hpc/bullsequana-x-supercomputers>

¹⁹ <https://www.sap.com/products/hana.html>

²⁰ https://docs.google.com/spreadsheets/d/1W6hfCN_F0TtBDgZOk2esW-oWKmz6cIs0ZVfgz6ho06s/edit#gid=1581861910

²¹ <https://atos.net/en/products/enterprise-servers/bullsequana-s>

²² https://atos.net/en/2019/news_2019_06_13/bullsequana-s800-becomes-the-highest-performing-server-on-the-market-world-wide-for-sap-hana-with-record-benchmark-result

```
-real 152m25.501s
```

```
user 0m6.294s
```

```
sys 0m3.084s
```

112 cores

```
time ./horizontal_run.sh -d SF-1000 -n 4
```

```
..... *
```

```
-real 175m55.426s
```

```
user 0m6.644s
```

```
sys 0m3.297s
```

224 cores

```
time ./horizontal_run.sh -d SF-1000 -n 4
```

```
..... *
```

```
-real 179m45.244s
```

```
user 0m6.566s
```

```
sys 0m3.300s
```

7.3 Summary

The results in this chapter show that MonetDB scales well when the memory footprint is large, but does not benefit from more cores than 56. NSCALE is now working intensively in optimizing databases on the BullSequana platform, building heavily on lessons learned in ACTiCLOUD. We have a strong belief that graph databases will run very well on the Bullsequana too and are eager to continue working with ACTiCLOUD partners optimizing databases after the project.

8 Scaling-up cloud resources to support large JVM-based databases: Numascale - KVM - Hyperscale JVM - Neo4j - MonetDB Lite

As the amount of data stored in databases keeps increasing, large databases require more and more memory in order to keep as much data as possible in-memory in order to speed up processing. The ACTiCLOUD platform enables large databases to scale-up providing them with potentially terabytes of shared main memory and hundreds of processing cores. The resources can be provisioned on demand and scale up and down using virtualization. To improve the performance of virtual machines, ACTiCLOUD's hypervisors expose NUMA topologies to the guest VMs through which they are also exposed to the Hyperscale JVM. By making the whole stack NUMA-aware and by exposing the topologies in each layer, ACTiCLOUD provides the foundations for large JVM-based database deployments.

8.1 Integration testing

To demonstrate the integration of the above in ACTiCLOUD we perform a set of experiments. For our experiments we deploy two different configurations of the same VM, one that is NUMA-oblivious and another that is NUMA-aware. Both configurations offer 48 virtual CPUs, pinned on 48 physical CPUs, and 192GBs of virtual memory. The key difference between the two configurations is that in the NUMA-aware configuration we take advantage of KVM's support for NUMA topologies and expose to the guest VM the actual NUMA topology of the underlying system. We use the Numascale testbed deployed in Oslo. The 192 GBs in the NUMA-aware instance are split in twelve 16GB-sized numa-nodes that are pinned on the first 12 physical numa-nodes of the testbed. The virtual numa-nodes are configured in such a way to reflect the same topology as the physical numa-nodes that they are pinned on. This configuration allows the Guest-OS and thus the Hyperscale JVM as well to understand the underlying memory topology. The following two snippets demonstrate the output of "numactl -H" from a NUMA-oblivious (i.e. with a single virtual NUMA-node) and a NUMA-aware VM instance (with multiple virtual NUMA-nodes, corresponding to the physical ones) respectively:

node distances:

```
node    0
0:    10
```

node distances:

```
node    0    1    2    3    4    5    6    7    8    9   10   11
0:    10    16    16    22    16    22   106   112   100   106   106   112
1:    16    10    22    16    22    16   106   112   100   106   106   112
2:    16    22    10    16    16    22   106   112   100   106   106   112
3:    22    16    16    10    22    16   106   112   100   106   106   112
4:    16    22    16    22    10    16   106   112   100   106   106   112
5:    22    16    22    16    16    10   106   112   100   106   106   112
6:   106   112   100   106   106   112    10   16   16   22   16   22
7:   106   112   100   106   106   112    16   10   22   16   22   16
```

```

8:  106  112  100  106  106  112  16  22  10  16  16  22
9:  106  112  100  106  106  112  22  16  16  10  22  16
10: 106  112  100  106  106  112  16  22  16  22  10  16
11: 106  112  100  106  106  112  22  16  22  16  16  10

```

On top of both the VMs we run two JVM-based database workloads, one for each database use-case of ACTiCLOUD. For MonetDB we run TPC-H²³ on top of MonetDBLite-Java v2.39 and for Neo4J we run LDBC SNB²⁴ on top of Neo4J v3.5.4. In all cases we use Hyperscale JVM for running the databases. As a result the software stack involved in those experiments integrates the following parts of the ACTiCLOUD architecture:

- Numascale platform (hardware)
- Numascale system libraries in the Host OS
- KVM-QEMU/Linux virtualization technology (hypervisor)
- Hyperscale JVM
- Neo4J (database)
- MonetDBLite-Java (database)

An illustration of the integration stack exploited in our experiments is presented in Figure 8.1.

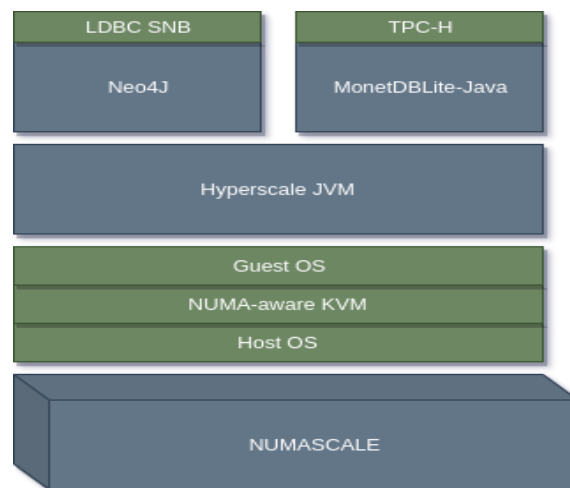


Figure 8.1: Integration stack.

Our experiment results demonstrate that in the case of the NUMA-aware VM the whole software stack becomes aware of the hardware NUMA topology allowing it to achieve better performance when running LDBC SNB on Neo4J and TPC-H on MonetDBLite-Java. The Numascale platform provides the necessary hardware resources for scaling up large JVM-based databases. The design principles of the NUMA-aware system libraries available on the ACTiCLOUD architecture are used to expose the underlying topology to the guest VMs and in turn to the Hyperscale. The JVM-based databases, Neo4J and MonetDBLite-Java remain NUMA-oblivious and rely on the hyperscale JVM in order to exploit the underlying Numa topology. A more detailed evaluation of the ACTiCLOUD system performance will be available in D4.5: "ACTiCLOUD Final Evaluation".

²³ <https://github.com/acticloud/tpch-monetdblite/blob/maxine/tpchbenchmark/pom.xml>

²⁴ <http://ldbcouncil.org/benchmarks/snb>

9 Efficient management of scale-up resources: Numascale - ACTiManager

9.1 Introduction

A specific component called ACTiManager.internal has been created to support large scale-up resources commonly used for the very large database solutions. ACTiManager.internal has been designed in a NUMA-aware fashion benefitting from the NUMA topology in very large systems such as Numascale. While ACTiManager.internal decides which NUMA node(s) an application can be allocated, the actual enforcement is done from the lower level. As a result, this has been done in the correct way by building on NUMA support in the kernel for modern Linux distributions from Centos and Ubuntu. In this section we provide details of how this integration was achieved.

9.2 Bringing up KVM and adding NUMA topology to the guest OS

In order to achieve NUMA-awareness in the guest OS of ACTiCLOUD we need to run the latest hosts OS's available as they have better toolchain and pthreads behavior. These are supported by CentOS 8 or Ubuntu 18.04.

9.2.1 Moving to CentOS 8

For this purpose Numascale had to re-install the root partition on the ACTiCLOUD server with CentOS 8 and prepared a newer 3.19 kernel with CentOS 8 and Numaconnect support. KVM on CentOS 8 natively supports NUMA topology as shown in the following commands:

```
numademo$ man 1 qemu-system-x86_64
...
-numa node[,mem=size][,cpus=firstcpu[-lastcpu]][,nodeid=node]
-numa node[,memdev=id][,cpus=firstcpu[-lastcpu]][,nodeid=node]
-numa dist,src=source,dst=destination,val=distance
-numa cpu,node-id=node[,socket-id=x][,core-id=y][,thread-id=z]
```

In this way one can define a NUMA node, assign RAM and virtual cpus (VCPUs) to it and also set the NUMA distance from a source node to a destination node.

Legacy vCPU assignment uses cpus option where firstcpu and lastcpu are CPU indexes. Each CPU option represent a contiguous range of CPU indexes (or a single vCPU if lastcpu is omitted). A non-contiguous set of vCPUs can be represented by providing multiple cpus options. If the "cpus" parameter is omitted on all nodes, vCPUs are automatically split between them.

For example, the following option assigns vCPUs 0, 1, 2 and 5 to a NUMA node:

```
-numa node,cpus=0-2,cpus=5
```

The "cpu" option is a new alternative to the "cpus" option which uses socket-id|core-id|thread-id properties to assign CPU objects to a node using topology layout properties of CPU. The set of properties is machine specific, and depends on the used machine type/smp options. The node-id property specifies the node to which the CPU object will be assigned. It is required for a node to be declared with the node option before it's used with cpu option.

For example:

```
-M pc \
-smp 1,sockets=2,maxcpus=2 \
-numa node,nodeid=0 -numa node,nodeid=1 \
-numa cpu,node-id=0,socket-id=0 -numa cpu,node-id=1,socket-id=1
```

9.3 Configuring KVM PASSING BOOT PARAMETERS for NUMA-awareness

Configuring KVM PASSING BOOT PARAMETERS for NUMA-awareness can be done with numactl. The set of options needed can be handled by the ACTiManager. It includes pinning of cores, ethernet adapters, storage and memory.

Here is a working example:

```
numactl --cpunodebind=0-11 qemu-system-x86_64 -nodefaults \
-enable-kvm -cpu host -machine q35 -nographic -serial mon:stdio \
-m 256G -smp sockets=16,threads=2,cores=4 \
-net nic,model=virtio \
-drive file=/tmp/bionic-server-cloudimg-amd64.img,if=virtio -boot c \
-drive file=/dev/sdb,format=raw,if=virtio \
-drive file=/dev/sdc,format=raw,if=virtio \
-drive file=/dev/sdd,format=raw,if=virtio \
-drive file=/dev/sde,format=raw,if=virtio \
-balloon virtio-numa node,mem=16G,nodeid=0,cpus=0-7 \
-numa node,mem=16G,nodeid=1,cpus=8-15 \
-numa node,mem=16G,nodeid=2,cpus=16-23 \
-numa node,mem=16G,nodeid=3,cpus=24-31 \
-numa node,mem=16G,nodeid=4,cpus=32-39 \
-numa node,mem=16G,nodeid=5,cpus=40-47 \
-numa node,mem=16G,nodeid=6,cpus=48-55 \
-numa node,mem=16G,nodeid=7,cpus=56-63 \
-numa node,mem=16G,nodeid=8,cpus=64-71 \
-numa node,mem=16G,nodeid=9,cpus=72-79 \
-numa node,mem=16G,nodeid=10,cpus=80-87 \
-numa node,mem=16G,nodeid=11,cpus=88-95 \
-numa node,mem=16G,nodeid=12,cpus=96-103 \
-numa node,mem=16G,nodeid=13,cpus=104-111 \
-numa node,mem=16G,nodeid=14,cpus=112-119 \
-numa node,mem=16G,nodeid=15,cpus=120-127
```

9.4 Extending ACTiManager.Internal to expose NUMA-awareness to Guest VMs

The internal component of ACTiManager takes into consideration the NUMA topology when placing VMs. More specifically, ACTiManager.internal pins VMs on a specific NUMA nodes taking into account the characteristics of the VMs and the NUMA topology to mitigate interference and enhance locality. We extend ACTiManager.internal to use the configuration feature provided above, so that the NUMA topology is pushed to the guest VM during VM instantiation. The test has been done and exposition of the NUMA topology to the guest VM has been performed successfully.

10 Interference-aware applications: MonetDB & ACTiManager

For certain applications, measuring performance indicators such as IPC (Instructions Per Cycle) or MPKI (Cache Misses Per Kilo Instruction) or other hardware metrics such as CPU and RAM utilization might not be enough to catch all performance issues. In these cases, it might be preferable to use application-specific metrics. For example, a database engine can have its own performance analyzer to decide if a query is running slow. As ACTiManager is oblivious to what applications run inside the VM that it manages, there is no out-of-the-box way for ACTiManager to be aware of such application-specific metrics.

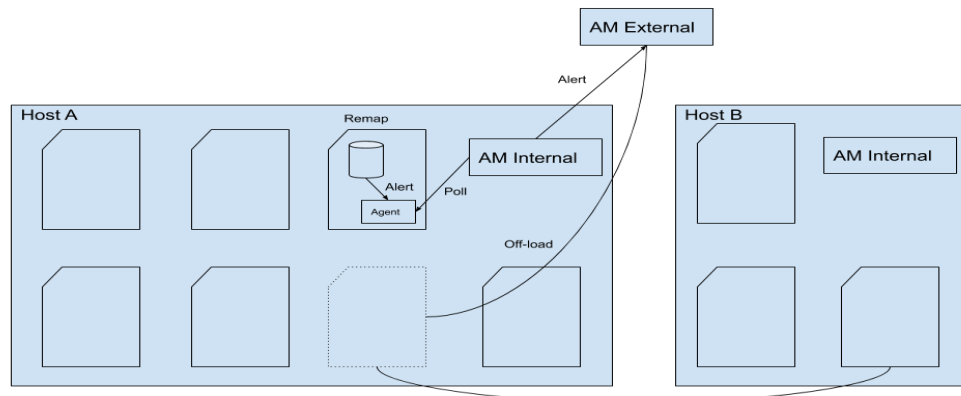


Figure 10.1: Interaction between PerformanceAlerter and other components.

To address this, an optional component, the PerformanceAlerter (see Figure 10.1) can be hosted in the VM together with the application. The PerformanceAlerter was described in D2.4 "Distributed Cloud Resource Manager v2.0". The application is then modified to call the PerformanceAlerter using a REST interface to notify ACTiManager that it experiences performance issues. The notification is a simple flag, set to either 0 or 1.

When a new VM arrives, ACTiManager internal will try to check if the PerformanceAlerter is enable in the VM. If it is enabled ACTiManager internal will continuously monitor the value sent by the PerformanceAlerter and act upon it. The code example below shows one way to poll the Performance Alerter on a given domain (VM).

```
def pollVm(self, vm):
    status = -1
    ip = Driver.getDomainIP(vm.domain)

    if ip != "-1":
        try:
            response = requests.get("http://" + str(ip) + ":5000/performancealert/1")
            print "Response from perfalert: " + str(response)
            if response.status_code == 200:
                json_response = response.json()
                print json_response
                status = str.split(str(json_response), ':')[1]
            else:
                status = -1
        except:
            status = -1

    print "Perfalert returns status: " + str(status) + " [0 = no issues, 1 = performance issues -1 = perfagent not responding]"
    return status
```

As a proof-of-concept, we have implemented a performance monitoring tool for the MonetDB database application (note that Neo4J could be similarly extended to use the PerformanceAlerter and notify ACTiManager for low performance due to interference). The tool is available as “perf_monitor.py” script from <https://github.com/acticloud/tpch-scripts>.

The script perf_monitor.py can be used to monitor the query performance over a relatively long running period. First, it executes each query in the benchmark N times and computes the average time of N-1 fastest executions. This average is regarded as the baseline performance of this query. Then, the script repeatedly executes the whole TPC-H benchmark query set for the time period as given by the option --duration. During each iteration, the queries are first randomly reordered for their executions.

After each query execution, the script compares this execution time against the baseline performance of this query to see if its performance has decreased. The performance deviation percentage is computed as:

```
devpercent = (current_exec_time - baseline_exec_time) / baseline_exec_time
```

If devpercent is larger than the threshold given by the option --degradation_threshold, then we have detected a “performance degradation”. Otherwise, we have a “performance normality”:

- If the performance status is normality, which is also the initial status, and the number of performance degradations has reached the number given by the option --patience (i.e., the patience level), then the performance status will be changed into degradation. The script also notifies the PerformanceAlerter about this change by using the REST API of the PerformanceAlerter to set the status to 1.
- If the performance status is degradation and the number of performance normalities has reached the patience level, the performance status will be changed into normality. The script also notifies the PerformanceAlerter about this change by setting its status to 0.

In Appendix C, we show a complete example of how to use the performance monitoring script to monitor TPC-H query executions.

11 Big data analytics integrations in ACTiCLOUD

Big data challenges exist at all levels in terms of data volume, from very large databases, such as astronomical databases, to very small databases, such as some industrial databases. To best serve (i.e., in terms of both performance and costs) a broad spectrum of business analytical applications that have varying or even contrasting demands, different hardware and software configurations are needed. In ACTiCLOUD, the two hardware systems that have highly divergent properties and focuses not only allow us to select the most suitable hardware and software stack to process different analytical workloads, but also enable both directions into which a database application²⁵ can scale, i.e., horizontally or vertically.

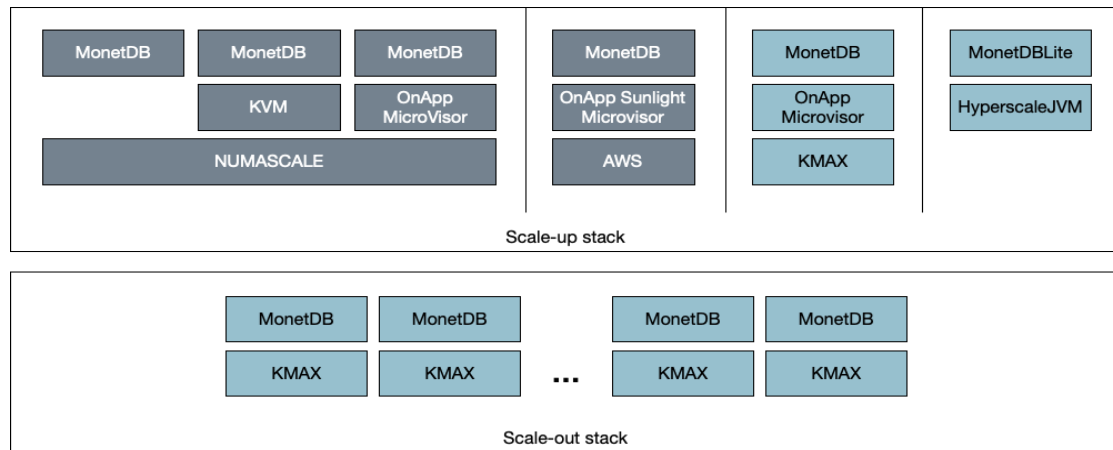


Figure 11.1: Integration stacks for scale-out and scale-up database applications.

Figure 11.1 gives an overview of the integration stacks we have realised and evaluated in ACTiCLOUD. On the vertical axis, we have the Numascale and AWS stacks to scale up to handle very large databases²⁶, while with the KMAX and HyperscaleJVM stacks, we can scale down to micro-levels (e.g., TPC-H scale factor 0.01, which is ~10MB). On the horizontal axis, the KMAX stack is particularly suitable for scale-out, with which we have run distributed MonetDB using the Air Traffic benchmark. Most of the evaluation results of these integrated stacks have been reported in D4.3 “ACTiCLOUD Intermediate Evaluation”. In the next deliverable, D4.5 “ACTiCLOUD Final Evaluation”, we will present new evaluation results, such as using the AWS stack to handle large data set, and comparing MonetDB and MonetDBLite against other popular open-source (embedded) database systems for micro-scale data sets on KMAX.

The evolution of MonetDB toward an ACTiCLOUD-aware database in the cloud is specified in task T3.3 “In-memory databases”. So far, all milestones have been archived. An overview is given in Appendix E. Here we highlight the integration related achievements:

1. MonetDB has been ported to and evaluated on both the project’s physical hardware systems provided by partners NSCALE and KALEAO, as well as in the virtual machines with KVM and OnApp's MicroVisor.
2. Distributed query processing features of MonetDB has been ported to and evaluated on KMAX.

²⁵ In this section, a “database application” should be read as a “relational database application”, unless otherwise stated.

²⁶ We have conducted the TPC-H benchmark using both stacks up to scale factor 1000 (i.e. ~1.1 TB). However, these stacks are not limited to this data size. We did not scale up further due to practical reasons, e.g. the price of AWS to run TPC-H SF1000.

3. An embedded version of MonetDB for Java, i.e., MonetDBLite-Java, has been implemented. It provides a native mapping of data types between the database and Java, and allows Java application developers to run MonetDB servers inside a JVM process and manage the database servers directly in their Java code.
 - a. MonetDBLite-Java has been ported to and evaluated on HyperscaleJVM, developed by UNIMAN.
 - b. MonetDBLite-Java has been ported to the Aarch64 architecture. Its evaluation on KMAX will be reported in the following deliverable D4.5 “ACTiCLOUD Final Evaluation”.

In addition to task T3.3, being ACTiCLOUD-enabled also means that MonetDB has become aware of and adaptive to its resource consumption. MonetDB have designed and developed MALCOM, a query memory footprint estimator for MonetDB to monitor resource utilization. Given a query, without executing it, MALCOM can give an estimation of the amount of memory MonetDB will need to achieve optimal performance²⁷. For the estimation, MALCOM keeps a dictionary of the actual memory consumption of MonetDB’s relational operators parameterised by different properties of their input data. This information can be produced by a MonetDB server as part of its profiling events for a tiny cost. MALCOM was reported in deliverable D3.3 “MonetDB ACTiCLOUD extensions v1.0”, and published in the XtremeCLOUD2018 workshop²⁸.

With respect to being flexible with resource consumption, we have implemented MDBconductor, which was reported in deliverable D3.7 “MonetDB ACTiCLOUD extensions v2.0”. MDBconductor conducts a fleet of MonetDB server (VM) instances. It can use MALCOM’s estimation to redirect an incoming query to the MonetDB instance with the most suitable (virtual) hardware resources. If none of the existing instances is suitable, MDBconductor will resize an existing instance or provision a new instance²⁹ to handle the query³⁰.

11.1 SQLPEL: a general benchmark platform

The integration and evaluation tasks in ACTiCLOUD requires considerable effort to keep track of a sizeable amount of different experiment settings³¹, so that we can make meaningful comparisons of the results among different database systems or different versions of the same database system. Inspired by this, we have designed and implemented (a prototype of) SQLPEL³², a general-purpose performance comparison platform for heterogeneous database systems, to help us keep track of comprehensive experiments settings and results. In this way, it would be much easier to monitor the progress of integrations, and compare the effects of different environmental settings and software versions.

²⁷ NB, with less memory, MonetDB might still be able to execute the query, but the execution time will most probably be (much) longer.

²⁸ Kersten, M.L., Zhang, Y., Katsogridakis, P., Koutsourakis, P., & Ruth, J.V. (2018). Database Resource Allocation Based on Resilient Intermediates. 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 314-319

²⁹ Without MALCOM, MDBconductor would still be able to operate. It can use a simpler strategy to redirect queries, e.g. round-robin or based on the load of the instances. However, MDBconductor will not be able to make informed decisions about when and how much to resize an instance or to provision a new instance.

³⁰ The description here is extremely simplified. In reality, MDBconductor would not carry out such heavy operations for a single query.

³¹ We used ~10 RDBMS in total with different combinations of benchmark data sets, hardware platforms, virtual machine technologies, and other hardware/software configurations.

³² The platform was originally published under the name SQLscalpel. Its name has been officially changed into SQLPEL, when we established the non-profit foundation “Stichting SQLPEL” in Feb. 2019 for its further development and take-up in the market.

The design of SQLPEL is based on various integration scenarios in ACTiCLOUD. The main goal of SQLPEL is to realise a Github-like platform in the cloud to share results of different benchmarks on different database systems. On this platform users can document detailed set-up of their experiments, upload and analyse experiment results, share the results and findings with other users, and share comments with each other. To enable users to exchange their benchmark results and make meaningful comparisons between different experiments, a considerable amount of environmental information needs to be stored together with the benchmark results. For instance, hardware or cloud instance configurations, OS version and other environmental configurations, DBMS versions, compilation parameters and their start-up time/runtime options, and benchmark configurations (e.g., data sizes, level of concurrencies, rate and size of updates, and number of runs of benchmark queries).

SQLPEL already takes into account an extensive set of basic hardware, software, DBMS and benchmark configuration parameters. This set can be easily extended to allow documenting more information. The screenshots below show the main features of SQLPEL.

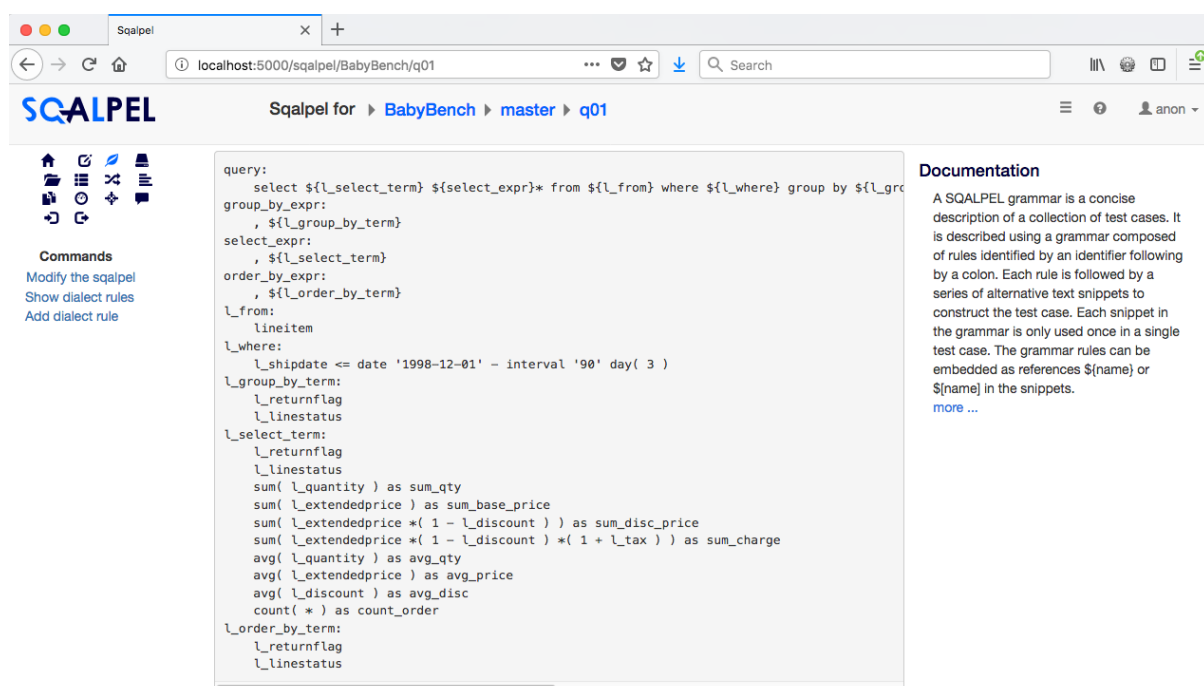


Figure 11.2: SQLPEL grammar for the TPC-H query 1.

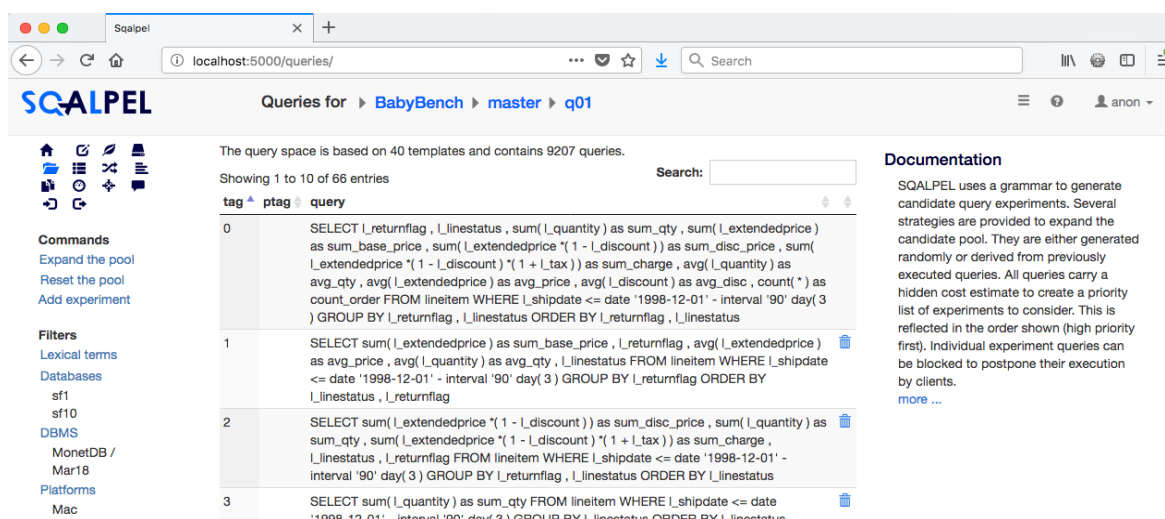


Figure 11.3: SQUALPEL's query pool.

Figure 11.2 shows the grammar which SQUALPEL has extracted from the TPC-H query 1. Using this grammar SQUALPEL can generate a large amount of alternative queries by including different sets of statements from the grammar and filling in different values for the literals. In this way, SQUALPEL can go far beyond the small set of queries that database benchmarks typically include. For instance, for the TPC-H queries 7 and 19, SQUALPEL can generate more than 100K query templates. The queries generated by SQUALPEL can be stored in a pool for later benchmarking use, as shown in Figure 11.3.

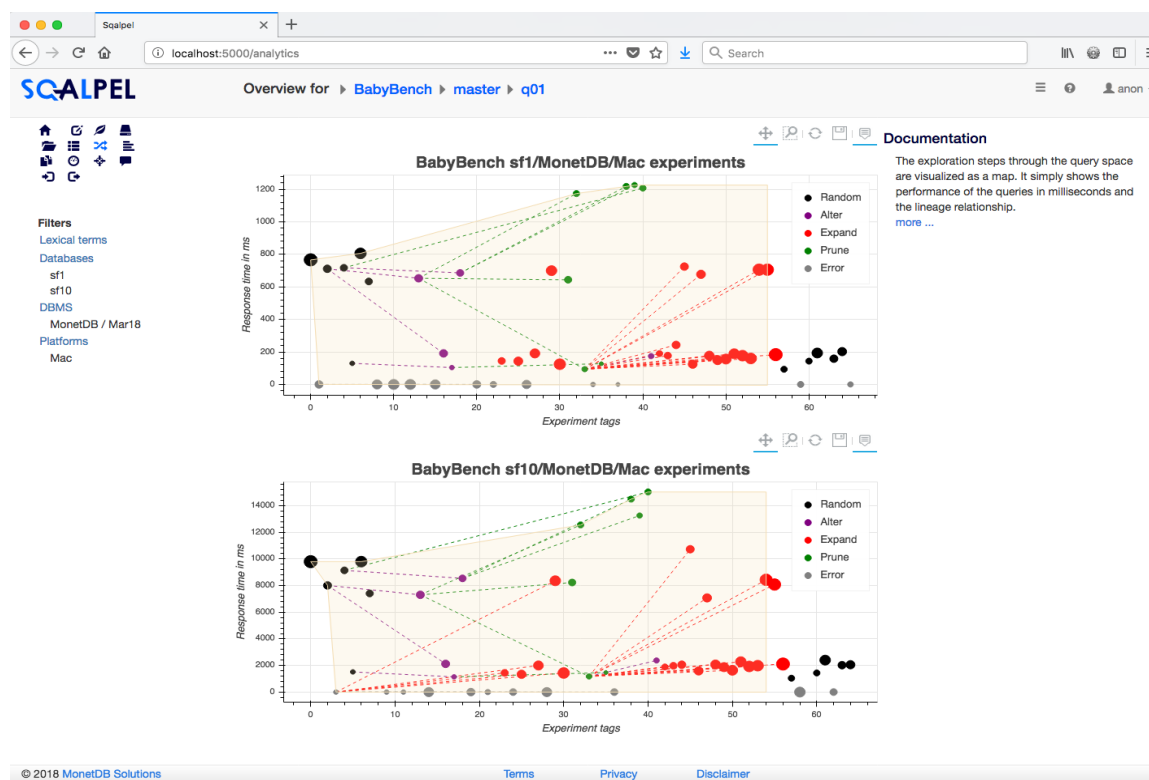


Figure 11.4: Experiments history.

From the query pool, a user can select queries to run on a machine of their own choice (including hardware, software and data set choices), i.e. SQUALPEL only provides the queries but leaves its users to define and conduct the benchmarking. After an experiment has been conducted, a user

can decide to upload the experiment results to SQLPEL, where they can be analysed with a few built-in visual analytics commands or exported in CSV for post-processing. Figure 11.4 shows the execution time of queries in a single experiment. The dashed lines illustrate the morphing action taken. The color coding for {alter, expand, prune} morphing is {purple, green, blue}. Queries that result in an error are shown as gray dots³³. The node size illustrates the number of components in the query. Hovering over a node shows the details of the run.



Figure 11.5: Principle components.

A principle component analysis of the lexical terms in the queries may indicate costly ones, as shown in Figure 11.5. From this graph, we can see that the dominant term in Q1 for a MonetDB installation is (see the green bar at the right side of the graph):

```
sum(1_extendedprice*(1 - 1_discount)*(1 + 1_tax)) as sum_charge
```

It is by far the most expensive component in terms of execution time. The underlying reason stems from the way MonetDB evaluates such expressions, which include type casts to guard against overflow and creation of fully materialized intermediates.

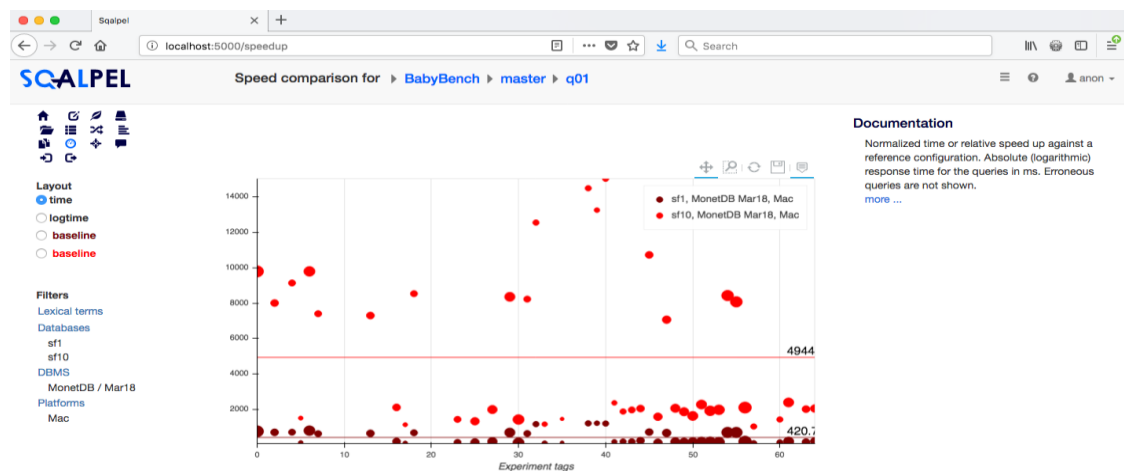


Figure 11.6: Query speedup.

Relative speedup between different versions of a system can be directly visualized, as shown in Figure 11.6. It should not come as a surprise that speedup factors of an individual query only tells

³³ This is normal, because SQLPEL can arbitrarily select rules from a query grammar to generate queries. There is no guarantee that a generated query contains valid syntax, nor can it be processed by different database systems. SQLPEL does not disqualify a query resulting in an error, because from that query SQLPEL can add or remove SQL statements to generate new queries, which can become valid queries later on.

part of the story. The figure shows that the base line query SF-1 Q1 runs about a factor 8 slower on a 10 times larger database instance. However, looking at the query variations it actually shows a spread of a factor 8-14. The outliers are of particular interest to dig into. The next step would be to determine their differences. For this we use a differential page, as shown in Figure 11.7. It highlights the differences in query formulation and gives an overview of the performance on various systems. This provides valuable insights to focus experimentation and engineering.

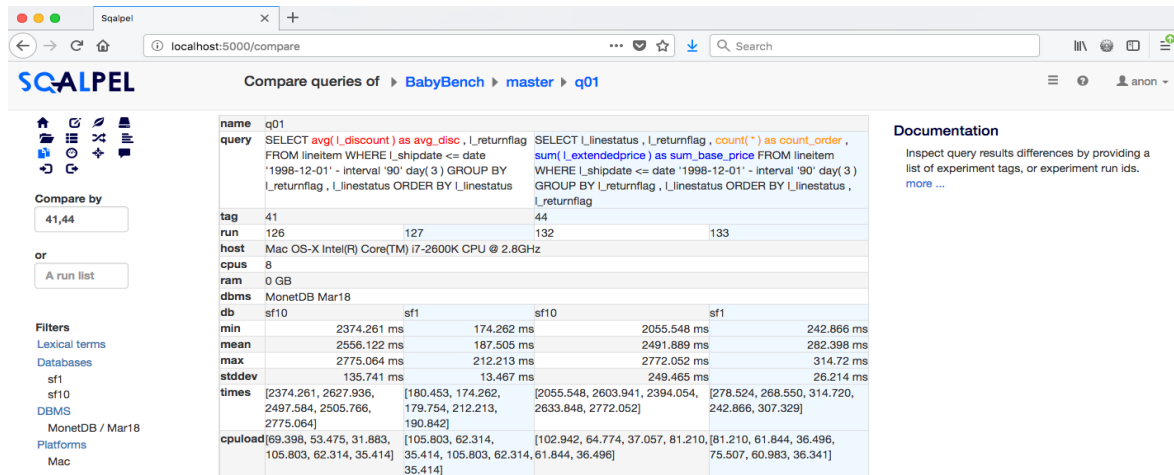


Figure 11.7: Query differentials.

SQUALPEL has been described in more detail in the following papers: DBtest2018³⁴, HiPEAC Info55³⁵ and CIDR2019³⁶.

11.2 Summary

During the ACTiCLOUD project, we have realised various integration stacks to support big data analytical applications from micro-scale to macro-scale, from a single DBMS server to distributed DBMS clusters. Therefore we have extended the RDBMS MonetDB with features to be ACTiCLOUD-architecture aware. In addition, we have developed a cloud platform SQUALPEL to ease the comparison of complex experiment results of benchmarking heterogeneous databases.

Thanks to ACTiCLOUD, MonetDB has taken its first step towards the cloud. We now can run MonetDB in (i) the microvisors of ACTiCLOUD, (ii) private cloud (e.g. Scilens) and (iii) public cloud (e.g., AWS). The evaluation results so far hint towards further improvements to enlarge its economic impact, such as the need for ‘swarm-based’ database processing. Swarm-based database processing recognises the need in modern cloud-based application where the database sizes are relatively small, but application providers are faced with 10s of thousands of such instances on beefy machines. This is particularly relevant for future business innovations on KMAX. A second opportunity is to realise a “resource elevator” which allows (database) applications to scale up and down the vertical hardware resources (in the cloud) automatically, flexibly and gracefully. This is particularly relevant for future business innovations on NSCALE and Onapp.

³⁴ M. Kersten, P. Koutsourakis, and Y. Zhang. “Finding pitfalls in query performance”. In proceedings of the 7th International Workshop on Testing Database Systems (DBTEST), colocated with 2018 ACM SIGMOD/PODS, Houston, TX, USA, Jun 10-15, 2018.

³⁵ M. Kersten and Y. Zhang. “Discriminative performance evaluation: separate the bold and the beautiful”. HiPEAC Info. no. 55, October 2018. <https://www.hipeac.net/magazine/7149/>

³⁶ M. Kersten, S. Manegold, Y. Zhang and P. Koutsourakis. “SQUALPEL: A database performance platform”. In proceedings of the 9th biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA, Jan. 13-16, 2019.

12 Conclusions and future work

This deliverable described the technical details regarding the integration of the final version 2.0 of the components of the ACTiCLOUD architecture, i.e., the MicroVisor, the OpenStack cloud framework, the ACTiManager, the HyperscaleJVM, the System Libraries, and the MonetDB and Neo4j databases, on top of the provided hardware platforms, KMAX and Numascale.

To maximise further the impact of the core components and the validity of the integration scenarios, and to bypass limitations and challenges that arose during the course of the project, we also included in our integration scenarios commodity hypervisors (KVM and Xen) and hardware platforms (Intel and AWS bare metal servers). In Deliverable D4.5 “ACTiCLOUD Final Evaluation” we will proceed with the evaluation of the various integration scenarios of the final prototype, putting together all components of the architecture.

Finally, because the ACTiManager architecture spans all levels of the computing stack, there are several interesting avenues for future work regarding integration. For example, the integration of ACTiManager with other technologies regarding resource management, e.g., Kubernetes, or the employment of a different virtualization technique, e.g., containers, would be very interesting scenarios.

Appendix A: Patches for Integrating ACTiManager on KMAX

QEMU EFI Firmware on Exynos

The patch for the EDK2 follows next:

```
diff --git a/ArmPkg/Library/ArmGenericTimerVirtCounterLib/ArmGenericTimerVirtCounterLib.c b/ArmPkg/Library/ArmGenericTimerVirtCounterLib/ArmGenericTimerVirtCounterLib.c
index 4bb1e1cde5..4f04f65a92 100644
--- a/ArmPkg/Library/ArmGenericTimerVirtCounterLib/ArmGenericTimerVirtCounterLib.c
+++ b/ArmPkg/Library/ArmGenericTimerVirtCounterLib/ArmGenericTimerVirtCounterLib.c
@@ -65,7 +65,8 @@ ArmGenericTimerGetTimerFreq (
    VOID
)
{
- return ArmReadCntFrq ();
+ return 24000000;
}

diff --git a/ArmVirtPkg/ArmVirtQemu.dsc b/ArmVirtPkg/ArmVirtQemu.dsc
index 885c6b14b8..0ded043e4c 100644
--- a/ArmVirtPkg/ArmVirtQemu.dsc
+++ b/ArmVirtPkg/ArmVirtQemu.dsc
@@ -121,7 +121,7 @@
#
# ARM Virtual Architectural Timer -- fetch frequency from QEMU (TCG) or KVM
#
- gArmTokenSpaceGuid.PcdArmArchTimerFreqInHz|0
+ gArmTokenSpaceGuid.PcdArmArchTimerFreqInHz|24000000

!if $(HTTP_BOOT_ENABLE) == TRUE
    gEfiNetworkPkgTokenSpaceGuid.PcdAllowHttpConnections|TRUE
```

The command for building the new QEMU-EFI firmware follows next:

```
# build -b RELEASE -a AARCH64 -t GCC49 -p ArmVirtPkg/ArmVirtQemu.dsc
```

QEMU EFI Firmware on Exynos

QEMU needs to be provided with the EFI firmware and a (64M) RW NVRAM image, in order to store EFI variables through the following command extension:

```
-pflash /usr/share/AAVMF/AAVMF_CODE.fd -pflash flash1.img
```

Custom QEMU binary and QEMU-EFI firmware

The patched QEMU binary is placed under /usr/local/bin and we also modified the distro-provided /usr/bin/kvm.

```
#!/bin/sh
```

```
exec taskset -c 4-7 /usr/local/bin/qemu-system-aarch64 -enable-kvm "$@"
```

VM Networking

We created the macvtap interfaces with specific MAC addresses and pass them to the QEMU., through the following commands:

```
# ip link add link eth0 name macvtap0 type macvtap mode bridge
# ip link set macvtap0 up
# ip link set macvtap0 address <mac address>
# <qemu command line> -net nic,model=virtio,macaddr=$(cat
/sys/class/net/macvtap0/address) -net tap,fd=3 3<>/dev/tap$(cat
/sys/class/net/macvtap0/ifindex)
```

To spawn QEMU VMs through QEMU's SLIRP, the following command should be used:

```
# <qemu command line> -net nic,model=virtio -net user
```

OpenStack modifications

In /etc/nova/nova.conf, we had to apply the following changes:

```
[libvirt]
# probably necessary due to the big.LITTLE arch
cpu_mode = host-passthrough
# necessary to avoid disk corruption during migration, also sane default
disk_cachemodes = file=none

# optional, websockets serial console for debugging purposes
# https://docs.openstack.org/nova/pike/contributor/testing/serial-console.html
# https://github.com/larsks/novaconsole
[serial_console]
enabled = true
base_url = ws://controller:6083/
proxycient_address = $my_ip
serialproxy_host = 0.0.0.0
serialproxy_port = 6083
```

When creating new images, we use the following image properties:

```
hw_disk_bus='scsi',      hw_firmware_type='uefi',      hw_scsi_model='virtio-scsi',  
os_command_line='console=ttyAMA0'
```

After setting up an L2 provider network with Openstack, with a macvtap linux agent on the compute nodes, we can run:

```
openstack port create --network provider --mac-address "14:5e:45:7f:a4:34" --  
enable compute-1-vm-1 (--host <host-id>)
```

With the above command, the interface works with the macvtap driver, but the neutron DHCP agent does not seem to work properly, so the VM fails to get automatically an IP address at boot time. In addition, this will not work probably during migrations. We are currently investigating whether it is possible to switch ports while migrating an instance. Otherwise, we will probably need to patch the neutron macvtap-agent.

Appendix B: Patches for Integrating ACTiManager with MicroVisor

ACTiManager.External

MicroVisor is integrated with OpenStack through a DevStack setup. Hence, we apply the following commands to integrate the aforementioned filters:

1. Copy filters/* in /opt/stack/nova/nova/scheduler/filters/
2. Copy weights/* in /opt/stack/nova/nova/scheduler/weights/
3. Restart nova-scheduler service

ACTiManager.internal

MicroVisor allows communication with the local instances through an API. In ACTiManager.internal we use the command that allows the pinning of a VM on specific cores through:

```
mvctl --mvmac <microvisor mac> --vcpu-pin -d <domain_id> -v <vcpu> -c <cpu>
```

Appendix C: Running MonetDB using more than 1 TB RAM

56 cores

```
time ./horizontal_run.sh -d SF-1000 -n 4  
SF-1000,default,01,150.356293678,164.948350191,158.469012677  
SF-1000,default,02,7.17917966843,8.50980162621,7.99676728248  
SF-1000,default,03,27.3114600182,27.9863877296,27.6133950353  
SF-1000,default,04,22.8039119244,23.5008728504,23.135048151  
SF-1000,default,05,25.2477800846,33.6248006821,29.6500020622  
SF-1000,default,06,8.18760442734,8.64464378357,8.35460340978  
SF-1000,default,07,54.7577672005,61.7250425816,58.280015111  
SF-1000,default,08,60.7236204147,98.7867298126,74.0918728115  
SF-1000,default,09,39.6817510128,40.9960584641,40.4720050693  
SF-1000,default,10,19.186416626,21.5116047859,20.1750450134
```

SF-1000,default,11,2.13810658455,2.44164156914,2.27002698183
SF-1000,default,12,12.1228473186,12.2815783024,12.1655828953
SF-1000,default,13,1114.22985005,1315.73805308,1234.4441067
SF-1000,default,14,3.30947780609,3.64907073975,3.50069785117
SF-1000,default,15,65.1560297012,128.367675543,91.7848629952
SF-1000,default,16,47.3144886494,53.2811272144,49.0660747885
SF-1000,default,17,38.0768649578,47.1424109936,40.7887011172
SF-1000,default,18,224.327656507,239.445295811,231.393054903
SF-1000,default,19,15.7938783169,16.9926931858,16.4369968176
SF-1000,default,20,22.1686193943,27.8222520351,24.6666442156
SF-1000,default,21,81.4250450134,98.9695763588,88.182879269
SF-1000,default,22,40.7284607887,42.5924534798,41.4845831395
-real 152m25.501s
user 0m6.294s
sys 0m3.084s

112 cores

time ./horizontal_run.sh -d SF-1000 -n 4
SF-1000,default,01,53.3499295712,58.2766027451,56.1487956643
SF-1000,default,02,11.8645589352,12.7590487003,12.3191530704
SF-1000,default,03,44.252903223,52.7223069668,47.2655479908
SF-1000,default,04,26.1406040192,35.332793951,29.1958357095
SF-1000,default,05,45.1970481873,49.3438258171,47.4099168777
SF-1000,default,06,7.57731342316,7.81257843971,7.7163513303
SF-1000,default,07,40.4852108955,44.8497469425,43.0476531385
SF-1000,default,08,83.529350996,109.950893402,94.636229217
SF-1000,default,09,60.5357170105,67.0784399509,64.2561222912
SF-1000,default,10,27.4188284874,40.255222559,32.0826572775
SF-1000,default,11,3.68794107437,4.24536204338,3.95804411173
SF-1000,default,12,12.8784985542,14.5902373791,13.3186339736
SF-1000,default,13,1181.60530782,1891.15279579,1580.47123337
SF-1000,default,14,3.83848905563,4.3910472393,4.04813313483
SF-1000,default,15,49.6398322582,69.8682394028,61.4811650515
SF-1000,default,16,44.0869758129,66.0569484234,57.5803402065
SF-1000,default,17,42.437584877,51.1799464226,47.2607491612
SF-1000,default,18,197.271804333,272.241749287,231.473254621
SF-1000,default,19,15.9244010448,17.7766597271,16.8926531076

```
SF-1000,default,20,38.884239912,46.7951574326,44.3666371703
SF-1000,default,21,88.3952288628,101.304565668,92.3624561428
SF-1000,default,22,47.905613184,51.660056591,49.5470964908
-real 175m55.426s
user 0m6.644s
sys 0m3.297s
```

224 cores

```
time ./horizontal_run.sh -d SF-1000 -n 4
SF-1000,default,01,77.1798136234,80.1342253685,78.7173765895
SF-1000,default,02,9.99750208855,14.2799870968,12.5585129857
SF-1000,default,03,40.1196074486,52.6199133396,46.4077450038
SF-1000,default,04,21.3140075207,27.4902842045,23.5185608268
SF-1000,default,05,45.8543379307,50.1613528728,48.8288126588
SF-1000,default,06,6.00046038628,6.25191116333,6.18215602637
SF-1000,default,07,36.4468808174,40.9099273682,38.3058379293
SF-1000,default,08,77.6787891388,221.294088602,146.230402589
SF-1000,default,09,45.5948028564,73.1688058376,64.5152151585
SF-1000,default,10,32.8596982956,41.0093486309,36.0061069727
SF-1000,default,11,3.38503289223,3.68599510193,3.5997609496
SF-1000,default,12,12.9956283569,14.0543148518,13.5695219636
SF-1000,default,13,821.135664225,1873.63925266,1326.1588493
SF-1000,default,14,3.56265950203,4.21064186096,3.79235845802
SF-1000,default,15,36.3397223949,100.910349369,60.3273343442
SF-1000,default,16,38.9541926384,59.1824002266,51.6649445295
SF-1000,default,17,56.0434737206,71.258793354,61.6466759445
SF-1000,default,18,210.748041391,245.649312258,234.312627554
SF-1000,default,19,22.0716838837,27.0760736465,24.2187087536
SF-1000,default,20,33.7574796677,47.8446519375,44.171692908
SF-1000,default,21,278.813402653,361.918828487,317.36125058
SF-1000,default,22,51.0559229851,53.5762217045,52.2041909695

-real 179m45.244s
user 0m6.566s
sys 0m3.300s
```

Appendix D: Example usage of MonetDB performance monitor script perf_monitor.py

We show here an example of how to use the MonetDB performance monitor script, perf_monitor.py, described in Section 10. To use this script, one needs to conduct the following three steps.

First, use the tpch_build.sh script (in the root directory of this repository) to generate a TPC-H dataset and load it into a MonetDB database:

```
./tpch_build.sh -s 1 -f /<path-to>/tpch
```

Second, start the just created database (this command can be copy-pasted from the final output of the tpch_build.sh script, or acquired later from this script using it -d option):

```
mserver5 --dbpath=/<path-to>/tpch/SF-1 --set monet_vault_key=/<path-to>/tpch/SF-1/.vaultkey
```

Finally, start the performance monitoring tool:

```
./perf_monitor.py -i 10 -p 5 -d 30 -t 0.5 -a 'http://127.0.0.1:5000/performancealert/1' SF-1
```

where the options mean:

- -i 10: execute the queries 10 times to obtain the baseline performance
- -p 5: collect 5 performance degradations before printing a warning
- -d 30: run the performance monitoring for 30 seconds (excluding the initiation time)
- -t 0.5: regard execution time increases of larger than 50% as performance degradations
- -a 'http://127.0.0.1:5000/performancealert/1': URL of the performance agent
- SF-1: name of the database

Note that perf_monitor.py assumes that a MonetDB server is serving the database SF-1 using default host and port number (i.e. localhost:50000). Also, the script assumes that the path to the MonetDB binary files has been properly added to \$PATH (otherwise, the script will not be able to run the queries using the MonetDB client tool mclient).

This script outputs the following information about the query performance:

```
dbname,seqno,query,exec_time,perf_dev,dev_pcnt,perf_stts
...
"SF-1",1,"q11",53.21,20.48,62.56%,0
...
"SF-1",6,"q04",252.20,136.79,118.53%,0
...
"SF-1",7,"q16",274.34,123.34,81.68%,0
...
"SF-1",7,"q03",143.90,19.37,15.56%,0
"SF-1",7,"q05",129.23,19.47,17.74%,0
"SF-1",7,"q07",218.61,13.90,6.79%,0
"SF-1",7,"q12",114.72,16.43,16.72%,0
"SF-1",7,"q19",141.55,46.74,49.29%,0
"SF-1",7,"q15",83.61,34.94,71.81%,0
"SF-1",7,"q14",76.92,34.29,80.45%,1
"SF-1",8,"q19",96.76,1.94,2.05%,1
"SF-1",8,"q21",690.01,49.60,7.75%,1
"SF-1",8,"q09",190.26,24.47,14.76%,1
```



```
"SF-1", 8, "q05", 125.95, 16.19, 14.75%, 1
"SF-1", 8, "q01", 930.22, 52.64, 6.00%, 0
"SF-1", 8, "q13", 254.95, 8.37, 3.40%, 0
...
```

where

- dbname is the name of the database
- seqno is the ID of this run
- query is the query ID
- exec_time is the execution time of this query
- perf_dev is the deviation of this execution time from its base execution time
- dev_pcnt is the percentage of the deviation of compared to its base execution time
- perf_stts is the performance status: 0 - normality, 1 - degradation

From this example output, we can see that perf_stts is set to 1 after there have been 5 executions whose dev_pcnt is larger than the degradation threshold 50% (i.e. $-t 0.5$), and that perf_stts is set back to 0 after there have been 5 executions whose dev_pcnt is less than 50%.

Appendix E: Overview archived milestones of evolving MonetDB

The evolution of MonetDB toward an ACTiCLOUD-aware database in the cloud is specified in task T3.3 “In-memory databases”. So far, all milestones have been archived:

1. MonetDB has been ported to and evaluated on both the project’s physical hardware systems provided by partners NSCALE and KALEAO, as well as in the virtual machines with KVM and OnApp's MicroVisor.
2. An in-memory-only option has been added into MonetDB to further reinforce MonetDB’s strength in supporting complex data-intensive statistical analysis applications. This option was first released in MonetDBLite in 2018. It has been ported back to MonetDB and is ready to be released in Q1 2020.
3. Distributed query processing features of MonetDB has been ported to and evaluated on KMAX.
4. MonetDB’s distributed database offerings has been extended to support predicates-based data partitioning (released in the Apr2019 version of MonetDB) and remote updates (further hardening continues after ACTiCLOUD). These features largely improve MonetDB’s usability for scale-out applications.
5. MonetDB’s replication service has been extended with a fast hot-snapshot feature, which helps increase the availability of the database. Hot-snapshot is ready to be released in Q1 2020.
6. An embedded version of MonetDB for Java, i.e., MonetDBLite-Java, has been implemented. It provides a native mapping of data types between the database and Java, and allows Java application developers to run MonetDB servers inside a JVM process and manage the database servers directly in their Java code.
 - a. MonetDBLite-Java has been ported to and evaluated on HyperscaleJVM, developed by partner UniMAN.
 - b. MonetDBLite-Java has been ported to the ARM64 architecture. Its evaluation on KMAX will be reported in the following deliverable D4.5 “ACTiCLOUD Final Evaluation”.
 - c. MonetDBLite-Java is under active maintenance. It was first released in 2018. Since then, its core library, MonetDBLite, has been rewritten, and is ready to be released in Q1 2020.