



## ACTiCLOUD: ACTivating resource efficiency and large databases in the CLOUD

Project No: 732366

H2020-ICT-2016-1

### D1.2: ACTiCLOUD Architecture

<b>Due date of deliverable:</b>	<b>M16 (2018/04/30)</b>
Actual submission date:	M16 (2018/04/30)

**Executive summary:** Deliverable D1.2 “ACTiCLOUD Architecture” reports the refined and final ACTiCLOUD architecture that comprises a set of core system components and the relevant APIs, based on the strategic objectives of the project that focus on improving cloud resource efficiency and enabling large, in-memory database applications on cloud offerings.

**List of authors:**

<b>Authors</b>	<b>Affiliation</b>
Georgios Goumas, Vasileios Karakostas, Konstantinos Nikas	ICCS
Atle Vesterkjær, Einar Rustad	NSCALE
Andrew Attwood	KALEAO
Michail Flouris	ONAPP
Christos Kotselidis, Foivos Zakkak	UNIMAN
Ying Zhang, Panagiotis Koutsourakis, Pedro Ferreira, Joeri van Ruth, Martin Kersten	MDBS
Jim Webber	NEO
Ewnetu Bayuh Lakew, Petter Svärd, Simon Kollberg	UMU

<b>Dissemination Level</b>	<b>X</b>	<b>PU (Public)</b>
		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:
<b>Nature</b>	<b>X</b>	<b>R (Report)</b>
		P (Prototype)
		D (Demonstrator)
		O (Other)

<b>Review Status</b>		Draft
		WP Leader accepted
		QA approved
	<b>X</b>	<b>Coordinator accepted</b>

**Revision History:**

Version	Author(s) (Affiliation)	Notes
0.1	Georgios Goumas, Vasileios Karakostas (ICCS)	TOC and initial content inserted
0.2	ALL	Content updated by all partners
0.3	Georgios Goumas, Vasileios Karakostas (ICCS)	Content updated
0.4	Michail Flouris (ONAPP), Ewnetu Bayuh Lakew (UMU)	Document reviewed
0.5	Georgios Goumas, Vasileios Karakostas (ICCS)	Comments and suggestions addressed
1.0	Georgios Goumas, Vasileios Karakostas (ICCS)	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

KALEAO

onapp



monetdb  
solutions

neo4j



**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**

<b>1</b>	<b>Introduction</b>	<b>11</b>
1.1	Purpose of this Document .....	11
1.2	Document Structure .....	11
<b>2</b>	<b>ACTiCLOUD objectives</b>	<b>12</b>
<b>3</b>	<b>CSP business scenarios and application use cases</b>	<b>13</b>
3.1	CSP business scenarios .....	13
3.2	Application use cases .....	18
3.2.1	Typical cloud applications.....	18
3.2.2	Database-centric applications.....	19
<b>4</b>	<b>ACTiCLOUD Architecture</b>	<b>26</b>
4.1	Overview .....	26
4.2	ACTiCLOUD::Hardware .....	27
4.2.1	Numascale platform .....	27
4.2.2	KMAX platform .....	28
4.3	ACTiCLOUD::Aggregation Layer .....	31
4.3.1	Numascale platform .....	32
4.3.2	KMAX platform .....	35
4.4	ACTiCLOUD::Hypervisor.....	36
4.5	ACTiCLOUD::CloudManager.....	40
4.6	ACTiCLOUD::ACTiManager .....	40
4.6.1	ACTiCLOUD::ACTiManager.internal .....	42
4.6.2	ACTiCLOUD::ACTiManager.external.....	43
4.6.3	ACTiManager - Cloud Manager roles and interaction .....	44
4.7	ACTiCLOUD::GuestOS::System Libraries.....	45
4.8	ACTiCLOUD::JVM.....	46
4.9	ACTiCLOUD::MonetDB .....	47
4.10	ACTiCLOUD::Neo4j .....	48
4.11	ACTiCLOUD::Application.....	48
<b>5</b>	<b>Embraced technologies</b>	<b>49</b>
<b>6</b>	<b>APIs</b>	<b>51</b>
6.1	Hardware API .....	51
6.2	MicroVisor API.....	51
6.3	OpenStack API .....	51
6.4	ACTiManager APIs .....	51

6.4.1	ACTiManager's subcomponents API .....	52
6.4.2	Cloud-native application resource management to ACTiManager API .....	52
6.4.3	Application demands to ACTiManager API .....	53
6.5	MonetDB to Guest OS API.....	53
6.6	Neo4j to JVM API.....	53
<b>7</b>	<b>Conclusions</b>	<b>54</b>
<b>8</b>	<b>References</b>	<b>55</b>

## Figures

Figure 4.1:	Base ACTiCLOUD architecture. ....	27
Figure 4.2:	Numascale Architecture.....	28
Figure 4.3:	KMAX platform.....	29
Figure 4.4:	KMAX blade.....	30
Figure 4.5:	KMAX compute node. ....	31
Figure 4.6:	KMAX Chassis layout. ....	31
Figure 4.7:	The NumaChip occupies one socket on the motherboard. ....	32
Figure 4.8:	The NumaConnect Card bridges the communication on the motherboard to other servers in a Numascale Shared Memory System.....	33
Figure 4.9:	NumaConnect cables connect the servers together. ....	33
Figure 4.10:	Unified Numascale system. ....	34
Figure 4.11:	The unified Numascale system allows the OS/Hypervisor to manage all memory as if it was a single server.....	34
Figure 4.12:	The KMAX compute node accelerates different scenarios and workloads through two FPGAs. ....	36
Figure 4.13:	Comparison of MicroVisor with the standard Xen architecture. ....	38
Figure 4.14:	The lightweight design of the MicroVisor architecture provides a distributed mode for utilizing resources on remote boards.....	39
Figure 4.15:	The MicroVisor forms a distributed storage platform using the storage resources of other network attached block devices. ....	39
Figure 4.16:	The “ACTiManager” follows a hierarchical approach, distinguishing the management of resources at node level and at site level. ....	41
Figure 4.17:	General architecture and operation of the “ACTiManager.internal” component that manages resources within a single node.....	42

Figure 4.18: General architecture and operation of the “ACTiManager.external” component that manages resources of a cloud site and across sites.....	44
Figure 4.19: Interaction between ACTiManager and OpenStack.....	45
Figure 4.20: Architecture subcomponents regarding the JVM.....	46
Figure 4.21: MonetDB subcomponents and their mutual relationships.....	47



**List of Abbreviation**

Abbreviation / Acronym	Meaning
ACID	Atomicity Consistency Isolation Durability
API	Application Programming Interface
CDR	Call Detail Records
CPU	Central processing unit
CSPs	Cloud Service Providers
DBaaS	Database as a Service
DPDK	Data Plane Development Kit
DVFS	Dynamic voltage and frequency scaling
ECC RAM	Error-correcting code Random-access memory
FPGA	Field-programmable Gate Array
FWaaS	Firewall as a Service
GC	Garbage Collection
HPC	High Performance Computing
IPC	Instructions per cycle
IPMI	Intelligent Platform Management Interface
IaaS	Infrastructure as a Service
JVM	Java Virtual Machine
KVM	Kernel-based Virtual Machine
LPAAE	Large Physical Address Extension
NAS	Network Attached Storage
NFV	Network Function Virtualization
NIC	Network Interface Card
NPU	Network Processing Unit
NUMA	Non-Uniform Memory Access
NVM	Non-Volatile Memory
OPEX	Operating Expenditure
OS	Operating System
OS	Operating Systems
PSoC	Programmable System-on-Chip
QoS	Quality of Service

RAID	Redundant Array of Independent Disks
SDN	Software Defined Network
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SO	Strategic Objective
SPU	Storage Processing Unit
SR-IOV	Single root IO virtualization
TCO	Total Cost of Ownership
TLB	Translation Lookaside Buffer
UFS	Unix File System
VDI	Virtual Desktop Instances
VM	Virtual Machine
vCPU	Virtual CPU

# 1 Introduction

ACTiCLOUD's vision is to develop a novel cloud architecture that will break the existing scale-up and share-nothing barriers, and enable the holistic management of physical resources both at the local cloud site and at the distributed levels. ACTiCLOUD targets drastically 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 D1.2 “ACTiCLOUD Architecture” is to describe the final ACTiCLOUD architecture together with the relevant APIs. The description is influenced by the use case scenarios and the requirements that were documented in the deliverable D1.1 “ACTiCLOUD requirements and base architecture”, and are briefly summarized in this document, too. This document is a key milestone for the implementation of the project, as it has served until now for the implementation of the Strawman prototype and will be serving until the end of the project as the main “handbook” for the implementation of the ACTiCLOUD components and their integration in the subsequent optimized prototypes.

## 1.2 Document Structure

The remainder of this document is structured in the following manner: Section 2 provides an overview of the ACTiCLOUD strategic objectives. Section 3 analyzes the end-user scenarios that ACTiCLOUD envisions to support. Section 4 describes the ACTiCLOUD architecture, broken down into components and subcomponents. Section 5 presents the embraced technologies, while Section 6 provides a high-level overview of the APIs. Finally, Section 7 concludes this deliverable.

## 2 ACTiCLOUD objectives

In this section we review the high-level objectives of ACTiCLOUD which serve as the starting point to define the ACTiCLOUD architecture. These strategic objectives are:

- **Effective utilization of cloud resources.** This strategic objective is sought by unchaining resource management and provisioning from the physical bounds of a single server and a single cloud site, together with the implementation of novel resource-aware allocation policies. This objective is further split into the following two explicit sub-objectives that are concurrently pursued:
  - **Resource efficiency:** Focusing on the requirements of Cloud Service Providers (CSPs) for reduced Total Cost of Ownership (TCO), ACTiCLOUD will drastically increase the resource efficiency of cloud infrastructures in terms of throughput per resource unit.
  - **Performance stability:** Focusing on the requirements of end-users, ACTiCLOUD will deliver performance stability to applications in terms of minimized performance variation compared to standalone execution.
- **Deployment of resource demanding applications in the cloud.** Leveraging the increased resource efficiency of the first strategic objective, ACTiCLOUD will engineer an increase in the applicability of cloud computing for resource demanding applications, having **a special focus on database applications** that heavily rely on in-memory data processing and, as such, cannot be effectively run in today's cloud environments. This strategic objective is further split into the following two sub-objectives:
  - **Scalability in resource provisioning:** Towards delivering the increased resource demands of the applications under consideration, ACTiCLOUD will significantly improve the scalability of resources that can be currently delivered in today's cloud systems.
  - **Elasticity in resource provisioning:** Responding to rapid changes in demand that are commonly found in cloud-based applications, ACTiCLOUD will address the ability of cloud systems to provision and deprovision huge amounts of resources in an elastic way.

### 3 CSP business scenarios and application use cases

This section provides an analysis of the end-user scenarios we envision to support during the course of the project. ACTiCLOUD targets two main end-user groups: (i) CSPs that operate or manage IaaS cloud data centers, and (ii) user applications that execute or could benefit from a cloud-enabled infrastructure.

In this section we conduct a top-down analysis from the perspective of CSPs and user applications to infer the requirements they pose on a next generation cloud resource manager like ACTiCLOUD. We divide this analysis into two segments: the first focuses on the **business scenarios** a CSP can benefit from, and the second focuses on **use cases** of user applications embracing both traditional cloud applications and next generation, database-driven ones. Particularly, we are interested in assessing the potential added business values of the technological advancements that are expected to result from ACTiCLOUD.

#### 3.1 CSP business scenarios

The de facto operating model of IaaS CSPs is to provision sufficient compute resources for their clients, i.e., cloud end-users. CSPs follow different approaches to resource management but typically each one will have a mechanism to carve out resources differently depending on the types of end-users that they may be catering for. Multiple cloud solutions exist including those from Google [GCP], Amazon [AWS], Microsoft [WAZ], and Rackspace [RAC]. Generally though, most of the cloud providers will not provide direct access to the underlying software for their cloud platform, but rather CSPs provide access to web-interfaces or cloud APIs that interact with the underlying platform, providing access to the Virtual Machines (VMs) or more generically “instances”. These virtual compute resources, i.e., the instances, can then be used by the end-users to run their workloads of choice. CSPs use their business strategies to determine how workloads should be matched with the underlying hardware resources. This usually leads to a level of overcommitment between virtual and physical resources. The predominant open-source software that is used to manage physical (hypervisors) and virtual (VM instances) cloud resources is OpenStack. Using OpenStack, the administrators of a cloud system will normally provide external access to an API and/or dashboard to expose or run certain operations once authorized (normally after payment). This will allow for the creation and destruction of virtual resources including VMs.

CSPs look to gain competitive advantage by exposing cloud interfaces that provide cutting edge functionality to the underlying cloud “stack” e.g., elastic services, DBaaS, FWaaS. However, in this environment CSPs must also be competitive on price not just functionality, which results in CSPs having to maximize the usage of the underlying hardware but with minimal impact on the performance of virtual machines and the expectations of customers.

The business scenarios detailed below have been specifically chosen as use cases that will demonstrate the benefits of ACTiCLOUD. There are many other use cases that can take advantage of ACTiCLOUD technologies, but the ones chosen here focus on the main architectural advantages that ACTiCLOUD will provide.

##### *Business scenario 1: Effective consolidation for increased revenue and reduced TCO*

**Scenario description:** This scenario is directly related to ACTiCLOUD’s strategic objective on effective utilization of cloud resources, and especially on resource efficiency. Under this scenario,

end-users specify the dimensions of the VMs that will be created within certain bounds that are typically prespecified by the CSP. Based on these dimensions, the resource manager selects a physical machine that can accommodate each VM in its entirety. This becomes a ‘bin-packing’ problem with workloads allocated based on feasible target allocations, and some strategy for determining where to place the workload if there are multiple possible candidates. These strategies are usually primitive, relying on round-robin, random or least-utilized allocations and disregard: (i) the detailed resource footprints (i.e. I/O, CPU, memory, cache intensity), (ii) the application profile (critical/non-critical, user-facing/batch), and (iii) the execution platform characteristics (e.g., CPU frequency and characteristics, memory hierarchy, system topology). This may cause resource fragmentation and/or contention caused by VM interference [DK13], ultimately leading to severe under-utilization of the platform.

Moreover, CSPs sell virtual shares of the underlying physical platforms to end-users. Certain types of workloads will be less intensive and less critical than others. With careful consolidation, there can be more physical resources left for running additional workloads. A different model of operation (similar to Amazon’s current practice) is based on auctioning off spot instances. These spot instances are best-effort virtual machines that are given very low priority by the cloud schedulers. When sufficient resources are left over to run a spot instance, it can be provisioned and then charged for.

ACTiCLOUD’s improved approach will identify the cases where the physical hardware resources are being underutilized and then perform effective consolidation. There are several actions that can be taken when workloads have been effectively consolidated, e.g., shutting down physical servers for energy saving or hosting additional workloads in the short term, or making the spare resources available in a cloud federation (see Business Scenario 4), or planning for a more conservative procurement of hardware infrastructure in the long term.

**Assumptions:**

- “Bursty” workloads are suitable candidates to be consolidated. These workloads, especially when not inter-dependent on other workloads running on the same platform, can be consolidated without having too much of a performance impact overall. Of course, as in several cases, these types of workloads may be latency critical. Therefore, it is the resource manager’s responsibility to maintain the QoS levels posed by the application. This is one of the major challenges that ACTiCLOUD needs to address.
- Other applications that fit well for consolidation and colocation include those that are mutually independent and use different resources at different times. For example, workloads that are memory intensive could be the preferred candidates for co-locating with compute intensive workloads as different types of resources will be utilized. In general, if all workloads are relatively large with respect to the underlying physical resources, a greater degree of resource fragmentation will occur. These workloads would also benefit from resource consolidation. Workloads that are insensitive to performance criteria are also good candidates for resource consolidation. Virtual desktop instances (VDIs) are a good potential candidate as they rely on user-interaction.
- Workloads that are stateless and operate on a problem set that can be paused and resumed are good candidates for benefiting from a mode of operation based on start/pause/stop/resume accompanied with an appropriate pricing model. For certain types of scientific workloads, this can be a good fit, especially if they need large amount of

memory resources. This type of workload could take advantage of the low-utilization periods [LCG14, LCG15, BCH13, DK14, RTG12] of large infrastructures supporting ACTiCLOUD to process workloads offline. If effective consolidation is carried out, any workload can be run on the ‘freed-up’ hardware.

- Stateless applications that run on a small data-set are best suited for rapid spawn-up instances that can be used when the rest of the platform has low utilization. Workloads that are non-critical (i.e., they do not pose hard latency requirements), or do not need to run online but could be used when other cloud applications are dormant or much quieter (e.g., outside of office hours) can benefit from effective consolidation.
- Overall, CSPs can benefit from many types of workload consolidation. However, applications in this business scenario should ideally not experience any performance impact, and if they do, this should be traded off with a mutually beneficial pricing model.

**Expected benefits:** Based on a combination of ACTiCLOUD resource aggregation and intelligent workload placement, CSPs will be able to utilize their platforms at a higher operational level. CSPs will also have the opportunity to run workloads for many end-users, with some having less stringent availability requirements than others. Workloads that have no need for being always on, would be ideal for utilizing idle cycles on physical machines. Over-provisioning is not a new feature, with many CSPs already benefiting from it. The difference with ACTiCLOUD is the intelligent monitoring system that matches workloads to the available physical resources. CSPs are more than aware that cloud operating expenditure (OPEX) far outweighs the infrastructure costs over the long run, and so will be highly supportive of any functionality that allows the resource utilization and hence the profits to be increased. Thus, effective consolidation of cloud resources will allow CSPs to (i) increase revenue by hosting additional users on the same infrastructure, (ii) reduce electricity costs, and (iii) plan for reduced capital expenses on hardware equipment.

### *Business scenario 2: Workload prioritization*

**Scenario description:** This scenario relates to ACTiCLOUD’s strategic objective on performance stability. This scenario considers that in a cloud environment there are usually many types of end-users who have different requirements for their workloads. Certain end-users will be willing to pay premiums to receive guarantees for certain performance levels (e.g., “gold” instances), whilst others will be willing to sacrifice performance for lower costs (e.g., “silver” instances). It is the role of the CSPs to match these requirements. If workloads are tagged with hints on how the performance should be managed, resources can be divided into shares of the physical resources. CSPs tend to offer premium packages that guarantee performance levels by reserving more resources, with the best performance requirements resulting in the end-user being offered a bare-metal, costly deployment. By offering fine-grained control on resource allocation, it is possible to map the workload requirements to specific physical resources through workload allocation heuristics.

#### **Assumptions:**

- Potentially, all applications could benefit from systems of prioritization with workloads requiring more resources or by taking precedence over workloads that are willing to sacrifice performance for cost savings. For prioritization to work, there must be a multi-tenant platform with multiple users, utilizing the platform at the same time. If certain

workloads require performance levels that cannot be provided at the same time as another workload, then this would mark the application as being a candidate for migration to another physical server.

- Applications should declare their priority class, and be well modeled and monitored in order to provide suitable prioritization. This requires flow of information from the user to the cloud management system. In addition low-level event monitoring should be available by means of an API to the orchestration platform. For the workload to benefit from decisions at the orchestration layer, the workload should be long running and have a sustained average throughput. Applications that need peak bursts can be supported through this business scenario with resources being allocated either a-priori, or better, through dynamic resource allocation on-the-fly to support the peak behavior. If done dynamically, it will help supporting the resource consolidation scenario.

**Expected benefits:** Most CSPs cater for a range of end-users, with some willing to sacrifice performance for price. By having different service levels, it gives options for the orchestration platform to react in a suitable time and provide the performance levels needed by the workloads. Prioritization and tiered service levels are always beneficial for CSPs, as they can tune price models and ultimately charge more for guaranteeing performance levels, if the platform supports it. Being able to support this dynamically requires platform support throughout the stack and is one of the main benefits of the holistic approach taken by ACTiCLOUD.

### *Business scenario 3: Hosting larger workloads*

**Scenario description:** This scenario closely relates to ACTiCLOUD's strategic objective on resource scalability. A business scenario that is not currently supported by existing platforms is that of having shared resources used to support workloads larger than that of a single server. Currently, Google, Amazon, etc. can only offer instances that will fit into one of their pre-baked instance types. Although large prices can be paid for special dedicated hardware to these Cloud vendors, this approach precludes all the benefits of cloud computing and hence its cost is excessively high.

Most applications that need large memory spaces, or have resource requirements in excess of that of a single machine have been re-engineered to be split into smaller, scalable components. For certain workloads, this type of splitting as popularized by methods such as map-reduce is suitable and can be deployed using current cloud solutions. For other workloads, such as large in-memory databases and others that require online processing of large datasets in-memory, this approach may be less than ideal and was one of the main motivations for ACTiCLOUD.

**Assumptions:** The following types of workloads are expected to be serviced by this scenario:

- Workloads that require TB's of data in-memory including large online databases.
- Workloads that would benefit from large amounts of memory resources, but cannot be applied to the cloud, because they do not currently fit into the prebaked size types provided by CSPs.
- Applications that require large amounts of resources, but cannot afford bare-metal deployments.
- Applications that currently do not fit in even the largest memory platforms available from hardware vendors.



- Typical examples of applications that fit these categories also include scientific simulations, data modeling, weather forecasting, and other large High Performance Computing (HPC) type workloads that are currently split into worker units using cluster technologies available today.

**Expected benefits:** This is a new area that ACTiCLOUD is tackling. There are similar software platforms, such as memcached [FIT04], that rely on sharing memory resources over the network but have large latency penalties due to the networking and processing overhead. It is unclear how regular customers will benefit from the much larger resource VMs once they become available. High-end CSPs whose customers are dealing with large datasets will benefit from the potential to offer those customers VMs capable of expanding beyond a single physical system, meeting the needs of customers growing memory demands. It is foreseen that these high-end customers would also be willing to pay a premium for such ACTiCLOUD enabled features, which will benefit the cloud software license providers, as well as the CSPs who will be able to charge premium prices for a disruptive, novel technology.

#### *Business scenario 4: Collaboration with sibling cloud sites*

**Scenario description:** This scenario assumes that a cloud site has established a collaboration or federation with a number of sibling cloud sites that are geographically distributed. This provides opportunities for more effective resource utilization and manageability. Regarding resource utilization, collaborations between sibling cloud sites can alleviate CSPs from overprovisioning their platforms, just to cope with seldom circumstances of peak workload traffic. In that case, excess traffic from one site can be directed to a collaborating one, to make enough space for the temporary high demands.

Cloud maintenance, on the other hand, is currently a difficult challenge for CSPs. Normally there are scripts or systems used to migrate VMs between machines during a planned maintenance window. Supporting mass migration in OpenStack is currently a complex task, with manual intervention usually required. It also relies on having spare equipment for migrating workloads away from the system targeted for maintenance. Storage needs to be replicated away from the targeted hardware, if there is not already a valid replica that is accessible from the standby machine. Maintenance and scheduled downtime may be carried out for several reasons including Cloud software upgrades, software security patches, or hardware upgrades/maintenance. Currently, the maximum number of VMs on a physical machine is in the order of 10-100s. Increasing the number of virtual machines per hypervisor also increases the impact of removing the hypervisor from the system, as each virtual machine and its associated resources must be relocated, e.g., firewalls, storage etc. CSPs are currently reducing the overhead of virtualization through the use of containers and unikernels. Combining this optimization with hypervisor memory increases (including the use of large shared memory aggregates as proposed in ACTiCLOUD) will see the number of virtual instances per hypervisor grow massively. Given the expected increase in scale of virtual instances, it will soon become unmanageable to evacuate virtual machines from large memory hypervisors using current methods.

#### **Assumptions:**

- Cloud sites have already established a collaboration. High-level administrative and security issues are orthogonal to the challenges covered by ACTiCLOUD and are not covered within this project.

- It is recognized that platform incompatibilities in hardware and software between the sibling sites can limit the applicability of this scenario. ACTiCLOUD operates under the assumption that the sibling cloud sites are compatible and can smoothly support migrations of workloads between them.
- This scenario deals with traffic peaks that are considered spatially local, i.e., one cloud site in the coalition suffers from the peak traffic, while there exists at least one other site that operates under normal traffic.
- All long-running applications will potentially be on hardware that is subject to a maintenance cycle.

**Expected benefits:** This scenario targets at decreasing resource waste that is created by the conservative provisioning policies of CSPs. Taking into consideration the collaboration with sibling sites, CSPs can plan for hardware resources that match better the typical and not the peak traffic of their sites, counting for the assistance of collaborating sites in the seldom cases of workload peaks or when maintenance is needed.

### *Business scenario 5: Enhanced dependability and availability*

**Scenario description:** Dependability is an objective that is not strategically addressed by ACTiCLOUD. However, this is a high challenge for CSPs, and ACTiCLOUD can potentially contribute to this direction as well. ACTiCLOUD will operate on a typical loop involving application and resource monitoring, analysis, decision making, and actuation with a focus on the optimization of resource allocation. Within this loop, anomalies related to hardware failures and security attacks can be detected and mitigated. ACTiCLOUD will not implement full functionality of this scenario, but some of its components could be utilized or easily extended to aid in this direction.

#### **Assumptions:**

- All applications are potentially subject to reliability or security issues. Aside from reliability issues due to software programming errors or software operating system bugs, which can be handled through upgrades, software can suffer from unreliable platforms.

**Expected benefits:** Enhanced dependability by being able to predict a number of hardware failures or security attacks. This can translate to timely maintenance and reallocation of resources without application interruption in the former case and mitigation of cyber-attacks in the latter case.

## **3.2 Application use cases**

### **3.2.1 Typical cloud applications**

Cloud installations typically host a variety of applications including web/mail/storage services, office and business applications, media servers, scientific simulations, social media applications, and many others. ACTiCLOUD has a special focus on database-centric applications that are analyzed more in depth in Section 3.2.2, where a number of use case scenarios are described towards building a Database-as-a-Service (DBaaS) cloud offering. In this section we discuss the properties and requirements of typical cloud applications.

To cope with the complexity of the ecosystem supporting cloud applications, we need to

emphasize the properties that are critical for a resource management system for an IaaS cloud. To this direction, we classify the generic cloud applications into two major classes, i.e., latency critical and batch applications, defined as follows:

- **Latency critical applications:** Applications of this class are typically user-facing, like web search or social media applications, which perform computation tasks only when there are user requests. As a result, their resource utilization footprint greatly varies over time and can be considered as unpredictable, because it is heavily dependent on human social behavior and interactions. Such applications are characterized by their long-running behavior and their sensitivity to the time it takes to respond to end-user requests. Indeed, they cannot tolerate interruption of their execution and delay in their service time. Interruptions of services and/or increased service response times can result in revenue reduction due to end-users service abandonment as far as opting for competing services, thus incurring long-term revenue loss. Therefore, it is desirable to maintain response time for a service below a desired value.
- **Batch applications:** Applications of this class are typically long-running workloads that do not heavily interact with human users. They perform offline analytical computations or scientific simulations, and their QoS is determined by their throughput (operations per second) or time to completion which lies in the order of several minutes, or even hours or days. As such, batch applications can tolerate interruption in the operation, as long as their average throughputs and deadlines are met.

#### *Use case 1: Execution of typical cloud applications*

This is the most generic and straightforward application use case for the ACTiCLOUD system. The requirement for ACTiCLOUD is to preserve the QoS levels of each hosted application as described in their respective Service Level Agreement (SLA). For latency-critical applications, QoS is typically expressed in latency time or requests per second. For batch applications, QoS is expressed in time to completion or throughput (operations per second).

#### **3.2.2 Database-centric applications**

ACTiCLOUD is heavily motivated by the high challenges to host database-centric applications in cloud infrastructures, especially when those applications are resource hungry. However, there are significant added values from hosting database-driven applications in the cloud, i.e.,

- **Flexibility, elasticity and scalability:** a new application instance can be up and running quickly. Instances are able to grow/shrink gracefully with the application's growth/decrease.
- **Cost reduction:** Both the reduction of human intervention and the pay-as-you-go pricing model will eventually lead to significantly lower costs.
- **Availability and reliability:** an application is always available when its users need it.

The DBaaS offerings in an ACTiCLOUD-enabled cloud data center shall facilitate the different types of database-centric applications that are analyzed in this section. In this analysis, we provide a generic use case description, detail the application properties, present their requirements, and discuss the benefits each application will enjoy by entering the cloud.

*Use case 2: Database applications with constant workload and intermittent uptime*

**Use case description:** Some Business Intelligence (BI) applications are used only during office hours, e.g., applications in education that monitor students' progress, and healthcare applications that process hospital claims. These use cases closely relate to ACTiCLOUD's strategic objective on resource efficiency, so that applications running in an ACTiCLOUD-enabled data center can enjoy advantageous prices.

**Application properties:**

- In principle, the applications are only used during predefined time frames, e.g., office hours.
- Applications' workloads, in terms of number of concurrent users and the amount of data, are fairly consistent and generally known a priori.
- Backups can be made outside applications' operational hours.
- Due to the privacy requirements of some applications (e.g., applications processing patient records or financial information), multi-instances are preferred over multi-tenant single-instance to serve many end users.

**Application requirements:**

- The applications should be up and running during their expected uptime.
- The applications should perform in accordant with some predefined SLA. For instance, the end-users should have the feeling that the applications are responsive to their actions (e.g., <1 sec).
- Failover should be in place relatively quickly, e.g., ~30 minutes.

**Cloud benefits:**

- Flexibility: With growing data, bigger instances can be launched easily. With growing number of users, more instances can be launched easily.
- Cost reduction: Pay-as-you-go for the needed resources only.
- Availability: Applications are available when needed.
- Reliability: Automatic backup during applications' down time. Quick restart or replacement of a failed instance.

*Use case 3: Database applications with constant workload and continuous uptime*

**Use case description:** This class of database-centric applications typically receives a fixed amount of data at regular time intervals. For instance, network performance monitoring systems continuously monitor the traffic in computer networks. Such applications are expected to be always up-and-running to process large amounts of data. In event of failures, their normal operations should be resumed as soon as possible. This use case closely relates to ACTiCLOUD's strategic objective on performance stability.

**Application properties:**

- The applications are constantly in use.
- Applications' workloads are fairly consistent in terms of number of concurrent users and the amount of data they process.

- The volume of data per application/database instance is generally in the order of 100GBs or larger.
- The number of concurrent users per application/database instance is relatively small.
- The number of instances needed (per client organization) can be large.

**Application requirements:**

- The applications must be up and running 24x7.
- The applications must be highly responsive to the users' tasks throughout their uptime.
- Failover should be in place quickly, e.g., within 30 seconds.
- The backup process shall not interfere with the application's normal operation.

**Cloud benefits:**

- Flexibility: With growing data size, larger instances can be launched easily. With growing number of users, more instances can be launched easily. In an ACTiCLOUD-enabled data center:
  - increased flexibility due to better hardware resource utilization across physical servers.
  - increased flexibility in dealing with geographically skewed workloads by using remote replicas to serve application users that are physically closer to the users. This not only facilitates performance reliability, but also leads to potential cost reduction.
- Availability: In a cloud environment, one can more easily workaround faulty hardware by (re)starting application instances on healthy hardware, which helps providing the 24x7 availability.
- Reliability: Automatic backup, quick restart or replacement of a failed instance. In an ACTiCLOUD-enabled data center:
  - increased reliability with backups on a remote site.
  - increased reliability due to better hardware resource utilization across physical servers.

***Use case 4: Database applications with predictable workload burst***

**Use case description:** Sometimes end-users need to conduct heavy computations in their analytical applications. For instance, from time to time, the data scientists of a big telecom company would want to run some heavy analytics on monthly/quarterly/yearly/etc. call detail records (CDR) of large customer groups. For such occasions, the data scientists would typically want to quickly set up and run several heavy instances of their analytical applications (i.e., with more hardware resources, particularly computational resources) for a relative short time period (e.g., hours, days). Such heavy instances can be either new application instances started on-demand, or expansions of existing normal application instances. This use case closely relates to both ACTiCLOUD strategic objectives on scalability and elasticity in resource provisioning respectively.

**Application properties:**

- The application instances are only used for a relative short time (e.g., hours, days).
- Large instances are needed to handle the heavy computational work.

- The number of concurrent users is small per instance.

**Application requirements:**

- An end user should be able to configure and start a heavy application instance easily and quickly; or an end user should be able to reconfigure and expand an existing application instance easily and quickly.
- Some start-up time for such a heavy instance is acceptable (e.g., <1 hour), since it might need to load a big amount of data (e.g., in the order of TBs).
- The application may not respond instantly to users' tasks, since it might be a long running query over a big amount of data.
- Failure of the application instance should be avoided during the lifetime of this instance, or at least during a long running query, because both re-computing the query and restarting the instance are time consuming and expensive.
- The application instance should be shut down or shrunk as soon as its computation jobs are finished.

**Cloud benefits:**

- Flexibility: In a cloud environment, one does not need to wait for the purchase of more hardware to support heavier computations.
  - In an ACTiCLOUD-enabled data center: cross server hardware can be used, even hardware from a remote site.
- Cost reduction: No need to purchase and maintain expensive hardware only for a short time. The high-end hardware is immediately available if-and-only-if needed.
  - In an ACTiCLOUD-enabled data center: one can even choose to allocate this kind of short-term large application instances on a remote site which can offer a lower price (e.g., because the remote site is located at a more affordable location, or the instance can be run during the quiet hours of that remote site).
- Availability: In a cloud environment, one can more easily workaround faulty hardware by starting application instances on healthy hardware. In an ACTiCLOUD-enabled data center:
  - hardware from different physical servers can be used for a single application instance. This makes it easier and quicker to allocate the necessary hardware for the large application instances.
  - an application instance can be even run on a remote site, if the local site does not have the necessary hardware immediately available. Again, this makes it easier and quicker to allocate the necessary hardware for the large application instances.

***Use case 5: Applications with unpredictable workload burst***

**Use case description:** This class of applications must be capable of reacting to sudden peaks in traffic that cannot be known in advance. For example, a breaking news event would instantly cause the majority of the users of a social media application to login and post messages. Such events can happen at any moment in time and cause a sudden requirement for additional resources beyond what was planned. When this happens, social media applications should stay operational delivering reasonable performance. Another good example is real-time seismic

information systems that can expect a huge workload peak after an earthquake. This use case closely relates to ACTiCLOUD's strategic objective on elasticity in resource provisioning.

**Application properties:**

- The workload burst is heavy but short (e.g., hours to days).
- It is unpredictable when a workload burst will happen.
- The application should be able to rapidly adapt to the new (much heavier) workload.

**Application requirements:**

- Usage/load information of all instances of an application must be constantly available, so that the application admins can detect the start and end of a workload burst.
- After a workload burst has been detected:
  - It should be possible to start a medium-sized new instance of the application within minutes.
  - It should be possible to expand a running application instance. The time to scale-up should be negligible, and it should not affect the performance of the running instance.
- After a workload burst has ended:
  - It should be possible to shut down some application instances as soon as possible (e.g., within minutes). All users of those instances should be transferred to the remaining instances transparently.
  - It should be possible to shrink some running application instances as soon as possible (e.g., within minutes), without affecting their user connections and their performance.
- Failover should be in place quickly, e.g., within 30 seconds.

**Cloud benefits:**

- Flexibility: In a cloud environment, one does not need to wait for the purchase of more hardware to support the heavier workload, which is undesirable in dealing with unpredicted workload burst.
  - In an ACTiCLOUD-enabled data center: cross server hardware, i.e., pools of resources from different physical servers, can be used.
- Cost reduction: No need to purchase and maintain expensive hardware only for a short time. The high-end hardware is immediately available if-and-only-if needed.
  - In an ACTiCLOUD-enabled data center: one can exploit the localities of remote sites to handle the sudden workload burst, e.g., choosing hardware resources from a remote site that can offer a lower price, or is geographically located closer to the workload burst.
- Availability: In a cloud environment, the underlying platform (e.g., the hardware resource manager or the VM manager) can more easily work around faulty hardware by starting application instances on healthy hardware. In an ACTiCLOUD-enabled data center:
  - hardware from different physical servers can be used for a single application instance. This makes it easier and quicker to allocate the necessary hardware to scale-out and/or scale-up the application instances.



- an application instance can even run on a remote site, if the local site does not have the necessary hardware immediately available. Again, this makes it easier and quicker to facilitate the necessary scale out and scale up.

#### *Use case 6: Analysis of social networks with high dynamism*

**Use case description:** A highly dynamic social network to provide opportunities for Page Rank, cluster detection, weak links etc. Such techniques are used within typical consumer web applications (e.g., social search and recommendations). They are also used within intelligence (e.g., monitoring/responding to terrorist cells) and as such have repercussions both for citizens' quality of life and their safety. This use case closely relates to both ACTiCLOUD strategic objectives on scalability and elasticity in resource provisioning respectively.

#### **Application properties:**

- The graphs are both large and dynamic.
- Writes occur at unpredictable times and volumes.
- Arbitrary ad-hoc queries and graph algorithms may be executed to gain insight into the data.
- In the first instance it will be assumed that analytical databases are single user for queries/algorithms.

#### **Application requirements:**

- Any queries issued by an end-user looking for insights should run as quickly as possible, even at the expense of overall utilization.
- Queries on the ACTiCLOUD platform should outperform those on a classical single instance of the database.

#### **Cloud benefits:**

- Flexibility: Analytical workloads are bulky in nature. The cloud can be used to provide compute and I/O elasticity such that the capacity of the system can expand for large queries and be shrunk for those times when small queries are running.
- Cost reduction: In an ACTiCLOUD-enabled data center, we expect to be able to utilize cloud elasticity to pre-provision the system only when we want to deliberately flood the database with large amounts of processing power for intensive algorithms on large graphs (e.g., Page Rank).

#### *Use case 7: Enterprise Operations*

**Use case description:** This use case replicates analytics in a representative production environment. The database will serve multiple concurrent users each with their own query workload and unpredictable frequency. Failures in the underlying hardware and software are expected, though the nature and timing of those failures are difficult to predict. Nonetheless, the database should appear to remain available to its users.

#### **Application properties:**

- The graphs are heterogeneous in size and structure.
- Writes occur at unpredictable times and volumes.



- Arbitrary ad-hoc queries and graph algorithms may be executed to gain insight into the data.

**Application requirements:**

- Any queries issued by an end-user looking for insights should run as quickly as possible, but should not starve other queries.
- Latency of queries on the ACTiCLOUD platform should outperform those on a classical single instance of the database.

**Cloud benefits:**

- Large queries can occupy any compute resource available to them, without starving others.
- Allow analytical queries to run alongside transactional queries without exhausting resources.

## 4 ACTiCLOUD Architecture

In this section we first describe the base architecture of ACTiCLOUD, as dictated by the project's strategic objectives, business scenarios, and application use cases. Then, we describe the architecture in more detail, providing information about the individual components.

### 4.1 Overview

The core components of the ACTiCLOUD architecture consist of:

- Entirely new components that need to be incorporated in the system,
- Components that already exist but need to be properly extended, and
- Other key components in the cloud stack that play an important role in the proposed architecture but will remain intact.

To meet its ambitious goals, ACTiCLOUD needs to consider a cloud system from the hardware level, up to the application and the level of geographically distributed cloud sites. In the next paragraphs we identify the core components of the ACTiCLOUD architecture and provide a short description of their role in the operation of the system (Figure 4.1).

**ACTiCLOUD::Hardware.** ACTiCLOUD considers state-of-the-art hardware (servers, CPU, memory modules, etc.) following two different flavors: x86-based systems that represent the typical ISA-wise component of a cloud datacenter (Numascale platform), and ARM-based systems that capture the features of low-power, microserver technologies that can serve both as cloud datacenter servers and as edge devices (KMAX platform).

**ACTiCLOUD::Aggregation layer.** This component is key to the ACTiCLOUD approach as it provides hardware and software support to pool resources at the rack scale.

**ACTiCLOUD::Hypervisor.** We consider extensions of state-of-the-art hypervisor technologies that will be able to provide mechanisms for allocating and managing resources at the rack scale.

**ACTiCLOUD::Cloud manager.** We embrace state-of-the-art cloud management technologies (OpenStack in our case) to utilize their already existing mechanisms in resource monitoring and management.

**ACTiCLOUD::ACTiManager.** The ACTiManager is the heart of the ACTiCLOUD system that will undertake the critical tasks of optimizing resource allocation, communication, and collaboration with sibling cloud sites dynamically.

**ACTiCLOUD::Guest OS.** This is the typical Guest OS layer inside a virtual machine that will not be changed by the ACTiCLOUD approach.

**ACTiCLOUD::Guest OS::System libraries.** This component will be extended to provide enhanced capabilities in memory allocation and topology awareness within the guest operating system.

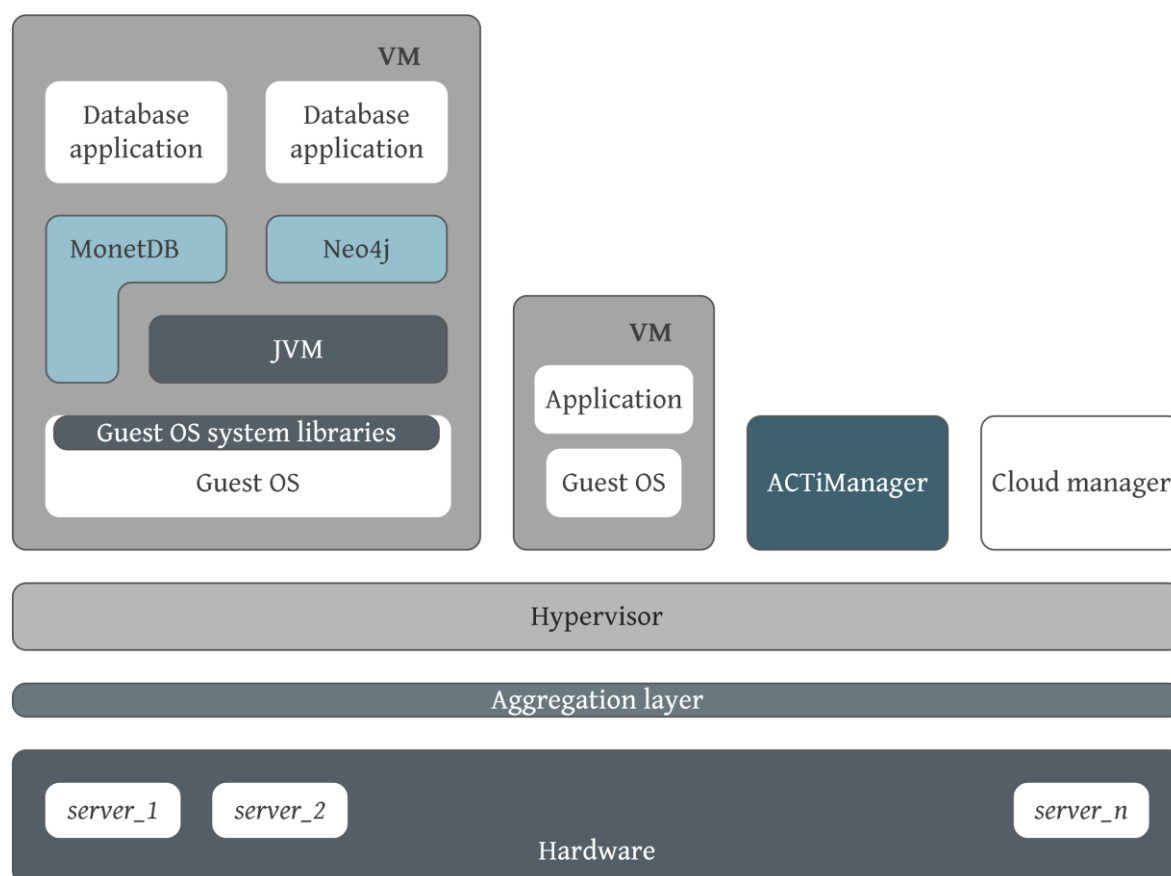
**ACTiCLOUD::JVM.** The Java Virtual Machine, being a core component of the state-of-the-art database systems and big data analytics frameworks, will be extended to support enhanced resource and memory allocation.

**ACTiCLOUD::MonetDB.** MonetDB operates both independently (i.e., as a normal database server) and in a JVM (i.e., the embedded MonetDBJavaLite version). MonetDB will be extended to be able to execute in a cloud environment, benefiting from the scalability, flexibility, and elasticity in

resource provisioning an ACTiCLOUD-enabled system can provide.

**ACTiCLOUD::Neo4j.** Neo4j operates solely on top of JVM. Neo4j will be extended to make full use of the resources of the state-of-the-art hardware resources offered at the rack scale by an ACTiCLOUD-enabled system.

**ACTiCLOUD::Application.** This is either a typical cloud application or a database-centric application executing on top of MonetDB or Neo4j.



**Figure 4.1:** Base ACTiCLOUD architecture.

## 4.2 ACTiCLOUD::Hardware

In this section we describe the architecture subcomponents regarding the hardware platforms. As ACTiCLOUD targets two hardware platforms, the Numascale and the KMAX platform, we split the description accordingly.

### 4.2.1 Numascale platform

Numascale's NumaConnect™ technology enables computer system vendors to build scalable shared memory servers with the functionality of enterprise mainframes at the cost level of clusters. The technology unites all the processors, memory, and I/O resources in the system in a fully virtualized environment that is controlled by standard operating systems (Figure 4.2).

NumaConnect enables significant cost savings in three dimensions: resource utilization, system management, and programmer productivity. According to long time users of both large shared memory systems and clusters, in environments with a variety of applications, the large shared memory systems provide a much higher degree of resource utilization due to the flexibility of unified system resources.

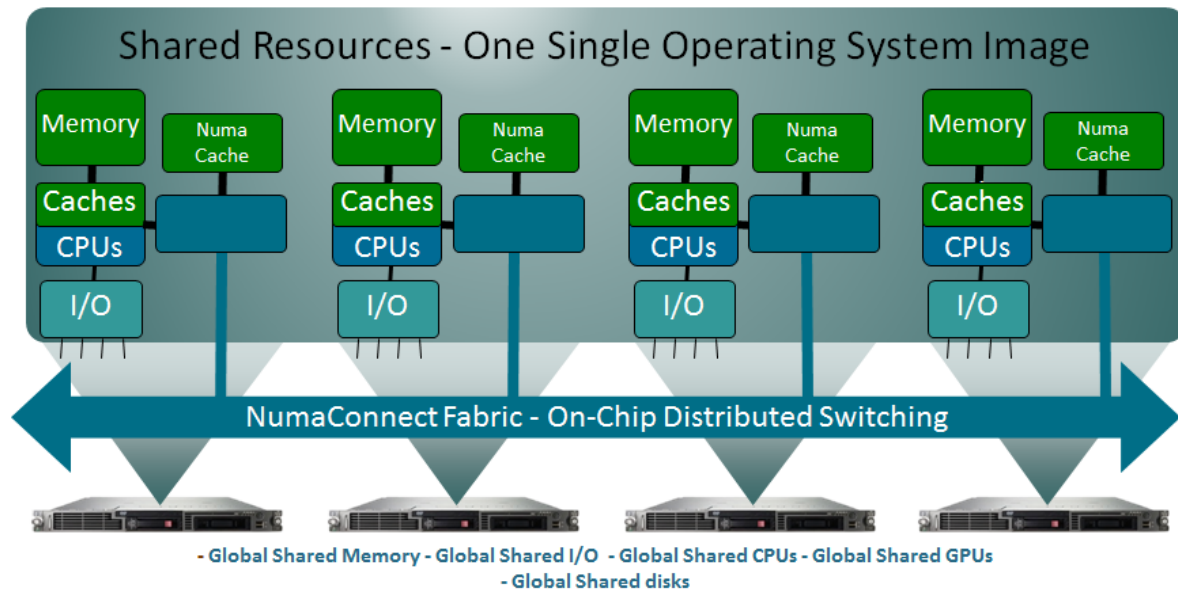


Figure 4.2: Numascale Architecture.

Systems based on NumaConnect will efficiently support all classes of applications using shared memory or message passing through all popular high level programming models. System size can be scaled to 4K nodes where each node can contain multiple processors. Memory size is limited by the 48-bit physical address range provided by the AMD Opteron processors resulting in a total system main memory of 256 TBytes. Support for newer processors (i.e., Intel Skylake) is in the making and will be available in 2018.

Parallel processing in a cluster requires explicit message passing programming, whereas shared memory systems can utilize compilers and other tools that are developed for multi-core processors. Parallel programming is a complex task and programs written for message passing normally contain more code than programs written for shared memory processing. Since all programs contain errors, the probability of errors in message passing programs is higher than in shared memory programs. A significant amount of software development time is consumed by debugging errors, further increasing the time to complete development of an application.

#### 4.2.2 KMAX platform

The Kaleao KMAX platform (Figure 4.3) is a new-generation server system architecture based on the principles of true convergence with the hardware design creating a “share-anything” resource scale-out platform. The KMAX chassis (Figure 4.6) provides management and cooling to

12 KMAX blades that present independent hyper-converged compute, storage, and switching units. Each KMAX blade (Figure 4.4) contains an embedded Ethernet switch that provides 2 \* 40Gb QSFP connections to switches external to the KMAX or can be used to stack a number of blades together, providing less external bandwidth. Each blade hosts 4 compute nodes, each node being provided with 2 \* 10Gb links from the blade switch. Compute nodes (shown in Figure 4.5) host the processing and storage subsystems of the KMAX. Each compute node contains 4 Exynos 7420 64 bit ARMv8 processors fabricated using the Samsung low power 14nm FinFet process. The Exynos utilizes the big.LITTLE architecture, providing 4 x Cortex-A57 cores running at 2.1GHz and 4 x Cortex-A53 cores running at 1.5GHz, giving a total of 8 cores per processor. The big.LITTLE arrangement enables the processor to shut down the A57 cores and run with just the A53 cores activated, enabling the processor to reduce its power consumption when running smaller workloads. Each of the 4 Exynos is connected to its own 4GB LPDDR4 RAM memory with bandwidth scaling up to 24.88GB per second. For storage, each Exynos is connected to its own 128GB UFS (Unix File System) drive that provides up to 300MB/s of local storage bandwidth. In summary each node has 4 Processing Units that consists of the following: 8 core ARM Exynos, 4GB LPDDR4 RAM and 128GB of UFS storage.

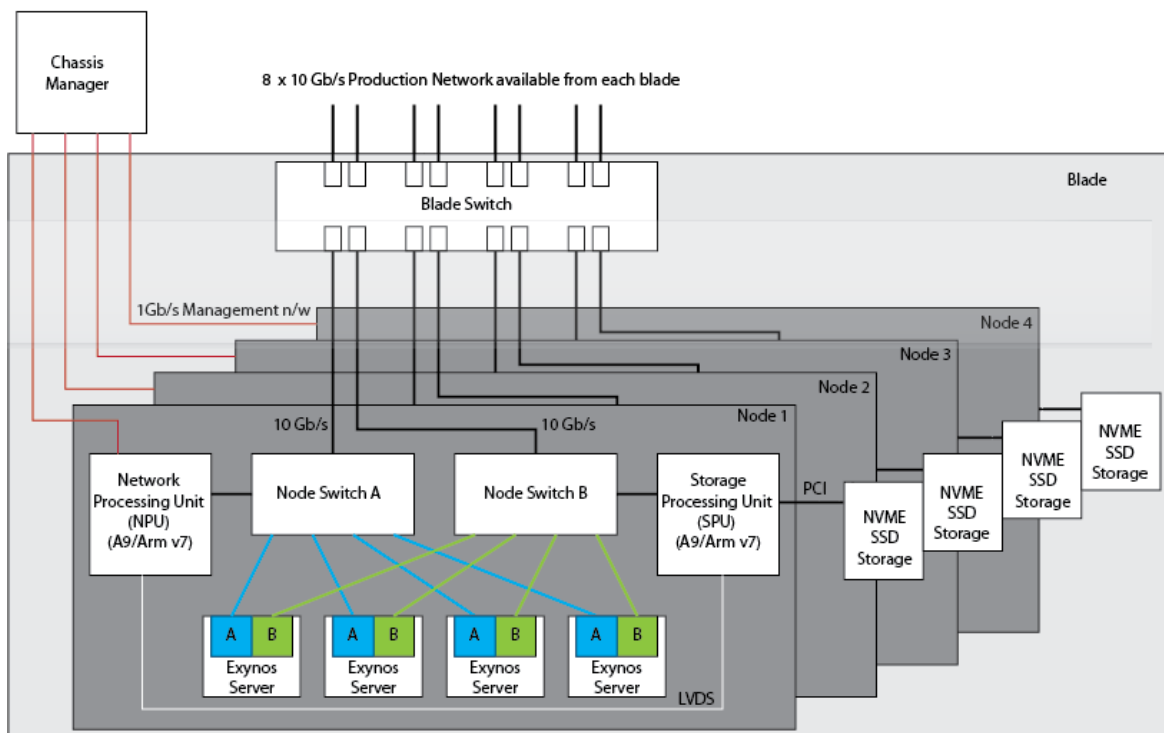
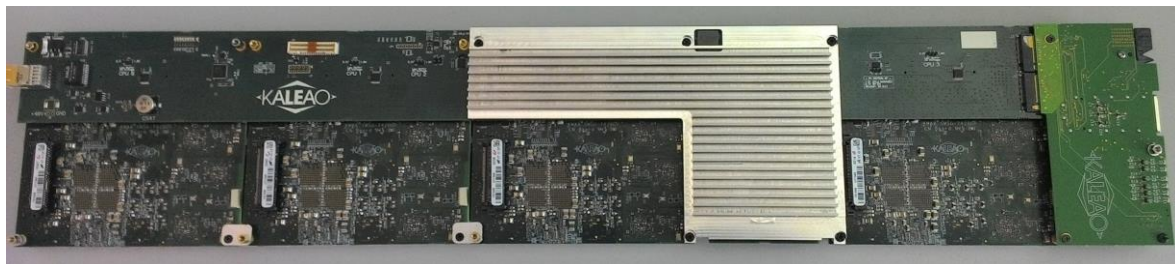


Figure 4.3: KMAX platform.

To provide external network access and additional storage to the four Processing Units on the compute node, there are 2 interconnected ARM A9 PSoc FPGA chips. We refer to the first PSoc as the Network Processing Unit (NPU) and the second as the Storage Processing Unit (SPU). Each of the four Exynos is connected via PCI links to both chips. The NPU is connected to the Blade switch using 2 \* 10Gb links, providing remote access to the compute node from the external 40Gb

connections and other compute nodes on the same blade. We refer to these links between the blade switch and the NPU as production networks A and B. The NPU creates physicalized interfaces that exist as “eth” endpoints on the PCI link for each exynos and are assigned to ingress and egress traffic to either production network A or B. Figure 4.5 shows four Exynos with interfaces a and b, that could be named as eth0 and eth1 within the host OS. These physicalized interfaces are then assigned to the external production network A or B 10Gb connections. It is easiest viewed as though the NPU creates two virtual switches and creates interfaces on the PCI line to the Exynos that are assigned to either switch. The Exynos are able to communicate on the same node using the switch on the NPU, whereas all external traffic is forwarded to the blade switch to be forwarded to a second NPU on the same blade or via either of 40Gb links that provide external access to the blade. Each PCI link to the NPU switch is capable of 2.5Gbps, providing non-blocking access to the production A and B networks from each Exynos.



**Figure 4.4:** KMAX blade.

The second PCI link from each Exynos is connected to the SPU FPGA and is utilized to provide access to the on node storage sub system. The PCI link provides an Ethernet interface on the node to carry traffic to a switch on the SPU. In turn, this will be connected to the storage fast path logic that will read and write ATAoE frames to the NVMe directly using PCI express gen 3. This provides a theoretical maximum of 2GB per second throughput to the disk. The A9 of the PSoc terminates management messages and configures the fast path hardware but all I/O traffic is forwarded directly from the PCI link of the Exynos to the PCI link of the disk without any CPU intervention.



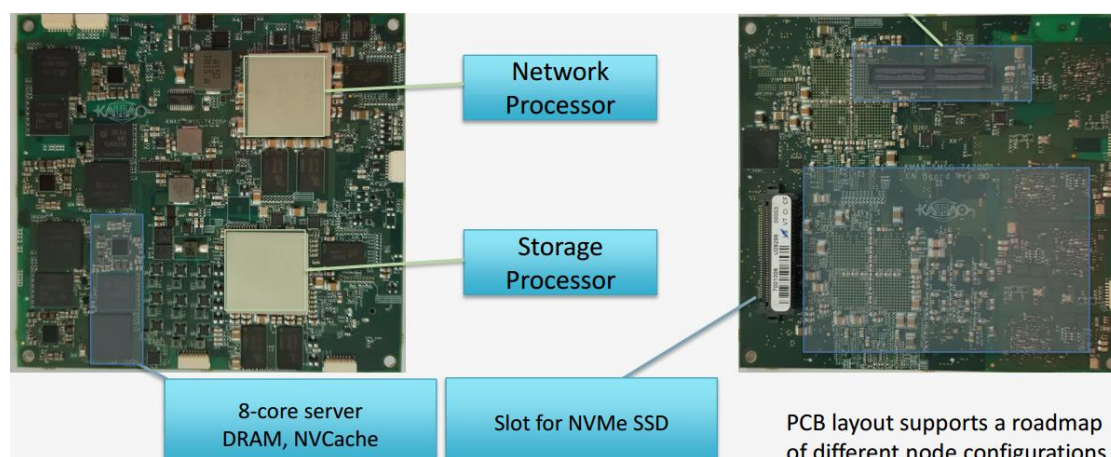


Figure 4.5: KMAX compute node.

The KMAX has been designed to condense the maximum number of ARM cores and storage into standard rack dimensions, as can be seen in Figure 4.6. A single 42U rack of such a system could therefore scale to provide applications access to over 20,000 cores with access up to a 350 TB pool of paged memory, 14 Tb/s of network access bandwidth and 6 PB of flash storage.

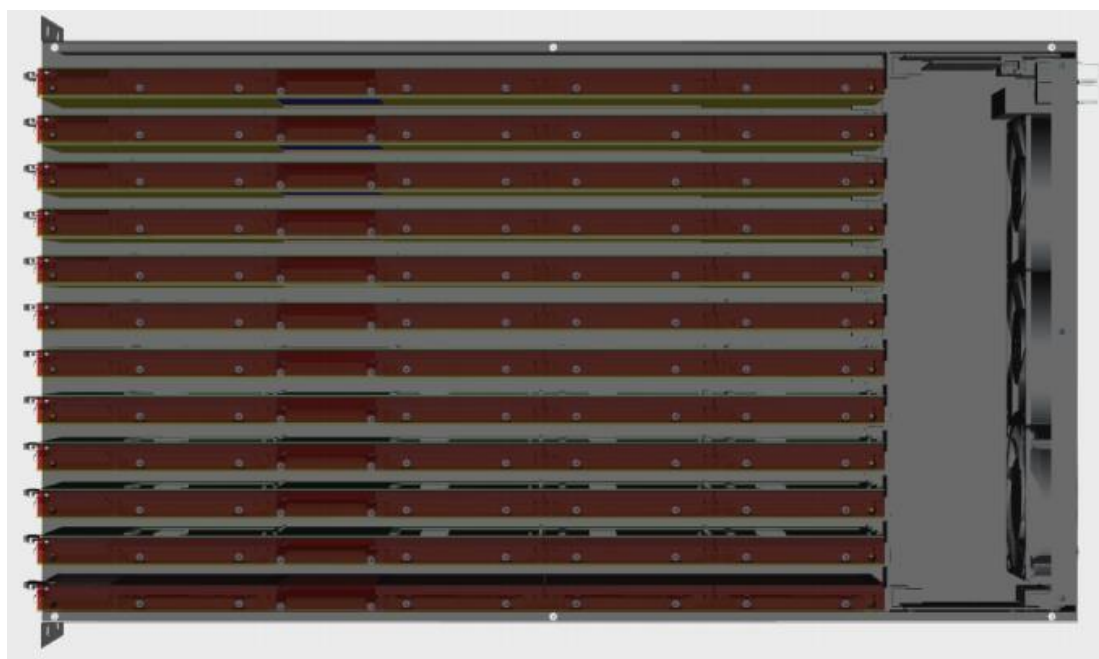


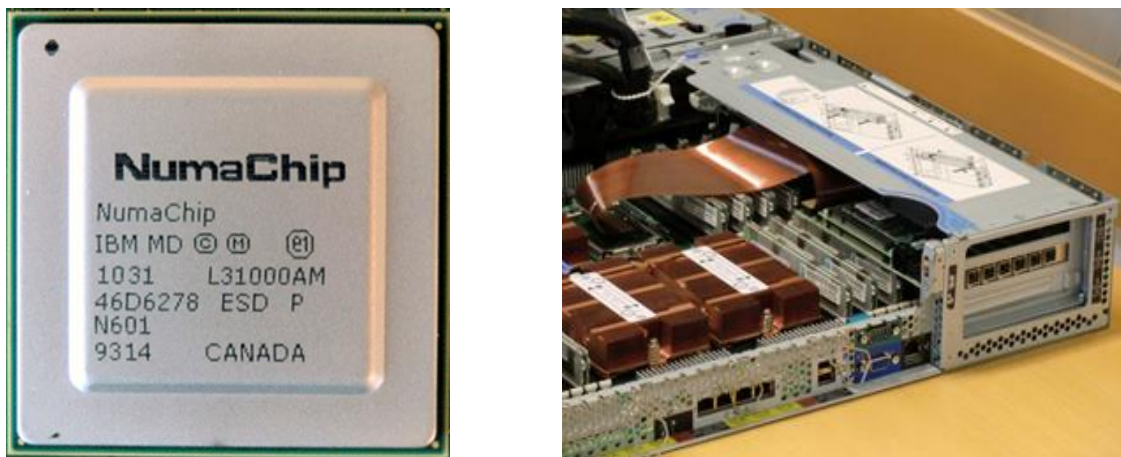
Figure 4.6: KMAX Chassis layout.

### 4.3 ACTiCLOUD::Aggregation Layer

Here we describe the architecture subcomponents regarding the Aggregation Layer. As the aggregation layer is tightly coupled with the relevant hardware support, we split this subsection into the Numascale and the KMAX platforms.

#### 4.3.1 Numascale platform

NumaChip contains an on-chip switch to connect to other nodes (servers) in a NumaChip based system (meaning that the other servers also have a NumaChip), eliminating the need to use a centralized switch. It is designed to communicate with the CPUs on the motherboard using the processor bus on a commodity server and bridge servers in a Cache Coherent Shared Memory manner. The end result is a large shared memory system where up to 4096 servers can be joined in a Single SMP. It works like this: (i) The NumaChip occupies one socket on the motherboard (Figure 5.6), (ii) The NumaConnect Card bridges the communication on the motherboard to other servers in a Numascale Shared Memory System (Figure 5.7), (iii) NumaConnect cables connect the servers together (Figure 5.8), and (iv) the end result is one unified system (Figures 6.9 and 6.10).



**Figure 4.7:** The NumaChip occupies one socket on the motherboard.





**Figure 4.8:** The NumaConnect Card bridges the communication on the motherboard to other servers in a Numascale Shared Memory System.



**Figure 4.9:** NumaConnect cables connect the servers together.

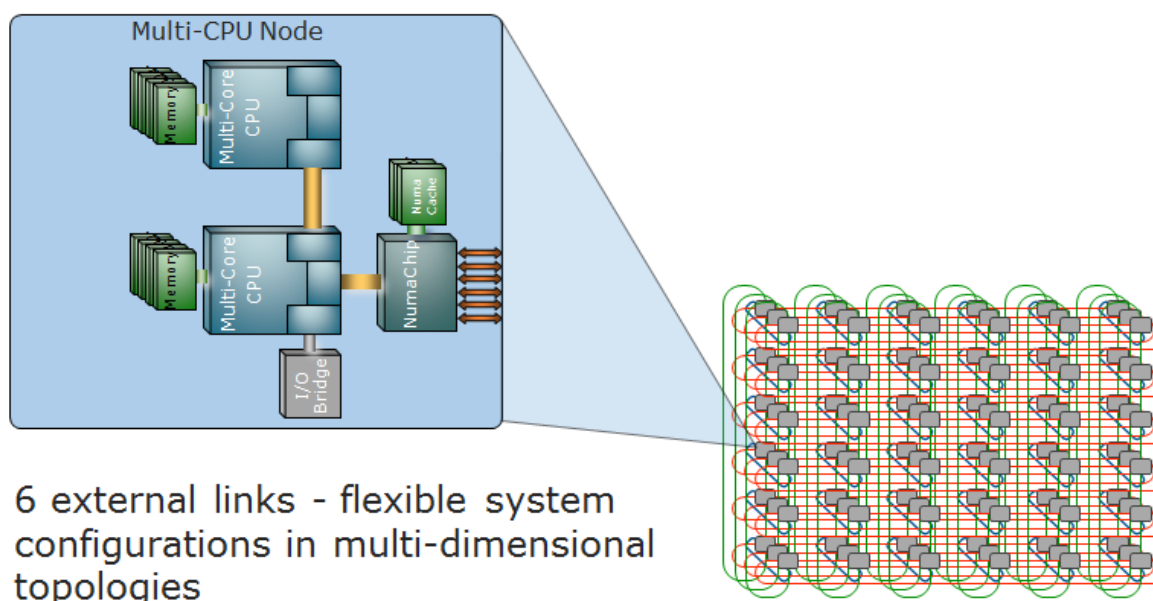


Figure 4.10: Unified Numascale system.

```
top - 09:29:57 up 20 days, 20:07, 6 users, load average: 8.32, 8.31, 8.32
Tasks: 3068 total, 3 running, 3065 sleeping, 0 stopped, 0 zombie
Cpu(s): 1.4%us, 0.1%sy, 0.0%ni, 98.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1583148160k total, 1185082348k used, 398065812k free, 4k buffers
Swap: 33046524k total, 0k used, 33046524k free, 4035576k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
202867	root	20	0	1119g	1.1t	524	R	815	74.1	3758:34	x.mod2as
197024	root	20	0	19168	8500	1204	R	37	0.0	403:40.55	htop
206397	root	20	0	13512	3704	924	R	14	0.0	0:04.48	top
197023	root	20	0	13512	3700	924	S	13	0.0	177:20.24	top
1399	root	20	0	0	0	0	S	3	0.0	1:03.79	ksoftirqd/348
2187	root	20	0	0	0	0	S	3	0.0	0:50.98	ksoftirqd/545
10	root	20	0	0	0	0	S	2	0.0	290:26.05	rcu_sched
11190	root	20	0	0	0	0	S	1	0.0	68:02.65	kworker/40:1
11177	root	20	0	0	0	0	S	1	0.0	173:47.85	kworker/24:1
11241	root	20	0	0	0	0	S	1	0.0	86:23.99	kworker/44:1
11668	root	20	0	0	0	0	S	1	0.0	53:37.71	kworker/348:1
11688	root	20	0	0	0	0	S	1	0.0	59:31.14	kworker/336:1
12035	root	20	0	0	0	0	S	1	0.0	20:37.15	kworker/545:1
11197	root	20	0	0	0	0	S	0	0.0	96:09.80	kworker/36:1
11203	root	20	0	0	0	0	S	0	0.0	148:15.71	kworker/32:1
11209	root	20	0	0	0	0	S	0	0.0	166:31.05	kworker/28:1
11233	root	20	0	0	0	0	S	0	0.0	82:54.52	kworker/48:1
11626	root	20	0	0	0	0	S	0	0.0	32:13.59	kworker/308:1
11710	root	20	0	0	0	0	S	0	0.0	26:04.65	kworker/357:1
11714	root	20	0	0	0	0	S	0	0.0	31:09.39	kworker/354:1
47717	root	20	0	7764	576	484	S	0	0.0	22:06.24	tail
47736	root	20	0	7764	576	484	S	0	0.0	22:13.57	tail

Figure 4.11: The unified Numascale system allows the OS/Hypervisor to manage all memory as if it was a single server.

All servers are acting as one cache coherent system without the need for a virtualization layer, or any centralized communication protocol.

The big differentiator for NumaConnect<sup>1</sup> compared to other high-speed interconnect technologies is the shared memory and cache coherency mechanisms. These features allow programs to access any memory location and any memory mapped I/O device in a multiprocessor system with high degree of efficiency. It provides scalable systems with a unified programming model that remains the same from the small multi-core machines used in laptops and desktops to the largest imaginable single system image machines that may contain thousands of processors. The architecture is commonly classified as ccNUMA or NUMA but the interconnect system can alternatively be used as low latency clustering interconnect.

There are a number of advantages for shared memory systems that lead experts to hold that architecture as the holy grail of computing compared to clusters:

- Any processor can access any data location through direct load and store operations, which leads to easier programming with usually less code to write and debug compared to distributed systems that support only message passing (e.g., MPI) programming models.
- Compilers can automatically exploit loop level parallelism that enables higher efficiency with less human effort.
- System administration relates to a unified system, as opposed to a large number of separate images in a cluster, requiring less effort to maintain.
- Resources can be mapped and used by any processor in the system giving optimal use of resources in a single image operating system environment.

The fabric routing is controlled through routing tables that are initialized from system software at the BIOS level. This software is called the NumaConnect Bootloader. The NumaConnect Bootloader ensures that all resources are available, unifies the resources in software, and presents all servers to the Linux kernel as one single system. When the operating system boots, it sees all the servers as one platform.

The Numascale related hardware events are link retries, CRC failures, nCache DIMM or NumaChip over-temperature, or AMD Northbridge watchdog timeouts. Issues will be reported by the kernel “critical” error reporting mechanism, which broadcasts to all login sessions, aiding pre-failure diagnosis and supplanting running Numascale's nc2-check tool while trying to reproduce the hang, after the fact. These Numascale related errors are on a system level and reported on kernel level on all partitions (VMs). They will be made available in the system logs and reported (/var/log/syslog), and on the console. The errors reporting will be implemented for the kernel running CentOS7 as a proof of concept. Numascale will assist in reporting the errors via the communication mechanisms that OnApp will integrate in the MicroVisor hypervisor platform.

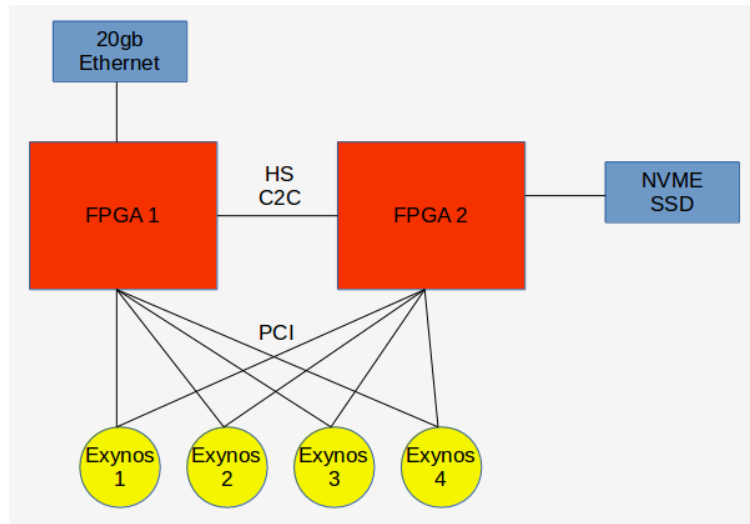
#### 4.3.2 KMAX platform

The KMAX compute board has been designed to enable custom modification to accelerate different customer deployment scenarios and workloads. This is achieved through the use of two Xilinx PSoc (Figure 4.12) that provide connectivity for the PCI Links of the Exynos and are connected to both the high speed external 20Gbps network provided by the blade switch and the

---

<sup>1</sup> [https://www.numascale.com/the-numachip-enables-cache-coherent-low-cost-shared-memory/numa\\_products.html](https://www.numascale.com/the-numachip-enables-cache-coherent-low-cost-shared-memory/numa_products.html)

2GB/s link to the NVMe drive. The FPGA is configured by default to provide NIC functionality and out of band management to the Exynos via the PCI links. It also provides Ethernet switching capability and will provide ATAoE (ATA over Ethernet) fastpath support for accelerated NVMe storage access from the Exynos.



**Figure 4.12:** The KMAX compute node accelerates different scenarios and workloads through two FPGAs.

With this configuration, the KMAX platform is able to support adaptation of the units that reside inside the FPGA to support and accelerate the hardware aggregation requirements identified by the application developers. For example, it is possible to support a system-wide virtual disk that aggregates multiple physical disks, whilst maintaining a scheme to determine the locality of disk offsets to individual Exynos. The translation between virtual and physical identities could be handled in hardware so the overhead of virtualizing and accessing remote disk offsets would be minimized. Additionally, for communication-intensive scenarios, KMAX will support zero copy DM transfers between user space applications on distant nodes.

#### 4.4 ACTiCLOUD::Hypervisor

In this section we describe the architecture subcomponents regarding the Hypervisor, focusing on the specific technology embraced i.e., OnApp's MicroVisor. Figure 4.13 shows how the MicroVisor differs from the standard Xen architecture from which it has been forked. Figure 4.14 shows how the lightweight design of the MicroVisor architecture can be used in its distributed mode for utilizing resources on remote boards. Finally, Figure 4.15 depicts how in particular the storage resources of another network-attached block device can be used to form a distributed storage platform.

The MicroVisor is designed to be a lightweight, distributed Hypervisor platform that is suited for emerging hardware platforms. Currently x86 (Intel/AMD) platforms are facing issues in scalability and have used NUMA architectures for increasing the number of processor sockets and addressable memory on a single board. Cores in NUMA hardware are linked with certain regions of memory but can access the entire system memory at a higher incurred latency cost

and at lower speeds. ARM based, micro-server hardware platforms incorporate many cores that have lower performance than the incumbent x86 (Intel/AMD) platforms, but are more energy efficient and have been promising to enter the data center market as a cost effective alternative [DCR15]. The MicroVisor has been designed to work on both x86 (Intel/AMD) and ARM hardware platforms. However, the MicroVisor is primarily designed to target the aforementioned low power ARM-based processors that have fewer resources.

The main difference of the MicroVisor from Xen and KVM is that there is no control domain (Dom0 in Xen terminology). The centralised Dom0 model means that split driver requests have to pass through the control domain for each request and it becomes a bottleneck when many requests are being served to multiple guest domains (DomUs). This difference is highlighted in Figure 5.12 where the traditional Xen Hypervisor platform is shown on the right with a control domain and on the left the MicroVisor is shown. In order to interface with hardware, which is one of the normal roles of the control domain in this new architecture, new driver domains are needed. These driver domains interface with the hardware. The current implementation of the MicroVisor uses the MiniOS [MOS] or a minimal Linux kernel for adding the driver support.

The other big difference between the MicroVisor and Xen is that network requests that are triggered by storage I/O requests, are not sent in the usual TCP/IP encapsulation but rather as Ethernet frames. These Ethernet frames ensure that the overhead of TCP/IP processing is removed and as such the virtualization overhead is reduced. Overall the use of Ethernet control frames for the management, control, and monitoring of the MicroVisor platform, along with the driver domain and removal of Dom0 ensures that the virtualization overhead is much lower than the current Xen implementations. This results in higher performance and better control for each of the resources and the guest domains.

Finally, the distributed model also allows for resources from remote MicroVisors to be shared and distributed, which fits the expected ACTiCLOUD architecture well. In the Numascale hardware-assisted case, the high-speed memory interconnect will allow for the MicroVisor to share resources on a very fast path.

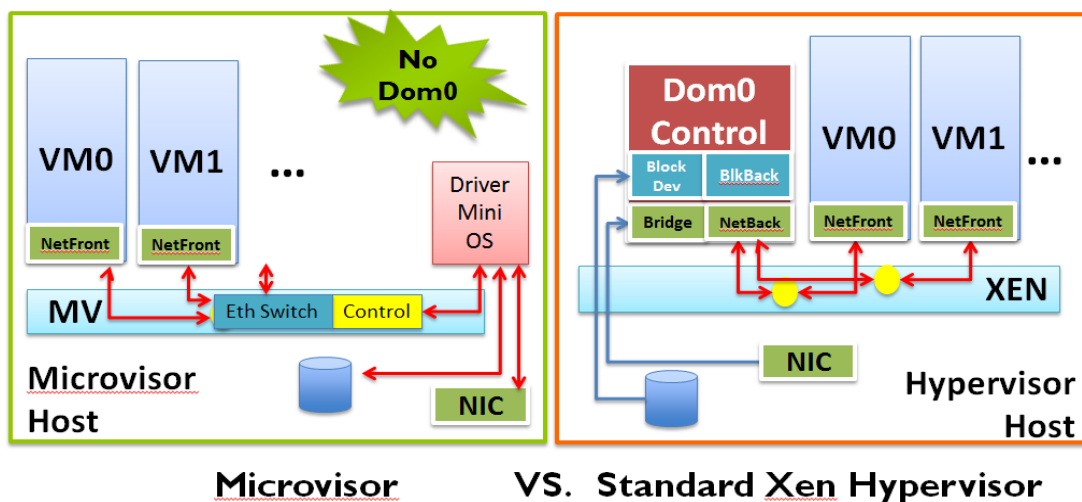
The Hypervisor consists of the following subcomponents:

- **Driver domains.** For the MicroVisor, the split driver model of Xen is further extended. Given that there is no control domain (no Dom0), a driver when required is launched as its own Virtual Machine (driver domain) that has access to the physical resource that can then be shared with the guests. The current implementation of the driver domain uses MiniOS as the host OS, which is a cut down minimalistic Xen kernel used for stub domains.
- **Resource scheduler.** The MicroVisor resource scheduler can be configured through the MicroVisor API. Pinning of resources can be carried out accordingly. The scheduler then decides how the resources are presented through the driver domains to the guest Virtual Machines and so can be used for QoS and performance configuration.
- **Orchestration API.** Control of the MicroVisor is carried out through the MicroVisor API. Assigning workloads, VMs and the connected resources is carried out through this API.
- **Virtual Switch.** An integrated internal switch to the MicroVisor handles packet forwarding. This will need to be worked on in the scope of ACTiCLOUD to manage the shared network resources from different hardware nodes to provide aggregated network resources that are linked with the guest VMs.

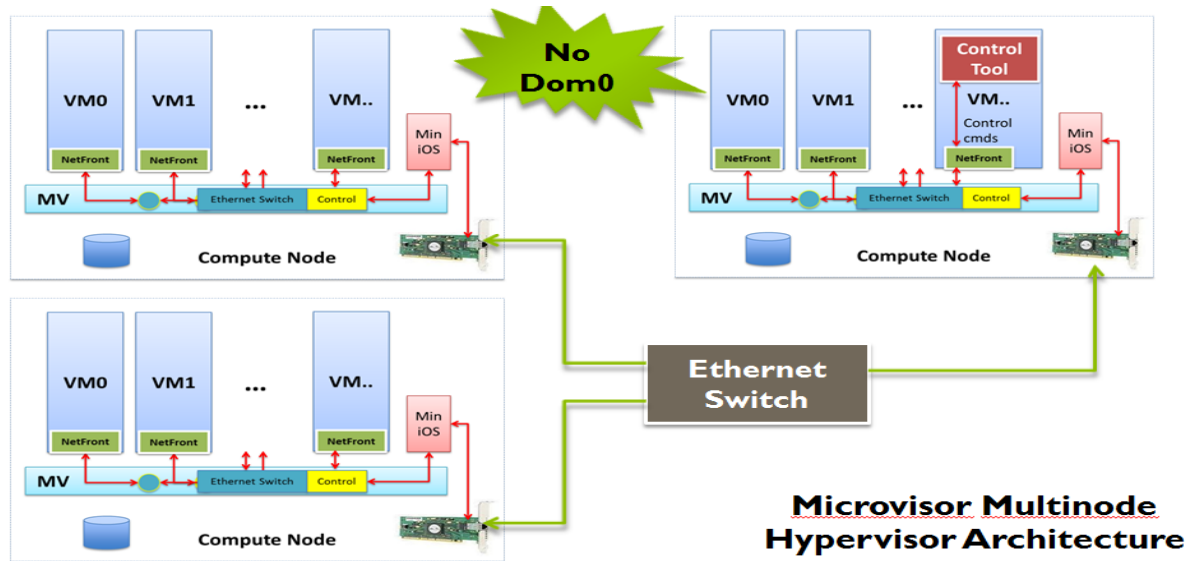


- **Ethernet packet handling.** The MicroVisor does not use TCP/IP. Instead Ethernet frames are used for controlling the MicroVisor. This avoids having the heavy TCP/IP stack within each domain, but does mean that the fault tolerance etc. of TCP has to be replicated. VMs residing on the MicroVisor can continue to use TCP/IP on top of Ethernet.
- **Monitoring system.** This system is built into the MicroVisor. Some values that would normally be exposed in GNU/Linux via the “/proc” interface, for instance, are exposed through the Monitoring API. Currently, some statistics about CPU, network, memory, and storage can be queried. Depending on the requirements of the higher level components, these can be extended.
  - **Monitoring API.** The monitoring values are stored using the round robin database (RRD) format. To get the current status of a particular MicroVisor an API is used, that describes the resource that is being interrogated and how frequently the monitoring should be carried out.

OnApp has developed a distributed, hyper-converged storage platform that has been developed for the OnApp Cloud platform as a separate storage product. OnApp Storage allows for Hypervisors to use direct attached storage as a distributed block-storage system. This has been used for many years with Xen and KVM and is used in a large part of the ONAPP customer base and is seen as a disruptive technology versus the conventional network attached storage (NAS) platforms that are usually used in the data center. The storage platform is in the process of being adapted for the MicroVisor platform (Figure 4.15), such that it can be used to share storage resources on the Kaleao KMAX and Numascale hardware platforms. This development work will be extended to attempt supporting the sharing of other resources, focusing on network and memory resources. CPU resources are hard to share and are likely hard to fully share without increased hardware support is beyond the scope of this project.

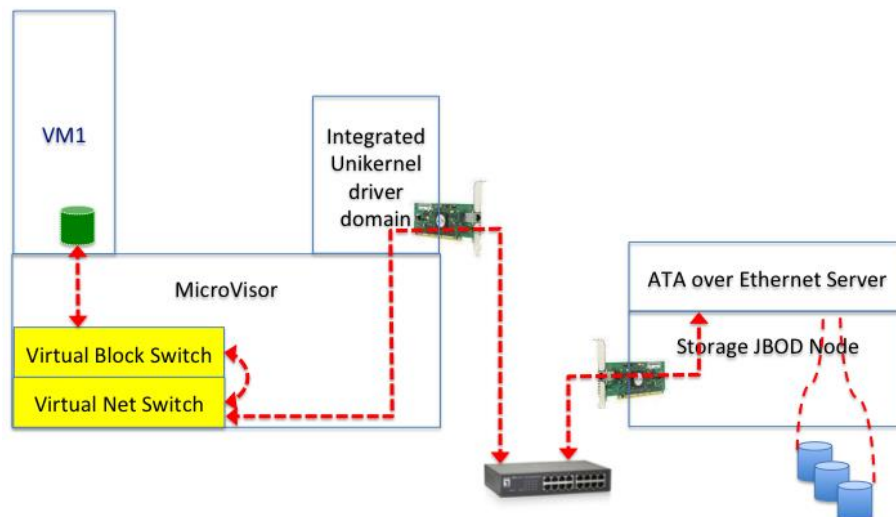


**Figure 4.13:** Comparison of MicroVisor with the standard Xen architecture.



**Figure 4.14:** The lightweight design of the MicroVisor architecture provides a distributed mode for utilizing resources on remote boards.

## MicroVisor Block IO Architecture



**Figure 4.15:** The MicroVisor forms a distributed storage platform using the storage resources of other network attached block devices.

## 4.5 ACTiCLOUD::CloudManager

The cloud manager controls large pools of compute, storage, and networking resources in a cloud site. It provides mechanisms to monitor, deploy, migrate, and terminate hosted applications. While large industrial players in cloud resource provisioning, such as Amazon, Microsoft, Google, and VMware, rely on in-house commercial software to manage their own clouds, OpenStack has become the defacto choice of most other public cloud providers and private in house clouds. We have chosen OpenStack as our cloud manager for ACTiCLOUD due to its open standards within the industry and its level of adoption in both small and large scale deployments. ACTiCLOUD aims to design and implement a hierarchy of resource schedulers that will operate dynamically at the rack and site levels by extending existing policies and collaborating with OpenStack, in order to maximize our impact and promote cloud interoperability and openness.

OpenStack is a rich and complex software environment with numerous capabilities for cloud management. Within ACTiCLOUD, the following OpenStack modules are directly relevant to our approach:

- **Nova**<sup>2</sup> is the primary computing engine behind OpenStack. It is used for deploying and managing large numbers of virtual machines and other instances to handle computing tasks. This module is required to spawn new VMs, migrate VMs, terminate VMs, and other important operations needed by the ACTiManager.
- **Ceilometer**<sup>3</sup> provides telemetry services, which allow the cloud to provide monitoring data that can be used for other management decisions (e.g., placement, modeling, and prioritization) and billing services to individual users of the cloud.
- **Heat**<sup>4</sup> is the orchestration component of OpenStack, which allows developers to store the requirements of a cloud application in a file that defines what resources are necessary for that application. In this way, it helps to manage the infrastructure needed for a cloud service to run. This module can be extended to accommodate policies related to applications' scaling (scale up/down, scale out/in) requirements.
- **Watcher**<sup>5</sup> provides a flexible resource optimization service for multi-tenant OpenStack-based clouds. Watcher provides a complete optimization loop, including a metrics receiver, optimization processor, and an action plan applier. This allows a wide range of cloud optimization goals, including the reduction of data center operating costs, increased system performance via intelligent virtual machine migration, increased energy efficiency. This module can be extended to accommodate part of the ACTiManager's logic.

## 4.6 ACTiCLOUD::ACTiManager

The ACTiManager aims to manage resources at both the node-level and the site-level. Hence, the architecture of ACTiManager follows a hierarchical approach, distinguishing the management of resources at each level. As shown in Figure 4.16, a specific ACTiManager component takes care of

---

<sup>2</sup> <https://docs.openstack.org/developer/nova/>

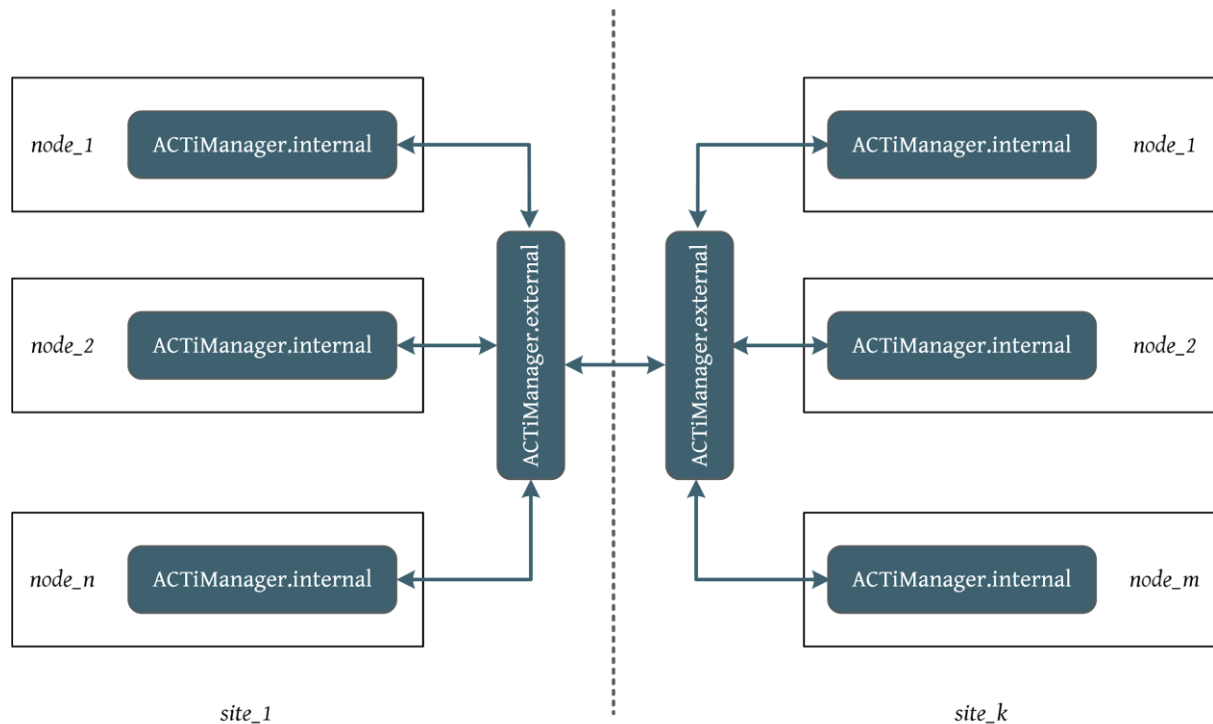
<sup>3</sup> <https://docs.openstack.org/developer/ceilometer/>

<sup>4</sup> <https://wiki.openstack.org/wiki/Heat>

<sup>5</sup> <https://wiki.openstack.org/wiki/Watcher>



resource orchestration internally within each node (ACTiManager.internal), and a specific ACTiManager component orchestrates resources across nodes within a cloud site and between distributed cloud sites (ACTiManager.external). Depending on the underlying architecture and setup of the cloud site, a node in one setup may be substantially different compared to another setup. To cope with this issue, within the frame of ACTiCLOUD we refer to a node as the atomic compute unit that is managed by a single hypervisor instance. Thus, in the frame of this project a node in the Numascale system is a rack of servers interconnected with Numaconnect (Figure 4.2), while a node in the KMAX platform is an Exynos server (Figure 4.3), both managed by one instance of the hypervisor (MicroVisor in our case).



**Figure 4.16:** The “ACTiManager” follows a hierarchical approach, distinguishing the management of resources at node level and at site level.

Regardless its level of operation (node or site), the ACTiManager consists of the following three subcomponents:

- The **Information Aggregator** component that is responsible for collecting information from the various monitoring facilities.
- The **Modeler** component that is responsible for providing models (i) to detect anomalies like interference, imbalance, overload, underload etc, (ii) to predict the impact of various actions (e.g., consolidation effect on the already executing application, migration time and failure probability), and (iii) to characterize application in terms of their resource footprint and co-execution behavior.
- The **Decision Maker** component that decides about the optimized resource allocation and initiates the relevant actions, including placement, prioritization, consolidation, migration,

interference mitigation, and resizing of the running VMs in ACTiCLOUD.

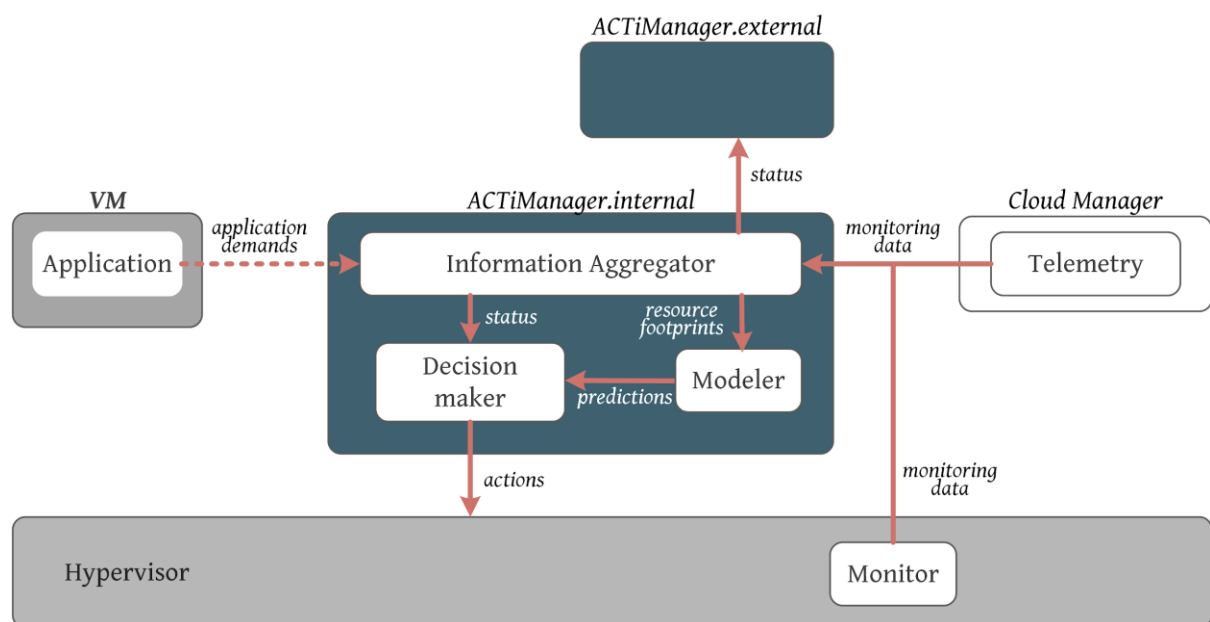
Next we describe the role of each subcomponent in more detail and its relevance to the level of operation, and provide more details on the interaction between the ACTiManager and the Cloud Manager.

#### 4.6.1 ACTiCLOUD::ACTiManager.internal

ACTiManager.internal is responsible for managing resources within a single node. To this direction, it collects information about the resource utilization of the executing applications within the node, performs sanity checks (e.g., detects interference, local overload/underload, underperforming applications) and takes relevant actions locally (e.g., remaps virtual to physical cores, moves data within the node, etc.). In case the local actions are unsuccessful or insufficient to enforce the desired policy, it informs the ACTiManager.external component. The general architecture and operation of the ACTiManager.internal component is shown in Figure 4.17.

The Information Aggregator collects information from three major sources:

- The **Telemetry** source that collects all the monitoring information that the standard agent of the Cloud Manager provides. In the context of OpenStack cloud manager, the Telemetry subcomponent can refer to Ceilometer.
- The **Monitor facilities within the hypervisor** that collect and make available information from the hardware monitoring facilities and events within the Hypervisor, e.g., CPU utilization, input/output operations per second (IOPs), instructions per cycle (PC), cache traffic, etc.
- The **application demands** reported with special interfaces directly to the ACTiManager. This is an optional source of information (denoted by the dotted line), as ACTiCLOUD does not demand this feature from applications, but can utilize it if it is available.



**Figure 4.17:** General architecture and operation of the “ACTiManager.internal” component that manages resources within a single node.

The Information Aggregator passes status information to the upper level of the ACTiManager hierarchy and also provides this information to the Modeler.

The Modeler serves two critical roles:

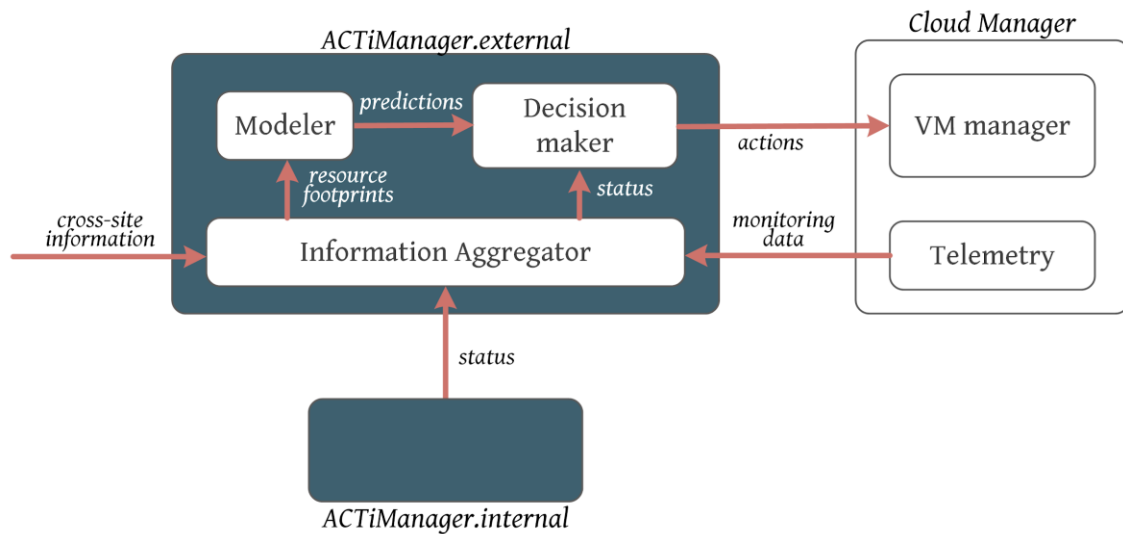
- It performs workload identification, classification, and correlation with the information gathered from the Information Aggregator.
- It predicts the potential for performance improvement, the cost of potential interference, and the cost of performing an action. Depending on the action under consideration, the Modeler may further predict the performance (cost-benefit) of the co-scheduling/co-location action (i.e., putting different VMs running close to each other), of the moving action, or of a resizing action.

The Modeler and the Information Aggregator both support the heart of the optimization processes that take place in ACTiCLOUD, i.e., the **Decision Maker** subcomponent.

The Decision Maker tries to optimize resource allocation at the node level by enforcing the policies and priorities that are defined by the users, VMs, or applications within ACTiCLOUD and by deciding on the appropriate consolidation of the running VMs/workloads within the node. To enforce its decisions, the Decision Maker in the ACTiManager.internal requests actions from the Hypervisor. Such actions include (but are not limited to) moving VMs and data within the node, freezing VMs, resizing VMs, and others.

#### 4.6.2 ACTiCLOUD::ACTiManager.external

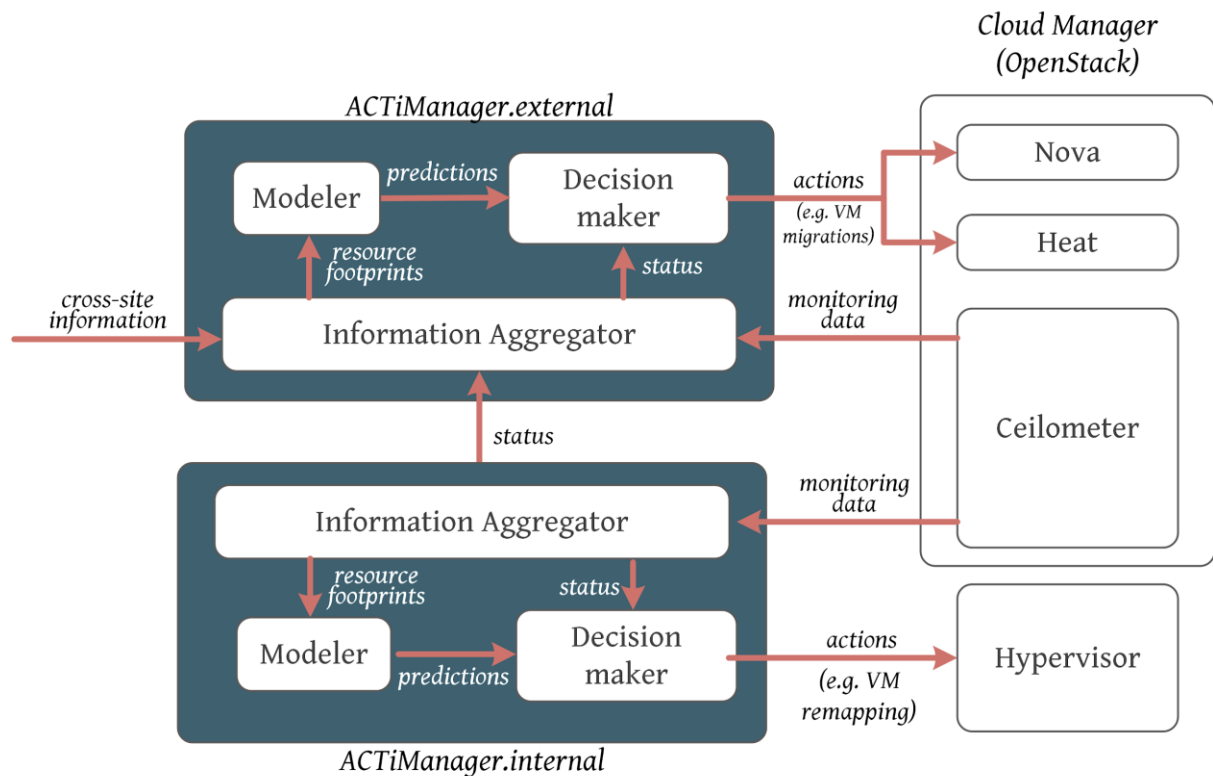
The operation and general architecture of the ACTiManager for the site level is shown in Figure 4.18. In this case, the Information Aggregator receives information (i) from the Telemetry facilities regarding the resource utilization of the platform and the running VMs, (ii) from the various ACTiManager subcomponents of each node regarding more detailed information on the execution status of each node, and (iii) from peer ACTiManager components of remote sibling sites. This information is passed to the Modeler which, in this case, provides modeling and prediction information for cross-node and cross-site migration actions in order to cope with problems like load imbalance between nodes, or site over/underutilization. The Decision Maker then decides on more profitable workload allocations and requests the relevant actions from the Cloud Manager.



**Figure 4.18:** General architecture and operation of the “ACTiManager.external” component that manages resources of a cloud site and across sites.

#### 4.6.3 ACTiManager - Cloud Manager roles and interaction

The ACTiManager and the Cloud Manager (OpenStack in our case) both lie at the same level of the base ACTiCLOUD architecture. Although OpenStack provides a rich set of capabilities to manage a cloud site, certain key features like interference-awareness, dynamic operation, and resource allocation optimization are missing. This is the part where the ACTiManager comes into play to interact with OpenStack and provide additional functionality. The core interaction between ACTiManager and OpenStack is shown in Figure 4.19. The Information Aggregator at both the node and site levels receives monitoring information from OpenStack’s Ceilometer, the Decision maker at the site level requests from Nova to take actions on VM placement and migrations, while the Decision maker at the node level requests from the hypervisor to take actions on VM co-scheduling and placement, i.e., re-pinning virtual CPUs to physical CPUs or moving data.



**Figure 4.19:** Interaction between ACTiManager and OpenStack.

#### 4.7 ACTiCLOUD::GuestOS::System Libraries

Here we describe the architecture subcomponents regarding the System Libraries of the Guest OS. These subcomponents include mainly optimizing the memory allocation library and the interaction with the guest operating system regarding memory allocation.

More specifically, the main subcomponent is a library that will be created to replace the built-in system library “libc”. There are several allocators which can be used instead of the standard build-in allocators. Some of them are suitable for general use and others are optimized for special demands of the applications. An allocator for general use must address speed, scalability, fragmentation, and avoidance of false sharing and lock contention. A well-known allocator is Hoard. Hoard performs well regarding speed, scalability, fragmentation and avoidance of false sharing and lock contention, but it lacks thread-awareness as it does not assign every thread its own private heap. Hoard also reuses empty memory blocks in other threads, which may result in returning remote memory from a malloc() call. The Numascale Memory Allocator, NCALOC, is an alternative memory allocator to the libc built in allocator.

The subcomponent of memory allocation involves tuning and adapting the memory allocator to key applications by extending and optimizing the NCALOC library. The optimization efforts will be driven by performance analysis of the specific database systems and applications of ACTiCLOUD to better adapt to their memory requirements and access patterns. In addition, we will focus on the performance effects of the kernel patches that affect memory allocation,

together with the partner code of JVM, NEO, and MonetDB. We will analyze the performance effect of these patches related to deployment (pinning) and scalability, supply a NUMA-aware optimized heap allocator, analyze runtime allocation behavior, and do profiling using the perf profiling tool.

#### 4.8 ACTiCLOUD::JVM

Here we describe the architecture subcomponents regarding the JVM. They refer specifically to Maxine VM, an open source virtual machine that is developed by UNIMAN. Figure 4.20 summarizes the subcomponents and shows their relation with each other.

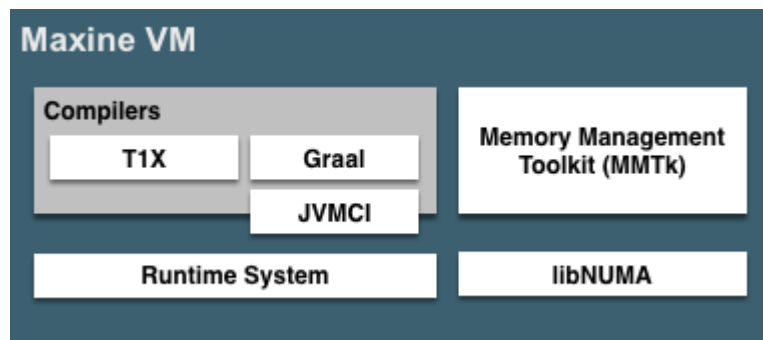


Figure 4.20: Architecture subcomponents regarding the JVM.

- **Maxine VM / T1X** compiler required for x86 and AArch64 on Numascale and Kaleao testbeds respectively. This is a template compiler (similar to an interpreter) for fast, but suboptimal, execution (short compilation time and therefore fast application response time).
- **Maxine VM / Graal** compiler required for x86 and AArch64 on Numascale and Kaleao testbeds respectively. This is the Just-In-Time (JIT) aggressive optimizing compiler that produces efficient code (slower response time due to code warmup but much lower execution time, i.e., peak performance).
  - **Maxine VM / Graal / JVMCI.** The Java Virtual Machine Compiler Interface (JVMCI) will enable Maxine VM to achieve an operational and up-to-date Graal compiler. In this way, our platform will utilize the best performing Java compiler in the market. Furthermore, by implementing JVMCI we automatically get Java 8 support which is necessary to run Neo4J workloads.
- **Maxine VM / MMTk.** The Memory Management Toolkit (MMTk) is a state-of-the-art collection of memory managers and Garbage Collection (GC) algorithms. By integrating it into Maxine VM we will be able to get the most complete GC toolkit in our VM which will consequently enable us to perform more detailed research towards NUMA-aware GC algorithms.

#### 4.9 ACTiCLOUD::MonetDB

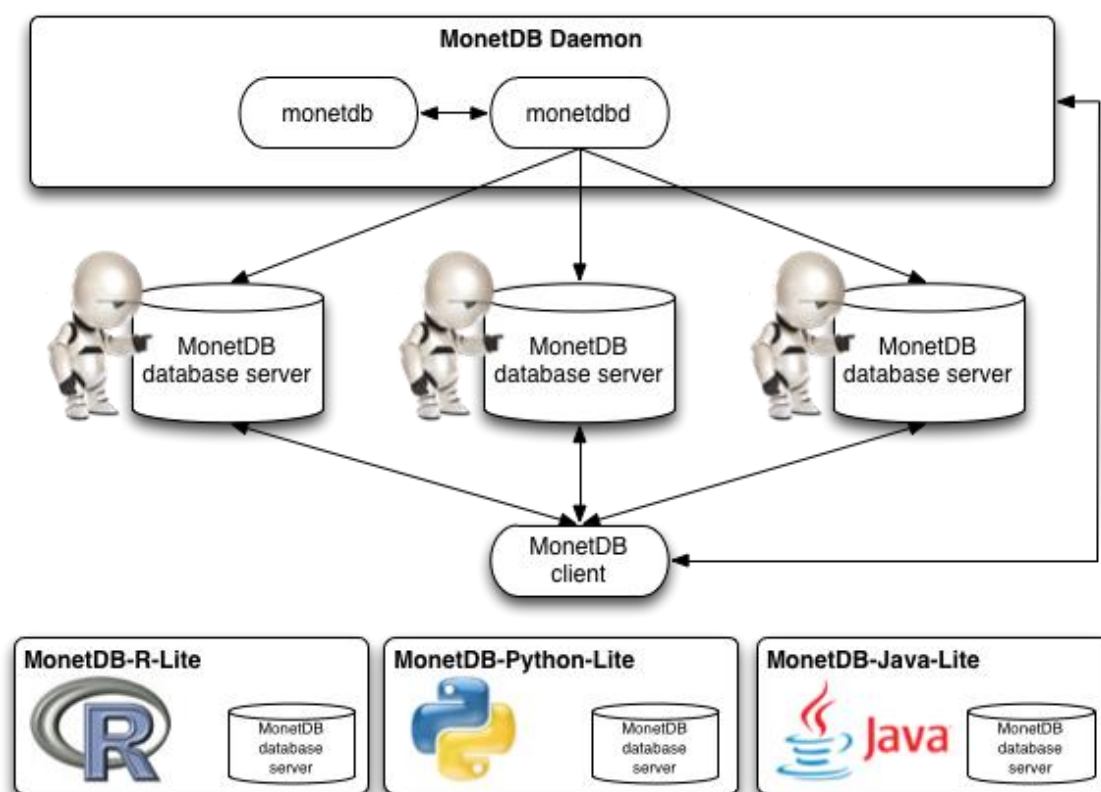


Figure 4.21: MonetDB subcomponents and their mutual relationships.

Here we describe the architecture subcomponents regarding MonetDB. Figure 4.21 summarizes them and shows their relations with each other.

- **MonetDB database server:** the main component of the MonetDB software suite. It contains a single C program called *mserver5*. Next to supporting SQL:2003 as its main query language, it also supports in-database execution of R, Python, and C code through SQL UDFs (User Defined Functions, a.k.a. stored procedures).
- **MonetDB daemon:** the MonetDB super server that controls the creation/destruction and start/stop of individual *mserver5* instances. It is only for Linux systems, and contains two C program:
  - **monetdbd:** the server of the MonetDB daemon.
  - **monetdb:** the client of the MonetDB daemon, interacts with *monetdbd*.
- **MonetDB client:** client programs and drivers in various programming languages to interact with *mserver5*.
  - **mclient:** is the MonetDB C-client. Next to communicating directly with *mserver5*, this particular client is also able to communicate with an *mserver5* through a MonetDB daemon.
- **Marvin:** the profiler of a MonetDB server.



- **MonetDB-R-lite, MonetDB-Python-lite, MonetDB-Java-lite:** embedded MonetDB server for R, Python and Java. These are stand-alone components.

For the deployment in cloud environment, depending on the needs of the applications, one can decide to put all MonetDB subcomponents into a single VM, or device the subcomponents into several groups and use one VM for each group. In general, it is recommended to deploy a MonetDB database server in a dedicated VM to avoid resource contention within the VM.

#### 4.10 ACTiCLOUD::Neo4j

Here we describe the architecture subcomponents regarding Neo4j:

- **Neo4j Drivers** client library, available in several programming languages (Java, Python, Javascript, and C#), for connecting to the Neo4j database through the **Bolt binary protocol**.
- **Neo4j Bolt Server** is responsible for accepting and managing client connections against the database and dispatching queries to the database and results back to the clients.
- **Cypher Planner/Optimizer** translates Cypher queries into optimized logical plans that satisfy the declarative requirements of the queries. Internally a cost-model and database statistics are used in order to estimate plans.
- **Cypher Runtime** is the layer implementing the various expression evaluators and operators on top of the Neo4j database needed to execute the logical plans chosen by the Cypher Planner/Optimizer.
- **Neo4j Kernel** is the core component in the Neo4j database responsible for low level I/O operations, e.g., efficiently storing/retrieving data to/from the disk, transaction management, indexing, checkpointing, entity locking.
- **Page Cache** provides a strategy for caching file blocks (called **Pages**) in memory with efficient syncing mechanism between memory and disk, and it is also responsible for managing read/write locks on such pages.

#### 4.11 ACTiCLOUD::Application

ACTiCLOUD does not require application modifications and will ensure transparent execution of applications in ACTiCLOUD-enabled systems. However, ACTiCLOUD provides the opportunity to the applications to communicate optionally with the ACTiManager through an interface (API). The applications can use this interface to inform the ACTiManager about the desired metrics of interests and demands; the ACTiManager will use this information to increase the utilization of the system while meeting the application's demands.



## 5 Embraced technologies

To proceed with the implementation of the ACTiCLOUD system based on the definition of the architecture as analyzed in the previous sections, we need to consolidate on specific technologies that will be part of the ACTiCLOUD's ecosystem. The main goal in embracing specific technologies is to maintain a good balance between (i) already existing components that are open-source, stable, and widely accepted, and (ii) components that will be able to deliver the advanced features and capabilities promised by ACTiCLOUD. In this way, we aim to increase the impact of our system by maximizing the effect of the new features and their compatibility with existing ecosystems. In particular, referring to the Figure 4.1 of the base ACTiCLOUD architecture, the following technologies will be embraced:

- **ACTiCLOUD::Hardware.** As analyzed in the previous paragraphs, ACTiCLOUD builds on top of the systems provided by its two hardware partners, i.e., NSCALE's platforms that are typical x86-based systems and KALEAO's platforms that are ARM-based systems. This will give us the opportunity to work on platforms that constitute the vast majority of hardware architectures of typical cloud datacenters, microservers, edge, and IoT devices.
- **ACTiCLOUD::Aggregation layer.** Aggregation of resources at the rack level is an advanced component provided by NSCALE and KALEAO platforms as described in Section 4.3.
- **ACTiCLOUD::Hypervisor.** The Hypervisor layer in the ACTiCLOUD architecture will be provided by OnApp's MicroVisor technology. MicroVisor is a lightweight virtualization technology that provides advanced capabilities for rack-level resource pooling and management. Where relevant, we may also experiment with alternative virtualization technologies (e.g., Linux/libvirt/KVM).
- **ACTiCLOUD::Cloud manager.** In this layer we will embrace OpenStack, the de facto open source cloud management system.
- **ACTiCLOUD::ACTiManager.** ACTiManager is the heart of the proposed architecture. It is a module that will be implemented from scratch and will be delivered as open source software.
- **ACTiCLOUD::Guest OS.** The Guest OS will not be changed by the ACTiCLOUD approach, as discussed in the previous section. In that sense, ACTiCLOUD-enabled systems will be able to operate with any type of Guest OS. However, our experimentation and any changes to the Guest OS system libraries (as described next) will be done on Linux-based systems.
- **ACTiCLOUD::Guest OS::System libraries.** Changes to the system libraries on the Guest OS, will be made on Linux system libraries (e.g., libc, libNUMA).
- **ACTiCLOUD::JVM.** ACTiCLOUD will enhance JVM technologies by working on the Maxine Virtual Machine, an open source virtual machine that is developed at UNIMAN. The emphasis in Maxine's architecture is on modular design and code reuse in the name of flexibility, configurability, and productivity for industrial and academic virtual machine researchers.
- **ACTiCLOUD::MonetDB.** MonetDB is a multi-threaded analytical database management system, primarily implemented in C. Linux is one of the primarily supported operating systems of MonetDB. MonetDB operates on any of the conventional Linux distributions, such as RedHat, Ubuntu, CentOS, and Fedora.

- **ACTiCLOUD::Neo4j.** Neo4j is a graph database which targets the JVM as runtime environment. For this reason Neo4j requires a JVM to run. Neo4j currently supports the following JVM implementations: OpenJdk, HotSpot(™) by Oracle, and IBM Jdk.
- **ACTiCLOUD:: (Database) Applications.** ACTiCLOUD is transparent to the technologies of the executing applications, either these are typical cloud applications or database applications. However, applications can inform the ACTiManager regarding their metrics of interest and expected demands from the system.

## 6 APIs

The technologies presented above affect to a large extent the APIs that will be used to enable communication and collaboration between the ACTiCLOUD components. ACTiCLOUD will try to minimize its intrusiveness on current technologies and will rely on existing APIs in order to maximize its compatibility with existing ecosystems. The exact APIs will be specified in the forthcoming period when the actual integration of the various components will take place. However, the following API families are foreseen to be applicable to the communication and collaboration of the ACTiCLOUD components.

### 6.1 Hardware API

The collection of monitoring information and hardware management will be based on standard APIs such as Intelligent Platform Management Interface (IPMI), more low-level monitoring facilities provided by the individual hardware components (e.g., performance counters monitoring provided by processor vendors), and platform-specific monitoring information (e.g., power and temperature sensors) provided by the hardware vendors, i.e., Numascale and Kaleao. These APIs will be utilized by the MicroVisor to gather this monitoring information and pass it to the ACTiManager via the OpenStack's telemetry mechanisms.

### 6.2 MicroVisor API

The MicroVisor offers a REST API for platform management, i.e., to gather information about the platform (e.g., alerts and statistics) and apply actions on the resources (e.g., create pools of resources, assign VMs on pools, etc.). The MicroVisor provides also a monitoring API for getting the current status of a particular MicroVisor, specifying the resource that is being probed and defining how frequently the monitoring should be carried out. The monitoring values are stored using the round robin database (RRD) format, and include statistics regarding CPU usage, network IO, block storage IO, and memory utilization, among others. The MicroVisor API will be utilized by OpenStack for collecting monitoring information and applying standard management operations, and by the ACTiManager for collecting advanced monitoring information system and applying intelligent optimization decisions.

### 6.3 OpenStack API

OpenStack's architecture is heavily affected by the notion of individual components and the extensive use of REST APIs for communication among them and with the external world (i.e., cloud operator, cloud-native application). For example, the Nova, Ceilometer, and Heat components of OpenStack provide individual APIs for managing VMs, collecting monitoring information, and orchestrating VM execution, respectively. ACTiCLOUD leverages these APIs and will extend them accordingly where necessary to enable the integration with the lower parts of the ACTiCLOUD architecture. Finally, these APIs will be utilized by the ACTiManager to collect monitoring information and apply intelligent optimization mechanisms.

### 6.4 ACTiManager APIs

ACTiManager is the most active ACTiCLOUD component as it assumes communication with the MicroVisor, the Cloud Manager (OpenStack), internally between its ACTiManager.Internal and ACTiManager.External components, and optionally with the Application (Figures 4.17, 4.18, and

4.19). More specifically, ACTiManager will utilize: (i) MicroVisor's REST API to collect information and request actions, (ii) OpenStack's RESTful API to collect information and request actions, (iii) a new API for internal ACTiManager communication, (iv) a new optional API to interact with cloud-native applications that need to apply their own custom resource management logic, and (v) a new optional API to collect application demands when the metric of interest cannot be inferred by simply using black-box techniques (i.e., CPU utilization, instructions per second, etc.). The former two were described previously, while the latter three are described next.

#### **6.4.1 ACTiManager's subcomponents API**

The architecture of the ACTiManager is based on the Internal component that focuses on the node level, and on the External component that focuses on the site level. Both the Internal and External components of the ACTiManager are further broken down into individual subcomponents, i.e., the Information Aggregator, the Modeler, and the Decision Maker, as described in Section 4.6. The communication among these subcomponents will be based on specific APIs, while the REST API principles among certain components will be applied if deemed necessary.

#### **6.4.2 Cloud-native application resource management to ACTiManager API**

This optional API will enable the cloudification of applications, i.e., applications that are aware of their execution in a cloud infrastructure and apply their own resource management to make the best out of such environment. The cloudification typically requires some application internal logic about getting the current use of the resources (i.e., collecting some monitoring statistics from the cloud), modeling of actions, and making decisions that will request an action from the cloud manager, e.g., to scale out or scale up.

In this section, we describe the necessary API in the context of cloudifying MonetDB and focusing on two individual scenarios that are of particular interest for the cloudification of MonetDB: (i) getting query results before a certain deadline, and (ii) getting query results within a monetary budget. This API provides MonetDB the means of communicating with the ACTiManager to collect information about resource usage and to request actions. Note that such an API would be useful (and can be extended) for other applications that want to apply their own resource management policies by communicating them to the cloud manager, too.

##### ***Query results within a specified time***

In this scenario, a database user, e.g., a data scientist, needs the results of a query before a specified time. MonetDB uses the ACTiManager's API to inquire information about the resources that are currently available to the user (e.g., number of physical CPUs, real memory usage, CPU and memory overcommitment), and estimates a worst case time cost for the execution of the query. In case the time estimation is within the specified deadline, MonetDB continues with the execution of the query. In case the time cost estimation is not within the specified deadline, MonetDB estimates the needed resources and the cost of allocating them, and then: it either requests these resources from the ACTiManager using this API (e.g., create, delete, start, pause, restart, resize a VM) and executes the query, or notifies the user that the query cannot be answered within the deadline.

##### ***Query results within a specified monetary budget***

In this scenario, the aforementioned database user needs the results of a query within a

monetary budget, assuming that the cloud pricing is affected by the resources that are requested and used (i.e., cores, memory, network, storage, etc.). Similar to the previous scenario, MonetDB uses the ACTiManager's API to inquire information about the resources that are currently available to the user (e.g., number of physical CPUs, real memory usage, CPU and memory overcommitment), and then estimates a worst case monetary cost for the execution of the query with the current resources. In case the monetary cost estimation is within the specified budget, MonetDB continues with the execution of the query. In case the monetary cost estimation is not within the budget constraints, MonetDB estimates the cost and the time for executing on less expensive resources, and then: it either requests these less expensive resources from the ACTiManager using this API (e.g., create, delete, start, pause, restart, resize a VM) and executes the query, or notifies the user that the query cannot be satisfied within the specified monetary budget.

#### **6.4.3 Application demands to ACTiManager API**

Sometimes the ACTiManager would want to monitor the performance of some applications (e.g., databases, web-servers) using more meaningful metrics about the real performance of the applications than simply looking into application-agnostic low-level performance monitoring events, such as CPU load, IPC, etc. Indeed, such applications hold and report many individual statistics about their performance, so that a user or a system administrator can configure them accordingly. These metrics can be for example: number of queries or transactions per second, average execution time per query, 95th percentile execution time per query, “cold” latency (how much time until the user starts getting “some” results), total latency (how much time until the user gets “all” results) and throughput (i.e., how much data is processed per time unit). This monitoring will occur by having the application itself posting to the ACTiManager periodically the metric of interest through an API call (e.g., `post_updated_metric()`), based on the internal statistics that the application holds. This API support by the ACTiManager and usage by the applications would require minimal modifications from the side of the applications towards their cloudification, and more importantly it would enable the ACTiManager to take much more informed and correct decisions about enforcing the requested policies without affecting the performance of the applications.

#### **6.5 MonetDB to Guest OS API**

MonetDB relies on standard POSIX APIs for its communication with the Guest OS, e.g., the pthread library and related multi-threading primitives, efficient implementation of memory mapped files for I/O of big data chunks, and malloc() for small data chunks (the size limit is configurable). No extensions to these APIs are anticipated within the frame of ACTiCLOUD.

#### **6.6 Neo4j to JVM API**

Neo4j relies on the JVM API. For some of the features relevant to the ACTiCLOUD requirements and architecture, Neo4j would require some new JVM APIs, such as listing all available NUMA cores, binding threads to specific NUMA cores, allocating memory on a specific NUMA location (also through Unsafe operations) and retrieving NUMA locality info associated to a running thread, e.g., on which NUMA core the thread is running.

## 7 Conclusions

In this deliverable, we first described business scenarios from the CSPs' point of view and the application use cases that we envision to support through the ACTiCLOUD project. We then described the base architecture of the ACTiCLOUD and dissected each main component into their relevant subcomponents. Finally, we outlined the technologies that we have embraced in the ACTiCLOUD implementation and provided a high-level overview of the APIs that will be used.

## 8 References

- [AWS] Amazon Web Services. <https://aws.amazon.com/ec2/>
- [BCH13] Barroso, L. A., Clidaras, J., & Hölzle, U. (2013). The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3), 1-154.
- [DCR15] “Data Center Rack Server Market by Data Center Type (Tier 1, Tier 2, Tier 3, Tier 4), by Services (Consulting, Installation and Support, Professional), & by End User (Mid-Size, Enterprises, Large) - Global Forecast to 2019”, Marketsandmarkets.com, Publishing Date: January 2015, Report Code: TC 2947.
- [DK13] C. Delimitrou and C. Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. *SIGPLAN Not.* 48, 4 (March 2013), 77-88.
- [DK14] Delimitrou, C. and Kozyrakis, C., 2014. Quasar: resource-efficient and QoS-aware cluster management. *ACM SIGPLAN Notices*, 49(4), pp.127-144.
- [FIT04] Fitzpatrick B.. 2004. Distributed caching with memcached. *Linux J.* 2004, 124 (August 2004), 5-.
- [GCP] Google Cloud Platform. <https://cloud.google.com/>
- [IGN12] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, Sjoerd Mullender, and Martin L Kersten. MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Eng. Bull.*, 35(1):40-45, 2012.
- [KAL] Kaleao - Innovative server platform architecture: <http://www.kaleao.com/>
- [LCG14] Lo, D., Cheng, L., Govindaraju, R., Barroso, L. A., & Kozyrakis, C. (2014, June). Towards energy proportionality for large-scale latency-critical workloads. In *ACM SIGARCH Computer Architecture News* (Vol. 42, No. 3, pp. 301-312). IEEE Press.
- [LCG15] Lo, D., Cheng, L., Govindaraju, R., Ranganathan, P., & Kozyrakis, C. (2015, June). Heracles: improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (pp. 450-462). ACM.
- [MIL13] Miller, J. J. (2013, March). Graph database applications and concepts with Neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA* (Vol. 2324).
- [MOS] MiniOS. <https://wiki.xenproject.org/wiki/Mini-OS>
- [MVM] Maxine Virtual Machine. [https://en.wikipedia.org/wiki/Maxine\\_Virtual\\_Machine](https://en.wikipedia.org/wiki/Maxine_Virtual_Machine)
- [NUM] Numascale Numaconnet: [https://www.numascale.com/numa\\_pdfs/numaconnect-white-paper.pdf](https://www.numascale.com/numa_pdfs/numaconnect-white-paper.pdf)
- [OAS] OnApp Integrated Storage: <https://docs.onapp.com/display/32AG/Integrated+Storage>
- [OPE] OpenStack: <https://www.openstack.org/>
- [RAC] Rackspace. <https://www.rackspace.com/cloud>
- [RAC15] X. Ragiadakou, M. Alvanos, J. Chesterfield, J. Thomson, M. Flouris. Microvisor: A Scalable Hypervisor Architecture for Microservers. In *EUROSERVER: Green Computing Node for European*

Micro-servers, HiPEAC 2015.

[RTG12] Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H. and Kozuch, M.A., 2012, October. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In Proceedings of the Third ACM Symposium on Cloud Computing (p. 7). ACM.

[WAZ] Windows Azure. <http://www.windowsazure.com/>