



ACTiCLOUD: ACTivating resource efficiency and large databases in the
CLOUD

Project No: 732366

H2020-ICT-2016-1

Technical Report on Distributed Cloud Resource Manager v1.0

18/12/2018

Executive summary: This technical report is based on ACTiCLOUD's Deliverable 2.2 that provides the initial version of ACTiCLOUD's cloud resource manager, named as ACTiManager.

List of authors:

| Authors | Affiliation |
|--|-------------|
| Georgios Goumas, Vasileios Karakostas, Konstantinos Nikas, Dimitrios Siakavaras, Stratos Psomadakis, Stefanos Gerangelos | ICCS |
| Ewnetu Bayuh Lakew, Petter Svärd, Simon Kollberg | UMU |

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



Confidentiality:

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

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

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

Table of Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 7 |
| 1.1 | Purpose of this document | 7 |
| 1.2 | Document structure | 7 |
| 2 | ACTiManager design and architecture | 8 |
| 2.1 | Relevance to ACTiCLOUD objectives, business scenarios and use cases | 8 |
| 2.2 | ACTiManager Design | 8 |
| 2.2.1 | Design principles | 8 |
| 2.2.2 | Overview | 10 |
| 2.2.3 | Detailed design and architecture | 11 |
| 3 | ACTiManager operation | 15 |
| 3.1 | Events | 15 |
| 3.2 | Actions | 16 |
| 3.3 | Models | 17 |
| 3.4 | The Lifecycle of a VM under ACTiManager | 18 |
| 4 | ACTiManager invocation scenarios | 20 |
| 4.1 | The VM creation scenario | 20 |
| 4.2 | The interference detection scenario | 20 |
| 4.3 | The under/overload and fragmentation detection scenarios | 21 |
| 4.4 | Imbalance detection scenario | 21 |
| 4.5 | Application under/over-performance | 21 |
| 5 | Module description | 22 |
| 5.1 | Overview | 22 |
| 5.2 | Internal and External sub-modules | 22 |
| 5.2.1 | Internal | 22 |
| 5.2.2 | External | 23 |
| 5.2.3 | Communication mechanism between Internal and External sub-modules | 24 |
| 5.2.4 | Communication support between ACTiManager.internal and hypervisor | 24 |
| 5.3 | Information Aggregator Component | 25 |
| 5.3.1 | Internal | 25 |
| 5.3.2 | External | 25 |
| 5.4 | Modeler Component | 25 |
| 5.4.1 | Internal | 25 |
| 5.4.2 | External | 27 |

| | | |
|----------|--------------------------------|-----------|
| 5.5 | Decision maker | 28 |
| 5.5.1 | Internal | 28 |
| 5.5.2 | External | 28 |
| 5.6 | Interaction with OpenStack | 29 |
| 6 | Summary and Future Work | 30 |

Figures

Figure 2.1: “ACTiManager” follows a hierarchical approach, distinguishing the management of resources at node level (ACTiManager.internal) and at site level (ACTiManager.external). 11

Figure 2.2: General architecture and operation of the “ACTiManager.internal” component that manages resources within a single node..... 12

Figure 2.3: General architecture and operation of the “ACTiManager.external” component that manages resources of a cloud site and across sites..... 13

Figure 2.4: Interaction between ACTiManager and OpenStack..... 14

Figure 3.1: The lifecycle of a VM under ACTiManager. The blue boxes represent the different stats that a VM can be in, while the white boxes denote the models that are used when transitioning from the current state (denoted by the dotted line) to the next state..... 19

List of Abbreviations and Acronyms

| Abbreviation / Acronym | Meaning |
|------------------------|--|
| AMQP | Advanced Message Queuing Protocol |
| CPU | Central Processing Unit |
| CSPs | Cloud Service Providers |
| DP | Decision Period |
| IOPS | Input/Output Operations per second |
| IPC | Instructions per Cycle |
| KVM | Kernel-based Virtual Machine |
| LLC | Last level Cache |
| MQTT | Message Queuing Telemetry Transport |
| pCPU | Physical CPU |
| QoS | Quality of Service |
| SLA | Service Level Agreement |
| STOMP | Streaming Text Oriented Messaging Protocol |
| vCPU | Virtual CPU |
| VM | Virtual Machine |

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

This document describes the design and the basic components for resource management and workload characterization of *ACTiManager*^{1 2}. ACTiManager is an advanced resource manager that will support the core goals of ACTiCLOUD towards resource-efficient IaaS cloud offerings. This version 1.0 implements the core functionality of ACTiManager.

1.2 Document structure

The document is organized as follows: Section 2 describes the design and the architecture of ACTiManager. Section 3 discusses in more detail the operation model of ACTiManager. Section 4 presents the cases and scenarios under which ACTiManager intervenes to perform efficient resource management. Section 5 describes in detail the current version 1.0 of ACTiManager. Finally, Section 6 concludes this document and provides directions for future work.

¹ “D1.2: “ACTiCLOUD architecture”, (<https://acticloud.eu/dissemination/deliverables>)

² Vasileios Karakostas, Georgios Goumas, Ewnetu Bayuh Lakew, Erik Elmroth, Stefanos Gerangelos, Simon Kolberg, Konstantinos Nikas, Stratos Psomadakis, Dimitrios Siakavaras, Petter Svard, Nectarios Koziris. “Efficient Resource Management for Data Centers: The ACTiCLOUD Approach”. In 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2018).

2 ACTiManager design and architecture

2.1 Relevance to ACTiCLOUD objectives, business scenarios and use cases

ACTiManager is the most critical component in the ACTiCLOUD architecture as it plays a central role in the realization of ACTiCLOUD's objectives towards next generation IaaS platforms. ACTiManager enhances state-of-the-art cloud management approaches with capabilities to:

- perform more effective resource allocation, increasing system throughput (*Strategic Objective S01.1 on resource efficiency*),
- provide applications with more strict performance guarantees (*Strategic Objective S01.2 on performance stability*).

In addition, as part of the ACTiCLOUD architecture, ACTiManager supports the delivery of scale-up and scale-out resources offered by the underlying hardware and hypervisor layers (*Strategic Objective S02.1 on scalability in resource provisioning*) and interacts with applications to respond to specific-tailored, dynamic resource requests (*Strategic Objective S02.2 on elasticity in resource provisioning*).

Moreover, ACTiManager is designed to support directly ACTiCLOUD's business scenarios [D1.2], also summarized below:

- Business scenario 1: Effective consolidation for increased revenue and reduced TCO,
- Business scenario 2: Workload prioritization,
- Business scenario 3: Hosting larger workloads,
- Business scenario 4: Collaboration with sibling cloud sites,
- Business scenario 5: Enhanced dependability and availability.

Finally, the direct focus of ACTiManager is to support "Use case 1: Execution of typical cloud applications". In addition, as the project evolves, ACTiManager is expected to play an important role on enabling the Use Cases 2-7 that are more relevant to database-driven applications [D1.2].

2.2 ACTiManager Design

2.2.1 Design principles

ACTiManager is designed with the following principles in mind:

1. To operate in the typical closed-loop control fashion based on: (a) monitoring and information aggregation, (b) information processing and modeling, and (c) decision making and actuation, leading respectively to the three core components of the module:
 - the *Information Aggregator*,
 - the *Modeler*, and
 - the *Decision Maker*.

2. To be scalable in large-scale cloud installations, operating at various levels, including node level, cloud site level, and inter-site. For this reason, ACTiManager follows a modular hierarchical design (see Figure 2.1), split into two sub-modules:
 - *ACTiManager.internal* whose goal is to manage resources at a fine-grain level within the node (e.g., mapping of virtual CPUs on physical CPUs, allocation of memory, etc.), and
 - *ACTiManager.external* whose goal is to manage resources within and across cloud sites, enforcing high-level policies for placement, load balance, efficient consolidation, energy efficiency, etc.
3. To minimize the modifications to an existing cloud management system and to operate as an “out-of-the-box” add-on component that can be plugged-in and out of an existing installation at will. Towards this direction, we put significant effort in interfacing with existing, well established components of the core cloud manager (OpenStack³ in our case), and minimizing the necessary changes to the core cloud manager itself for integration with ACTiManager.
4. To support the relevant ACTiCLOUD’s strategic objectives, business scenarios, and use cases:
 - ACTiManager relies on online characterization of the VMs (or applications) and incorporates in their feature list characteristics that describe a VM’s potential to suffer from or create interference (i.e., a “noisy” or a “sensitive” VM regarding the use of resources, respectively) by the co-location of another VM within the same executing node.
 - A “noisy” VM uses a significant part of some shared resources that might affect the performance of another VM that is co-located with. An example of “noisy” application for the shared last level cache (LLC) is a streaming application whose performance is not improved by having more cache; but, because it uses as much cache as it has access to, it might affect the performance of other applications that share that cache with.
 - A “sensitive” VM is a VM whose performance depends on the resources that are assigned to it. An example of “sensitive” application regarding the shared LLC is a cache-friendly application, whose performance improves as it gets more cache. Co-locating “noisy” with “sensitive” applications can severely affect the performance of the sensitive application due to interference in shared resources.
 - To dynamically identify the characteristics of the VMs, ACTiManager logically splits the site infrastructure into (i) a number of nodes operating as “laboratory” nodes, and (ii) the rest (vast majority) of the site nodes as “production” nodes⁴.

³ <https://www.openstack.org/>

⁴ Dejan Novaković, Nedeljko Vasić, Stanko Novaković, Dejan Kostić, and Ricardo Bianchini. 2013. DeepDive: transparently identifying and managing performance interference in virtualized environments. In Proceedings of the 2013 USENIX conference on Annual Technical Conference (USENIX ATC'13).

More specifically, ACTiManager uses the "laboratory" node(s) for quickly characterizing the VMs and extracting the aforementioned information (i.e., "noisy" or "sensitive" behavior) in an execution environment that is free from any source of interference. The rest of the compute nodes, i.e., the vast majority of the resources, are treated as usual compute or "production" nodes. More details about the lifecycle of a VM are provided in Section 3.

- ACTiManager assumes that the cloud site administration may distinguish between high-priority, latency-critical VMs (i.e., "gold" instances) and low-priority, batch VMs (i.e., "silver" instances), potentially with a different billing policy. The reader may refer to [D1.2] (page 15) for a more detailed analysis of Business Scenario 2.
- ACTiManager assumes that some applications may incorporate special functionality to expose their desired metric of interest and achieved Quality of Service (QoS). If this functionality is provided by the application, ACTiManager is able to utilize it to act towards maintaining the desired QoS of applications.

2.2.2 Overview

The design of ACTiManager has been described in [D1.2]. To make this document self-contained, we repeat some information below. The reader can skip to Section 3 if already familiar with the ACTiManager design and architecture.

As shown in Figure 2.1 and described above, a specific ACTiManager instance takes care of 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 ([D1.2], Figure 4.2), while a node in the KMAX platform is an Exynos server ([D1.2], Figure 4.3), both managed by one instance of the hypervisor (MicroVisor in our case).

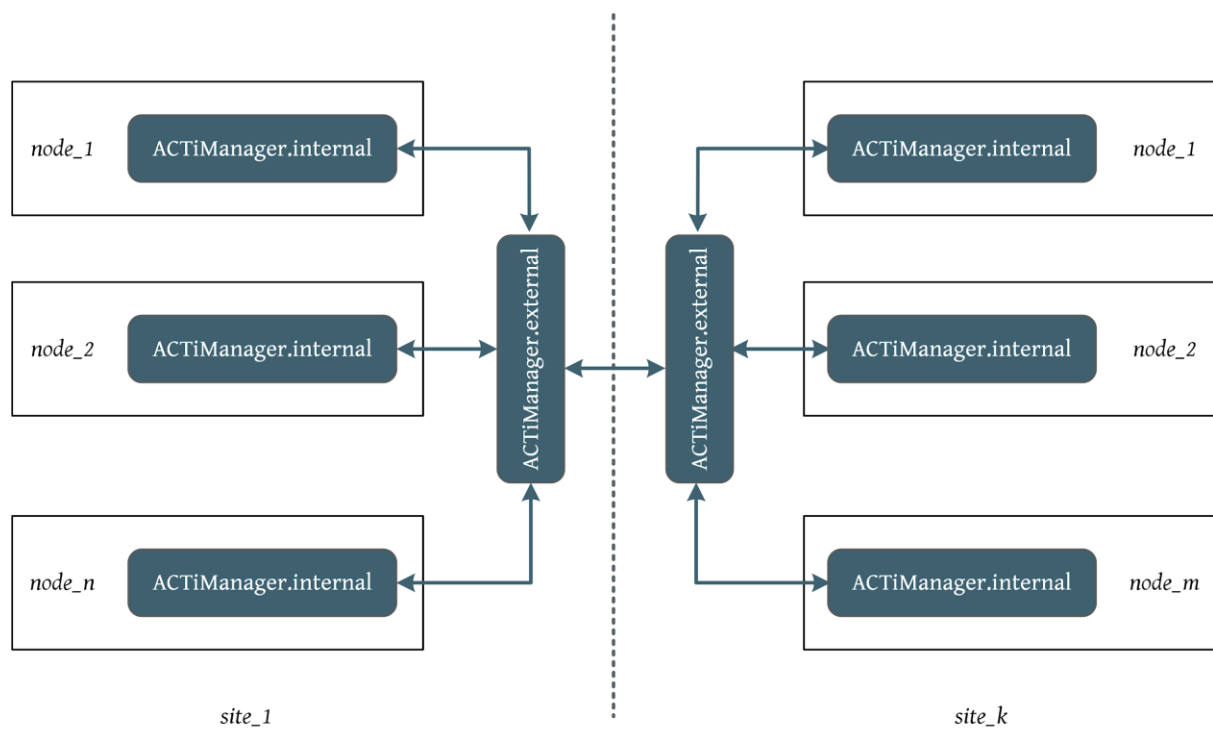


Figure 2.1: “ACTiManager” follows a hierarchical approach, distinguishing the management of resources at node level (ACTiManager.internal) and at site level (ACTiManager.external).

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

- 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 characterize the execution of the applications as “noisy” or “sensitive” regarding the use of resources, (ii) to detect anomalies like interference, imbalance, overload, underload etc., (iii) to predict the impact of various actions (e.g., consolidation effect on the already executing application, migration time, and failure probability), and (iv) to characterize applications in terms of their resource footprint and co-execution behavior on a multi-tenant system.
- 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 core cloud manager.

2.2.3 Detailed design and architecture

ACTiCLOUD::ACTiManager.internal

ACTiManager.internal is responsible for managing resources within a single node. To this end, it

collects information about the resource utilization of the running VMs within the node, performs sanity checks (e.g., detects interference, local overload/underload, underperformance of VMs/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 sub-module. The general architecture and operation of the ACTiManager.internal sub-module is shown in Figure 2.2.

The **Information Aggregator** collects information from three major sources:

- The *Telemetry*⁵ source that collects all the monitoring information that the standard agent of the Cloud Manager provides. In the context of OpenStack, the Telemetry subcomponent can refer to Ceilometer⁶.
- 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 (IPC), 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.

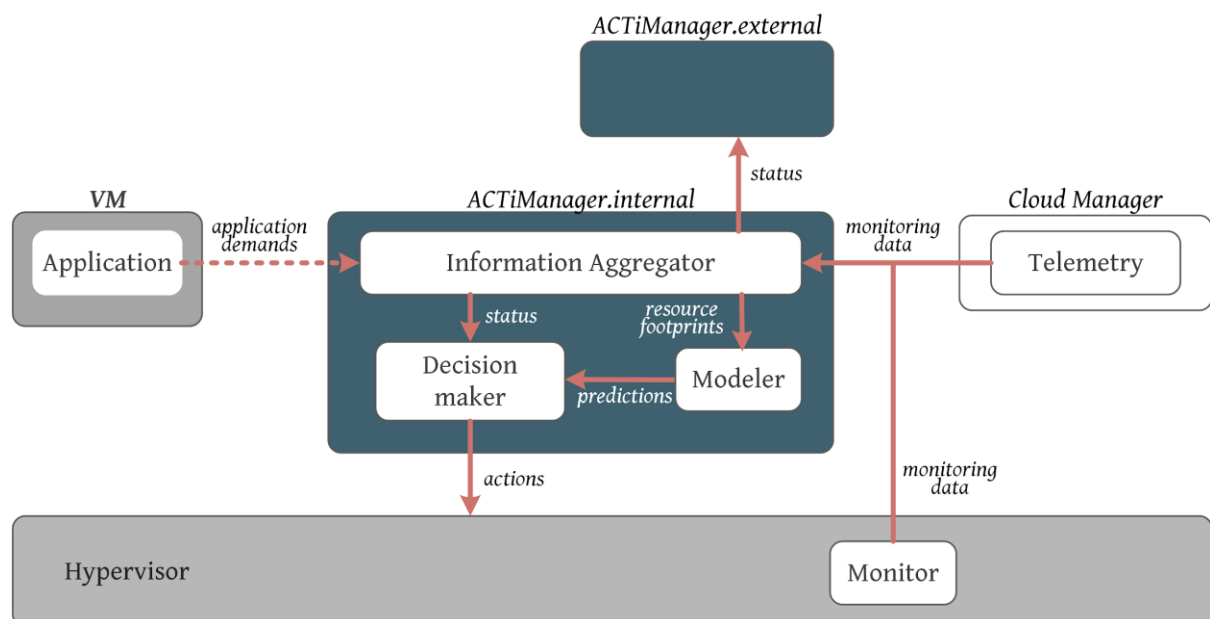


Figure 2.2: General architecture and operation of the “ACTiManager.internal” component that manages resources within a single node.

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:

⁵ <https://wiki.openstack.org/wiki/Telemetry>

⁶ <https://docs.openstack.org/ceilometer/latest/>

- 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 a co-location action (i.e., putting different VMs running close to each other), of a moving action, or of a resizing action.

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

ACTiCLOUD::ACTiManager.external

The operation and general architecture of the ACTiManager for the site level is shown in Figure 2.3. 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 placement and migration actions in order to cope with problems like load imbalance between nodes, or site over/under-utilization. The Decision Maker then decides on more profitable workload allocations and requests the relevant actions from the core cloud manager.

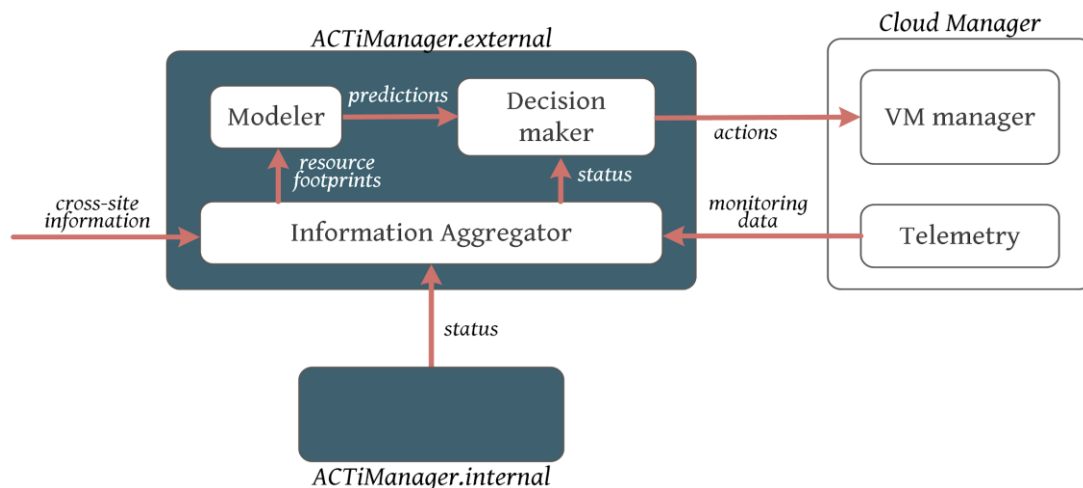


Figure 2.3: General architecture and operation of the “ACTiManager.external” component that manages resources of a cloud site and across sites.

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, prioritization, and resource allocation optimization are missing. This is the part where the ACTiManager comes into play to interact with OpenStack and to provide additional functionality.

The core interaction between ACTiManager and OpenStack is shown in Figure 2.4. 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 migration, while the Decision Maker at the node level requests from the hypervisor to take actions on VM co-scheduling and placement, i.e., re-mapping virtual CPUs to physical CPUs or moving memory. Besides, additional features such as the possibility of creating and/or adding a VM into multiple instance groups (e.g., an application can belong to either a "gold" or "silver" prioritization group based on the agreed SLA and billing policy, as well as to "noisy" or "sensitive" characterization group regarding the use of shared resources, based on its runtime behavior) are added inside OpenStack to augment both initial and continuous placement decisions. In general, ACTiManager.external should be viewed as a module that improves OpenStack by either introducing entirely new components, by adding new features to existing components of OpenStack, and by using the APIs of OpenStack.

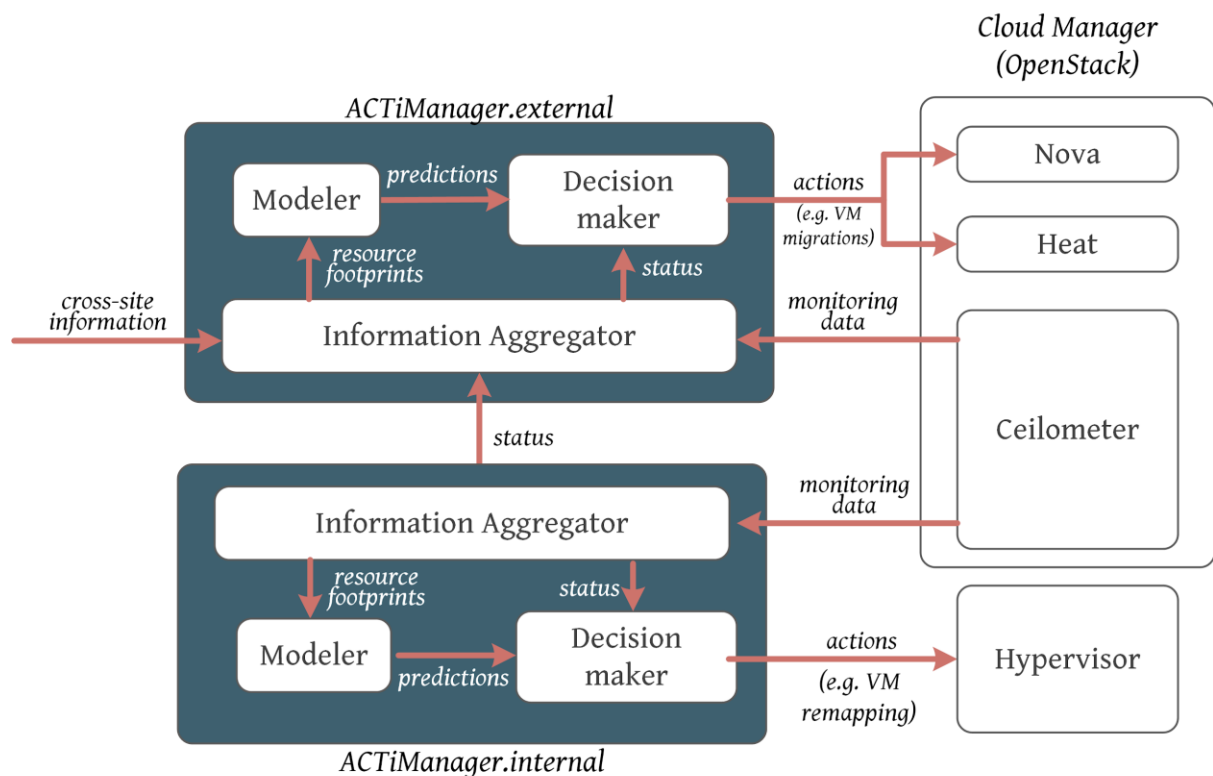


Figure 2.4: Interaction between ACTiManager and OpenStack.

3 ACTiManager operation

ACTiManager is invoked periodically in specific Decision Periods (DP) that can be different for the internal and external sub-modules. In the current version of ACTiManager, Decision Periods for both internal and external sub-modules are set in the order of a few minutes, as ACTiManager needs to track longer-term behavior and gather monitoring data over sufficiently large periods to avoid temporary utilization spikes. Analysis of the impact of this parameter is left for a later stage towards final fine-tuning and optimization of the system. At the end of each Decision Period, ACTiManager analyzes the status of the system, decides on any identified event based on models, and performs any required actions. Next we describe these events, actions, and models that define the operation of ACTiManager. Note that we may extend these lists during the course of the project.

3.1 Events

Events that trigger the decision making and actuation logic in the internal and external components of ACTiManager include:

- ACTiManager.internal events:
 - **Node's total execution load changed:** A VM has finished its execution or a new VM has been placed in that node.
 - **Node under/over-utilization detected:** Node resources are under/over-utilized, e.g., a low or high percentage of the node's processing cores or memory are used.
 - **Interference detected:** One or more of the VMs that execute in the node, experience performance degradation due to interference in shared resources (e.g., cores, caches, memory, network, storage).
- ACTiManager.external events:
 - **Creation of a new VM or new characterization period for an already running VM due to behavior change:** A newly created VM is placed in the "laboratory" node for characterization in order to extract its "fingerprint" regarding its performance and behavior, i.e., CPU and memory utilization, IPC, IOPS, cache misses, and to classify it as a "noisy" or "sensitive" VM regarding its use of resources. In addition, a VM is placed in the "laboratory" node when ACTiManager observes changes in its behavior and/or performance. Alternatively, the cloud user can directly provide this fingerprint information to ACTiManager (similarly to as providing the application's metric of interest); in that case, the newly created VM is managed directly by the "placement of a VM" event.
 - **Placement of a VM (end of characterization period):** A VM that was in a "laboratory" node until now, has just been characterized and needs to be placed in the production system. In case the cloud user provides directly the VM's fingerprint information, the same event takes place just when that VM is created.

- **Site under/over-utilization:** Site resources are under/over-utilized (e.g., a high percentage of the site's nodes are under/over-utilized in terms of cores or memory).
- **Site imbalance:** Some site resources are heavily utilized, while others are not (e.g., the difference of workload between the node with the heaviest workload compared to the node with lightest workload exceeds a threshold).
- **Node(s) interference:** One or more nodes report interference problems that cannot be mitigated at the node level by the ACTiManager.internal.
- **Notification from remote sites:** One or more of the sibling remote sites report under/over-utilization.
- **Application under/over-performance:** Application manager (owner) reports the performance of the application and indicates the need for changing the configuration of the allocated VMs, e.g., by changing the resources of the VM in terms of cores or memory (scale-up/down), or by changing the number of VMs (scale-out/in).

3.2 Actions

On the occurrence of the aforementioned events, ACTiManager is able to select from a palette of actions that include the following actions:

- **ACTiManager.internal actions:**
 - **VM (re)mapping:** A VM's virtual CPUs and memory are (re)mapped within the node. This responds to map a newcomer VM or to remap a VM that suffers from or causes interference to a more isolated place in the node (e.g., in a different CPU socket or NUMA node).
 - **Capping:** ACTiManager.internal changes the number of resources (e.g., CPU utilization) that are currently assigned to a VM, as a response to application's resource utilization. This should be done within the maximum allowable range in order to reduce energy footprint.
 - **Report interference:** ACTiManager.internal turns to the external component for interference resolution, in case internal actions (through remapping) are not effective.
 - **Report under/over-utilization:** ACTiManager.internal reports to the external component whether the node experiences under/over-utilization of resources.
- **ACTiManager.external actions:**
 - **VM migration:** One or more VMs are migrated as a response to: (i) end of Characterization Period (see below), (ii) node interference, (iii) underutilization (evacuation of node and shutting down),
 - **VM control (pause.restart):** A VM (e.g., a silver one) is temporarily frozen as a response to interference or over-utilization. For example, assuming that a gold and a silver VM are running in a single compute node, ACTiManager.external may

pause the silver VM, in order to allow the gold VM to meet its QoS level. Later, ACTiManager.external may decide to restart the silver VM.

- **VM termination:** A VM is terminated/destroyed.
- **Notification to remote sites:** ACTiManager reports to sibling site over-utilization (request for assistance) or underutilization (advertisement of availability).
- **VM Scaling:** ACTiManager.external changes the number (scale-out/in) and the size (scale-up/down) of VMs as a response to application under or over-performance.

3.3 Models

Deciding on which action to select under the occurrence of one or more events is the responsibility of the Decision maker. Decision making and event invocation are supported by the Modeling subcomponent. A list of the models is presented below:

- ACTiManager.internal *models*:
 - **Characterization model:** This model extracts a VM's fingerprint regarding its use of resources, e.g., cores, memory, etc., and classifies its execution to either "noisy" or "sensitive". This model is the key ingredient of the characterization phase that occurs while the VM is executed in the "laboratory" node.
 - **Interference model:** This model classifies the execution of a VM to either "interference suffering" or "interference free" when co-located with other VMs.
 - **(Re)mapping cost model:** This model predicts the performance of a VM for a certain mapping of its resources in the node, taking into account the characteristics of the VMs that currently run in that node. It also predicts the time required to remap a VM (i.e., its cores, memory, both) within the node, and predicts the new performance after the remapping action.
 - **Node under/over-utilization model:** This model detects when a node is under/over-utilized.
- ACTiManager.external *models*:
 - **VM placement model:** The model that decides in which node to place a new VM, taking into account the characteristics of the currently running VMs in all or some of the site's nodes.
 - **Site under/over-utilization model:** The model that detects when a site is under/over-utilized.
 - **Site imbalance model:** The model that detects when a site is unbalanced regarding the utilization of its nodes.
 - **Migration cost model:** The model that predicts the time, bandwidth requirements, and failure probability to migrate a VM.

3.4 The Lifecycle of a VM under ACTiManager

Figure 3.1 summarizes the lifecycle of a VM within an ACTiManager-enabled system. ACTiManager heavily relies on online monitoring and analysis of a VM's "health" status.

To be able to detect any anomalies and take corrective actions, the system needs to have a solid understanding of VMs' execution characteristics, involving their performance, resource demands, required QoS levels, potential to suffer from or create interference, etc.

To accomplish this, every newly spawned VM starts its execution in the "laboratory" node. During its execution in the "laboratory" node, ACTiManager collects monitoring information and, based on the characterization model, it creates the fingerprint of the VM, e.g., "noisy" or "sensitive" regarding resource utilization. The VM is executed and analyzed in the "laboratory" node for a specific *Characterization period*.

Upon the end of this Characterization period, ACTiManager decides the node in which the VM will continue its execution in the production environment, by checking if the resources that the VM requires are available in any node (number of cores, memory, etc.) taking into account the priorities (gold or silver) and the characteristics (fingerprint, noisy, sensitive) of the rest of the VMs/applications that run currently in the site's nodes.

ACTiManager.internal and ACTiManager.external wake up in specific *decision periods* (that can be different for the two components) and check for various events, such as change in execution load, interference detection, and under/over-utilization, among others. In case none of the above events occur, ACTiManager keeps the current configuration and waits for the next decision period.

In case any of the above events occurs, ACTiManager decides how to resolve that issue based on the various models and by applying actions at node or site level.

Finally, in case the behavior of a VM changes significantly (e.g., change in fingerprint or performance), ACTiManager may migrate that VM to the "laboratory" node to perform another characterization phase and extract a new fingerprint.

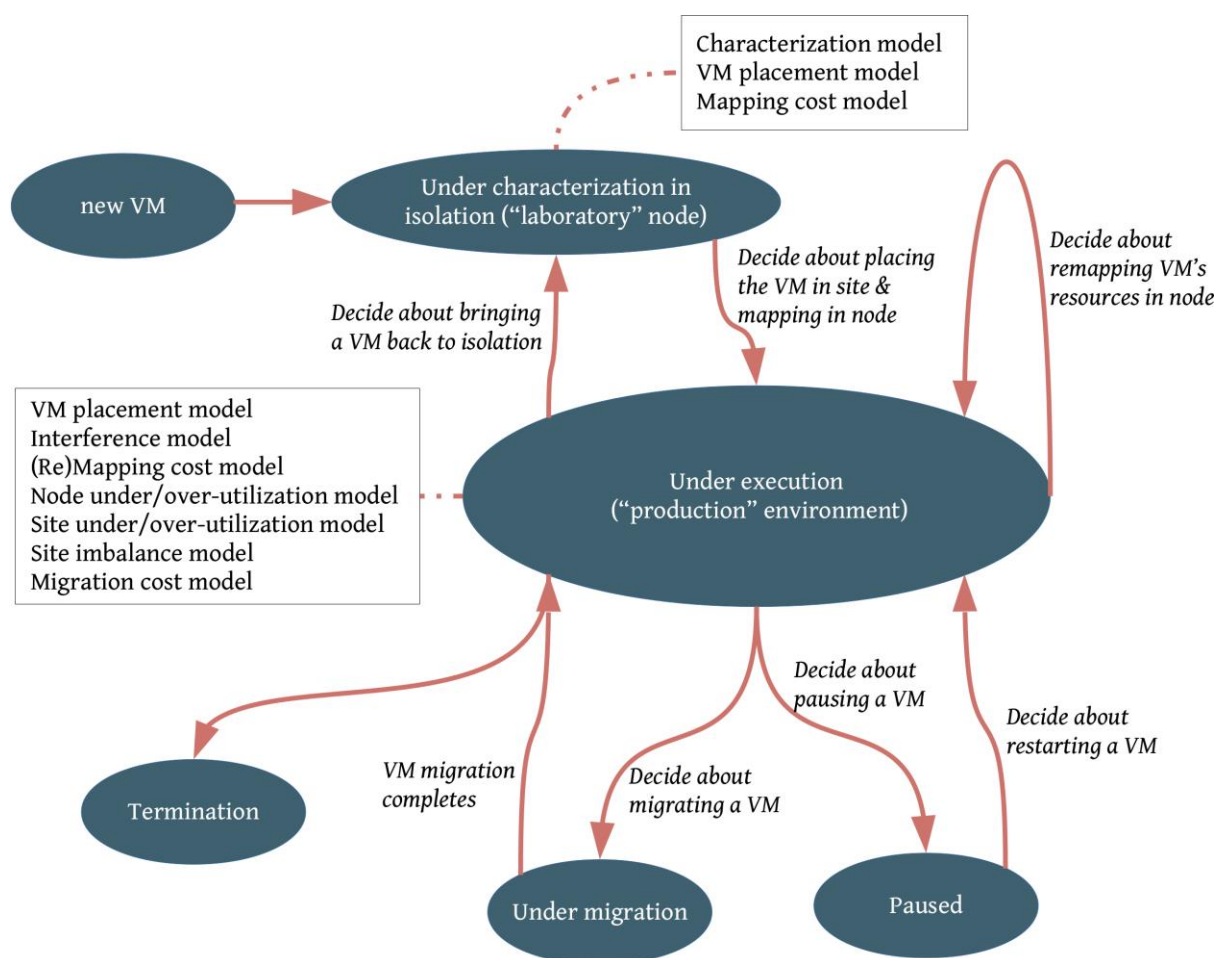


Figure 3.1: The lifecycle of a VM under ACTiManager. The blue boxes represent the different stats that a VM can be in, while the white boxes denote the models that are used when transitioning from the current state (denoted by the dotted line) to the next state.

4 ACTiManager invocation scenarios

In this section we describe in more detail the various invocation scenarios of ACTiManager. The basic functionality for supporting these scenarios has been implemented as part of ACTiManager v1.0. The next release of ACTiManager will further extend this functionality with more advanced capabilities and optimizations. All these scenarios, except for the VM creation scenario, are initiated after a Decision Period quantum finishes; at that point, ACTiManager checks for any of the following scenarios and applies actions to mitigate them. In the VM creation scenario, ACTiManager is invoked every time a VM is created (and destroyed).

4.1 The VM creation scenario

ACTiManager is invoked every time a VM is created. ACTiManager places the newly created VM in the "laboratory" node for characterization and for extracting its fingerprint information. Once the characterization period for that VM finishes, ACTiManager decides where to place it in the production nodes of the site, taking into account the required resources of the VM and the characteristics of the rest of the VMs in the nodes, in order to avoid interference and respect the SLA of the high-priority gold VMs. Note that ACTiManager may: (i) skip the characterization step when the cloud user provides the fingerprint information, and (ii) bring a long-running VM back to the "laboratory" node for characterization, when the fingerprint and/or performance of that VM exhibits unpredictable behavior.

4.2 The interference detection scenario

ACTiManager detects any performance interference cases, after a Decision Period quantum expires. The identification of performance interference is part of the ACTiManager.internal's responsibilities. ACTiManager.internal identifies interference based on the fingerprint of the VM and, if provided, based on the application's metric of interest. Once ACTiManager.internal identifies performance interference, it has to understand which shared resources cause the performance degradation and which VM is responsible for. Based on this information, ACTiManager.internal first tries to resolve interference locally at node level, by taking local actions, such as VM remapping or capping. In case the actions of ACTiManager.internal are not sufficient, ACTiManager.internal notifies the ACTiManager.external about the interference issue, to apply mitigation actions at site level. Indeed, ACTiManager.external can decide to migrate to a different compute node either the "sensitive" VM that suffers from interference or the "noisy" VM that causes interference. Another available action is that of pausing; assuming the scenario of having a gold and a silver VM running in the same node and the gold VM experiencing performance degradation due to interference, ACTiManager.external can decide to pause the execution of the silver VM (which is typically assumed as a VM that runs batch applications) and restart it again later, in order to improve the performance of the gold VM and meet its QoS level for as long as it is deemed necessary. Another option is to re-characterize the "noisy" or the "sensitive" VM in order to better understand whether this behavior occurs due to some phase change in its execution.

4.3 The under/overload and fragmentation detection scenarios

ACTiManager detects system underload, overload, or fragmentation, after a Decision Period quantum expires. These events entail different courses of actions. ACTiManager collects statistics about the execution of the VMs in the nodes and the node utilization, and uses threshold-based algorithms to detect underload and overload. The thresholds are periodically checked and events are generated when they are violated so that the Decision Maker can take corrective actions. For example, when the system is determined underloaded, that is an indicator to either admit more VMs to maximize revenue, or to consolidate the VMs to minimize energy cost. On the other hand, detecting system overload is an indicator for VM migration, pause, or termination to satisfy demands of high priority applications.

Fragmentation detection is also performed by checking the allocated resource distribution for each VM. The decision maker tries to compose resources for a VM from a nearby node as much as possible, in NUMA architectures, as is the case with the Numascale platform. This resource distribution is periodically adjusted depending on the behavior of neighbor VMs.

4.4 Imbalance detection scenario

ACTiManager is invoked periodically in specific *Decision Periods (DP)* that can be different for the internal and external components, to check for load imbalance among the site's nodes. For every Decision Period, ACTiManager.internal checks the utilization of the resources at node level, and in case of under-/over-utilization, it reports that event to the ACTiManager.external. In this way, ACTiManager.external has complete view of all nodes' utilization, and can identify whether load imbalance among the site's compute nodes exists. In case load imbalance is detected, ACTiManager.external can perform VM migration actions based on the available models to balance the work among the site's compute nodes, while taking into consideration the necessity of avoiding performance interference. In addition, load imbalance can be detected when a newly created VM has been characterized and is about to be placed in the production system, or when a VM is destroyed.

4.5 Application under/over-performance

While under or over-performance of application is determined and managed by the cloud user or application owner, the responsibility of ACTiManager lies in providing APIs that would help the cloud user to realize his/her desires. For example, the cloud user may want to add or reduce the number of instances for an application; it is the responsibility of the ACTiManager to handle such actions transparently. Once the ACTiManager receives the request, it makes decisions about where to place the additional VM(s) if the request is for additional VM(s) or which VM(s) to terminate if the request is for VM(s) removal.

5 Module description

In this section we describe in detail the software components of ACTiManager that have been implemented in this deliverable (ACTiManager v1.0).

Our initial implementation and first release of ACTiManager focused on supporting the entire lifecycle of a VM, as described in Section 3.4. Towards that goal, we created and placed accordingly all the functional components that are required to support the invocation scenarios of ACTiManager, as described in Section 4, and integrated them with OpenStack.

We have also performed extensive work regarding the models that support the decision making logic of ACTiManager, such as the remapping cost model, the node under/over-utilization model, the VM placement model, and the VM migration cost model, that are described in Section 3.3. In the current version of ACTiManager, we have used simple threshold-based models in order to focus more on the software architecture and the functionality of the ACTiManager. However, we briefly discuss our work in progress regarding the more sophisticated parts of the models that will be included in the next release.

Finally, our first implementation of ACTiManager assumes commodity, off-the-shelf, hardware platforms (Intel/AMD) and hypervisor technology (QEMU/KVM⁷), in order to avoid any dependencies that could arise during the integration of the MicroVisor (the hypervisor technology that is provided by ONAPP) with the ACTiCLOUD's hardware platforms, i.e., the Numascale and KMAX platforms (that are provided by NSCALE and KALEAO, respectively). Hence, our initial implementation is independent of the underlying hardware, it only depends on the hypervisor, and its integration with OpenStack, and consequently can be used on the Numascale and KMAX platforms with KVM as hypervisor. Our next step for integrating with ONAPP's MicroVisor requires the completion of: (i) porting the MicroVisor on the two hardware platforms, and (ii) integrating the MicroVisor with the Pike version of OpenStack.

5.1 Overview

Our initial version of ACTiManager involves the implementation of the basic components of its two sub-modules: (a) ACTiManager.internal and (b) ACTiManager.external. ACTiManager is primarily developed as a separate component that utilizes and complements an OpenStack installation and its core components. The current version of ACTiManager is based on the Pike⁸ version of OpenStack. As OpenStack is implemented in Python, we decided to implement ACTiManager in Python, too. In addition, both ACTiManager.internal and ACTiManager.external sub-modules consist of three separate components that are responsible for different tasks at the two levels: (i) information aggregator, (ii) modeler, and (iii) decision maker.

5.2 Internal and External sub-modules

5.2.1 Internal

The initial version of the ACTiManager.internal sub-module has been implemented as a daemon service. It executes in a continuous loop in which it sleeps for a predefined time, named as

⁷ https://www.linux-kvm.org/page/Main_Page

⁸ <https://www.openstack.org/software/pike/>

Decision Period (DP). Then, it wakes up and evaluates the current state of the node and takes the appropriate decision at node level. More specifically, it gathers monitoring information through its Information Aggregator component and checks via its Modeler component for the existence of the following conditions:

- whether total execution workload has changed compared to the previous Decision Period,
- whether node under/over-utilization exists,
- whether performance interference exists.

Based on these conditions, ACTiManager.internal proceeds to the corresponding decision. The main actions that the internal sub-module is capable of applying in this version of ACTiManager are:

- to (re)map a VM's virtual CPUs and memory within the cores and the NUMA nodes of the node,
- to report interference to the external sub-module via the communication mechanism that is described in Section 5.2.3,
- to report node under/over-utilization to the external sub-module via the communication mechanism.

5.2.2 External

The initial version of the ACTiManager.external sub-module has been implemented as a daemon service. It executes in a continuous loop in which it sleeps for a predefined time, named as Decision Period (DP). Then, it wakes up and evaluates the current state of the site, taking into consideration the state of all individual nodes, to apply the appropriate decision at site level. More specifically, it gathers monitoring information through the Information Aggregator component (and the internal sub-module) and checks via the Modeler component for the existence of the following conditions:

- whether site imbalance exists,
- whether site under/over-utilization occurs,
- whether there are any paused VMs that could be restarted in the site,
- whether the characterization period for those VM(s), if any, that are in the "laboratory" node(s) has been completed, and
- whether there are any messages in the communication mechanism between the external and internal sub-modules for interference detection (the internal sub-module notifies the external).

Based on these conditions, ACTiManager.external proceeds to the corresponding decision. The main actions that the external sub-module is capable of applying in this version of ACTiManager are:

- to migrate one or more VMs from one node to another, and

- to pause one or more VMs.

5.2.3 Communication mechanism between Internal and External sub-modules

The internal and external sub-modules need to exchange information. For example, ACTiManager.internal needs to post important events that it cannot handle itself, to the external sub-module, which in turn proceeds to applying the corresponding high-level action. This communication occurs via the mechanism that has been implemented using RabbitMQ⁹. RabbitMQ is an open source message broker software that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT), and other protocols. The events of communication between the internal and external sub-modules are:

- periodic update of node monitoring information,
- detection of node under/over-utilization, and
- detection of interference between VMs in the same node.

5.2.4 Communication support between ACTiManager.internal and hypervisor

The internal sub-module communicates with the node's hypervisor using a wrapper library, which provides a hypervisor-agnostic API. This library essentially provides an interface, which will then be implemented by hypervisor-specific components. This library augments the Modeler and the Decision Maker, via the Information Aggregator, with detailed performance statistics, and enables the Decision Maker to apply the (re)mapping and capping actions.

In the current version of ACTiManager, we have implemented a "libvirt-specific helper library", that provides primarily KVM-specific operations. As libvirt supports also the Xen hypervisor, this library can be extended to support Xen as well. The libvirt helper library provides the internal sub-module with the ability to apply actions at node level and gather system-level topology and monitoring information that is currently unavailable on Openstack. More specifically, the helper library provides the ability to:

- query and change dynamically the mapping of virtual CPUs (vCPUs) to physical CPUs (pCPUs),
- query the node's NUMA information and topology,
- tune the NUMA memory options for the VMs,
- get fine-grained node and VM statistics (CPU and memory statistics, measurements using performance counters, etc.).

Note that those operations that are already supported by OpenStack are omitted from these components, such as VM (live) migration, coarse-grained monitoring information and statistics about the VMs and the node, etc.

⁹ <https://www.rabbitmq.com/>

5.3 Information Aggregator Component

5.3.1 Internal

The Information aggregator component of the internal sub-module is responsible for gathering monitoring information at the node level. More specifically, in the current version, it retrieves information metrics about:

- the topology of the node, e.g., number of physical cpus, amount of RAM, NUMA configuration,
- the mapping of virtual CPUs to physical CPUs, and
- runtime execution metrics such as CPU utilization about the VMs.

5.3.2 External

The Information aggregator component of the external sub-module is responsible for the high-level view of the site infrastructure. More specifically, in the current version, it retrieves information metrics about:

- the total number of active nodes,
- the total number of running VMs,
- the number of running VMs per node,
- the characteristics of the running VMs (gold, silver, noisy, sensitive),
- the number of paused VMs, and
- the number of characterized VMs that need to be placed.

5.4 Modeler Component

5.4.1 Internal

The Modeler component of the internal sub-module is responsible for the following functions:

VM characterization model

This model extracts a VM's fingerprint regarding its use of resources, e.g., cores, memory, etc., and classifies its execution to either "noisy" or "sensitive". This model is the core of the characterization phase that occurs while the VM is executed in the "laboratory" node. In this version of ACTiManager, we have implemented the functionality: (i) for accessing and gathering various monitoring metrics, such as CPU, memory, and network utilization, that are fed as input to the characterization model and (ii) for characterizing the execution of the VM using simple thresholds on the aforementioned metrics. Our ongoing work focuses on more intelligent and accurate characterization modeling.

Interference detection model

This model classifies the execution of a VM to either “interference suffering” or “interference free” when co-located with other VMs. The interference classification depends on the

characteristics of both the target VM and the co-located VMs. The model takes as input the information retrieved from the characterization, i.e., the VMs' fingerprints regarding the use of resources, "noisy" or "sensitive" behavior, etc., and combines that with monitoring information about its current execution. In this version of ACTiManager, the interference detection model uses simple rules identifying scenarios based on: (i) the prioritization groups of the VM (gold, silver), (ii) the characterization status of the VM (noisy, sensitive), and (iii) simple thresholds for monitoring metrics in the current node.

Remapping cost model

This model predicts the time required to remap a VM (cores, memory, both) within the node, and the expected performance improvement/degradation after the remapping action. This model targets the main characteristics of our two hardware platforms, i.e., the NUMA architecture of the Numascale system, and the big.LITTLE cores of the Exynos boards that constitute the KMAX system. Thus, we split this model into two parts:

- **Remapping cores and memory in a NUMA system.** The Numascale platform consists of multiple servers that implement the NUMA architecture themselves. As the Numascale platform introduces an additional level of NUMA shared-memory across multiple servers, the mapping of VM's cores and memory becomes an important factor in defining its performance in such systems. This model focuses on the cost of (re)mapping cores and memory, or just the memory, of a running VM. More specifically, the model predicts how much time it would take to remap both the cores and the memory, or just the memory of a VM, from one NUMA node to another. In addition, the model predicts the performance of the VM under the new configuration, in case a part or the whole memory of a running VM is remapped on a different NUMA node away from the assigned cores. Such actions could be the result of ACTiManager's decision making; for example, in order to create free memory on one or more NUMA nodes to accommodate the creation of a new large VM, ACTiManager may remap the memory of running VMs. Based on this model, ACTiManager will select the VM that will suffer less from performance degradation due to accessing remote NUMA nodes.
- **Remapping cores in big.LITTLE systems.** The KMAX platform consists of multiple Exynos boards, with each board implementing ARM's "big.LITTLE" architecture. big.LITTLE is a heterogeneous computing architecture, that couples relatively more powerful and power-hungry processor cores (big) with relatively battery-saving and slower ones (LITTLE). The execution of a VM on a big/LITTLE core may have significant impact on its performance. However, the impact on performance is not the same for all applications, and it depends on the application itself. Hence, the mapping of VMs on cores matters. An intuitive approach for mapping VMs to cores is to prioritize the gold VMs by mapping them on the big cores, while mapping the silver VMs on the LITTLE cores. Our model targets the mapping of multiple VMs, when they belong on the same prioritization class, based on a different approach. The idea is to use various metrics from the footprint of the VM and low-level monitoring events from performance counters, to predict the impact on performance when remapping a VM from a big to a LITTLE core, and vice versa. Our current and ongoing work has been focusing on identifying and combining those metrics.

In the current version of ACTiManager, the two aforementioned parts of this model use simple threshold-based and linear models based on CPU and memory usage for predicting the remapping cost in NUMA and big.LITTLE systems.

5.4.2 External

The Modeler component of the external sub-module is responsible for the following functions:

Site Under/Overutilization model

This model detects when a site is under/over-utilized. For the detection of this scenario in v1.0 of ACTiManager, a minimum and a maximum number of VMs per node are defined. In case the total number of VMs is less than the minimum number multiplied by the total number of nodes, i.e., the ratio of VMs per node is low, then a Site Underutilization event is raised. Similarly, in case the total number of VMs is more than the maximum number multiplied by the total number of nodes, i.e., the ratio of VMs per node is high, then a Site Overutilization event is raised. Our future work will focus on integrating also characteristics of the VMs and statistics from the host resources.

Imbalance detection model

This model detects when a site is unbalanced regarding the utilization of its nodes. In this version of ACTiManager, the Modeler checks for imbalance in a coarse-grained manner. More specifically, we use a maximum difference threshold regarding the number of VMs that are executed on the node with the heaviest load compared to the node with the lightest load. In case this limit has been reached or exceeded, then an Imbalance event is raised.

Migration prediction model

This model is responsible for estimating the migration cost of a VM from one node to another. Our ongoing effort on this model has primarily focused on live VM migration within QEMU/KVM and is based on previous research¹⁰. We have observed that the most significant parameter that affects the migration procedure is the number of dirty pages per iteration of the migration, i.e., the rate at which the VM writes memory affects the feasibility and the time needed for performing a VM migration action. We combine this dirty page write together with the total available network bandwidth to predict at a relatively high accuracy if the migration will succeed (we focus on the pre-copy algorithm¹¹) and to estimate the total time of migration and the total downtime that the user will experience. Note that this model is not part of the current version of ACTiManager; instead, the current implementation uses simple threshold-based and linear models based on memory usage for predicting the feasibility and the cost of migration, respectively.

¹⁰ Jinshi Zhang, Eddie Dong, Jian Li, and Haibing Guan. "MigVisor: Accurate Prediction of VM Live Migration Behavior using a Working-Set Pattern Model". In Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '17).

¹¹ https://en.wikipedia.org/wiki/Live_migration

5.5 Decision maker

5.5.1 Internal

The Decision Maker component of ACTiManager.internal proceeds to the corresponding decision based on the aggregated information and the output of the models. The main actions of the internal sub-module in this version of ACTiManager are to (re)map a VM's resources among the NUMA nodes of the node, and to report interference and node under/over-utilization to the external sub-module via the communication mechanism.

5.5.2 External

Placement

The Decision Maker component of ACTiManager.external considers various placement policies when placing a VM. The VMs priority class (e.g., gold, silver) and its runtime behavior (noisy, sensitive, fingerprint) are taken into account when a VM is placed in a node, together with the current load of the various nodes.

In our initial version of ACTiManager, VMs with the same priority class or similar runtime behavior are placed in different nodes as much as possible to avoid resource contention. Our approach to implement the placement functionality of the Decision Maker is through interacting and modifying existing components of OpenStack. More specifically, we modify OpenStack to be able to create:

- *Global instance groups*, i.e., instance groups that are not scoped to a single project as in vanilla OpenStack, but rather allow for membership of instances from all the projects in a cloud.
- *Multi-group instance memberships* that allow instances to be added as a member to multiple groups, instead of just being member to only one group as in vanilla OpenStack. Each instance group is taken into consideration by the scheduler when the instance is being placed on a node.

These modifications allow us to create multiple groups with different policies (e.g., anti-affinity policy which prohibits VMs to be placed on the same node together with other VMs that belong in the same group) which in turn allows for multiple placement policies for a single VM. In this way, ACTiManager can group instances to guide the placement process in a more sophisticated way than was possible before in vanilla OpenStack. For example, instance A can be added to both a global anti-affinity group called "gold" and to another global anti-affinity group called "noisy", while instance B can be "gold" and "quiet". Based on this information, the scheduling policies can be modified to make more informed decisions when placing VMs.

For the next version of ACTiManager, we will look into implementing more placement policies. These policies could improve the placement by taking into consideration the characteristics of the hardware platforms, i.e., by introducing the NUMA-awareness concept for the Numascale architecture.

5.6 Interaction with OpenStack

Our current version of ACTiManager is able to interact with OpenStack's Nova component through the NOVA API to:

- create global instance groups,
- add and remove instances from groups,
- migrate instances,
- pause/restart instances,
- vertically scale / resize instances.

The OpenStack modifications of our current version of ACTiManager target both the CLI and Nova components to support the following functionalities:

- Global instance groups
- Multi-group membership
- Adding and removing instances from groups

Finally, the following features are planned to be implemented in OpenStack for the next version of ACTiManager:

- In the Nova component:
 - New placement policies (through filters and/or weighers) will be implemented to improve the placement logic by taking into consideration the characteristics of the hardware platforms, as discussed above in Section 5.4.
 - Live resizing (also known as vertical scaling) will be considered to dynamically scale-up/down a VM. Resizing is currently performed through cold migration, which interrupts the VM and restarts it under a new instance. The live-resizing feature would provide support for scaling-up/down without interrupting the VM. Such feature has been considered by the OpenStack community¹² as well.
- In the Heat¹³ component:
 - Live resizing might also be implemented in Heat to support autoscaling.

¹² <https://blueprints.launchpad.net/nova/+spec/instance-live-resize>

¹³ <https://wiki.openstack.org/wiki/Heat>

6 Summary and Future Work

This document described the initial version 1.0 of ACTiManager that focused on implementing the basic functionality for supporting the entire lifecycle of VMs using simple threshold-based modeling logic and targeting commodity systems. Our future work will focus on developing more sophisticated models, integrating with MicroVisor, and providing further specialized support for the hardware platforms of the project.