Data Centres Optimization for Energy-Efficient and EnvironmentalLy Friendly INternet

Funding scheme: Specific Targeted Research Projects – STREP
Co-funded by the European Commission within the Seventh Framework Programme

Project no. 609140

Strategic objective: FP7-SMARTCITIES-2013 (ICT-2013.6.2)

Start date of project: October 1st, 2013 (36 months duration)

Deliverable 4.4

Implementation of Cross DC Workload Orchestrator and VM Manager

Due date: 03/01/2016

Submission date: 05/04/2016

Deliverable leader: UCL

Author list: Stuart Clayman (UCL), Michał Balcerkiewicz (PSNC), Ariel Oleksiak (PSNC), Tomasz Ciesielczyk (PSNC), Wojciech Piątek (PSNC), Dimitrios Siakavaras (GRNET), Vangelis Angelou(GRNET), Aniello Reale (WIND), Domenico Gallico (IRT), Matteo Biancani (IRT)

Dissemination Level

☒	PU:	Public
☐	PP:	Restricted to other programme participants (including the Commission Services)
☐	RE:	Restricted to a group specified by the consortium (including the Commission Services)
☐	CO:	Confidential, only for members of the consortium (including the Commission Services)

List of Contributors

Participant	Contributor
UCL	Stuart Clayman
PSNC	Michał Balcerkiewicz, Ariel Oleksiak, Tomasz Ciesielczyk, Wojciech Piątek
GRNET	Dimitrios Siakavaras, Vangelis Angelou
WIND	Aniello Reale
IRT	Domenico Gallico, Matteo Biancani

Amendment History

Version	Date	Partners	Description/Comments
0.1	03/01/2016	UCL	Initial draft
0.2	16/02/2016	PSNC	Added Workload Orchestrator and VM Manager
0.3	28/02/2016	UCL	Extra Additions
0.4	07/03/2016	GRNET	Reworked VM Manager section and added Dolfin Info DB description
0.5	17/03/2016	PSNC	VM Manager deployment, improved Monitoring Data Collector
0.6	18/03/2016	UCL	Updated Cross DC Workload Orchestrator
0.7	24/03/2016	PSNC	Monitoring section improvements, Conclusions added, Tables and Figures updated, Abbreviations and References updated, fixed typos
0.8	30/03/2016	WIND	Internal Review
0.9	31/03/2016	PSNC	Addressed comments provided by the WIND team
1.0	31/03/2016	WIND	Final Review
1.0	4/04/2016	IRT	Sanity check
1.0	5/04/2016	WIND	Addressed comments provided by IRT

Table of Contents

Figures Summary

Tables Summary

Abbreviations

AP	Action Point
API	Application Programming Interface
DB	Database
DC	Data Centre
DCI	Data Centre Interconnection
DoW	Description of Work
EC	European Commission
eCOP	Energy Consumption and Optimization Platform
HTTP	HyperText Transfer Protocol
iPERF	The network bandwidth measurement tool
IPR	Intellectual Properties Rights
JSON	JavaScript Object Notation
PM	Person Months
PMO	Project Management Office
REST	Representational State Transfer
QCT	Quality Check Team
SDC	Synergetic Data Centre
ToC	Table of Content
TMC	Technical Management Committee
URL	Uniform Resource Locator
VM	Virtual Machine
WP	Work Package

Executive Summary

The goal of the DOLFIN project is to deliver a set of frameworks and tools that combined together lead to decreased energy consumption in comparison to traditional Data Centre solutions. DOLFIN offers a broad scope of energy savings mechanisms, targeted at single Data Centre (racks, servers and cooling systems) and multiple Data Centres (VM migrations). This document focuses on energy optimizations within multiple Data Centres environments, which are realized with the help of the VM Manager , the Monitoring Data Collector, the Cross DC Orchestrator, and the DOLFIN Info Database modules.

This document presents the design and implementation attributes and a technical description of the Cross DC Orchestrator, the VM Manager, the Monitoring Data Collector, and the DOLFIN Info Database design and implementation. The purpose of these modules is to enable cross Data Centre interoperability.

This includes a detailed description of:

- Cross DC Orchestrator - the module responsible for interacting between the various Data Centres to determine if a Virtual Machine can be transferred from one DC to another DC
- VM Manager – module available for interfacing with several hypervisors (e.g. Openstack, Libvirt etc.) and supporting multiple actions including, but not limited to, list VMs running on a host, start/stop a VM, VM migration.
- Monitoring Data Collector – module responsible for throughput measurements between Data Centres. Knowing amount of maximum bandwidth available at given time helps to take accurate decisions regarding VM migrations
- DOLFIN Info Database – offers an abstracted and logically-centralised information manipulation (including information collection, aggregation / processing, storage / indexing and distribution) across the other DOLFIN architectural components.

The document presents the main interactions of this part of the DOLFIN system together with an overview of the implemented components and the messages they send.

1. Introduction

This document presents the design and implementation attributes of the Cross DC Orchestrator, the VM Manager, the Monitoring Data Collector, and the DOLFIN Info Database systems.

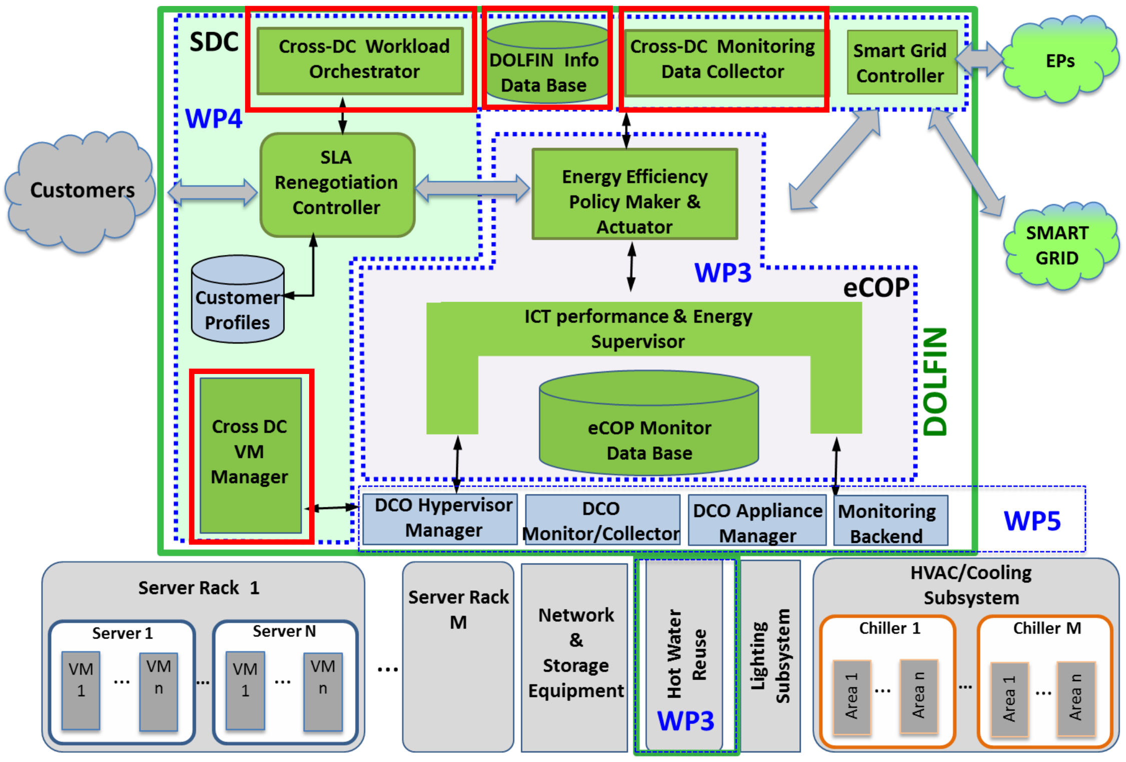

Figure 1. Overall DOLFIN architecture

The elements of the SDC create a distributed infrastructure that interacts with and is reliant on the elements of the eCOP system within each Data Centre. The energy-conscious SDC provides a dynamic, service-effective and energy-efficient allocation of load demands, across a distributed network of co-operating DCs. The three main elements of the SDC within a DC are:

1. The Cross DC Workload Orchestrator
2. The Cross DC VM Manager,
3. The Cross DC Monitoring Data Collector, and
4. The DOLFIN Info Database

The three first elements interact with identical components within each Data Centre, finally forming sets of distributed systems allowing the various DOLFIN-enabled DCs to seamlessly interoperate.

In addition the, SDC provides the Smart Grid Controller for the integration with the Smart Grid environment, being responsible for assisting the Grid towards achieving energy stabilisation, providing responses on the changing demands for energy. This element is not part of the distributed infrastructure as it is per-DC, and interacts with the Smart Grid only.

Figure 2 offers a high-level view of cross DC interactions, showing the distributed interaction of the Cross-DC elements.

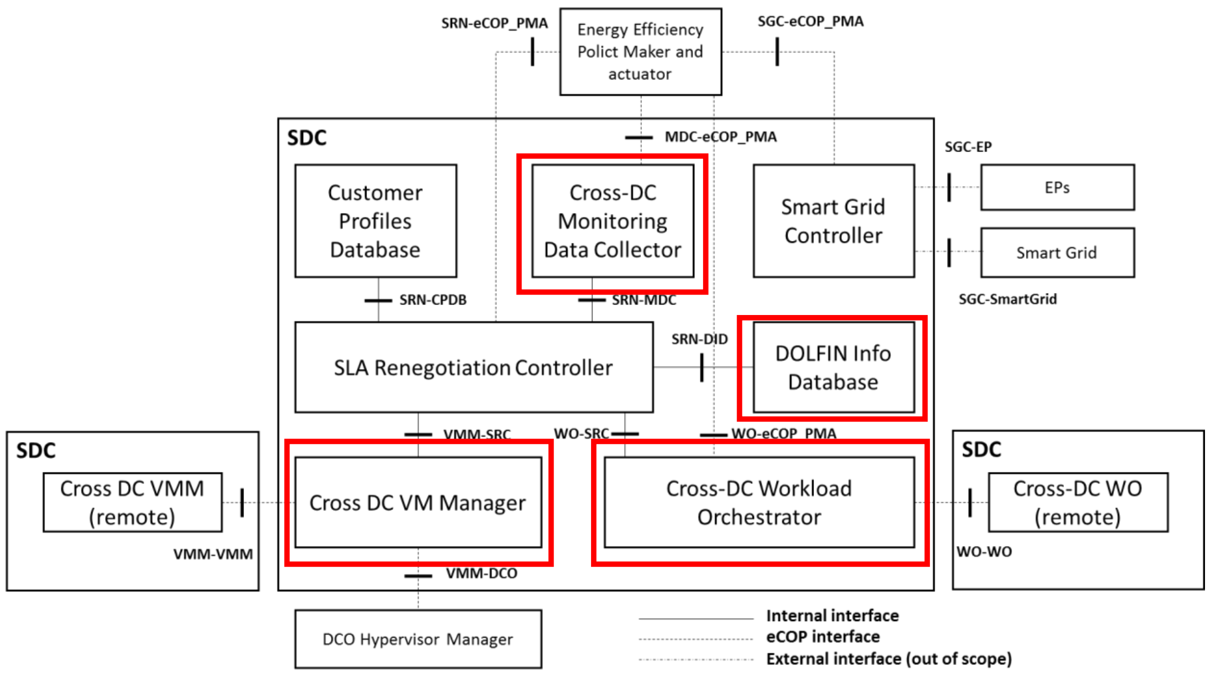

Figure 2. Modules and Interfaces

Of particular note is the Cross DC Workload Orchestrator that is itself a fully distributed system. Any local Energy Actuator can interact with its local Cross DC Orchestrator in order to determine if another DC is able to get extra VM load from another DC. The set of Cross DC Orchestrators interact amongst themselves in order to determine if a VM, or set of VMs, can be placed in a different DC. The Energy Actuator does not need to interact with other DCs itself.

Furthermore, the Cross DC VM Managers in the various DCs interact in order to negotiate the high level details of a VM migration process. Moreover, the Cross DC VM Managers interact with their local Hypervisor Managers in order to undertake the actual migration. Both the Hypervisor Managers and the migration process are existing technology that the SDC relies on, however the Cross DC VM Managers provide the lifecycle management and negotiation for setting up the migration process and also to provide the status details to other SDC components once the migration has been effected.

When a VM is relocated to another DC, the Cross DC Monitoring Data Collector of the remote DC will collect details for the VM and transmit those back to the originating DC, via its Cross DC Monitoring Data Collector. Such details will include live VM info that can be used for billing purposes.

2. Synergetic Data Centres (SDC) Components

To achieve energy efficiency, at the cloud level, the DOLFIN eCOP collaborates with the Synergetic Data Centres (SDC) module. For cross Data Centre interaction the SDC utilizes the following components:

- The Cross-DC Workload Orchestrator, which is a distributed software element that gets the SDC decisions and does most of the different types of resource optimisation including energy optimisation and management of their trade-offs in a cross DC optimization scenario. It handles DC migration message requests by performing negotiation with other SDS Workload Orchestrators located in distributed Data Centres in order to find one willing to receive the migrated VM.
- The Cross-DC VM Manager, which realises a DC Interconnect (DCI) interface and performs the actual migration of VMs cross DCs. It will typically apply a set of standard alternatives for coping with high/peak workloads, more precisely with allocation of VMs, data, services and tasks.
- The Cross-DC Monitoring Data Collector, which collects knowledge not only from the Synergetic Data Centre resources, but also from the network routers and in general from the network resources point of view. This is an important prerequisite to achieve energy efficiency across DCs.
- The DOLFIN Information Data Base (IDB), which offers abstracted and logically-centralised information manipulation (including information collection, aggregation / processing, storage / indexing and distribution) across all DOLFIN architectural components.

2.1. Components Interactions

The interactions of the main Synergetic Data Centres (SDC) components are presented below. The interactions are highlighted by flow diagrams that show the requests and the responses between each of the components.

2.1.1. Cross-DC Workload Orchestrator

This diagram presents the cross-DC VM Manager interaction with the Cross-DC Workload orchestrator. The interaction is initiated by the Workload Orchestrator (CDC.WO) posting a message such as CREATE, START, SHUTDOWN, RESTART, TERMINATE or MIGRATE a VM, along with additional data and parameters. The Cross-DC VM Manager initially acknowledges the request and replies the outcome of the action. Messages details are presented in [2].

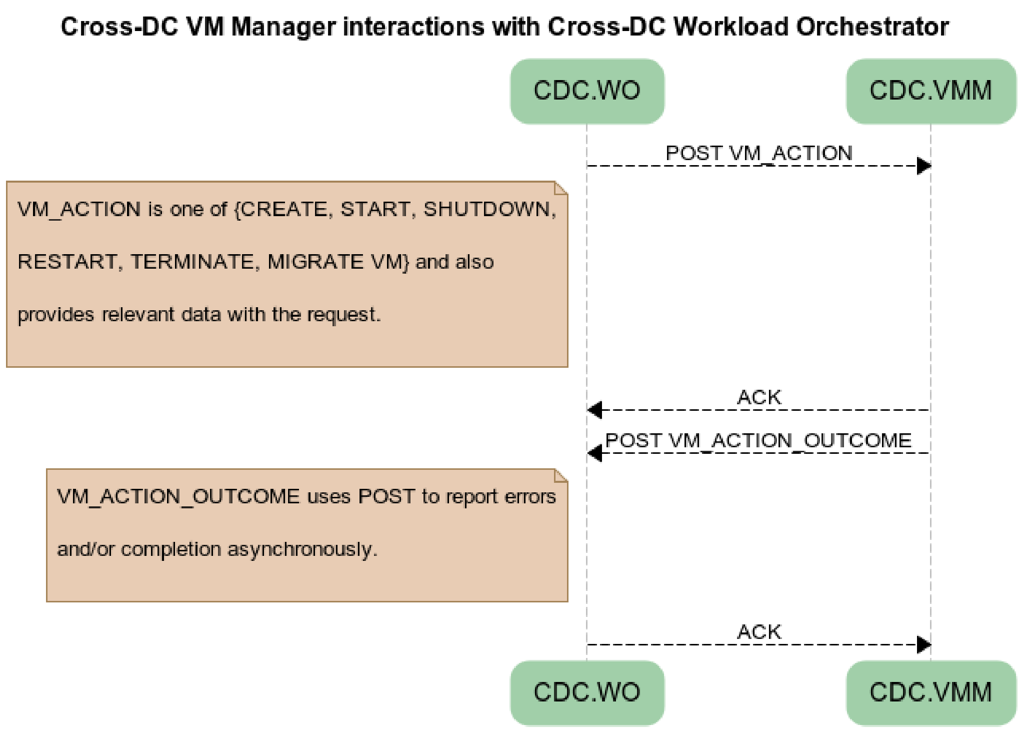
Cross-DC VM Manager interactions with Cross-DC Workload Orchestrator
CDC.WO
CDC.VMM
POST VM_ACTION
VM_ACTION is one of {CREATE, START, SHUTDOWN, RESTART, TERMINATE, MIGRATE VM} and also provides relevant data with the request.
ACK
POST VM_ACTION_OUTCOME
VM_ACTION_OUTCOME uses POST to report errors and/or completion asynchronously.
ACK
CDC.WO
CDC.VMM

Figure 3. Receiving request for VM actions from the Workload Orchestrator

This diagram presents the interface for requesting and performing a cross-DC migration. The process is supported by the Cross-DC Workload Orchestrator which is in charge of negotiating with the federated DCs.

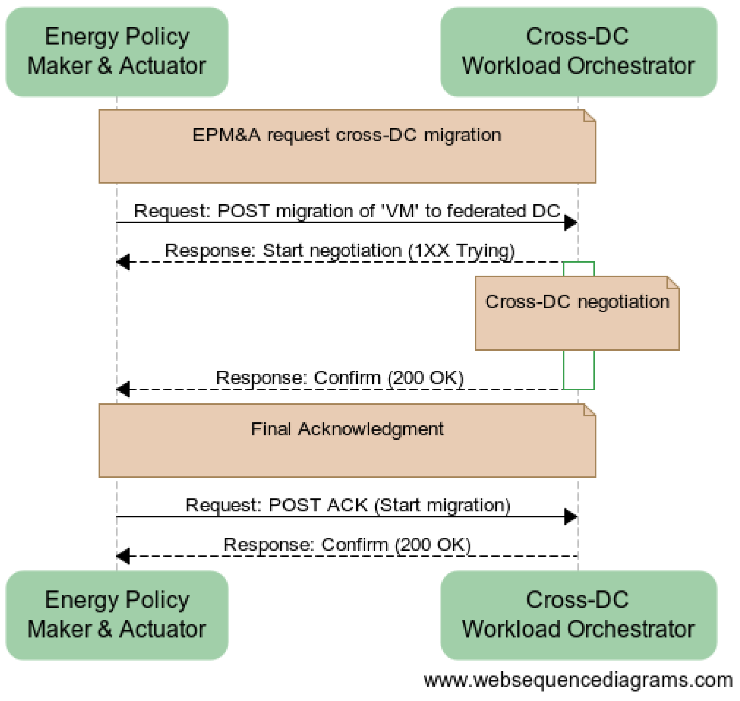
Energy Policy Maker & Actuator
Cross-DC Workload Orchestrator
EPM&A request cross-DC migration
Request: POST migration of 'VM' to federated DC
Response: Start negotiation (1XX Trying)
Cross-DC negotiation
Response: Confirm (200 OK)
Final Acknowledgment
Request: POST ACK (Start migration)
Response: Confirm (200 OK)
Energy Policy Maker & Actuator
Cross-DC Workload Orchestrator
www.websequencediagrams.com

Figure 4. Interface for cross-DC VM migration

2.1.2. Cross-DC Monitoring Data Collector

This diagram presents the cross-DC VM Manager interaction with the Cross-Dc Monitoring Data Collector. The interaction is quite simple having a post action message request and an ACK reply. Messages details are presented in [2].

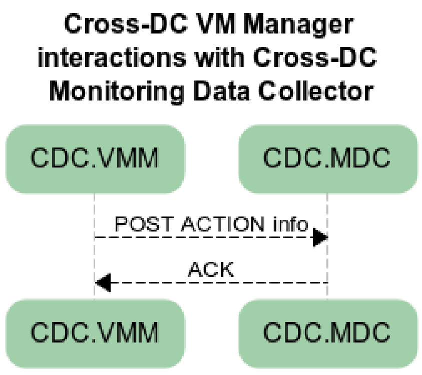
Cross-DC VM Manager interactions with Cross-DC Monitoring Data Collector
CDC.VMM
CDC.MDC
POST ACTION info
ACK
CDC.VMM
CDC.MDC

Figure 5. CDC VMM POSTs statistics of actions undertaken to the MDC

2.1.3. DOLFIN Information Database

This diagram presents the cross-DC VM Manager interaction with the DOLFIN Information Data Base. The interaction is quite simple having a get action message request and an ACK reply. Messages details are presented in [2].

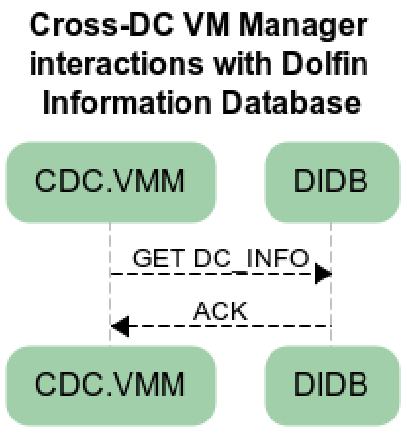
Cross-DC VM Manager interactions with Dolfin Information Database
CDC.VMM
DIDB
GET DC_INFO
ACK
CDC.VMM
DIDB

Figure 6. Polling the DIDB for info

2.1.4. Remote Cross-DC VM Manager

This diagram presents the interaction between two cross-DC VM Managers. Each VM Manager (VMM) may request to a) get information of a specific VM, b) migrate live or offline a VM and c) get information of the migration request. Messages details are presented in [2].

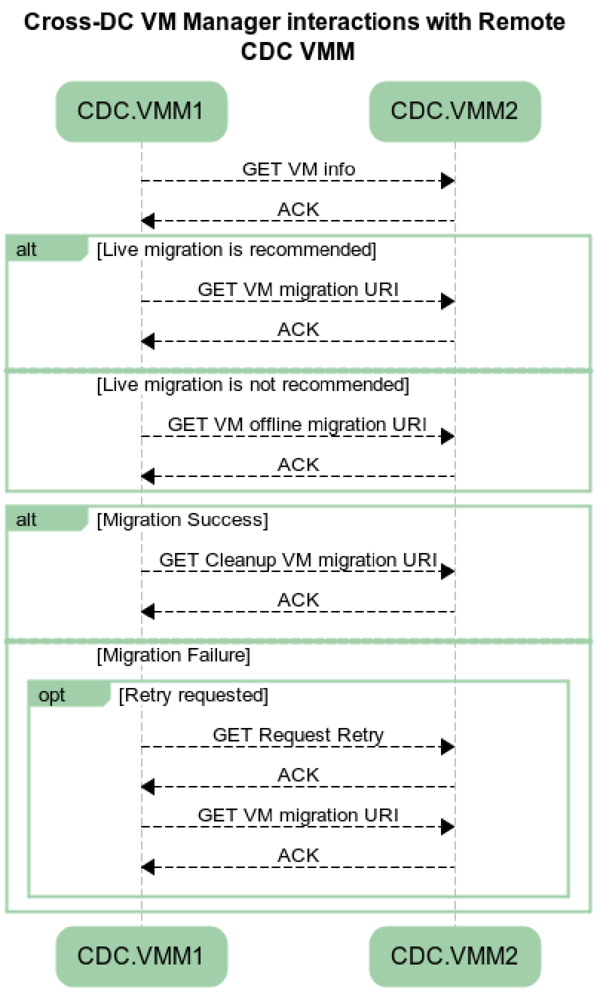

Figure 7. Coordinating with a remote instance of the VMM

2.1.5. DCO Hypervisor Manager

This diagram presents the interaction between a cross-DC VM Manager and two CDO Hypervisor Managers. Although most actions are straightforward and implemented in the relevant DCO HM APIs, the case for cross-DC migration is highlighted. Simple acknowledgements to requests are omitted for clarity. Messages details are presented in [2].

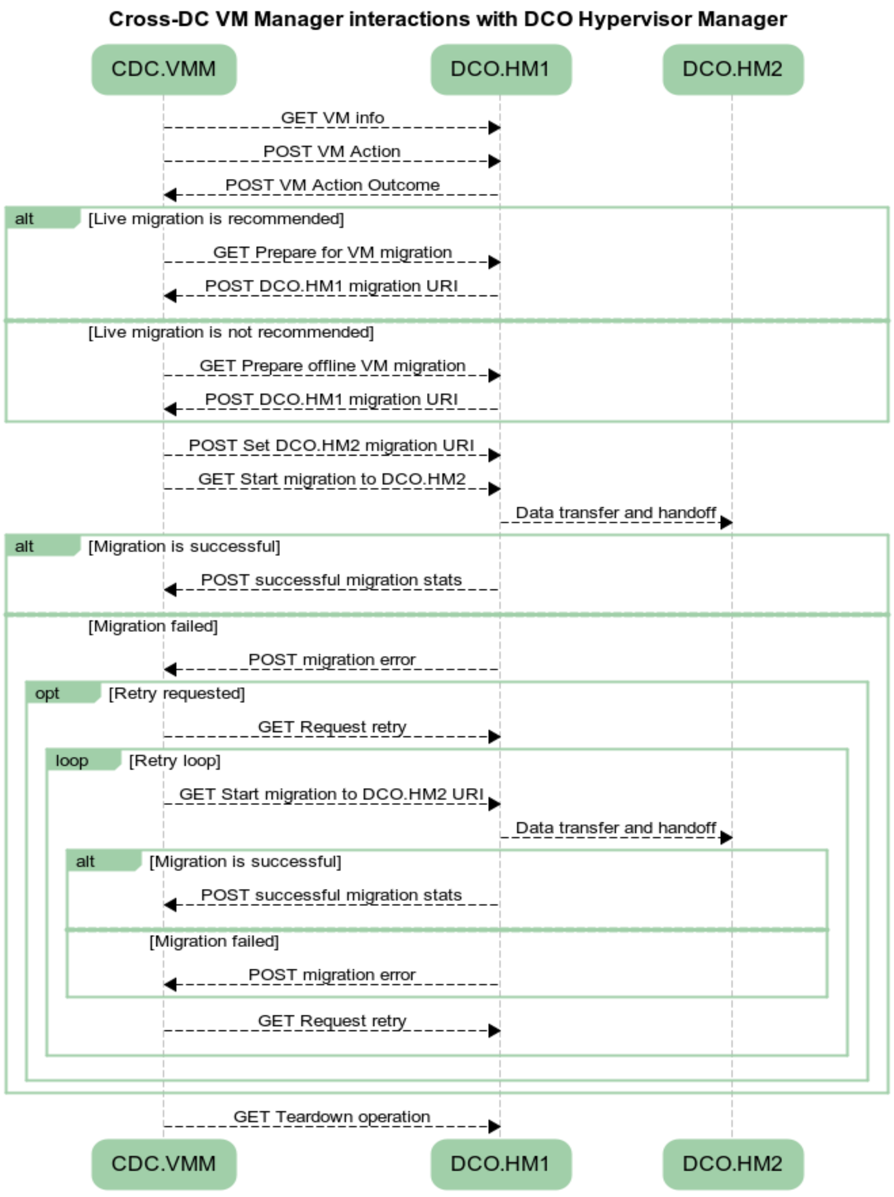
Cross-DC VM Manager interactions with DCO Hypervisor Manager
CDC.VMM
DCO.HM1
DCO.HM2
GET VM info
POST VM Action
POST VM Action Outcome
alt
[Live migration is recommended]
GET Prepare for VM migration
POST DCO.HM1 migration URI
[Live migration is not recommended]
GET Prepare offline VM migration
POST DCO.HM1 migration URI
POST Set DCO.HM2 migration URI
GET Start migration to DCO.HM2
Data transfer and handoff
alt
[Migration is successful]
POST successful migration stats
[Migration failed]
POST migration error
opt
[Retry requested]
GET Request retry
loop
[Retry loop]
GET Start migration to DCO.HM2 URI
Data transfer and handoff
alt
[Migration is successful]
POST successful migration stats
[Migration failed]
POST migration error
GET Request retry
GET Teardown operation
CDC.VMM
DCO.HM1
DCO.HM2

Figure 8. Interactions with the DCO HM module

2.2. Cross-DC Workload Orchestrator

The Cross DC Workload Orchestrator is a fully distributed system. Any local Energy Actuator can interact with its local Cross DC Workload Orchestrator in order to determine if another DC is able to get extra VM load from a different DC. The set of Cross DC Orchestrators interact amongst themselves in order to determine if a VM, or set of VMs, can be placed in a different DC. The Energy Actuator does not need to interact with other DCs itself.

The Cross-DC Workload Orchestration is also in charge of managing the full lifecycle of the virtual routers in the network and the allocation of the applications running on the virtual nodes. Moreover, it closely collaborates with the Monitoring Data Collector, which collects and manages all of the monitoring data received from the DC to DC network, including virtual and physical resources.

2.2.1 Architecture

The Cross-DC Workload Orchestrator is triggered by the local Policy Maker, which asks the Cross-DC Orchestrator if it can place a certain Virtual Machine (VM) with some specified attributes. Such a process is undertaken only if the Energy Actuator is unable to place a VM on the local DC. The Cross-DC Workload Orchestrator acts as a fully distributed sub-system, whereby all of the Cross-DC Orchestrators in all of the DCs in a group can interact with one another. They interact in order to determine if a VM can potentially be moved.

Figure 9 shows the interactions that happen whenever the Cross DC Workload Orchestrator is triggered:

1. The local Policy Maker starts the process to request a migration to a federated DC invoking the Cross DC Workload Orchestrator
2. The Cross DC Workload Orchestrator interact with the other Cross DC Workload Orchestrators, located in different federated DC, in order to determine if a VM can potentially be moved there.
3. Each invoked Cross DC Workload Orchestrator forward the request to its local Policy Maker
4. The Policy Maker accepts or rejects the request
5. The invoked Cross DC Workload Orchestrator forwards the answer to the Cross DC Workload Orchestrator which initiated the process
6. When a federated DC willing to take the VM is found, the local Cross DC Virtual Machine Manager is instructed to perform the migration.

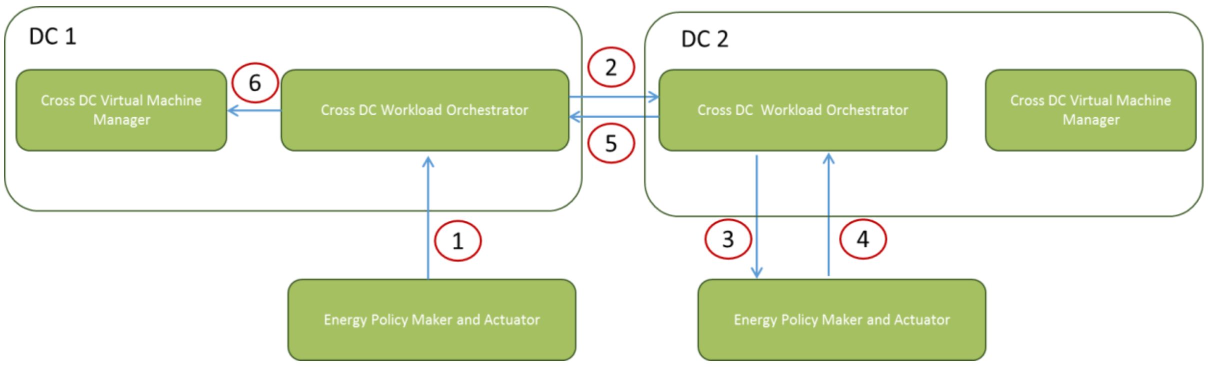
DC 1
Cross DC Virtual Machine Manager
6
Cross DC Workload Orchestrator
2
5
DC 2
Cross DC Workload Orchestrator
Cross DC Virtual Machine Manager
1
3
4
Energy Policy Maker and Actuator
Energy Policy Maker and Actuator

Figure 9. Cross DC Workload Orchestrator module in the VM migration phase

The Cross DC Workload Orchestration component is also responsible for the placement of the virtual routers, the allocation of service components and functions running on the routers as applications. In particular, it is responsible for instantiating the virtual routers as it is the software element which does most of the management and orchestration and is in charge of managing the full lifecycle of the virtual routers.

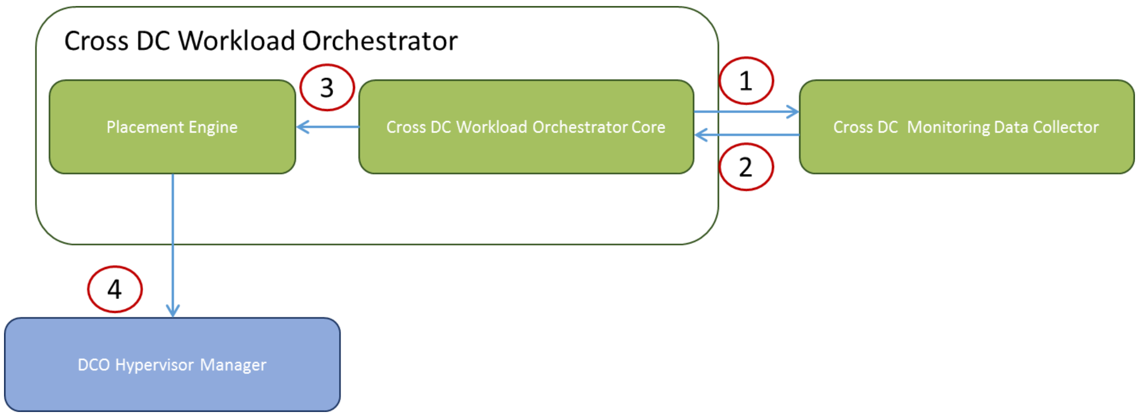
Cross DC Workload Orchestrator
Placement Engine
3
Cross DC Workload Orchestrator Core
1
2
Cross DC Monitoring Data Collector
4
DCO Hypervisor Manager

Figure 10. Cross DC Workload Orchestrator module in the network management phase

As shown in Figure 10, the Cross DC Workload Orchestrator can periodically poll the Cross DC Monitoring Data Collector to get the network assets status and metrics. If the need arises to increase the number of virtual routers, the Placement Engine will decide where to locate them, and will communicate the decision to the DCO Hypervisor Manager, which will translate the decision to commands executed by the local Hypervisors.

2.3 VM Manager

The Cross-DC VM Manager (CDC VMM) is responsible for the coordination with the local DC components on each site in order to allocate, migrate and de-allocate VMs to actual computing, storage or network resources. These resources are exposed from the DCO Hypervisor directly, but are managed in a higher level by the ICT and EPM&A DOLFIN components that reside in each DC. The component uses a RESTful API to interact with the various components of the architecture, whose calls are defined later in this document. It receives directives from the Cross-DC Workload Orchestrator, polls the DOLFIN Information DataBase for data and reports to the Cross-DC Monitoring Data Collector. It also coordinates with the DCO Hypervisor Manager and if necessary with other components of the DC. Thus, part of the API has to interface with the respective DCO Manager, and will therefore be DC specific.

2.3.1 Architecture

The Cross DC VM Manager's main responsibility is to manage a VM's lifecycle across DOLFIN-enabled DCs. Although it is primarily responsible for delegating such actions to the relevant DC on behalf of

DOLFIN, it also has to coordinate complicated actions (such as cross-DC migration of VMs) with other CDC VMMs, as per the CDC WO wishes. In Figure 11, a birds-eye view of such interactions is outlined:

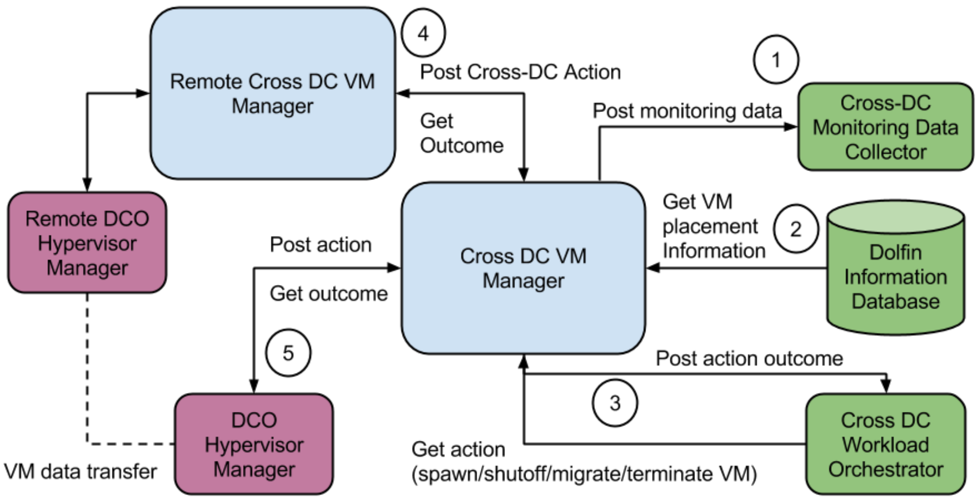
Remote Cross DC VM Manager
4
Post Cross-DC Action
Get Outcome
1
Post monitoring data
Cross-DC Monitoring Data Collector
Remote DCO Hypervisor Manager
Get VM placement Information
2
Dolfin Information Database
Post action
Get outcome
Cross DC VM Manager
5
Post action outcome
3
DCO Hypervisor Manager
Cross DC Workload Orchestrator
VM data transfer
Get action (spawn/shutoff/migrate/terminate VM)

Figure 11. Overview of CDC VM Manager's interaction with other DOLFIN modules

In essence, CDC VMM has to coordinate with:

1. ***CDC Monitoring Data Collector:*** as the Monitoring Data Collector is responsible for monitoring maximum achievable bandwidth between remotely located Data Centres, CDC VMM will be provided with information related to file transfer over the network. Knowing amount of time required to move large files (VM snapshots) between Data Centres, CDC VMM will be able to take optimal decision with regards to SLA (i.e. VM shutdown time is required to be minimal thus CDC VMM will choose routes with high bandwidth to quickly migrate VM).

2. ***DOLFIN Information Database:*** In order for the CDC VMM to correctly perform the requested actions it needs information for each DC, contained in the DOLFIN Info DB. Mostly, a description of the DC or DCs that partake in a specific operation needs to be known in advance, so that the CDC VMM can alter its action plan or cancel the operation altogether. Of interest therefore is the topology of the DC, the Hypervisor Manager (and hypervisor) used by each DC, the storage transport scheme preferred and relevant network setup (e.g. firewalls between DCs).

3. ***Cross DC Workload Orchestrator:*** Given that the CDC VMM is a delegation module for actions decided by the CDC WO, the relevant interactions include requests for local VM actions (spawn, start, shutdown, terminate) that are delegated to the relevant DCO HM and requests for migration to other DCs. The latter case is handled by the CDC WO with regards to the negotiation (i.e. finding a DC willing to receive the migrated VM) and the CDC VMM is thus notified only to coordinate the migration. It will however respond when an error occurs or a migration cannot be coordinated for some reason (e.g. hypervisor incompatibility).

4. ***Remote CDC VMM:*** Communication with a remote CDC VMM is only necessary when a migration is being negotiated. At such time, technical info about the VM under migration need to be pushed to the receiving side (to decide placement and compatibility) and a migration (destination) URI needs to be propagated to the sender. Once a connection is established, the CDC VMM delegates the migration to the DCO HM and, ultimately, the hypervisor.

5. ***DCO Hypervisor Manager:*** The CDC VMM will need to delegate most actions to the DCO HM, and will therefore need to use APIs that are common among most DC installations. Although the first target is the OpenStack API, it would be desirable to be able to communicate with DCO HMs that provides the AWS EC2 or VMWare vSphere API.

2.4 Cross DC Monitoring Data Collector

The Cross-DC Monitoring Data Collector is a service that is in charge of performing network throughput measurements between distributed Data Centres. Such measurements are one of the factors taken into account when deciding if VM migration is viable.

2.4.1 Architecture

Monitoring Data Collector is a stand-alone module responsible for providing bandwidth statistics between Data Centres. Such statistics are used to calculate the speed of moving large data volumes (entire Data Centres) to another location. The decision is taken by Policy Maker module based on multiple energy-related factors.

The measurements are performed using an open-source command line tool iPerf [3]. It is a client-server framework that determines maximum network bandwidth by pushing artificial traffic through the network. Once peak throughput is found, iPerf generates a report with the amount of data transferred and timestamp. Next, Monitoring Data Collector reads the report and saves the results into the DOLFIN Information Database. It should be noted that only TCP streams are utilized by iPerf as this closely simulates file transfers over the network.

The role of main components of the Monitoring Data Collector is as follows:

- REST API controller – handles message requests from eCOP Policy Maker module and returns path objects that contain bandwidth statistics for them
- Path Computation – performs Depth-First path computation on topology retrieved from DOLFIN Info Database and returns a shortest path between two topology elements (nodes) that meets maximum bandwidth criteria for that path
- Measurement Manager – runs asynchronously on separate thread, its job is to coordinate work of distributed iPerf instances
- SSH Retriever – reads and parses measurement reports generated by iPerf instances running in distributed Data Centres

Monitoring Data Collector has been authored in Java language making it possible to run either on Linux or Windows machines. The REST API is exposed with the help of CXF framework, whereas SSH client relies on Java Secure Channel library (JSCH). The Monitoring Data Collector service has ability to self-hosting within Jetty container therefore deployment complexity has been reduced to minimum.

The monitoring Data Collector internal structure and external interactions is shown on Figure 12:

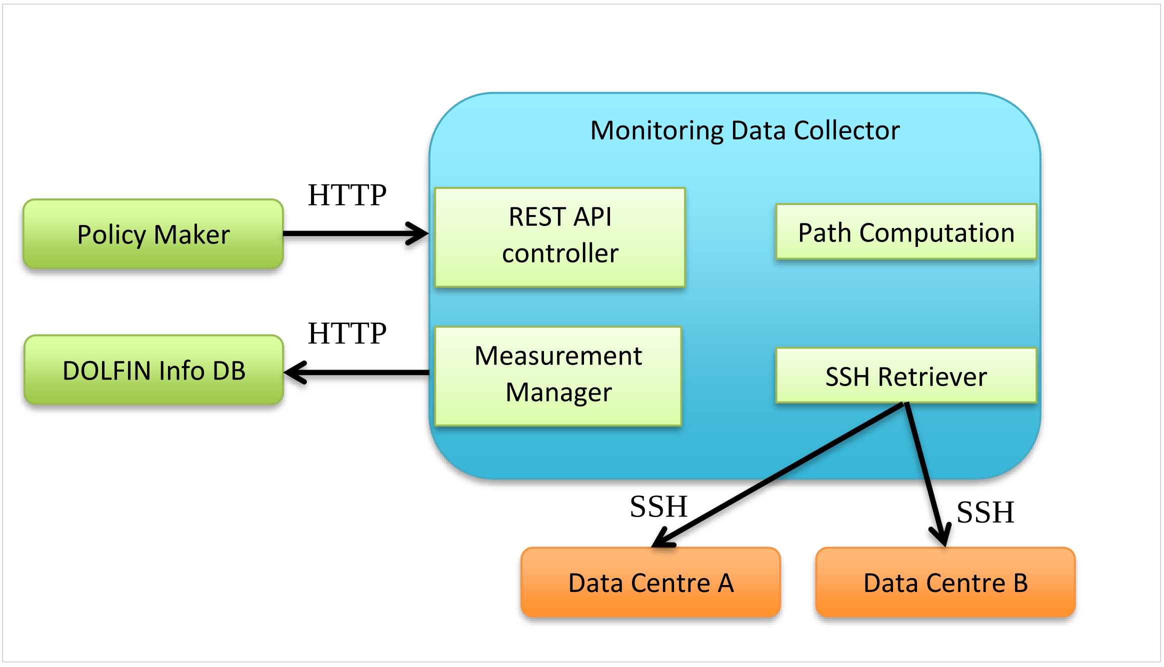

Figure 12. Monitoring Data Collector components and interactions

Monitoring Data Collector interacts with two DOLFIN modules:

- DOLFIN Info Database – used as a storage medium for measurements results obtained from iPerf instances located at remote Data Centres, as shown on diagram below:

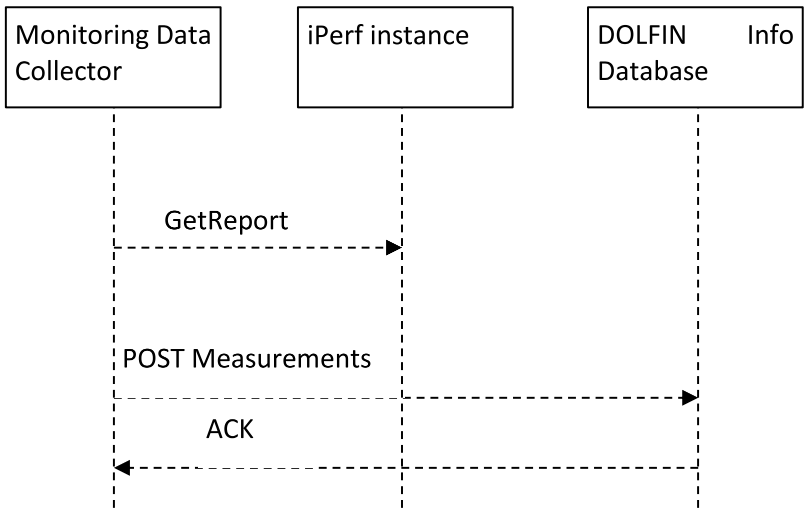

Figure 13. Monitoring Data collector saving measurements into DOLFIN Info DB

- Policy Maker – exposed REST API is consumed by Policy Maker to help take decision regarding Data Centres migration, this is shown on Figure 14:

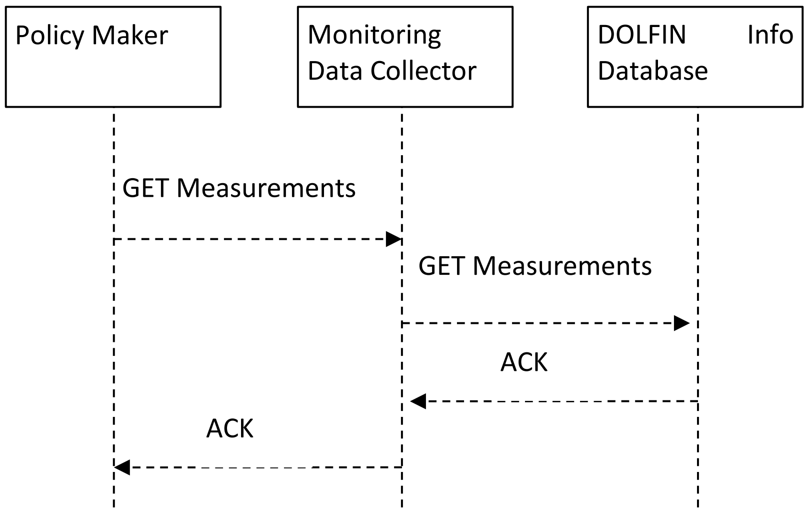

Figure 14. Policy Maker retrieving measurement results

2.5 DOLFIN Information Database

The DOLFIN Information Data Base (IDB) offers abstracted and logically-centralised information manipulation (including information collection, aggregation / processing, storage / indexing and distribution) across all DOLFIN architectural components. It includes information of the local energy requirements e.g. Demand/Response requests from the local Smart Grid operator.

The DOLFIN InfoDB provides storage and indexing, along with processing functionalities. It maintains a registry, storing specifications for the available information to be collected, retrieved, or disseminated. Moreover, it may apply aggregation functions (e.g. MAX, MIN, AVERAGE) to the collected data before they are stored or disseminated or filter data at the aggregation level for optimisation purposes.

In order to:

- manage systems in the DCs in a smart optimized, and automated way and as much as possible in "real time"
- achieve energy savings and the resulting C02 emission reductions
- meet the different demands from EPs in terms of optimization or different distribution of the power consumption of the smart grids

the DOLFIN expert system needs to have the topology data of infrastructures in DCs, the electricity consumption and infrastructure performance data of hardware infrastructures of the DCs constantly available. This info is also stored in the DOLFIN Energy Information Base.

2.5.1 Architecture

The DOLFIN Information Data Base (InfoDB) uses two separate interfaces for communication with the DOLFIN software entities:

1. the Information Management Interface is used for information manipulation configuration, including the information sources/sinks registration to the Information Base, the management of internal Information Base, information manipulation functions and the establishment, operation and optimisation of information flows; and
2. the Information Exchange Interface that offers the actual information exchange capability to the DOLFIN components. These two interfaces allow a unified yet abstracted information handling in all types of participating entities.

The DOLFIN Information Database module needs to store and manipulate a wide range of data provided and required by different DOLFIN components. As such, the DOLFIN Information Database needs to provide:

1. An API to define raw data storage parameters, aggregated metrics computed from raw data and the retention and access schemes for the relevant data. This API is in line with the Information Management Interface defined above, and exposed to the CDC Monitoring Data Collector module, SLA Renegotiation Module and the Smart Grid Controller.
2. An API to receive raw and aggregated monitoring data from CDC MDC. These data are stored for reference in the DOLFIN DB and are potentially processed further to derive new metrics/KPIs as needed. This API forms part of the Information Exchange Interface described in the previous section.
3. A way to query the database for information. This functionality is available to any DOLFIN module, and makes the DOLFIN DB the primary information repository for all participating DCs.
4. The modular design of the DB enables the Storage Broker to compute metrics through offline analysis of the historical data stored in the DB. Thus, a module can extract higher level information from the data stored in the DB.

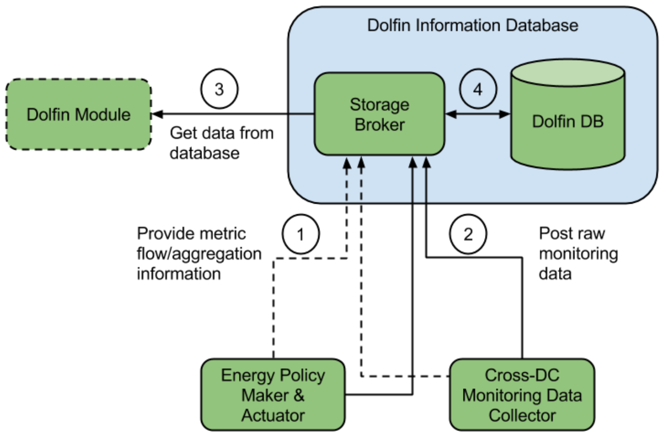
Dolfin Information Database
Dolfin Module
3
Get data from database
Storage Broker
4
Dolfin DB
Provide metric flow/aggregation information
1
2
Post raw monitoring data
Energy Policy Maker & Actuator
Cross-DC Monitoring Data Collector

Figure 15. High level depiction of DOLFIN Information DB's interactions

The Storage Broker module of the DB provides a simple interface that abstracts the peculiarities of the internal storage architecture. The Storage Broker can be extended to interface with different datastore through different backend-specific plugins. Thus, a large number of backends can be used for information storage.

3. Implementation

In the previous section we presented the components of the SDC, plus their interactions. In this section we present the implementation details of:

1. The Cross DC Workload Orchestrator
2. The Cross DC VM Manager,
3. The Cross DC Monitoring Data Collector, and
4. The DOLFIN Info Database

3.1. Cross DC Workload Orchestrator

The Cross DC Workload Orchestrator Core is the main component of the Cross DC Workload Orchestrator. It acts as a control point for the platform by sending out the command and acts as a management element for the platform by collecting monitoring data and enabling reactive behaviour. It also provides the placement mechanism with all necessary information to perform the steps needed to prepare the DC in order to be able to accommodate load from other DCs.

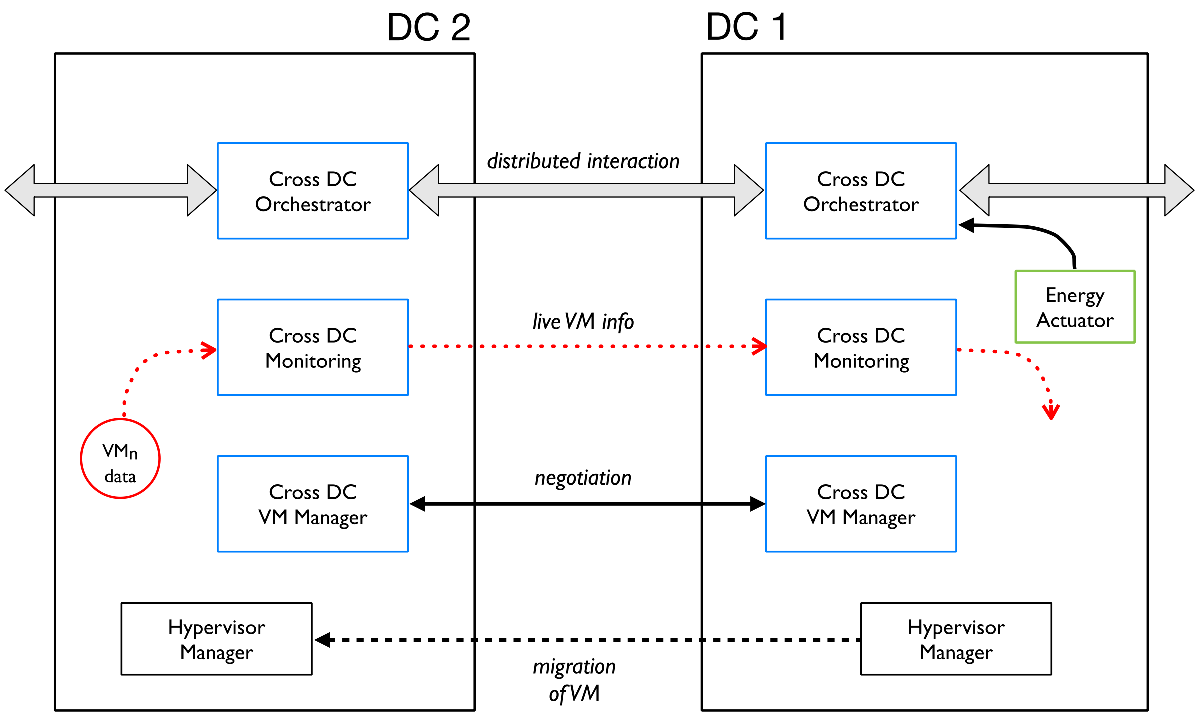

Figure 16. High-level view of the cross-DC workload relocation interactions

1. The Energy Actuator of the eCOP asks the local Cross-DC Orchestrator if it can place a certain Virtual Machine (VM) with some specified attributes. Such a process is undertaken only if the eCOP Energy Actuator is unable to place a VM on the local DC (DC1 in this case) in which case it triggers the Cross-DC Orchestrator.
2. The Cross DC Orchestrator of DC1 will interact with Cross DC Orchestrators of the DOLFIN-enabled Synergetic DCs. It will make a request to place or relocate a VM with the same specified VM attributes that the Energy Actuator provided. In this instance the Cross DC Orchestrator of DC2 is messaged.
3. The Cross DC Orchestrator of DC2 provides the response to the Cross DC Orchestrator of DC1. The response can be one of two outcomes:
    a. NO. DC2 is unable to accept the VM. i.e. placement is not possible on DC2.
    b. YES. DC2 can accept the VM. i.e. placement is possible on DC2.
4. The Cross DC Orchestrator of DC1 will inform the Energy Actuator that made the original request with the outcome. This will result in one of two situations:
    a. On NO, the eCOP Energy Actuator is unable to relocate a VM so it will have to leave the VM at DC1. No further steps with other DCs are taken at this time.
    b. On YES, the eCOP Energy Actuator will initiate the necessary actions to export the VM, and will move onto step 5.
5. The Cross DC Orchestrator will trigger a request to the Cross DC VM Manager of DC1 to initiate a cross DC VM transfer.
6. The Cross DC VM Manager of DC1 will tell the Cross DC VM Manager of DC2 that the VM needs to be placed in DC2.
7. The Cross DC VM Managers will then be responsible for the transfer of the specified VM.

3.1.1. Installation

To build the Cross DC Orchestrator Java 7 or Java 8 tool need to be installed in the system, the build process generates a jar file that can be executed as a standalone server.

In order to run the server the following command should be executed:

```
$ ant jar
```

3.1.2. API

The base URL is: http://ip:port/cdc/api and the data format on the API is JSON. The following functions are exposed by the API:

Request

This is the REST call made from the eCOP Energy Actuator to the local Cross DC Orchestrator. It contains all of the necessary details of a running VM that a remote DC will need to make a decision. The call contains info on:

- VM start time
- The elapsed time since the VM started
- The amount of cpu used by the VM
- The amount of memory used by the VM
- The amount of energy used by the VM
- The VM ID and VM name

Service URL	**HTTP**	**Response code**	**Description**
/cdc /placement	POST	200 (success) 405 (Invalid input)	Request a new migration.

Request example:

```
{
      "message":"placement_request",
      "payload":{
            "starttime": 1457512595765,
            "elapsedTime": 335633,
            "cpuSys": 1.36899995803833,
            "cpuTotal": 10.37600040435791,
            "cpuUser": 9.006999969482422,
            "memory": 33131,
            "energy": 1.1315979003906165,
            "energyTotal": 120.41549568176269,
            "vmId": 20,
            "vmName": "Router-20"
      },
      "timestamp": 1457512934598
}
```

Remote Request

This is the REST call made from the local Cross DC Orchestrator to a remote Cross DC Orchestrator. It takes the original eCOP request and elaborated it with:

- the info about the local Cross DC Orchestrator ID and address
- a request ID

Service URL	HTTP	Response code	Description
/cdc /remote_placement	POST	200 (success) 405 (Invalid input)	Request a distributed mi-gration and placement.

Request example:

```
{
	"message":"remote_placement_request",
	"dc_id": 3,
	"dc_host":  "128.40.39.142:7700",
	"request_id": "3/1423",
	"payload":{
		"starttime": 1457512595765,
		"cpuSys": 1.36899995803833,
		"cpuTotal": 10.37600040435791,
		"cpuUser": 9.006999969482422,
		"elapsedTime": 335633,
		"energy": 1.1315979003906165,
		"energyTotal": 120.41549568176269,
		"memory": 33131,
		"vmId": 20,
		"vmName": "Router-20"
	},
	"timestamp": 1457512934604
}
```

Response example, if the remove Cross DC orchestrator is ***not*** able to do a placement:

```
{
	"message":"remote_placement_response",
	"dc_id": 3,
	"dc_host":  "128.40.39.142:7700",
	"request_id": "3/1423",
	"payload":{
		"status": "NO"
	},
	"timestamp": 1457512951229
}
```

Response example, if the remove Cross DC orchestrator is able to do a placement:

```
{
      "message":"remote_placement_response",
      "dc_id": 3,
      "dc_host":  "128.40.39.142:7700",
      "request_id": "3/1423",
      "payload":{
            "status": "YES",
            "dc_id": 2,
            "dc_host":  "10.111.122.116:5000"
      },
      "timestamp": 1457512969694
}
```

The response contains the ID of the Data Centre that can accept the Virtual Machine, plus the address of the host that the Cross DC VM Manager must talk to in order to activate the migration.

3.2. VM Hypervisor Manager

This section describes in more detail the VM migration process. The CDC VMM after the negotiation initiates the migration process of the VM by sending appropriate request (Figure 17) to VM Hypervisor Manager which is a part of VM Manager using API described in the section 3.2.2.

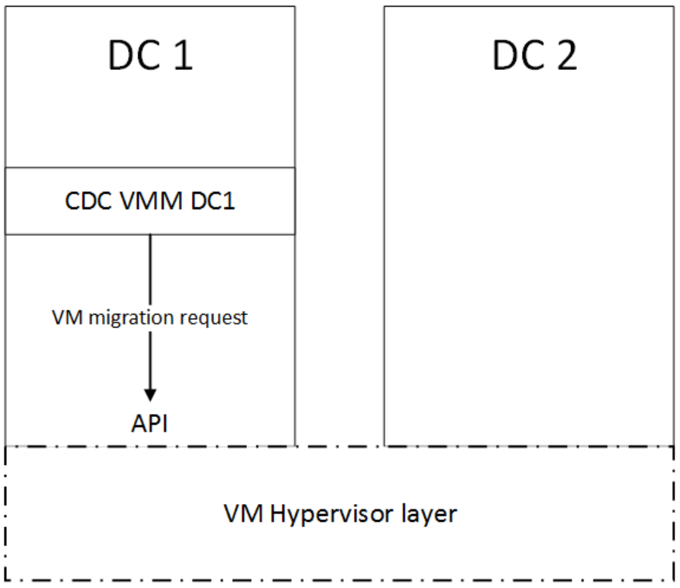

Figure 17. VM migration process

The deployment of the VM Hypervisor Manager component within the Dolfin project is realized twofold. The first case follows the OpenStack installation architecture, while the second one is based on the VMware approach.

Former installation assumes the usage of both KVM and VMware ESX as the hypervisors and organization of OpenStack modules around them. Figure 18 shows installed services and the role of particular nodes inside such installation.

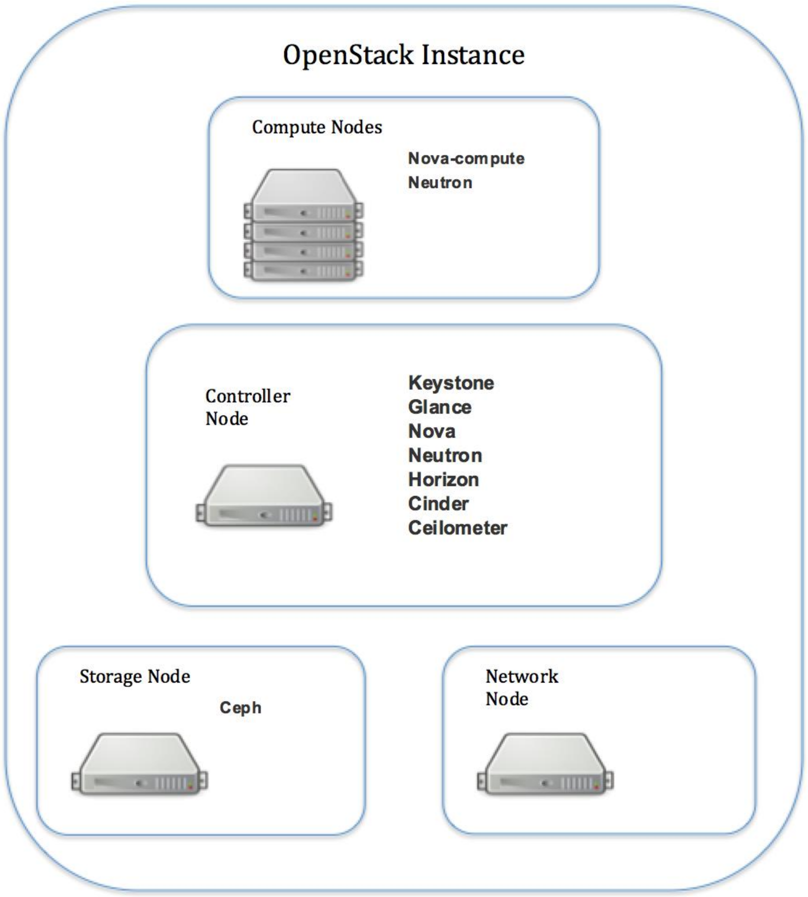

Figure 18. OpenStack Deployment Architecture

Within the latter case, a VMware solution, like vSphere with Operations Management [2], could be deployed. This powerful virtualization and management environment offers a wide range of features ranging from providing a virtualization layer, through live virtual machines migration up to the automation of virtual machines management operations. Figure 19 shows an overview of the vSphere architecture.

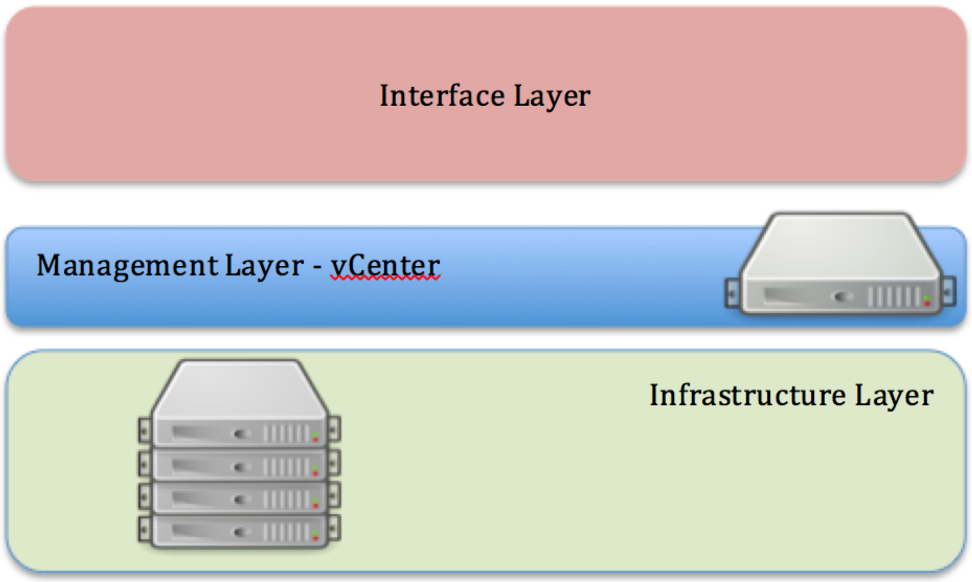

Figure 19. vSphere Deployment Architecture

The key elements of the presented architecture are services running within the Infrastructure Layer and interacting with the Management Layer. They include:

VMware vSphere Hypervisor: ESX - provides a robust, production-proven, high-performance virtualization layer. It enables multiple virtual machines to share hardware resources with performance that can match native throughput.

VMware vSphere vMotion - enables live migration of virtual machines between servers and across virtual switches with no disruption to users or loss of service, eliminating the need to schedule application downtime for planned server maintenance.

VMware vSphere Storage vMotion - enables live migration of virtual-machine disks with no disruption to users, eliminating the need to schedule application downtime for planned storage maintenance or storage migrations.

VMware vSphere Distributed Resource Scheduler - provides dynamic, hardware-independent load balancing and resource allocation for virtual machines in a cluster, using policy-driven automation to reduce management complexity while meeting SLAs.

VMware vSphere Distributed Power Management - automates energy efficiency in vSphere Distributed Resource Scheduler clusters by continuously optimizing server power consumption within each cluster.

The following table summarizes the role of these components within the VM Manager

Component Name	***Role***	***Corresponding DOLFIN modules***
vSphere Hypervisor: ESX	Virtualization layer	DCO Hypervisor Manager
vSphere vMotion	VM migration	CDC VMM, DCO Hypervisor Manager
vSphere Storage vMotion	VM migration	CDC VMM, DCO Hypervisor Manager
vSphere Distributed Resource Scheduler	Load management	CDC VMM
vSphere Distributed Power Management	Power policies for load management	CDC VMM
vCenter	Manager virtualized environment	CDC VMM, Remote CDC VMM

Table 3-1. vSphere component roles within the VM Manager

Both approaches to VM Hypervisor Manager component (by the means of OpenStack and VMware vSpere) are deployed at PSNC testbed based on FusionServer XH620 V3 server nodes. Each node comes with the following configuration:

- processor: Intel Xeon E5-2600 v3
- memory: 64GB DDR4, 2133MHz
- storage: 1TB, 7200RPM
- network: 10GbE

Both solutions come with special interfaces that can be used to interact with their components. OpenStack offers a Dashboard to manage the infrastructure together with a dedicated API to access the underlying services. In case of VMware vSphere, Interface Layer offers, apart from VMware vSphere Web Access, VMware vSphere SDK that provides a standard interface for VMware and third-party solutions to access the VMware vSphere.

3.2.1. Installation

To build the VM Manager the following tools need to be installed in the system:
- Java 8
- Maven

The following command installs them in debian-based distros:

```
$ apt-get install maven oracle-java8-installer
```

The build process generates a jar file that can be loaded as a Jetty service through the src/main/webapp/WEB-INF/web.xml file. The server stubs were generated by the swagger-codegen project to create a JAX-RS server.

In order to run the server, the following command has to be executed:

```
$ mvn clean package jetty:run
```

The swagger listing can be viewed here:

http://localhost:8080/swagger.json

Note that if the host has been configured to be something other than localhost, the calls will be directed to that host and not localhost.

3.2.2. API

The VM Manager API primarily allows migrating VMs across servers of a single data centre or even across multiple data centres.

The base URL is: **Error! Hyperlink reference not valid.** and the data format on the API is JSON. The functions listed in the following text are exposed by the API.

Migrations

Service URL	HTTP	Response code	Description
/vmm/api/migrations	POST	200 (success) 405 (Invalid input) 500 (Internal Server error)	Starts a new migration.

Request example:

```
{
	"id": 1,
	"vm": {
		"id": 5,
		"uuid": "apache VM 1",
		"orig_dc": {
			"id": 3,
			"name": "node 3",
			"endpoint":"http://ip:5000/v2.0/",
			"mgmtsw": "openstack",
			"hypervisor": {
				"id": 2,
				"name": "libvirt",
				"defaultURI: "qemu:///system"
		}
	},
	"destination: {
		"id": 2,
		"name": "node 2",
		"endpoint:" "http://ip2:5000/v2.0/",
		"mgmtsw": "openstack",
		"hypervisor": {
			"id": 2,
			"name": "libvirt",
			"defaultURI: "qemu:///system"
		}
	},
	"startDate": "11/3/2016 20:44:32",
	"lastUpdate": "11/3/2016 20:44:32",
	"tags": [
		"Apache"
	],
	"status": "start",
	"policy": "restartable",
	"complete": false
}
```

This operation does not include a response body.

Service URL	HTTP	Response code	Description
/vmm/api/migrations/findByStatus	GET	200 (success) 400 (Invalid status value)	Find migrations by status.

The inputs to this operation are supplied as get parameters added in the request URL. Example:
GET http://ip/vmm/api/migrations/findByStatus?status=networking HTTP/1.1

This operation does not include a request body.

Response example:

```
{
      "id": 1,
      "vm": {
            "id": 5,
            "uuid": "apache VM 1",
            "orig_dc": {
                  "id": 3,
                  "name": "node 3",
                  "endpoint":"http://ip:5000/v2.0/",
                  "mgmtsw": "openstack",
                  "hypervisor": {
                        "id": 2,
                        "name": "libvirt",
                        "defaultURI: "qemu:///system"
            }
      },
      "destination: {
            "id": 2,
            "name": "node 2",
            "endpoint:" "http://ip2:5000/v2.0/",
            "mgmtsw": "openstack",
            "hypervisor": {
                  "id": 2,
                  "name": "libvirt",
                  "defaultURI: "qemu:///system"
            }
      },
      "startDate": "11/3/2016 20:44:32",
      "lastUpdate": "11/3/2016 20:54:02",
      "tags": [
            "Apache"

      ],
      "status": "networking",
      "policy": "restartable",
      "complete": false
}
```

Service URL	HTTP	Response code	Description

/vmm/api/migrations/findByTags	GET	200 (success) 400 (Invalid status value)	Find migrations by status.

The inputs to this operation are supplied as get parameters added in the request URL. Example:
GET http://ip/vmm/api/migrations/findByTags?status=Apache,MySql HTTP/1.1

This operation does not include a request body.

Response example:

```
{
      "id": 1,
      "vm": {
            "id": 5,
            "uuid": "apache VM 1",
            "orig_dc": {
                  "id": 3,
                  "name": "node 3",
                  "endpoint":"http://ip:5000/v2.0/",
                  "mgmtsw": "openstack",
                  "hypervisor": {
                        "id": 2,
                        "name": "libvirt",
                        "defaultURI: "qemu:///system"
            }
      },
      "destination: {
            "id": 2,
            "name": "node 2",
            "endpoint:" "http://ip2:5000/v2.0/",
            "mgmtsw": "openstack",
            "hypervisor": {
                  "id": 2,
                  "name": "libvirt",
                  "defaultURI: "qemu:///system"
            }
      },
      "startDate": "11/3/2016 20:44:32",
      "lastUpdate": "11/3/2016 20:54:02",
      "tags": [
            "Apache"

      ],
      "status": "networking",
      "policy": "restartable",
      "complete": false
}
```

Service URL	HTTP	Response code	Description
/vmm/api/migrations/{migId}	GET	200 (success) 400 (Invalid ID supplied) 404 (Migration not found)	Find migration by ID.

The input to this operation is supplied along with the URL path as {migId}. Example:
GET http://ip/vmm/api/migrations/1 HTTP/1.1

This operation does not include a request body.

Response example:

```
{
      “id”: 1,
      “vm”: {
            “id”: 5,
            “uuid”: “apache VM 1”,
            “orig_dc”: {
                  “id”: 3,
                  “name”: "node 3",
                  “endpoint”:”http://ip:5000/v2.0/”,
                  “mgmtsw”: “openstack”,
                  “hypervisor”: {
                        “id”: 2,
                        “name”: “libvirt”,
                        “defaultURI”: “qemu:///system”
            }
      },
      “destination: {
            “id”: 2,
            “name”: "node 2",
            “endpoint:” “http://ip2:5000/v2.0/”,
            “mgmtsw”: “openstack”,
            “hypervisor”: {
                  “id”: 2,
                  “name”: “libvirt”,
                  “defaultURI: “qemu:///system”
            }
      },
      “startDate”: “11/3/2016 20:44:32”,
      “lastUpdate”: “11/3/2016 20:54:02”,
      “tags”: [
            “Apache”

      ],
      “status”: “networking”,
      “policy”: “restartable”,
      “complete”: false
}
```

Service URL	HTTP	Response code	Description
/vmm/api/migrations/migId	DELETE	400 (Invalid ID supplied)	Cancels a migration.

The input to this operation is supplied along with the URL path as {migId}. Example:
DELETE http://ip/vmm/api/migrations/1 HTTP/1.1

This operation does not include a request body.

This operation does not include a response body.

Service URL	HTTP	Response code	Description
/vmm/api/info	GET	200 (success) 404 (No info found)	Get all relevant info for a VMM instance.

Response example:

```
{
        "id”: 2,
        "name”: "regionOne",
        "endpoint”:”http://ip:5000/v2.0/”,
        "mgmtsw”: "openstack”,
        "hypervisor”: {
              "id”: 2,
              "name”: "libvirt”,
              "defaultURI”: "qemu:///system”
        }
}
```

3.3. Cross DC Monitoring Data Collector

3.3.1. Installation

Prerequisites

In order to properly run Monitoring Data Collector, the following requirements must be met:

- Java 1.6 or greater installed on machine where the service will be deployed
- IP address accessible from the Internet in case measurement results should be served to remote clients
- Outgoing SSH connections to Data Center hosts
- Optionally message queue server RabbitMQ can be installed
- iPerf 3 measurement tool installed in each Data Center
- DOLFIN Information Database

Configuration parameters

Before starting Monitoring Data Collector few configuration parameters must be provided. These parameters are used to initialize and drive behaviour of the service and can be found at {app_root}/etc/mdc.properties file. This file is organized as a key-value list, meaning only values can be altered. The following property entries are defined:

Key	***Value***
ip	IP address either in canonical or domain form. If the service should be accessible from Internet, the IP address of the deployment machine must be provided
port	[1024-65535] port number on which the service expects to receive requests
cli_port	[0 | 1024-65535] port number for accepting telnet connections. For security reasons only connections from the machine the service is running on are allowed. Port 0 disables telnet server
log_messages	[true | false] if set to true, raw http messages will be logged to a separate file
threads	[0-1024] size of internal thread pool for task processing
dolfin_db	Url to DOLFIN Info Database service. If empty no database will be used
topology	filename with topology in xml format
measure_rate	[0-1440] determines frequency of collecting measurements, in minutes

Table 3-2. Monitoring Data Collector configuration parameters

Additional logging facilities can be configured by editing the /etc/log4j.properties file.

Topology file is a human-readable xml file that describes inter-DC topology. Example topology file with two nodes representing Data Centers and a link between them:

```
<topology>
      <nodes>
            <node>
                  <name>DC master</name>
                  <host>dcmaster.com</host>
                  <user>admin</user>
                  <password>******</password>
                  <iperf>home/admin/iperf</iperf>
                  <ifname>eth0</ifname>
            </node>
            <node>
                  <name>DC backup</name>
                  <host>dcbackup.com</host>
                  <user>admin</user>
                  <password>******</password>
                  <iperf>home/admin/iperf</iperf>
                  <ifname>eth0</ifname>
            </node>
      </nodes>
      <links>
            <link>
                  <name>DC master-backup</name>
                  <source>DC master</source>
                  <target>DC backup</target>
            </link>
      </links>
</topology>
```

It should be noted that topology file poses a great security threat. Even though all passwords contained in topology file are kept in encrypted form, it is relatively easy for an attacker to recover the original passwords. Therefore it is strongly advised to lock down the machine on which the service is running.

Starting and stopping Monitoring Data Collector

Prior to starting the Monitoring Data Collector, it must be ensured that DOLFIN Information Database service is available. To learn how to start DOLFIN Info DB see section X.X.X.

It is expected that nodes described in topology file have iPerf tool installed. To install iPerf on Linux the following commands should be invoked:

```
# first make sure system is up-to-date
$ sudo apt-get update
$ sudo apt-get upgrade

$ iPerf installation
$ sudo apt-get install iperf
```

Once configuration has been set, the service can be started with provided 'start.sh' script file. After a couple of seconds instance will be ready for taking requests, or an error message will be displayed in case of failure (port already in use is most common error that prevents service from starting). To stop running the instance, script file 'stop.sh' should be called.

Command line

Monitoring Data Collector is supplied with simple command line (no support for commands completion when pressing the 'tab' key). This facility is meant to be a helper tool that allows to inspect state of the service without the need of using external clients. Available commands are presented in Table 3-3:

Command name	***Description***
exit | quit	Leaves command line and returns to OS shell
help	Prints a list of available commands
reload	Reloads configuration parameters – equivalent to starting the service with new parameters
password [password]	Outputs encrypted password
topology	Prints nodes and links that make up topology
bandwidth	Prints bandwidth statistics between all nodes

Table 3-3. Monitoring Data Collector command line parameters

3.3.2. API

The Monitoring Data Collector API primarily allows querying the topology and measurement results. Also some configuration parameters can be altered directly with this API.

The base URL is: http://ip:port/mdc/api and the data format on the API is JSON.
The following data types are accessible and can be manipulated on the REST API:

- topology – a collection of nodes and links
- config – configuration parameters
- measurements – most up to date throughput measurements

Topology

The operations allowed on topology resources are:

Service URL	HTTP	Response code	Description
/mdc/api/toplogy	GET	200	Returns current topology

This operation does not accept a request body.

Response example:

```
{
        “nodes”: [
                {
                        “name”: “node A”,
                        “host”: “hostA.net”,
                        “password”: “******”,
                        “iperf”: “home/admin/iperf”
                },
                {
                        “name”: “node B”,
                        “host”: “hostB.net”,
                        “password”: “******”,
                        “iperf”: “home/admin/iperf”
                }
        ],
        “links”: [
                {
                        “name”: “node1-node2”,
                        “source”: “node 1”,
                        “target”: “node 2”,
                }
        ]
}
```

Service URL	HTTP	Response code	Description
/mdc/api/toplogy	POST	200 400 (invalid input)	Updates (overrides) current topology

Upon receiving valid topology update request, Measurement Manager will discard obtained measurements and will start collecting new ones.

This operation does not include a response body.

Request example:

```
{
       "nodes": [
              {
                     "name": "Data Centre A",
                     "host": "dcA.com",
                     "password": "******",
                     "iperf": "home/admin/iperf"
              },
              {
                     "name": "Data Centre B",
                     "host": "dcB.com",
                     "password": "******",
                     "iperf": "home/admin/iperf"
              },
              {
                     "name": "Data Centre C",
                     "host": "dcC.com",
                     "password": "******",
                     "iperf": "home/admin/iperf"
              }
       ],
       "links": [
              {
                     "name": "DC A-B",
                     "source": "Data Centre A",
                     "target": "Data Centre B",
              },
              {
                     "name": "DC B-C",
                     "source": "Data Centre B",
                     "target": "Data Centre C",
              },
              {
                     "name": "DC A-C",
                     "source": "Data Centre A",
                     "target": "Data Centre C",
              }
       ]
}
```

Config

The operations allowed on config resources are:

Service URL	HTTP	Response code	Description
/mdc/api/config	GET	200	Returns current configu-ration

This operation does not accept a request body.

Response example:

```
{
       "measurementRate": 120
}
```

Service URL	HTTP	Response code	Description
/mdc/api/config	POST	200 400 (invalid input)	Sets new configuration

Upon receiving valid configuration update request, Measurement Manager will collect measure-ment results with new frequency.

This operation does not include a response body.

Request example:

```
{
       "measurementRate": 60
}
```

Measurements

The operations allowed on measurement resources are:

Service URL	HTTP	Response code	Description
/mdc/api/measurements	POST	200 400 (invalid input)	Retrieves bandwidth values be-tween start and end nodes

Request parameters:

Parameter	Type	Description
startNode	xsd:String	Identifier of the start node
endNode	xsd:String	Identifier of the end node

Response example:

```
[
	{
		“startNode”: “node 1”,
		“endNode”: “node 2”,
		“bandwidth”: 1000000
	}
]
```

3.4. DOLFIN Information Database

3.4.1 Installation

To install the infoDB, a running DBMS is necessary to be operating on the host machine and a copy of the storage broker software obtained as a source code package To obtain the code, the necessary prerequisites and compile the storage broker from source on a Debian compatible distribution:

```
$ git clone Error! Hyperlink reference not valid.

$ mvn clean package jetty:run
```

To configure the software for various DBMS's, the properties of the src/main/config/hibernate.cfg.xml file should be appropriately modified. The default settings assume a MySQL DB backend:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
      <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver
      </property>
      <!-- Hack to autocreate DB if it doesn't exist -->
      <property name="hibernate.connection.url">

      jdbc:mysql://localhost:3306/infodb?createDatabaseIfNotExist=true
      </property>
      <property name="hibernate.connection.username">root</property>
      <property name="hibernate.connection.password">root</property>
      <property name="hibernate.dialect">
            org.hibernate.dialect.MySQLDialect
      </property>
      <property name="show_sql">true</property>

      <!-- In theory, the following property should create and maintain the
table
           schema automatically. See also values create and create-drop -->
      <property name="hibernate.hbm2ddl.auto">update</property>

      <mapping resource="SLA.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

3.4.2 API

SLAs

Service URL	HTTP	Response code	Description
/infodb/api/sla	POST	200 (success) 301 (Unauthorized) 405 (Invalid input) 500 (Internal Server error)	Create a new SLA.

Request example:

```
{
        “id”: 1,
        “vm_uuid”: “debian-apache”,
        “host_id”: “node-1”,
        “current”: 3,
        “restriction”: 1,
        “type”: <VAGG>
}
```

This operation does not include a response body.

Service URL	HTTP	Response code	Description
/infodb/api/sla/findByType	GET	200 (success) 400 (Invalid type)	Finds SLAs by type.

The inputs to this operation are supplied as get parameters added in the request URL. Example:
GET http://ip/infodb/api/sla/findByType?type=Private HTTP/1.1

This operation does not include a request body.

Response example:

```
[
        {
                “id”: 1,
                “vm_uuid”: “debian-apache”,
                “host_id”: “node-1”,
                “current”: 3,
                “restriction”: 1,
                “type”: “Private”
        }, {
                “id”: 2,
                “vm_uuid”: “debian-mysql”,
                “host_id”: “node-1”,
                “current”: 1,
                “restriction”: 3,
                “type”: “Private”
        }
]
```

Service URL	HTTP	Response code	Description
/infodb/api/sla/{slaType}/{vmUuid}	GET	200 (success) 400 (Invalid input) 404 (SLA not found)	Finds SLAs by type and VM uuid.

The inputs to this operation are supplied along with the URL path as {slaType} and {vmUuid}.
Example:
GET http://ip/infodb/api/sla/Private/debian-apache HTTP/1.1

This operation does not include a request body.

Response example:

```
{
	“id”: 1,
	“vm_uuid”: “debian-apache”,
	“host_id”: “node-1”,
	“current”: 3,
	“restriction”: 1,
	“type”: “Private”
}
```

Service URL	HTTP	Response code	Description
/infodb/api/sla/{slaId}	GET	200 (success) 400 (Invalid ID supplied) 404 (SLA not found)	Find SLAs by ID.

The inputs to this operation are supplied along with the URL path as {slaId}. Example:
GET http://ip/infodb/api/sla/1 HTTP/1.1

This operation does not include a request body.

Response example:

```
{
	“id”: 1,
	“vm_uuid”: “debian-apache”,
	“host_id”: “node-1”,
	“current”: 3,
	“restriction”: 1,
	“type”: “Private”
}
```

Service URL	HTTP	Response code	Description
/infodb/api/sla/{slaId}	DELETE	400 (Invalid ID supplied)	Deletes an SLA.

The inputs to this operation are supplied along with the URL path as {slaId}. Example:
DELETE http://ip/infodb/api/sla/1 HTTP/1.1

This operation does not include a request body.

This operation does not include a response body.

4. Conclusions

This document presented the high level architecture and technical implementation details of the components developed within the Work Package 4 activity. The approach accompanied in creating this document was to add and update information that already existed in the D4.1: “Synergetic Data Centres for energy efficiency (Design)”([D4.1]) deliverable.

In particular, the deliverable describes modules, procedures and APIs that enable live migration of entire DOLFIN-aware Data Centres in fully automatic way. Such functionality, when properly explored, offers new powerful opportunities for energy savings by HPC operators.

Moreover, as the DOLFIN project seeks for energy reductions in every aspect, it also considers network as source of potential savings. In this regard, the Monitoring Data Collector helps determine the best routes for Data Centre migrations.

Additionally, the DOLFIN Information Database that serves as common data storage for DOLFIN modules is explained as well as its functionalities.

This work also presents various interoperable interfaces exposed towards the modules developed within the WP3 activity. The integration part requires development skill in order to implement the appropriate interfaces between the WP4 and WP3 components. However, adaptation and installation should not be highly complicated and in most cases are expected to be carried out straightforward after a careful reading of this material. By combining modules created by both activities, the ultimate goal should be achieved – migration of distributed Data Centres based on energy factors.

References

(1) D2.2 - DOLFIN requirements and system architecture, [Online]. Available: http://www.dolfin-fp7.eu/wp-content/uploads/2014/10/DOLFIN_D2.2_UCL_30-09-2014_FF.pdf

(2) D4.1 – Synergetic Data Centres for energy efficiency (SDC)(Design), [Online]. Available: http://www.dolfin-fp7.eu/wp-content/uploads/2015/07/DOLFIN_D4.1_UCL_FF-20150430.pdf

(3) iPerf – The network bandwidth measurement tool, http://iperf.fr