ascens

www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

Technical Report: TR20120500 - The Science Cloud Case Study

Overview and Scenarios

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **7.6.2010**

Lead contractor for deliverable: **LMU**
Author(s): **Philip Mayer (LMU), Christian Kroiss (LMU), José Velasco (Zimory)**

Due date of deliverable: **July 04th, 2012**
Actual submission date: **July 04th, 2012**
Revision: **V1**
Classification: **PU**

Project coordinator: **Martin Wirsing (LMU)**
Tel: **+49 89 2180 9154**
Fax: **+49 89 2180 9175**
E-mail: **wirsing@lmu.de**

Partners: **LMU, UNIPI, UDF, Fraunhofer, UJF-Verimag, UNIMORE, ULB, EPFL, VW, Zimory, UL, IMT, Mobsya, CUNI**

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

This document describes the Science Cloud Platform (SCP), which is a platform-as-a-service cloud providing the ability to execute applications in a robust manner while keeping to their SLAs.

The cloud consists of individual instances which may join or leave the cloud at will. Thus, the SCP is an exercise in volunteer computing and has no central coordinator. Individual instances of the cloud software work together to keep applications running, thus forming ensembles.

In this document, we discuss the architecture of the SCP and provide a high-level overview of an implementation. Three scenarios are provided which may serve as starting points for validating the methods, tools, and languages of ASCENS.

Contents

1 Introduction

Cloud computing refers to provisioning resources such as virtual machines, storage space, processing power, or applications to consumers "on the net": Consumers can use these resources without having to install hard- or software themselves and can dynamically add and remove new resources. A common use case is simply renting space, processing power, or applications from a cloud provider and using it over the Internet. Larger companies or universities may also buy a cloud solution and install it locally; thus offering a "private cloud" with space, power, or applications to their employees.

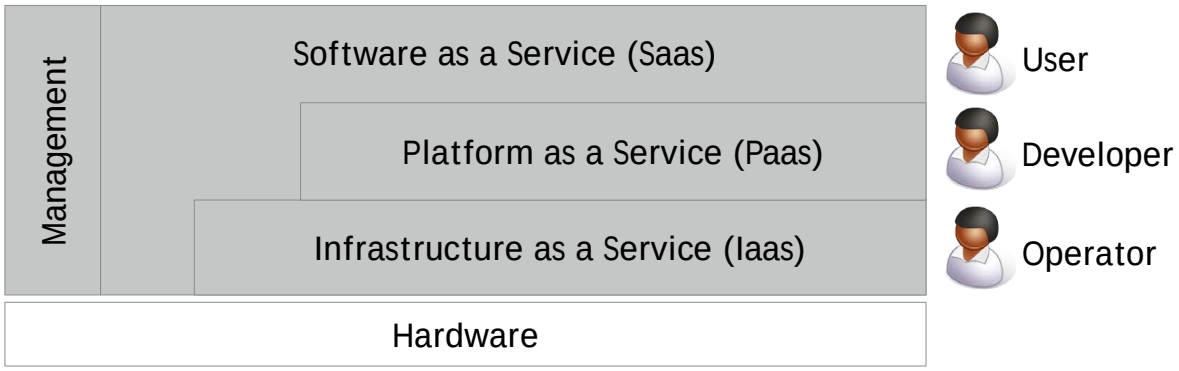

Figure 1: Cloud Solution Types

Depending on which resources and services are provided, we can distinguish three different layers of cloud solutions (see Figure 1), which may build upon one another or be offered to the customer as-is. Management tools are required across all solutions.

- On the lowest level, *Infrastructure as a Service (IaaS)* solutions provide resources such as virtual machines, virtual network switches, and simply data storage. Commonly, such solutions include tools and APIs for management, for example, starting and stopping VMs or creating new (virtual) networks. An example is the Zimory Enterprise Cloud product. These solutions will be managed, and provided by operators.

- On the next level, *Platform as a Service (PaaS)* solutions provide development and execution platforms for cloud applications. A PaaS solution may for example provide a framework for writing applications which can either be supplied with adequate resources and distributed automatically, or request additional resources (processing power or storage) themselves. An example is the Science Cloud Platform (SCP) which we will detail below. This level is interesting for developers of applications.

- Finally, *Software as a Service (SaaS)* solutions no longer provide raw resources but applications, for example a complete e-mail, calendar, or document collaboration solution. An example is Google Apps, or any application running within the SCP. This level, finally, is offered to users.

The term *cloud* refers to the idea of using resources without the need to know (from a users perspective) where and how they are provided — they are just "there", somewhere (apparently) in a white, fluffy, opaque mass that is the net.

An important property of clouds is that they are "always-on": The resources in a cloud will, from a user perspective, never fail (i.e. the cloud provider must ensure data redundancy, process migration, and load balancing), given the customer or his software does not exceed the Service Level Agreement

(SLA) — for example, if only a certain amount of storage space was paid for, only this amount of space is provided (though in a fail-safe manner). One of the advantages of virtual resources in a cloud, however, is that adding or removing resources is relatively painless and ideally can be performed automatically (again, based on SLAs or direct customer involvement).

In the ASCENS science cloud computing case study, we will be concerned with a Platform as a Service (PaaS) solution, i.e. a cloud on the second layer of Figure 1. The goal of the case study is providing a software system (called the Science Cloud Platform, SCP) which will, installed on multiple virtual or non-virtual machines, form a cloud providing a platform for application execution (for providing SaaS solutions). In particular, we will provide scenarios of how individual SCP instances work together to achieve important cloud properties (such as fail-safe operation) and how they go about acquiring new resources or discarding ones no longer required.

The Science Cloud Platform may run on IaaS solutions which enables it to provision new virtual machines or virtual networks or shut them down again. One such solution is the Zimory Enterprise Cloud. In the other direction, developers may use the Science Cloud Platform API to write SaaS applications which run on top of the SCP, providing any kind of application to end users.

Input for this document was the ASCENS Description of Work (Part B) [ea11], the deliverable D7.1 [vRA+11] (which this document supersedes), and the case study working meeting in Berlin in May 2012.

2 Science Cloud Overview

The basic idea of the science cloud is *cloud computing in the scientific environment*, i.e. to be used by scientists for data storage, grid-like calculations, or providing services (servers) to other people. Individual users or universities can join in, thus providing resources to the community. The science cloud is an exercise in *volunteer computing*. Each user can join or leave at will, which requires that the cloud parts have to work together in a peer-to-peer fashion.

The science cloud is a collection of notebooks, desktops, servers, or virtual machines running the Science Cloud Platform (SCP). Each (virtual) machine is running (usually) one instance of the Science Cloud Platform; we call such instances Science Cloud Platform instances (SCPi). Each SCPi is considered to be a *service component* in the ASCENS sense.

Multiple SCPs communicate over the Internet (IP protocol), thus forming a *cloud* and within this cloud one or more *service component ensembles*. We call one such ensemble a Science Cloud Platform ensemble (SCPe). Exactly which SCPis form an ensemble is dynamic and depends on the properties of the SCPis. We usually says that an ensemble consists of SCPis which work together to run one application in a fail-safe manner and under consideration of the *Service Level Agreement* (SLA) of that application, which may require a certain number of active SCPis, a certain latency between the parts, or have restrictions on processing power or on memory.

The SCP is a platform as a service which provides a platform for *application execution* as functionality (requirement 1 in [vRA+11]). Applications are written for and executed on top of the SCP, which is done by using *an application programming interface* (API) and library (requirement 3 in [vRA+11]). These applications (*apps*) provide SaaSs to users. Data storage functionality (requirement 2 in [vRA+11]) is provided by such an application.

Although the science cloud is an exercise in volunteer computing, there may be parts which are under the control of a commercial or academic entity (for example, a server farm which runs an IaaS such as the Zimory cloud). In this case, the goal of this entity is using as little resources as possible, thus preserving energy and reducing wear on the system.

On a technical level, the Science Cloud Platform is written in Java and is thus independent of the underlying operating system (requirement 8 in [vRA+11]).

2.1 Science Cloud Platform Components and Ensembles

Each instance of the Science Cloud Platform, running on a physical or virtual machine, is considered to be a service component in the ASCENS sense. Figure 2 shows the functionality required by a Science Cloud Platform instance.

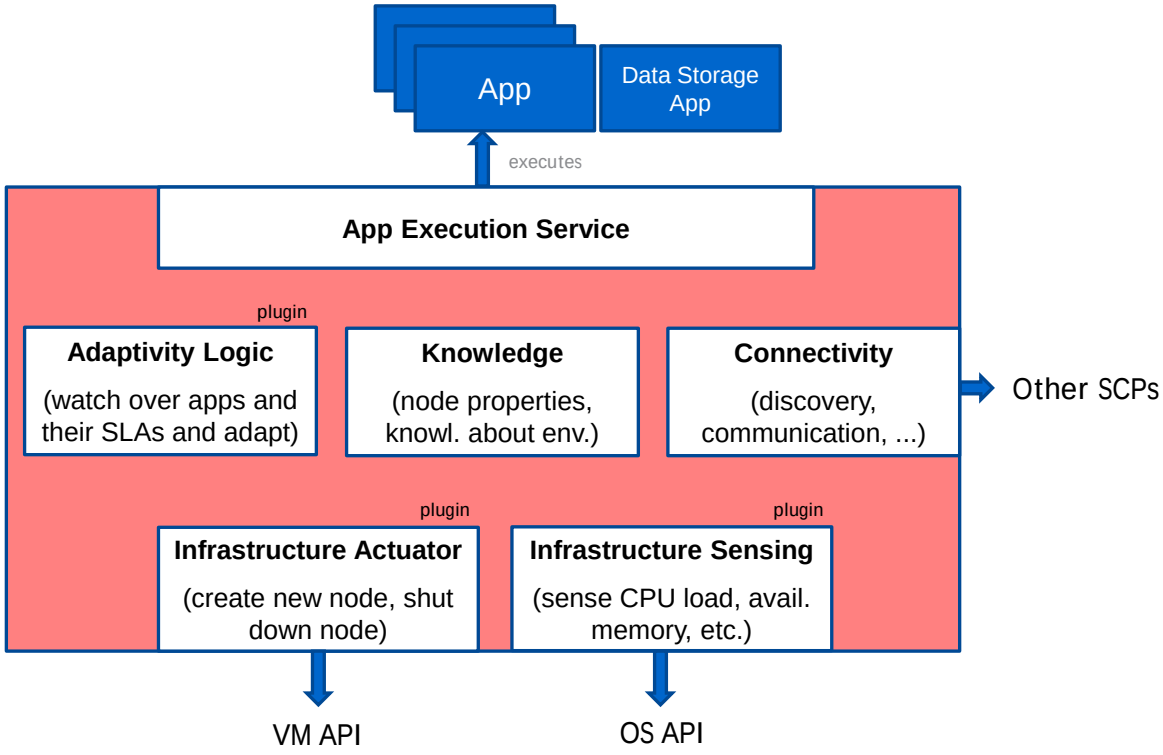

Figure 2: Functionality of a SCP instance

Firstly, each SCPi has *knowledge*. The knowledge consist of what the SCPi knows about itself (properties set by developers), about its infrastructure (CPU load, available memory), and about other SCPis (acquired through the network). Since there is no global coordinator, each SCPi must build its own world view and act upon the knowledge available. The SCPi may acquire knowledge about its infrastructure using an *infrastructure sensing plug-in*, which provides information about static values such as processor speed, available memory, available disk space, number of cores etc. and dynamic values such as currently used memory, disk space, or CPU load.

SCPi properties are important when specifying conditions (Service Level Agreements, SLAs) for the applications. For example, when looking for a new SCPi to execute an application, low latency between the SCPis might be interesting. Other requirements may be harder: For example, an application may simply not fit on an SCPi because of space; another may require a certain amount of memory.

Secondly, each SCPi has a *connectivity component* which enables it to talk to other SCPis over the network. The protocol followed by these communications must enable SCPis to find one another and establish links, for example by manually entering a network address or by a discovery mechanism. Furthermore, SCPis must be able to query others for knowledge and distribute their own. Finally, the protocol must support exchange of data and applications.

Since the Science Cloud Platform is based on both universities as well as individuals providing resources, scientists are involved on both the PaaS and SaaS levels. We imagine that scientists are able to start, stop, and configure SCPis and write applications for the SCP (PaaS level), but also to simply use them (“end user” or SaaS level).

As indicated in the previous section, an SCPi is adaptive and can react to conditions such as overload, shutdown of other SCPis, etc. Furthermore, it must watch over the apps executed and guarantee their SLAs. This functionality is performed in an *adaptivity logic* component. The adaptivity logic is exchangeable, application-independent, and has a direct relation to the SLAs of applications. We can image that for different test cases within ASCENS, different SLAs and matching adaptivity logic implementations are used. The adaptivity logic itself can be written in a standard programming language or custom domain-specific languages or rules. The adaptivity logic can adapt the SCPi on two levels:

- *On the platform level*: Applications may be moved between SCPis, thus balancing load or compensating for lost SCPis,
- *On the infrastructure level*: If the SCPi runs on an IaaS, an API may be available for creating new resources (virtual machines, virtual networks) and shutting them down. This API is provided, if available, through an *infrastructure actuator plug-in*.

Finally, each SCPi provides the application execution service to upper levels. The applications run on the platform must implement some API for the platform to be able to work with them (i.e. starting, stopping, working with data, etc.). One example of an app is the data storage service which allows users to store data in the cloud.

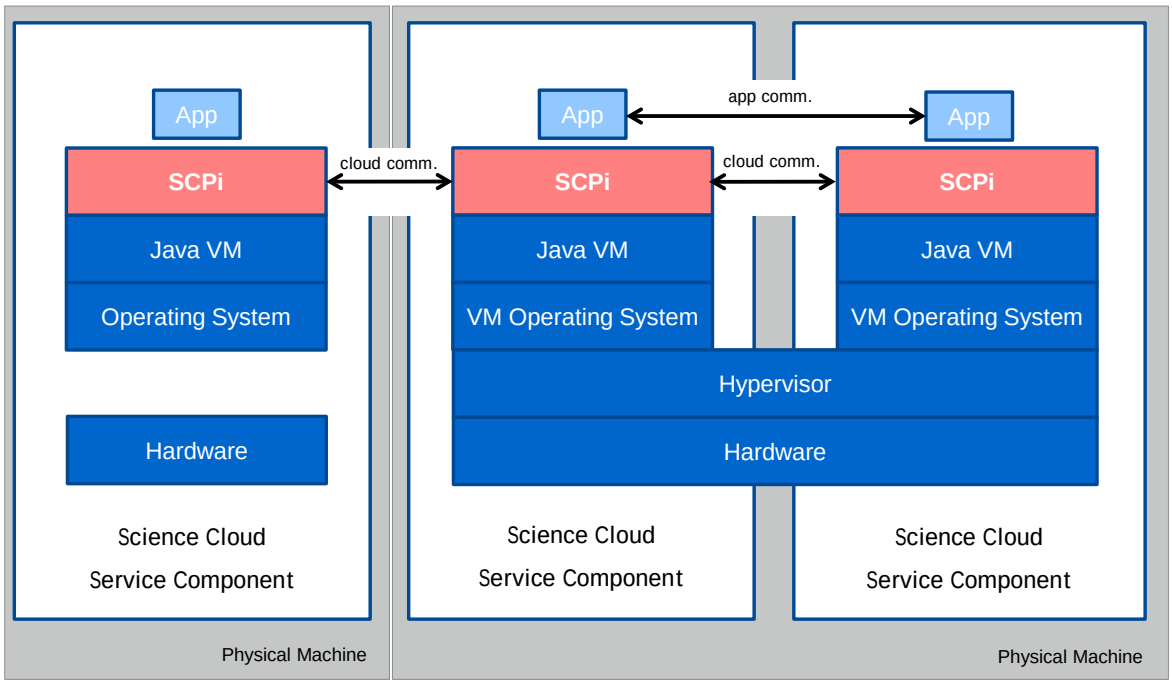

Figure 3: SCPi communication

Each SCPi is running on a stack of other software, which includes the Java virtual machine, an operating system, possibly a virtual machine and the Hypervisor the machine runs in, and actual hardware. As shown in Figure 3, communication as discussed in the previous section takes place between the individual SCPis; additionally, applications may also choose to communicate on upper levels.

A *Science Cloud Platform Ensemble* (SCPe) consists of individual SCPis based on a set of properties of the SCPis and/or the SLAs of applications. We usually says that an ensemble consists of SCPis

which *work together to run one application in a fail-safe manner and under consideration of the SLA of that application*, which may require a certain number of SCPis, a certain latency between the parts, or have restrictions on processing power or on memory.

At runtime, an ensemble may gain new SCPis or lose them depending on the behavior of the SCPIs itself and also on the load generated by the application itself or other applications running on the SCPIs.

2.2 Application Execution Service

All scenarios in the case study involve application distribution and migration. Seen from a high level, the *application execution service* provides the ability to run an application on the science cloud platform. Since an application provides a SaaS solution, it is intended to be used by "end users".

For developers, the service provides the ability to first write and then deploy an application (i.e. a piece of executable Java code) on the Science Cloud Platform, which is done by using the interface of one SCPi to upload a file which includes the code, data, and SLA meta information. An application can be either a singleton (i.e. run on only one SCPi at a time) or a multi-instance application (running on multiple SCPis in parallel). In either case, the cloud ensures that if a SCPi the application runs on goes down, it is restored and run on another SCPi. Application developers must follow three simple rules:

- The application must implement an interface of the SCP which enables the SCPi to request startup, shutdown, snapshots etc. This includes a means of addressing the application via a web interface: Each app must be accessible via a web site.
- All applications need to store their data in a place specified by the platform and be able to resume from snapshots of this data. Snapshots are requested by the platform. If an SCPi fails, the last backup-ed snapshot is used to start the application on another SCPi.
- Multi-instance applications (which support and request in their SLA multiple running instances) must coordinate their data via application-level communication. Snapshots are provided here as well but are only used if a SCPi fails as above. If a new instance is created, it is started from scratch and must coordinate with other instances to identify which part it has to fulfill.

Performance-based migrations or adding additional instances will be performed by the platform. The SCPi sensor part and the infrastructure measure performance as CPU load, memory load, disk space load, network usage, or other measures.

Based on node overload, new application instances are created (if possible) or applications moved to another SCPi (if the overloaded SCPi runs multiple applications). However, if a link overloads, moving an application or data out of a node with an overloaded link will be problematic. A possible adaptation strategy here might be to shut down the node, waiting for another one (with a possibly bigger link) to take over.

2.3 SCP Self-Adaptivity & Autonomy

The science cloud provides examples of adaptive ensembles and autonomous service components: Each SCPi is considered to be autonomic in the sense that it may join and leave the cloud at will. The science cloud is thus a dynamic cloud and works without a central coordinator in a peer-to-peer fashion. Still, the cloud must addresses the following issues:

- *Fail-Safe Operation*: Continue working if a SCPi fails, i.e. applications are still running. This requires some setup before an actual failure occurs.

- *Load Balancing / Throughput*: Parallel execution if the load is high, but not before that.

- *Energy conservation*: Shutting down virtual machines or de-configuring virtual networks if not required (this feature requires IaaS support).

To address these issues, each SCPi must continually monitor itself. In general, we can imagine the following adaptivity triggers and resolutions:

- *SCPi fails, disappears, or appears*: Failing SCPi attempts to notify other SCPis; other SCPis must take over responsibilities. If a new SCPi appears, take over applications.

- *SCPi or link with high load, or idle*: Move applications to another SCPi, receive applications from another SCPi, or create a new SCPi in a VM. If an SCPi is idle, take over applications from another SCPi, or shut down.

Through monitoring itself and its environment and the ability to adapt the behavior of the SCPis, the science cloud is *self-adaptive*. The individual self-* attributes are:

- *Self-Healing*. The aim of the science cloud platform as a whole is providing application execution. If an SCPi fails or is shut down, the applications executing must be made available / resumed elsewhere. Precautions are taken beforehand such that this is possible.

- *Self-Configuring*. Each SCPi stays conscious of newly added SCPis or disappearing SCPis in the network and adapts itself to make use of newly available resources or disappearing resources.

- *Self-Optimizing*. If an SCPi reaches its capacity (consistent high CPU load or swapping), it may transfer some of the load to another SCPi. The same applies to overloaded links in the network.

Obviously, these properties can not be provided by one instance of the Science Cloud Platform alone; multiple SCPis have to collaborate in order to make this a reality.

2.4 Summary

In this section, we have discussed the Science Cloud Platform (SCP), which consists of individual instances (SCPis) forming ensembles (SCPes). The SCP is built on autonomous nodes (requirement 4 in [vRA+11]), shows robustness under unstable conditions (requirement 5), can adapt to changing environments (req. 6) which includes utilizing an underlying IaaS if available (req. 7). Being based on Java, the SCP is able to run on various operating systems (req. 8).

The specification, execution, validation, and evaluation of the Science Cloud Platform instances (SCPis), the Science Cloud Platform Ensembles (SCPes) and the proper adaptation mechanisms used to ensure the functionality described in this chapter are further detailed in the next chapter.

3 Implementation: Component Properties, SLAs, and Adaptivity

In this chapter, we discuss some of the implementation details of the Science Cloud Platform which we require in the next chapter for describing the scenarios of this case study.

3.1 Node and Link Properties

There are *two basic resources* in the science cloud: (Virtual) machines and (virtual) network links. On a more detailed level, each of these provides more fine-grained resources to the SCPis, such as processor speed, memory, or bandwidth. In the following, we describe these resources on an abstract level. Theoretically, most of these resources might be considered to be modifiable, i.e. we can imagine that we can add to, remove, or change these resources at will. In practise, however, we have to restrict adaptivity on the resource level to what the underlying IaaS solution provides.

Which resources are available is stored at each SCPi in a local knowledge base which can be imagined as a place where data can be stored and retrieved given a key or a query. The individual concepts stored on the platform are typed with meta-concepts in an ontology. An ensemble is a collection of SCPis according to some properties and does not store any knowledge itself.

We use the term *node* for an SCPi along with its infrastructure, i.e. a virtual or physical machine which has the following static properties:

- A unique identifier (UUID/URN)
- Number of processors and processor cores
- Speed of each processor
- Main memory available to the SCPi
- Disk space available to the SCPi
- Trust level (i.e. data security)
- Access rights (i.e. whether Internet access from this node is allowed, whether printing is allowed, etc.)

The following dynamic properties can be imagined:

- Current load for each processor
- Current main memory consumption
- Current disk space consumption
- Current energy consumption of the system
- Location of the node (specified by the user)

Each SCPi will run applications. Thus, we add the following properties:

- Number of applications running
- Current load of each application
- Current disk space consumption of each application
- Current memory consumption of each application

A novel feature of the Science Cloud will be to also take the network into consideration. A connection between two nodes in the above sense is termed a *link*. Since we cannot represent the links as first-order citizens, we attach information about the links to the nodes:

- List of links available (maximum)
- List of links currently in use

A link has the following static properties:

- The two nodes linked together
- Maximum speed
- Link type (physical, virtual)
- Hops (Number of traversed routers, not SCPis) between the two nodes
- Trust level (i.e. data security)

Dynamic properties can be imagined as follows:

- Current latency (one-way transmission delay)
- Current bandwidth (bit rate)
- Current load (i.e. bandwidth available)

Dynamically allocating and de-allocating resources (nodes and links) is dependent on whether an IaaS is available, and on the IaaS functionality. Additionally, we might consider changing resource properties from the lists above through the SCPis if this is required and sensible.

3.2 SLAs: Constraining applications

When placing applications in the cloud, they are deployed with a Service Level Agreement (SLA). The SLA specifies the environment in which the application may be run or which QoS levels must hold. The SCP must ensure these SLAs on deployment and by adapting and using self-* properties while the application is running on the cloud.

SLAs on the SCP may specify the following:

- Minimum disk space required
- Minimum memory required
- Performance: Minimum available processor power (cores times speed)
- Performance: Maximum request time (measured using a load function in the application)
- Maximum latency between application instances (if the application is running on more than one node)
- Minimum bandwidth between application instances (if the application is running on more than one node)
- Minimum bandwidth to the Internet
- Minimum Trust Level (where applications may be placed, or constraints on the links via which applications may be transferred)

- Location constraints (fix the location of an application)

SLAs are monitored by the adaptivity logic within an individual SCPis. As pointed out above, the SLAs and the adaptivity component must use the same concepts. The SCPi adaptivity component must take SLAs into account both as adaptation triggers (for example when CPU load is too high) as well as constraints on external adaptation (for example when a node fails).

3.3 Using IaaS support

The Science Cloud Platform can run on any operating system which supports the Java virtual machine. In general, however, these will be either physical machines such as server, desktop or notebook computers running a standard operating system (Windows, Linux, MacOS), or virtual machines created by a cloud IaaS solution (for example, the Zimory Enterprise Cloud). Links between the SCPis are established using standard TCP/UDP/IP networking and are normally based on hardware as well. However, depending on the IaaS solution, virtual networks (VPNs or VLANs) may be established which have special properties (for example, encryption).

SCPis may use IaaS solutions if available to perform the following tasks:

- Create a new virtual machine which runs the SCPi (and thus joins the cloud when booted).
- Shut down a virtual machine (to conserve energy).

When creating a new virtual machine, the following parameters can be changed:

- Basic VM properties (CPU, RAM, Storage)
- Network interfaces (number, IP range, VLAN)

The features available will of course depend on the IaaS solution and on the adapters available to the SCPi implementation.

3.4 Handling Adaptivity

The question of how adaptivity can be supported in ensemble systems is of central importance to the ASCENS project and is researched on various levels in the individual work packages. Thus, there will be different solutions for different concerns. As we have shown in Figure 2, handling the adaptivity in an SCPi is considered to be an exchangeable part of the implementation which ensures that experiments on the source level are possible.

The adaptation logic has the task of monitoring the SCPi and its environment, and acting to failures, additions, overload, or idling to satisfy the application SLAs while conserving energy. Thus, we have a sensing and an acting side to the logic.

On the *sensing* (or *monitoring*) *side*, the adaptivity logic has full access to the knowledge base of the SCPi, which includes both the static and dynamic values we have discussed above. These can be contrasted against the SLAs of applications present, and against the overall goal of reducing energy consumption.

On the *actuator side*, the adaptivity logic must work with other SCPis to distribute applications as required by the SLAs. For example, if an SCPi experiences consistent high CPU load, another instance of a running application may be spawned on a second node. Furthermore, the adaptivity logic has access to the IaaS infrastructure (if available), which enables it to add or remove VMs.

The connectivity part of each SCPi (again, Figure 2) provides the ability to communicate with other SCPis to retrieve additional knowledge not (yet) in the knowledge base, and to request a move of application instances.

3.5 Conclusion

In this chapter, we have discussed some of the implementation details of the Science Cloud Platform. In the next section, we describe scenarios which can be used as a starting point for applying methods, models, and patterns in the ASCENS project.

4 Scenarios

In this section, we provide abstract scenarios for the science cloud case study which can be adapted as required for the individual techniques used. We discuss three scenarios:

1. Deploying a distributed application with an SLA to an existing cloud ensemble, i.e. to existing linked SCPis.
2. High load on one SCPi, which leads to a new node being created in an underlying IaaS and an application being moved there.
3. An SCPi goes offline, which means the applications there has to be made available on one or more other nodes.

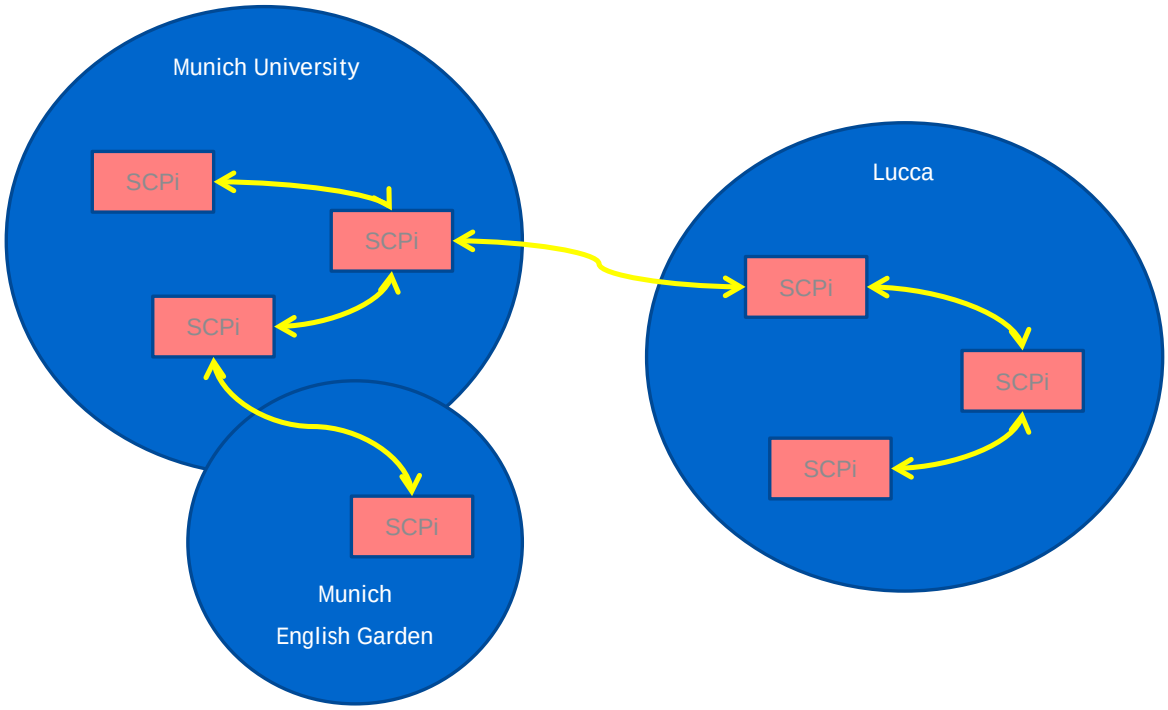

Figure 4: SCP scenario with seven SCPis

In all scenarios, we initially consider the following setup: A cloud is already deployed and running, but without any application deployed. There are seven SCPis in total: Three at LMU Munich, three at IMT Lucca, and one mobile device (in the English Garden in Munich) — see Figure 4.

The link between the machines inside LMU Munich and inside IMT Lucca has little latency and high throughput (under 50ms latency, 1 GBit/s throughput), while the link between the two cities has

a higher latency and less throughput (around 150 ms, 100 MBit/s throughput). The link to the mobile device has 400ms latency and 400 kbit/s throughput.

One of the IMT Lucca machines is a physical desktop machine, while the other two are virtual machines inside the Zimory Enterprise Cloud. In Munich, two machines are desktop computers and one is a VM. The mobile device is a notebook computer.

4.1 Application Deployment Scenario

On one of the machines at LMU Munich, a new application with the following SLA is deployed:

- Multi-instance application (at least 2 instances)
- Link between the instances must be under 50 ms in latency

Based on the latency requirement, the application instances may all be run at LMU Munich or in Lucca, but not both at the same time. It may not be run on the mobile node since two instances are required and the latency between the mobile node and the others is too high. Since the initial deployment is in Munich and the cloud is otherwise not loaded, the first SCPi will take one instance and start it. It requests a second instance in Munich to also load and start the application. This completes deployment.

4.2 High Load Scenario using IaaS

We consider the cloud again. This time, we have a singleton application which currently runs on one of the virtual machines at IMT Lucca. This application runs alone on its node and experiences consistently high CPU load. Since the application is a singleton, no additional instances can be spawned. Furthermore, all other machines feature the same amount of CPU power.

Thus, the adaptivity decision is made to spawn a new VM with more processing power and run the application there. The SCPi currently running the application instructs the Zimory platform to start a new VM. Once the VM is up, an SCP instance is injected into the machine and run. Once the instance joins the cloud, it is recognized and the application is moved there.

Note: This scenario can also be used without an IaaS if we consider the case that a new SCPi joins the cloud voluntarily. In this case, the new node may be chosen as the new place for the application.

4.3 Node Shutdown Scenario

Finally, we consider again our cloud. An application has been deployed on one of the desktop computers at LMU Munich with the following SLA:

- Singleton application
- Minimum CPU speed: 2 cores, 2 GHz each

As usual, this application is copied to at least one other (backup) node at deployment. All nodes in the cloud except for the mobile device are possible for this. While the application runs, regular snapshots are copied from the first node to the backup node.

Now, the initial deployment node in Munich fails without warning. Since the other deployment (backup) node is monitoring this node, it recognizes that the node and thus application is no longer available. It designates itself as primary node for this application and distributes the application to a new backup node. Then, it starts the application with the latest snapshot.

5 Conclusion

This document has described the science cloud case study of the ASCENS project. The aim of the case study is to provide a proving ground for ASCENS languages, methods, and tools in the cloud context.

The science cloud is a peer-to-peer platform-as-a-service which can run applications for users (thus providing SaaS), and is able to use an underlying cloud infrastructure (IaaS), among others the Zimory Enterprise Cloud. Depending on the application type, available resources, and SLAs, the Science Cloud Platform instances (SCPis) form ensembles (Science Cloud Platform ensembles, SCPes), whose nodes (SCPis) work together to keep applications running while keeping their SLAs.

In section 4, we have provided three scenarios which can be used to test-drive ASCENS methods against the case study. The scenarios can be extended and adapted as required by the task at hand.

References

[ea11] Martin Wirsing et al. ASCENS — Autonomic Service Component Ensembles v2.2. Framework 7 Project Description Part B, June 2011.

[vRA+11] Nikola Šerbedžija, Stephan Reiter, Maximilian Ahrens, José Velasco, Carlo Pinciroli, Nicklas Hoch, and Bernd Werther. D7.1: First Report on WP7: Requirement Specification and Scenario Description of the ASCENS Case Studies. ASCENS Deliverable, November 2011.