Project acronym:	**PrEstoCloud**
Project full name:	**Proactive Cloud Resources Management at the Edge for efficient Real-Time Big Data Processing**
Grant agreement number:	**732339**

D2.1 Scientific and Technological State-of-the-Art Analysis

Deliverable Editor:	**Quentin Jacquemart (CNRS)**
Other contributors:	**ActiveEon, CNRS, ICCS, NissaTech, Software AG**
Deliverable Reviewers:	**Dimitris Apostolou (ICCS), Baruch Altmann (LiveU)**
Deliverable due date:	**30/04/2017**
Submission date:	**11/05/2017 (original); 30/03/2018 (resubmission)**
Distribution level:	**Public**
Version:	**2.0**

This document is part of a research project funded by the Horizon 2020 Framework Programme of the European Union

Change Log

Version	Date	Amended by	Changes
0.1a	Apr. 10, 2017	Birkan Dag (Software AG)	Description of Software AG's Apama
0.1b	Apr. 13, 2017	Fabien Hermenier (CNRS)	Contribution regarding Cloud computing technologies and BtrPlace
0.1c	Apr. 13, 2017	Dimitris Apostolou (ICCS)	Contribution regarding Context-driven detection and situation awareness, and cloud adaptativity
0.2	Apr. 18-20, 2017	Quentin Jacquemart (CNRS)	Integration of partners' contributions
0.3	Apr. 21, 2017	Quentin Jacquemart (CNRS)	Added SOTA wrt. SDN and Virtual Networks
0.1d	Apr. 24, 2017	Iyad Alshabani (ActiveEon)	Contribution regarding Hybrid and Extended Cloud Management
0.4	Apr. 25, 2017	Quentin Jacquemart (CNRS)	Merged current version with contribution from ActiveEon
0.4b	Apr. 26, 2017	Quentin Jacquemart (CNRS)	Added document Introduction
0.4c	Apr. 25-26, 2017	Guillaume Urvoy-Keller (CNRS)	Restructured Chapter 5
0.1e	Apr. 27, 2017	Nenad Stojanovic (NissaTech)	Contribution regarding CEP, Siddhi, and workload prediction
0.5	Apr. 28, 2017	Quentin Jacquemart (CNRS)	Integration of NissaTech contribution into main document
0.6	Apr. 30, 2017	Quentin Jacquemart (CNRS)	Added Conclusion
0.6b	May 02, 2017	Quentin Jacquemart (CNRS)	Added Executive Summary section
0.6c	May 03, 2017	Quentin Jacquemart (CNRS)	Integrated Birgit Helbig's feedback
0.7	May 05-08, 2017	Quentin Jacquemart (CNRS)	Modified deliverable according to Dimitris Apostolou's comments
0.8	May 10, 2017	Quentin Jacquemart (CNRS)	Including Dimitris Apostolou's comments from 2nd review
1.0	May 10, 2017	Quentin Jacquemart (CNRS)	Finalized document
1.0b	Feb. 01, 2018	Birkan Dag (Software-AG)	Extended input for Apama
1.0c	Feb. 19, 2018	Quentin Jacquemart (CNRS)	New structure for document; marking down required changes for resubmission.
1.1	Mar. 09, 2018	Fabien Hermenier (CNRS), Pascal Urso (ActiveEon), Dimitris Apostolou (ICCS), Yiannis Verginadis (ICCS), Nenad Stojanovic (NissaTech), Giannis Ledakis (UBITECH), Panagiotis Gkouvas (UBITECH), Dirk Mayer (Software AG)	Input for revision
1.2	Mar. 12-23, 2018	Quentin Jacquemart (CNRS), Guillaume Urvoy-Keller (CNRS)	Integrating changes

Version	**Date**	**Amended by**	**Changes**
1.2	Mar. 27, 2018	Dimitris Apostolou (ICCS)	Internal review
1.3	Mar. 27, 2018	Quentin Jacquemart (CNRS), Guillaume Urvoy-Keller (CNRS)	Integrating changes wrt. internal review
1.4	Mar. 29, 2018	Andreas Tsagkaropoulos (ICCS)	Modifications following internal review
2.0	Mar. 29, 2018	Quentin Jacquemart (CNRS), Guillaume Urvoy-Keller (CNRS)	Integration and final changes

Contents

Executive Summary 8

Response to Reviewers 9

1 Introduction 12
1.1 Context and Motivation . . . 12
1.2 Project Ambition . . . 13
1.3 Deliverable Scope and Structure . . . 15

2 Hybrid and Federated Clouds 16
2.1 Introduction . . . 16
2.2 Management of Cloud Applications and Infrastructure . . . 18
2.3 Provisioning and Management at the Edge . . . 18
2.4 Distributed Applications Monitoring . . . 19
2.5 Amazon GreenGrass . . . 20

3 Context-driven Cloud Adaptivity 24
3.1 Context Detection & Situation Awareness . . . 24
3.2 Complex Events Processing . . . 25
3.3 Workload Prediction . . . 26
3.3.1 Existing Algorithms . . . 26
3.3.2 Scalable Predictions . . . 27
3.3.3 Conclusion . . . 34
3.4 Cloud Adaptivity . . . 34
3.4.1 Cloud Resource (re)Configuration . . . 35
3.4.2 Application Fragmentation & Refactoring . . . 41

4 Technological Assets 43
4.1 ProActive . . . 43
4.1.1 Active Objects . . . 43
4.1.2 Workflows and Scheduling . . . 44
4.1.3 Cloud Automation . . . 45
4.2 BtrPlace: Virtual Machine Placement under Constraints . . . 45
4.2.1 Context . . . 45
4.2.2 Industrial solutions . . . 46
4.2.3 Academic solutions . . . 46
4.2.4 Why BtrPlace . . . 47
4.2.5 Comparison with Kubernetes and Kubevirt . . . 48
4.3 Networking . . . 48
4.3.1 Software-Defined Networking . . . 48
4.3.2 Network Function Virtualization . . . 52
4.3.3 Inter-Cloud Networking . . . 53

4.3.4 Interconnection Technologies . . . 56
4.3.5 Towards an on-demand VPN service for PrEstoCloud . . . 57
4.4 Complex Events Processing Engines . . . 58
4.4.1 Apama . . . 58
4.4.2 Siddhi . . . 65
4.4.3 Comparison of CEP engines . . . 69

5 Conclusion **72**

Bibliography **75**

List of Figures

1.1 The PrEstoCloud conceptual architecture 14

2.1 AWS Greengrass high-level overview (source [207]) 21

3.1 Event stream processing 25
3.2 General Complex Event Processing 26
3.3 Storm topology (source: [22]) 27
3.4 Supervised algorithms (work with labeled data) 27
3.5 Unsupervised algorithms (work with unlabeled data) 28
3.6 Time series forecast example 28
3.7 Determining lags using ACF 30
3.8 Determining lags using PACF 30
3.9 Example of VAR usage for prediction of two parameters (source: [100]) 31
3.10 Example of an artificial neural network 32
3.11 Part of a recurrent neural network 33
3.12 *Unrolled* view of recurrent neural networks 33
3.13 LSTM cell 34
3.14 LSTM network and gradient flow 34
3.15 Summary of available mechanisms for holistic application scalability (source: [238]) 36

4.1 ProActive overview 44
4.2 SDN data packet treatment process 49
4.3 SDN network structure (source: [127]) 51
4.4 AWS VPC network architecture: use of an internet gateway to connect to the internet (source: [18]) 54
4.5 AWS VPC network architecture: use of a VPN gateway to connect to a personal infrastructure (source: [18]) 55
4.6 Azure networks connection models: Site-to-Site connection model (source: [42]) 55
4.7 Azure networks connection models: VNet-toVNet connection model (source: [152]) 56
4.8 The Forrester Wave: Big Data Streaming Analytics, Q1 2016 (source: [85]) 58
4.9 Apama Correlator 59
4.10 WSO2 CEP architecture 66
4.11 High-level architecture of Siddhi 67
4.12 Siddhi basic query 69

List of Tables

2.1 AWS Greengrass vs. PrEstoCloud 23
4.1 Comparison between CEP engines 70

Executive Summary

Deliverable name: D2.1: Scientific and Technological State-of-the-Art Analysis.

The goal of this project deliverable is to survey the state-of-the-art of the research literature in the areas of application to the PrEstoCloud project.

To this end, we first present the context of the PrEstoCloud project, as well as the scientific ambition of the project in terms of advancing the management of the infrastructure and the deployment of applications running in federated and hybrid clouds. In these circumstances, we also wish to optimize the use of the computing power available on edge devices, that, unlike most IoT platforms, would not be considered only as data producers, but also as active players in the processing of data.

Next, we review the existing methods to manage hybrid and federated clouds, as well as application monitoring on distributed systems, including IoT devices. We show that there is an evident lack of available standards between the multiple cloud providers platforms, and, therefore, no current best-common denominator.

Afterwards, we introduce the notions of *situation awareness* and *application context*, that PrEstoCloud will use as the basis to recommend application *reconfiguration*, i.e. to change the way the application is deployed over the whole infrastructure, in order to better optimize the use of the infrastructure. This review includes the techniques necessary to detect and predict application workload, and also the cloud adaptivity techniques, including horizontal and vertical scaling, and migration issues.

From this scientific state-of-the-art, we move towards the review of the technological aspects of PrEstoCloud. We discuss the set of tools that will be used as building blocks for the PrEstoCloud solution. This set includes mature software platforms available from within the consortium, such as ProActive (a cloud broker), BtrPlace (a scheduler under constraints), and Apama (a CEP engine). We also include extended discussions about available functionalities within public and private IaaS platforms, other available platform softwares (e.g. CEP engines and data brokers), and comparisons between them.

Finally, we detail how the sum of these elements can result in an advanced framework that will leverage application and device meta-data in order to finely match the infrastructure and its deployment to the available underlying platform, thereby extending the existing paradigms of cloud computing and edge computing, particularly in the context of applications processing big data streams.

Response to Reviewers

This Section lists the comments made by the project reviewers during the last project review, along with our answers detailing changes and pointers to corresponding sections.

COMMENT:

1. Chapter 3.3 and 4 overlap. BtrPlace is for virtual machine placement, in which context? Multi-clouds? Why is it not presented as part of Section 3.3? VM migration has several problems as discussed in the paper. Containers solve part of these problems. However, the project seems to rely on the BtrPlace which is for handling VMs. The decision of not using kubernettes is not convincing (coupled with the underlying cloud).
2. What are the features of BtrPlace for making it suitable for the project?

We have significantly reshaped the structure of the Deliverable in order to avoid having this overlap between the (previous) chapters 3 and 4. The previous document was organized following the original structure included in the Description of Work. The new structure delineates clearly the scientific state of the art from the technical one, where we describe some tools that will be underlying the PrEstoCloud platform.
The description and discussion about BtrPlace has been extended. Its positioning within the project and compared to direct competitors is detailed in Section 4.2.

COMMENT:

1. Regarding Complex Event Processing, only two systems are described, one with details while for the other a very high level overview is presented. There are many systems, most of them open source that are not even named, e.g., Storm, Flink, Kafka streaming, Spark streaming.
2. Does the critique to Storm (it does not exploit federated clouds) apply to other CEP technologies (e.g. Apama and others)?
3. Current CEP systems can be deployed on a distributed system for scaling query processing, Can Apama be deployed on top of a distributed system?

We have provided an overview of the main CEPs on the market, and provided a comparison among them in Section 4.4.

COMMENT:

1. The description of Apama is pure promotional material. Please provide technical details
2. Which kind of analytics are provided? There are some code snippets for Siddhi, but nothing for Apama Which kind of license does Apama have?

We included technical material in the description of Apama, so as to match the description outlined for Siddhi. It is available in Section 4.4.1.

COMMENT:

1. One of the mentioned drawbacks of Esper is that type of license is LGPL. The consortium claims that this is bad for exploitation. Why?
2. What license is going to be used by the consortium?

Considerations about software licensing used for the PrEstoCloud platform are outside of the scope of this deliverable. They will be discussed in the Deliverables related to exploitation.
As for the comment wrt. Esper, we just state that the commercial licenses are expensive.

COMMENT:

1. Explain the figures (what is showing Fig. 3.3? units? Legend?)

Figure 3.3 is now Figure 3.6. The labels have been added.

COMMENT:

1. Section 3.2 is very basic. Which technique is going to be used?
2. At some point windows are mentioned. Are these techniques going to be applied to CEP windows?

Section 3.2 is now Section 3.3. It has been extended to provide a description of more techniques, as well as adding a conclusion section detailing their possible usage.

COMMENT:

1. At the end of the document it is mentioned that 2 recommenders will be implemented. Which kind of recommenders?
2. Provide an example of the adaptation based on multiple streams.

Chapter 5 has been updated to clarify these issues.

COMMENT:

1. It is also mentioned that fragments of the data will be deployed across the processing topology what is the processing topology? Which kind of data intensive applications will you target? With data at rest?

PrEstoCloud will primarily target custom-made applications which are composed of microservices, i.e., small, independent processes communicating with each other by using language-agnostic APIs. Moreover, it targets data-intensive applications, i.e., applications which use a data parallel approach to process large volumes of data. We are going to focus mainly on applications that engage and process streaming

big data streams. Although out of these applications there will be data to be stored, we will not address any challenges with respect to optimized storage of data at rest.
The document has been updated to reflect this position more prominently.

COMMENT:

1. The work to be done on virtual networks does not present any challenge

We have updated and extended Section 4.3 to outline the challenges more prominently.

COMMENT:

1. Open questions are not always identified. Please provide them and some critical review of the different systems.

We have updated the sections within the main document body, as well as the conclusion, to better identify open questions and the way we intend to address them.

Chapter 1

Introduction

1.1 Context and Motivation

Cloud computing, and, in particular, the *Infrastructure-as-a-Service* (IaaS) model has profoundly transformed the way IT infrastructure and services are conceived and deployed. Through the use of *virtualization technology*, public cloud vendors – e.g. Amazon, Google, and Microsoft – have given their customers access to an accessible, customizable, and reliable platform to use as the back end for their applications. Moreover, due to its inner ability to scale on demand, seemingly infinitely, the **public cloud** has become the natural habitat for many *big data* driven applications.

However, running an application primarily in a public cloud – particularly a big data application – implies that the data to be processed needs to be transferred within the cloud boundaries. These data transfers may result in increasing service latency, which results in a low perceived *quality of experience* by end-users, on top of being expensive both in time and money for the application manager. Moreover, privacy constraints may prevent some business areas to rely on third-party platforms.

Following the success of public clouds, integrated software platforms, such as OpenStack, were made available so that private data centers could rely on the same technologies as public cloud. These data centers are privately owned – e.g. by a corporation, a university, etc – and are used locally. Because the size of the physical infrastructure is limited, these **private clouds** do not offer as much scaling ability as public clouds, but the latency of services is orders of magnitude lower as data locality is exploited.

While initially research focused on the operation of applications within a single cloud, i.e. confined to the physical boundaries of one data center, recent trends focus on optimal distribution of computational tasks between multiple clouds, leading to so-called **multi-cloud** applications. A popular example of multi-cloud deployments is the **hybrid cloud**, where a public cloud is used to supplement the computing power available within a private cloud when in need, e.g. during a burst peak demand.

At the same time, the recent proliferation of Internet of Things (IoT) devices results in a constant *steam* of data being transmitted from the edge of the network – where these IoT devices are connected – to a consumer application. While these IoT devices may be streaming data back to the consumer application directly, most deployments rely on a *data broker* that sits between the IoT devices and the application, to serve as a central relay point between all IoT devices and all (sub) components of the application. This deployment is known as a *publisher/subscriber* model, or *pub/sub* for short. All of these elements (the broker, the applications components) are commonly deployed in (private or public) clouds.

This approach is sub-optimal in the sense that data *always* needs to be transferred back from the edge of the network to a more central cloud before it can be processed. Consequently, the architecture completely disregards the ability of IoT devices to (pre)process data on their own. The **edge computing** paradigm was introduced to overcome these limitations. It aims to extend cloud services up to the edge of the network, i.e. include all devices, in particular those that produce the data, in the processing chain.

1.2 Project Ambition

The PrEstoCloud project is located at the *crossroads* between cloud computing, big data processing, and edge computing.

One of the main benefits of the cloud computing model is the elasticity of the infrastructure. It allows users to manage the size of the required deployment for their application. However, building an automated system that *accurately* matches infrastructural needs in order to avoid either over-provisioning or under-provisioning is still challenging. Over-provisioning means the underlying application infrastructure is too big for its needs, therefore needlessly increasing the hosting costs. Under-provisioning means the underlying application infrastructure is too small for its needs, leading to increasing response delays as the application load increases, resulting in a bad quality of experience for end-users.

The main ambition of the PrEstoCloud project is to provide a platform able to finely adjust the application deployment based on its needs. In a nutshell, PrEstoCloud aims to pro-actively adjust the used infrastructure and the application deployment in order to optimize the use of cloud and edge resources. To achieve this, PrEstoCloud relies on a number of pillars, that we introduce here:

- a **micro-service** architecture: PrEstoCloud will primarily target custom-made applications, made out of micro-services, i.e. small, independent processes able to communicate with each other using APIs;
- **code-level annotations**: the application source code, in particular, critical functions and micro-services, will be annotated by the developer with meta-data providing information with respect to its complexity, its deployment constraints, etc.;
- **deployment requirements**: application-level placement and scalability requirements are formulated by the application administrator (DevOps), e.g. to enforce Service Level Agreements (SLA) constraints;
- **context detection** and **situation prediction**: based on changes in the application-level data and meta-data, the PrEstoCloud platform will infer the application context in order to recommend adaptations to the deployment.

Please note that the main beneficiary target of the PrEstoCloud platform are custom data-intensive applications, i.e. applications which use a data-in-parallel approach to process large volumes of data, and in particular, engage and process big data streams.

Figure 1.1 depicts the high-level view of the architecture for the PrEstoCloud platform. It is composed of 3 main groups, which we call *layers*.

1. The **meta-management** layer includes components related to the decision logic for enabling cloud dynamicity. It includes
   - components related to situation detection, i.e. that infer the application context based on meta-data obtained from the infrastructure on which the application is running;
   - a component related to workload prediction, i.e. that predicts future incoming workload levels based on past application behaviour and current infrastructure state;
   - components that recommend adaptations to the infrastructure, e.g. the migration of a micro-service from a private cloud to a public cloud.

   The state-of-the-art of technologies related to these functionalities are presented in Chapter 3.

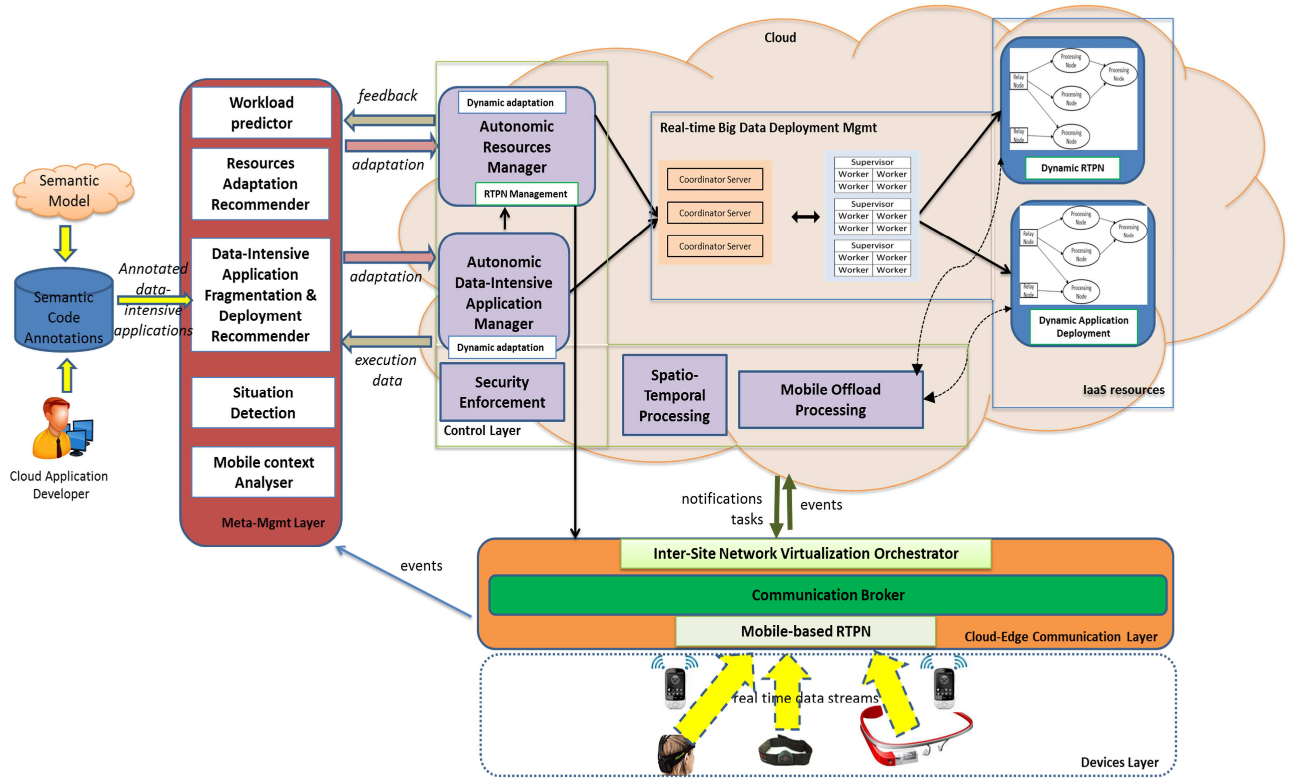

Cloud
Semantic Model
Semantic Code Annotations
Annotated data-intensive applications
Cloud Application Developer
Workload predictor
Resources Adaptation Recommender
Data-Intensive Application Fragmentation & Deployment Recommender
Situation Detection
Mobile context Analyser
Meta-Mgmt Layer
feedback
adaptation
adaptation
execution data
Dynamic adaptation
Autonomic Resources Manager
RTPN Management
Autonomic Data-Intensive Application Manager
Dynamic adaptation
Security Enforcement
Control Layer
Real-time Big Data Deployment Mgmt
Coordinator Server
Supervisor
Worker
Dynamic RTPN
Dynamic Application Deployment
IaaS resources
Spatio-Temporal Processing
Mobile Offload Processing
notifications
tasks
events
events
Inter-Site Network Virtualization Orchestrator
Communication Broker
Mobile-based RTPN
Cloud-Edge Communication Layer
real time data streams
Devices Layer

Figure 1.1: The PrEstoCloud conceptual architecture

2. The **control** layer includes components related to the commissioning, de-comissionning, and deployment of the application (fragments) onto either (public or private) cloud resources and/or edge devices. It includes
   - a cloud broker able to interface with a myriad of private and public clouds, and that is able to monitor the status of the deployed instances. The state-of-the-art related to this component is located in Section 4.1.
   - a scheduler able to solve placement problems under constraints. It is used to enforce the recommendations of the meta-management layers, and optimize the placement of all application micro-services across the global infrastructure. The state-of-the-art related to this component is located in Section 4.2.
3. The **cloud-edge communication** layer includes components enabling seamless communications between the distributed instances and components that are part of the infrastructure. It includes
   - a data broker used as an intermediate between the edge devices which generate data streams and the components of the PrEstoCloud platform, notably to implement pub-sub functionalities. The full details related to this component are located in PrEstoCloud deliverable D3.1 (*Communication Broker*).
   - an inter-site network overlay component responsible for creating a secured overlay network between the cloud(s) where the PrEstoCloud platform is deployed, the cloud(s) where the application micro-services are deployed, and the edge devices. The state-of-the-art related to this component is located in Section 4.3.

Please note that the full details related to the methodology behind the conceptual architecture presented in Figure 1.1, as well as full details about each of the individual layers and components are available as part of PrEstoCloud D2.3 – Conceptual Architecture.

1.3 Deliverable Scope and Structure

This Deliverable reviews the state-of-the-art of available technologies in the domains relevant to the PrEsto-Cloud ambition, summarized in Section 1.2. It is organized as follows:

- In Chapter 2, we discuss the challenges of federated clouds, reviewing efforts to handle and monitor applications that span over multiple (private and public) clouds. We also survey works that consider the management of edge/IoT devices, when the application managing these devices and consuming their data has a local presence (processing tasks are directly executed or offloaded on the edge device) and extends up the cloud. We put the emphasis on Amazon Greengrass, a recent initiative of Amazon Web Services to organize and manage IoT devices from the cloud. Indeed, Greengrass appears as a direct competitor to PrEstoCloud.

- In Chapter 3, we review existing works related to the core of PrEstoCloud, namely the adaptivity of federated cloud applications. As the meta-management layer of PrEstoCloud that will gear the adaptivity of the distributed application will have to process a huge amount of meta-data, we will rely of complex event processing technology to handle the meta-data stream. Next, we will focus on the key issue of workload predictions, that will help us to distinguish between transient issues and real trends that impose to reconfigure some parts of the applications. Lastly, we will review the existing literature on cloud adaptivity that encompasses many dimensions, including (i) horizontal and vertical scaling, (ii) live migrations of VMs or containers, and (iii) different strategies to trigger the configuration.

- Chapter 4 provides a technological overview of some tools that will be used to build the PrEstoCloud solution. We first introduce ProActive from ActiveEon, that will be the heart of the control layer of PrEstoCloud. ProActive enables to provision resources in public and private clouds and to execute workflows that lead to dispatching computing tasks on those resources. Next, we present BtrPlace, a VM placement algorithm that is not bound to any specific VM provisioning and management solution and enables to optimize the placement of VMs on resources materialized as physical server in a private data center. BtrPlace will be extended for PrEstoCloud to the case of federated clouds. We further position BtrPlace with respect to the existing literature and Kubernetes and Kubertvirt. The next section of Chapter 4 is focused on the networking layer, as we will need to be able to set-up an overlay on demand to interconnect all the resources in the private, public could and up to the edge. We introduce the Software Defined Networking (SDN) and Network Function Virtualization (NFV) paradigms that might be useful to build such an overlay. We also present the networking solutions offered by leading cloud providers (AWS, Azure, etc). The end of the Chapter is dedicated to the review of various CEP engines available on the market.

Chapter 2

Hybrid and Federated Clouds

In this Chapter, we discuss the challenges of federated clouds, and review efforts to handle and monitor applications that span over multiple (private and public) clouds. Moreover, we survey works related to the management of edge/IoT devices, when the application managing these devices and consuming their data has a local presence (processing tasks are directly executed or offloaded on the edge device) and extends up the cloud. Finally, we present Amazon Greengrass, a novel IoT platform available for use, and compare it to the foreseen PrEstoCloud platform.

2.1 Introduction

Over the last decade, the paradigm of **cloud computing** has considerably transformed the way IT infrastructure and service deployment is achieved. Through the use of hardware virtualization technology, previously prohibitively costly infrastructure could be reached thanks to the economies of scale achievable through multitenancy. Arguably, the three most popular service models for cloud computing are

- Software as a Service (SaaS): a specific software is configured, deployed, and available for use by end users, usually by connecting through the Internet to a specific interface (e.g. a web interface), or using a specific client application that relies on the network to interact with the back-end software (e.g. an application on a smartphone). The main advantage in this service model for the customer is the outsourcing of the application installation, configuration, management, and upgrade to the cloud provider.

- Platform as a Service (PaaS): a specific software environment is configured and available for cloud users to deploy they own application within this environment. The main advantage in this service model for the customer is to not to have to worry about the underlying, possibly complex configuration of the host operating system and the dedicated software environment.

- Infrastructure as a Service (IaaS): full access to the virtual environment is available to the customer, who can decide to install and configure all the software environment they please. Typically, virtual machines can be commissioned, configured (in terms of virtual hardware), and decommissioned through the cloud's interface (usually, a web interface), or via a custom API for automation purposes. The main advantage in this service model is the full access to the infrastructure, giving complete freedom over the type, amount, and configuration of the machines, without needing to invest in dedicated hardware upfront.

Moreover, in all cases, a key element of cloud services is their ability to scale according to the customer's need. In the case of SaaS, it can be adding users on the fly, e.g. adding a new email account to an email service. In the case of PaaS, it can be to increase the available resources to the application, e.g. increase the amount of workers for a Tomcat application. In the case of IaaS, it is the ability to either increase the number of virtual machines instantiated within the infrastructure, or to change the specification of the virtual machines (e.g. add RAM, storage, etc). For the remainder of the document, we primarily consider the IaaS service model.

These services were first popularized by Amazon, with a service called *EC2*; since then renamed *Web Services*. Amazon gives its customers the ability to commission and decommission virtual machines, which can then be customized, e.g. connected to the Internet to host services, or process data stored within the Amazon EC2 storage space. Because the physical infrastructure deployed by Amazon is so large – dozens of geographically dispersed physical data centers – the infrastructure is shared among multiple customers. Hence, these clouds are **multi-tenants**. Multi-tenancy is an important characteristic that allows the cost of the physical infrastructure to be shared among all customers. Amazon pays the upfront cost of the initial investment for the location and physical hardware, and amortizes this cost by renting out computer powers to *all* of its customers.

Moreover, due to the gargantuan size of the underlying physical infrastructure, public clouds offer seemingly limitless scaling possibilities: customers are able to request more storage, RAM, CPU, etc, without ever appearing to have hit the physical wall of feasibility.

Because cloud customers do not have information regarding the physical deployment of the data center, the cloud *manager* is in charge of *scheduling* all customers' tasks onto physical hardware, i.e. to choose a physical machine (that the customer does not know about and cannot choose) that will execute the customer's virtual machine. Consequently, a single hardware equipment will be used to power virtual machines from multiple customers. This kind of cloud service is referred to as a **public** cloud, and is characterized by multi-tenancy alongside reliance on virtualization. Nowadays, public IaaS clouds are many, from multiple vendors. To name a few: Google, Microsoft, Oracle, etc.

From public clouds, the virtualization technology has gradually spread to **private** clouds. Private clouds rely on the same virtualization principles as public clouds, but are mono-tenant. Usually, the physical infrastructure is hosted locally, e.g. by a corporation, a university, etc, and used for their own needs. The advantage for private clouds to rely on virtualization technology in this situation is to enable a better overall use for the infrastructure, as the physical hardware can be shared better among competing processes, and also have better isolation, which leads to better resilience. Unlike public clouds, private clouds do not appear to scale infinitely, and are usually sized to withstand a specific workload, estimated high-enough to meet most of the demand.

More recently, the trend has been to consider the cloud as a generic entity, regardless of the physical location of the infrastructure. The application is then run over multiple clouds. This situation is called **multi-cloud**. This category may be divided into four further subcategories:

- *community* clouds, which mutually share the infrastructure;
- **hybrid** clouds which rely on the use of at least one public cloud and at least one private cloud [26];
- **federated** clouds, which, like the community clouds mutualize infrastructure costs, but, in addition, provide a unique interface as entry point.

At the same time, Bonomi et al. [32] identify latency and jitter as a dominant concern in systems that require rapid response. Therefore, Zhang et al. [257] consider placement of applications on geographically distributed clouds. Nearby clouds generally perform better in terms of latency; this consideration eventually leads to the conception of fog computing.

Fog computing is a sub-paradigm included within **edge computing**, whose aim is to extend cloud computing and services to the edge of the network. The distinguishing characteristics of edge computing are its *proximity* to end-users, its dense geographical distribution, and its support for mobility. Services are hosted where they are used: at the network edge, and even on end-devices such as IoT devices, or access points. By hosting services locally, service latency is reduced and the quality of service (QoS) is improved, resulting in superior user-experience [156]. This paradigm is ideally suited for industrial applications, since in-network processing can improve energy efficiency, and data delivery reliability, due to the reduced communication and congestion [136], [191].

2.2 Management of Cloud Applications and Infrastructure

One of the main benefits of the cloud computing model is the elasticity of the infrastructure that allows the user to finely manage the size and configuration of their computing fleet. This adaptation can be driven by indications, coming either from the user, or from an automated system. One great challenge of cloud computing is to build such an *automated* resource *allocator*/de-allocator, that will allow to accurately match the actual usage in order to avoid under-provisioning or over-provisioning. Under-provisioning leads to performance issues and additional latency; and over-provisioning uselessly increases the user’s bill, as well as poorly exploit the cloud resources. As an alternative to allocating and de-allocing, an option to optimize resource usage is to *reconfigure* the resources allocated to the application. This was explored in [125] for Amazon EC2. It also requires being able to accurately model variations in the workload, which is not an easy task, particularly for public clouds, where only few trace data are available. Reiss et al. [200] recently paved the way to model a wokload dynamicity that is experienced in (public and private) heterogeneous cloud computing platforms.

Duplyakin et al. [61] propose a multi-cloud environment to process user requests. In this system, the users specify the percentage of the resources to be used in each cloud computing environment. If the user preferences are not satisfied due to a lack of resources, the system will balance the load progressively on the already-deployed instances until satisfaction of the user requirements. This approach allows the best possible use of hybrid clouds, but requires an intrusive solution installed inside a virtual machine.

Kailasam et al. [116] position themselves in the High Performance Computing (HPC) domain, and consider the optimization of the execution time in a hybrid cloud context. They propose three heuristic-based scheduling methods that adapt themselves to the evolution of the resources of the workload and the availability of the clouds. This approach allows the use of hybrid clouds in the context of HPC, but requires modifications to the application, or, more specifically, to the application’s task scheduler.

Leitner et al. [139] propose a model that enables running applications to burst into a different cloud infrastructure. The authors propose a framework for the creation of elastic cloud applications. This framework enables the monitoring of the performance and decide when to burst to a public cloud and in the other direction where to consolidate into the private cloud, but this approach needs the re-writing of the applications.

Jung et al. [115] target the optimization of the performance of the analysis of a huge-volume-and-loosely-coupled data in a distributed computing environment. Data is divided into pieces that are analyzed by an algorithm, which determines the nodes that will be used to process each data block. This approach is interesting, but very specific to Big Data environment.

Other research works have been achieved on the hybrid cloud, but focusing on economical aspects, e.g. [84], [87], [232], [233].

2.3 Provisioning and Management at the Edge

With the emergence of IoT applications and devices, it became evident that the scalability and elasticity needs of the applications and infrastructure varies widely from the enterprise applications to the embedded systems. However, in a lot of approaches, the application itself is implemented on the centralized cloud infrastructure and IoT technologies are only used as data providers, not as possible distributed or in-network processing nodes [136]. As an alternative, Laukkarinen et al. [137] propose the embedded cloud as a method of distributing processing and expanding resources between different IoT technologies. The authors focus on the heterogeneity challenge and homogenize data accessing and processing of heterogeneous IoT technologies for the end user.

However, the cloud management for the IoT platforms is critical and still presents issues. AWS IoT and Bluemix IoT Solutions are based on non-flexible, layered architectures [4] where the bottom layer includes the IoT deployment; the communication framework is managed by a middleware layer that provides access to the low-level hardware to the top level application layer. This application executes the business logic and processes the sensor data [142]. Such architectures imply that the business logic is executed only in the application layer and the IoT devices should be deployed with their own software, able to provide data streams to read from [250]. However, in practice, the provisioning of IoT devices is performed manually, which makes it difficult to react to changes, both in terms of infrastructure, and with respect to application requirements. Still, some specific IoT devices, known as gateways have emerged (e.g., Intel IoT gateway, Smart Things Hub, and Rasberry Pi). They offer functionalities in terms of processing, storage, and memory resources that can be considered as execution environment. These units could participate to the whole system, and be used for processing by specific offloading mechanisms.

Additionally, Sehgal et al. [206] address the problems of managing resource-constrained devices, which are often used for building IoT solutions, by adopting some network management protocols. INOX [46] is a robust and adaptable Platform for IoT that provides enhanced application deployment capabilities by creating a resource overlay to virtualize the underlying IoT infrastructure. In the literature, an additional abstraction layer located on top of the IoT infrastructure is frequently used. It allows keeping the underlying infrastructure untouched when deploying an IoT solution, e.g. [140], [141], [157].

Given the chaotic and ad-hoc nature of wireless IoT environments, fault-tolerant cloud management becomes critically important for IoT-based edge/fog computing systems. This is especially true for industrial applications, which require high degrees of reliability. Proactive fault tolerance can be achieved through two complementary mechanisms. First, multi-path networking can mask network failures: multiple parallel network paths are active simultaneously for the communication between two network nodes. Second, node replication can mitigate node failures: a virtualized node is replicated onto multiple physical nodes. Until recently, researchers assumed only one node or link would fail at the same time [92]. When the probability of multiple failures happening simultaneously is not negligible, the entire distribution of failures must be considered. Recently, Spinnewyn et al. [222] developed a mathematical model for availability in such environments. In this work, the placement algorithm combines node and link replication for each deployed service. This combination is realized by placing multiple duplicates of one and the same service component. The service is then available when at least one of its duplicates is accessible.

2.4 Distributed Applications Monitoring

In [7], the authors give a comprehensive state of the art of the existing commercial monitoring systems such as Monitis [155], Uptime Cloud Monitor [101], LogicMonitor [146], Nimsoft [225], Nagios [160], SPAE by SHALB [210], CloudWatch [11], OpenNebula [174], CloudHarmony [47], Windows Azure FC [226]. The conclusion of this study is that most of the solution lack standards format and metrics for the cloud layers.

Several monitoring specifications, standards and tools have been devised. Some of them are Java Management eXtension (JMX) standardized in 2003 for monitoring and managing java-based applications, Simple Network Management Protocol (SNMP) widely used for monitoring and detecting anomalies on network devices and systems (e.g. switch, router, server, etc.), Nagios [160] for the supervision of applications, servers, network devices, business processes, etc., or again Opsview [181] for the monitoring of network, physical, virtual and cloud-based servers, application server, etc.

More specifically to ESB solutions, [192] follows JMX standards to propose a service-monitoring framework for ESB-based services that allows getting more information on hosted services (number of invocations, results, response time, etc.) and processes (e.g. number of exchanged messages between a BPEL engine and services).

The proposal is designed to monitor various parameters while limiting the overhead. However, the proposed solution does not integrate neither the monitoring of the used underlying computing resources parameters (that can have an impact on QoS and scalability of the hosted ESB), nor a module for the detection or prediction of complex patterns (symptoms). In [192], authors propose a monitoring framework for ESB with a set of monitoring mechanisms that exploit JBI monitoring capabilities. The proposed solution is also based on JMX and allows getting information related to the qualitative ESB's parameters (e.g. uptime), exchanged messages (e.g. sent and received requests, sent and received replies, etc.), hosted services (invocations timestamps, response time, etc.) and processes performance. Similarly, to [255], authors of [192] do not integrate the monitoring of the underlying computing resources that can impact the QoS and scalability of the ESB. However, they exploit event processing technology functionalities for the detection or prediction of complex symptoms.

Regarding the correlation, aggregation and filtering of monitored data to identify or predict complex patterns, event-processing technologies are more and more used for tracking exceptional events or situations that can appear on various components [71]. Business Activity Monitoring (BAM) solutions are also introduced in order to gather, aggregate, analyze, correlate and present monitoring data that can come from many levels of heterogeneous enterprise systems. BAM solutions are generally based on event-processing technologies [192].

2.5 Amazon GreenGrass

It is important to the PrEstoCloud consortium to research on competing cloud technologies, and clearly understand the way that cloud computing, edge computing and fog computing continuously evolve. Towards this action, we analyzed, installed and tested the tools that Amazon AWS provides related to IoT and edge computing, as Amazon is one of the leading Cloud vendors [72] and usually considered as the leader and reference point for all competitors [237], on the IaaS market but also with an extensive portfolio of interconnected services.

One of the most related products to PrEstoCloud's vision regarding the combined exploitation of cloud and edge resources is the AWS Greengrass [207]. AWS Greengrass is a software that extends AWS cloud capabilities to local IoT devices, in order to collect and analyze data closer to the source of information, while also establishing secure communication among each other through local networks. Its main purpose is to enable IoT devices to rapidly (in near real-time) respond to local events and minimize the cost of transmitting IoT data to the cloud. For example, AWS Greengrass enables machine learning inference locally on Greengrass Core devices using models that are built and trained in the cloud. To achieve this, developers who use AWS Greengrass can author serverless code (AWS Lambda functions [209]) in the cloud and deploy it to devices for local execution of applications. In the following schema of an AWS Greengrass deployment, the envisioned deployment architecture is depicted.

AWS Greengrass introduces an architecture comprised of 3 basic components: the Greengrass Core, the Greengrass Group, and the IoT Device SDK. Any IoT device (running Linux and supporting ARM or x86 architectures) that uses AWS IoT Device SDK can be configured to interact with a Greengrass Core through a secured local network. Such devices along with Greengrass Cores and Lambda functions are organized into Greengrass Groups that correspond to collections of inter-communicating entities. These may conceptually represent physical areas that group IoT devices (e.g. a building floor or a house). Such Groups can contain up to 200 local devices.

The AWS Greengrass Core is a compute node hosted on a local device that bears the appropriate software to enable the use of local device resources like cameras, serial ports, or GPUs. Its objective is to allow the rapid access and processing of IoT data while transparently use AWS cloud resources for management, analytics, and storage. Specifically, it provides the following functionalities:

- Allows deployment and execution of local applications created using AWS Lambda functions, and mana-

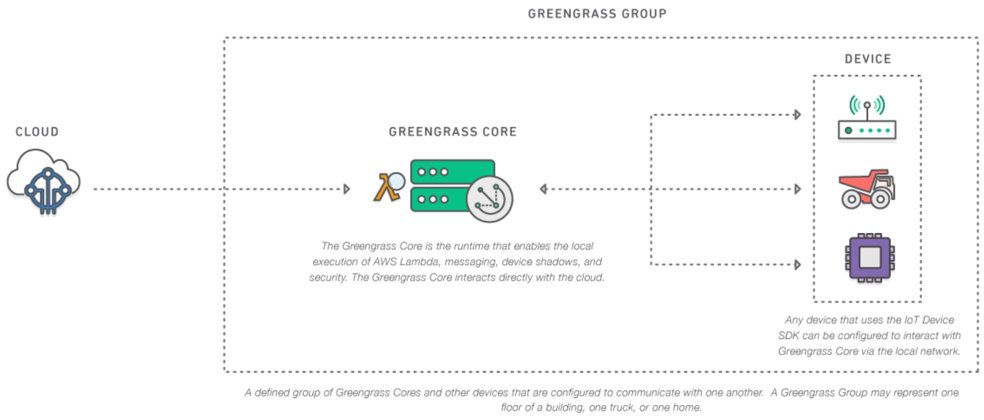

GREENGRASS GROUP
DEVICE
CLOUD
GREENGRASS CORE
The Greengrass Core is the runtime that enables the local execution of AWS Lambda, messaging, device shadows, and security. The Greengrass Core interacts directly with the cloud.
Any device that uses the IoT Device SDK can be configured to interact with Greengrass Core via the local network.
A defined group of Greengrass Cores and other devices that are configured to communicate with one another. A Greengrass Group may represent one floor of a building, one truck, or one home.

Figure 2.1: AWS Greengrass high-level overview (source [207])

ged through the deployment API.

- Enables local messaging between devices over a secure network using a managed subscription scheme through the MQTT protocol
- Ensures secure connections between devices and the cloud using device authentication and authorization.
- Provides secure, over-the-air, software updates of user-defined Lambda functions.

The Greengrass Core can pass messages between Devices, Lambda functions and even AWS cloud using the MQTT (Message Queuing Telemetry Transport) Protocol. These components can interact using pre-defined subscriptions. A subscription consists of a source, a target, and a topic, where the source is the originator of the message and the target is the destination of the message. This approach is related to the Communication Broker that PrEstoCloud envisions which also presents the basic functionalities of the traditional publish-subscribe paradigm. The AWS Greengrass Core SDK enables Lambda functions to interact with the AWS Greengrass Core on which they run in order to publish messages (using event topics), interact with the local Thing Shadows service (i.e. JSON-based "virtualization" of IoT devices), or invoke other deployed Lambda functions. This SDK is used exclusively for writing Lambda functions running in the Lambda runtime environment on an AWS Greengrass Core. Lambda functions running on an AWS Greengrass Core can interact with AWS cloud services directly using the AWS SDK. Regarding the usage of AWS Lamda functions, developers can select one of the pre-existing Lamda functions that are available, or have to develop their own on one of the supported languages (Java, Python, NodeJS) by creating specific code that implements the needed AWS SDK interfaces.

In addition, AWS Greengrass offers another advanced feature, relevant to PrEstoCloud, called Device Shadow. This allows AWS Greengrass enabled devices within an AWS Greengrass Group to interact by storing, retrieving or even modifying the state of another AWS device, using a JSON document. This feature is used in cases where communication between the nodes is interrupted, so that the device state is synced asynchronously. In intermittent communication scenarios, PrEstoCloud aims to provide a similar solution, but through the usage of the Spatio-temporal Library that can enable the ad-hoc communication between devices using a full-mesh topology unlike the AWS Greengrass where at least one device should be used as a hub. This solution will present greater flexibility and resilience in cases where there isn't only intermittent communication between the edge and the cloud but also among the edge resources.

In PrEstoCloud, we focus our research efforts into the creation of similar but also complementary functionality, in the sense that we aim to allow the execution of certain application fragments at the edge devices but also efficiently decide, at run-time, the shifting of processing tasks from edge devices to multi-cloud resources and vice-versa. We highlight the main three differentiations among AWS Greengrass and PrEstocloud.

First, AWS Greengrass, for obvious reasons, allows the interconnection of the edge layer with only cloud resources that are managed by Amazon (i.e. AWS). PrEstoCloud will enable the use of private or public clouds by considering in each case which is the most efficient cloud resource to use. Thus, PrEstoCloud will enable the common exploitation of edge and multi-cloud resources.

Second, in AWS Greengrass the execution of processing jobs (AWS Lambda functions) at the extreme edge of the network is bounded by the fact that the AWS Greengrass Core is statically preconfigured to be hosted on certain edge devices. The Greengrass Core acts as a hub that can communicate with a limited number of other devices, while the rest IoT devices (with AWS IoT Device SDK installed) are considered only as producers of events or data streams. Bringing the computation at the extreme edge on a per device level, and benefiting from the computational resources from cameras and mobile devices (wherever this is possible) is something that PrEstoCloud aspires to deliver and it is something that it is not available in similar solutions like AWS Greengrass.

The third important differentiation factor corresponds to the development of the AWS Lambda functions. The application developer is in charge for statically defining what can be executed on an AWS Greengrass Core and what on the AWS cloud. On the other side, PrEstoCloud allows the application developer and the DevOps to properly define their constraints but also provide hints about application fragments that can be executed either on cloud resources, edge resources or both. From that point on PrEstoCloud is in charge to decide during run-time (based on the current state of the resources used, the application fragments status and the current and the predicted workload) to offload/onload processing tasks on certain (or types of) edge resources to/from multi-cloud resources. We summarize our findings with respect to the differences and the complementarity of the two platforms in Table 2.1.

Besides AWS Greengrass that is based on the deployment of serverless functions (Lamdas) at the edge, using feeds of events delivered through MQTT protocol, by the IoT devices there is another relevant tool, called Amazon Kinesis [208]. This tool is relevant to PrEstoCloud since it focuses on efficiently ingesting real-time data such as video, audio, and IoT telemetry data for machine learning, analytics, and other applications.

Factor examined	AWS Greengrass	PrEstoCloud
Supported Cloud	AWS	OpenStack, AWS, Google Cloud
Computation@Edge	Yes, at Greengrass core devices (Raspberry Pi 3 is a typical example)	Yes, at local processing units but also at devices at the extreme edge that have adequate processing capabilities
Communication with Nodes	Through MQTT protocol, allowing IoT nodes to report their state	MQTT
Scalability Support	Yes (only through AWS resources)	Yes (using multiple cloud and edge resources)
Target Devices	Nodes are IoT that only provide content	Nodes can be some smart devices with bilateral communication
Dealing with connectivity problems	Device shadows; allows devices to synchronize when connectivity is restored; at least one hub node (Greengrass Core) should exist	Spatio-Temporal Library; when used, services on different devices can have ad-hoc connectivity; full-mesh topology support
Over-the air updates	Yes	Yes (through JPPF support [112])
Can be used at mobile phones	No, but phones can be used as IoT devices that produce data	Yes, PrEstoCloud agent shall provide monitoring information and potentially allow the local execution of predefined methods
Predictive scaling of resources	No	Yes
How distributed processing is achieved?	Execution of Lamda fuctions at the edge	Distributed processing using a framework like JPPF
Support for Streaming scenarios	Yes, but Kinesis is better suited for this	Yes
Dynamic Offloading/Onloading support	No (Lamda fuctions predefine what will be executed at the edge)	Yes

Table 2.1: AWS Greengrass vs. PrEstoCloud

Chapter 3

Context-driven Cloud Adaptivity

In this Chapter, we review existing works related to the core of PrEstoCloud, i.e. the adaptivity of federated cloud applications. The meta-management layer of PrEstoCloud will leverage a huge amount of meta-data to provide the adaptivity of the distributed application infrastructure. We will rely of complex event processing technology to handle the meta-data stream.

We will then focus on the key issue of workload predictions, that will help us to distinguish between transient issues and real trends that impose to reconfigure some parts of the applications. Lastly, we will review the existing literature on the many dimensions of cloud adaptivity including scaling possibilities and migration issues.

3.1 Context Detection & Situation Awareness

Situation Awareness (SA) refers to the "perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future" [68], [76]. To realize systems for Situation Awareness, individual pieces of raw information (e.g. sensor data) should be interpreted into a higher, domain-relevant concept called *situation*, which is an abstract state of affairs interesting to specific applications. The power of using "situations" lies in their ability to provide a simple, human-understandable representation of, for instance, sensor data [147]. In the context of dynamic computing systems, situation is defined as an event occurrence that might require a reaction [3].

In PrEstoCloud, we follow this definition and we will address situation awareness through complex event processing technologies capable of processing in real-time a large number of events generated by a variety of distributed cloud and edge computing resources as well as other data generating sensors.

Context, in general, can be described as useful and related information that can be used to characterize any entity and situation in a computing environment [196]. The notion of context-aware computing is generally the ability for the devices to adapt their behavior to the surrounding environment [2]. This topic has received more attention in recent years due to the fact that computing devices and computers, especially mobile ones, are being equipped with multiple and more accurate sensing capabilities than before. Context-detection refers to the capability of a system to be aware of its physical environment and to respond proactively and intelligently [1]. The use of context enables services and systems to cope with the dynamic nature of the Internet [215], [216].

To reflect the varying nature of context and to ensure a universal applicability of context-aware systems, context is typically represented at different levels of abstraction [149]. At the first level of raw context sources there are context data coming from sensor devices, or user applications. At the next levels, context is represented using abstraction approaches of varying complexity. The work by Bettini et al. [28] reviews models of context that range from key-value models, to mark-up schemes, graphical models, object-oriented models, logic-based models and ontology-based models. In [234] an ontological model of the W4H classification for context was proposed, including the five dimensions of context: Who, What, Where, When, and Why. In practice, we often distinguish context-aware systems in terms of types of applications: mobility context-awareness, location-awareness and time-awareness. With respect to edge computing contextual information, many research works focus on battery level, cell connection state and its bandwidth, WiFi connection state and its bandwidth, Bluetooth state, the congestion level of the connection (RTT) to VMs on the cloud, and the signal strength of cell and WiFi connection. (see for example [258]).

Relevant to the design of context-aware applications are modeling languages, which take context explicitly into account. The first such effort was ContextUML a UML-based modeling language which was specifically designed for context-aware Web service development and applies model-driven development principles; see [214]. ContextUML considers that context contains any information that can be used by a Web service to adjust its execution and output. ContextUML has been adopted for the development of a model-driven platform, called ContextServ, which is used to develop context-aware Web applications; see [216].

3.2 Complex Events Processing

In today's digital business age, enterprises are rapidly moving applications to an event driven model to achieve the speed, agility and responsiveness they need to remain competitive. At the same time, organizations are dealing with ever-increasing volumes of data from an ever-increasing number of data-generating sources. They need to do this so that they can respond to opportunities and threats by capturing the correct context and taking the most appropriate action. Orthogonally to the volume of data is the speed at which the data needs to be analyzed and the viable time window to act. Not only must they analyze "big data in motion" in "real time", they also need technology that delivers "performance at scale" by supporting a wide number of concurrent applications while consuming large volumes of changing data. And this must be done in real time so the business can respond to events immediately. Another important dimension to performance is in the speed of evolution. Deployed applications are commonly outdated in some way immediately after deployment; new threats and opportunities arrive second by second, and the ability to react to this simply and quickly provides an enterprise with a massive advantage.

Event-driven architectures enable digital businesses to detect changes in circumstances, discern the impact of those changes and respond quickly. These architectures enable businesses to exploit opportunities and forestall threats that are often nestled within events. These key attributes are summarized as "monitor, analyze and act". The events go through the CEP engine, where there are analyzed. The results are immediately usable.

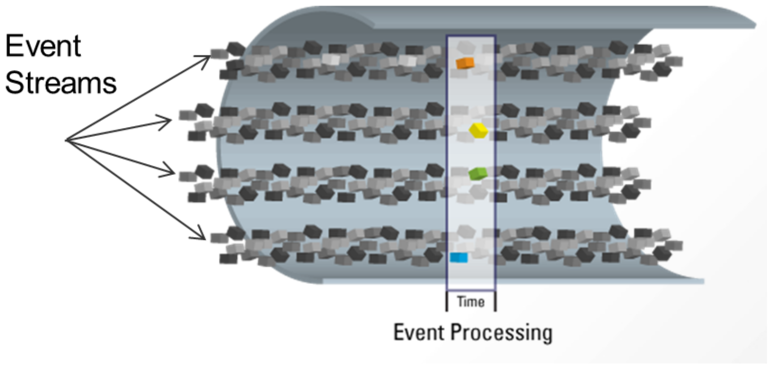


Figure 3.1: Event stream processing

CEP is already matured technology and the current commonly used solutions are there for many years. There were many acquisitions, so that the best engines are finally owned by best (software) vendors. Historically, CEP was seen as the real-time pattern detector, i.e. the system that increases so called "real-time situational awareness". The detection is done through the so-called CEP engine, i.e. a pattern-recognition software that does fast and in-memory matching between input streams and a pattern representing a situation of interest. Situations of interest are represented as complex event patterns, which are formalized in a language specific for the selected CEP engine. As input, Complex Event Processing takes streams of real-time data which can be very intensive, since the CEP engine is designed for very fast computing: a matching situation of interest should be detected within milliseconds. Although this definition is still valid, there is a change in the definition of the concept of "situation". Indeed, the "situation" has become more complex than that what can be usually described with a "traditional" complex event pattern.

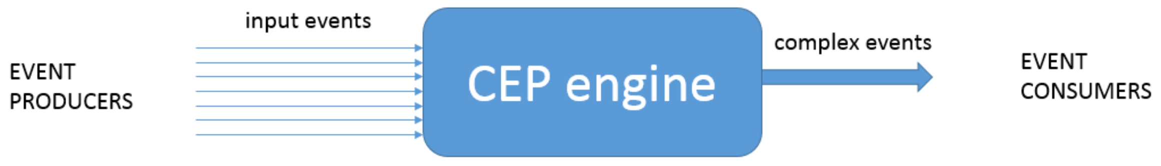


Figure 3.2: General Complex Event Processing

The general architecture presented in Figure 3.2 is usually suitable for detecting situations which are less challenging, being represented through relatively simple patterns (e.g. those that depend on a direct comparison of the values in input streams). Indeed, basic CEP brings the value for detecting interesting situations, but not really for creating real-time situational awareness, due to too simplistic representation. For example, it is possible to detect easily that the speed of a user is decreasing in a short period of time, but it is much more difficult to define when the number of users are slowing down in an (moving) area, although this latter request seems not much more complicated than the first one. Indeed, the difference is coming not from the complexity of the particular conditions, but rather from the complexity of the pattern as a whole. This is usually resolved through introducing processing pipelines which connect isolated computations into a complex detection process.

Event Processing Networks (EPNs) [73] are the next level of abstractions, which represent networks of processing elements that altogether generate a value for the user. They can be rather simple or have a more complex structures. The main issue is that not only one CEP processing step, but rather many of them can be connected, which increases the processing power, and also introduces a huge issue with the synchronization of different processing steps. Those are processed by event processing agents which create further events that are relevant to event consumers.

One of the modern processing structures which reflects EPN nature is Storm [22], illustrated in Figure 3.3. A spout is a source of streams in a topology. Generally, spouts read tuples from an external source and emit them into the topology. All processing in topologies is done in bolts. Bolts can do anything from filtering, functions, aggregations, joins, talking to databases and more. Bolts can thus do simple stream transformations. Doing complex stream transformations often requires multiple steps and thus, multiple bolts.

3.3 Workload Prediction

3.3.1 Existing Algorithms

There are several approaches for predictive monitoring in computing systems and equipment, both from the academic literature as well as from commercial vendors. An example of the latter is the Accenture predictive

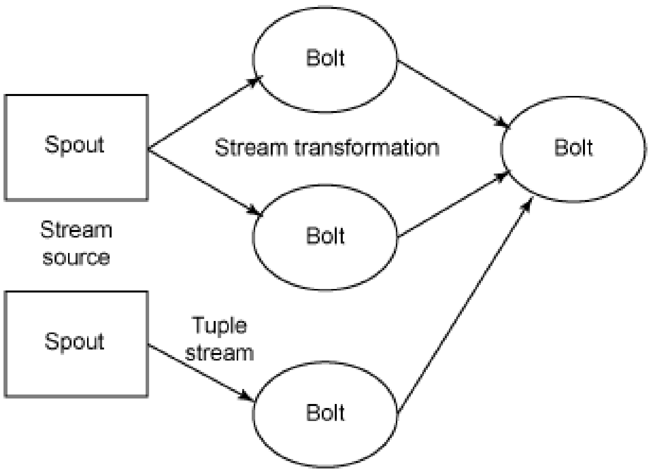


Figure 3.3: Storm topology (source: [22])

monitoring solution (www.accenture.com), which extracts sensor information from equipment, control systems and business systems, and then uses analytical models to predict equipment faults and suboptimal performance before the equipment fails. Dean et al. [56] proposed an unsupervised behaviour learning for predicting performance anomalies in virtualized cloud systems. Sharma et al. [212] addressed the problem determination and diagnosis in shared dynamic clouds, while Guan & Fu [86] developed an adaptive anomaly identification approach by exploring metric subspace in cloud computing infrastructures. In summary, there is an active research stream employing predictive models for detecting cloud infrastructure shortcomings or anomalies but there has not been to date an approach that combines predictive capabilities with the ability to recommend cloud resource adaptations.

3.3.2 Scalable Predictions

One of the main drawbacks of the prediction algorithms is the scalability. In this Section we provide an overview of algorithms which can be used for scalable predictions (in the case of big data scenarios).

Machine learning algorithms represent a group of algorithms which learn (create a model) out of data, based on optimization of some cost function [218]. We make a distinction between:

- supervised algorithms – which create a model using labeled data
- unsupervised algorithms – which create a model using unlabeled data

Examples of supervised algorithms (Figure 3.4) are classification and regression, while clustering and dimensionality reduction belong to the group of unsupervised techniques (Figure 3.5).

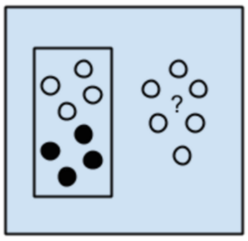


Figure 3.4: Supervised algorithms (work with labeled data)

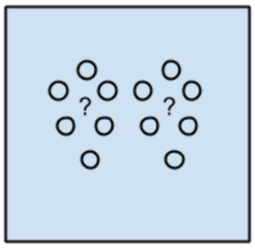


Figure 3.5: Unsupervised algorithms (work with unlabeled data)

Classification algorithms use labeled training data to generate a model which will be able to predict labels of unseen data. Algorithms such as Logistic regression, KNN (K nearest neighbors), SVM (support vector machines), decision trees and NN (neural networks) belong to this group.

Regression algorithms (such as Linear regression) try to predict values of dependent variable based on values of one or more independent (explanatory) variables.

Time series prediction (forecasting) represents an important area of machine learning with application in many different domains. Comparing to "traditional" (non time series) datasets, time series are specific in having dependence between observations – a time dimension. This time dimension provides structure and is a source of additional information that must be taken into account. An example of time series forecasting is given on Figure 3.6. The original time series is given in blue and is splitted into two parts, training part and test part. Predictions for the training data are given in green, while predictions on unseen data are given in red. We can see that predicted values are pretty close to the original values, implying that the prediction model is doing a good job.

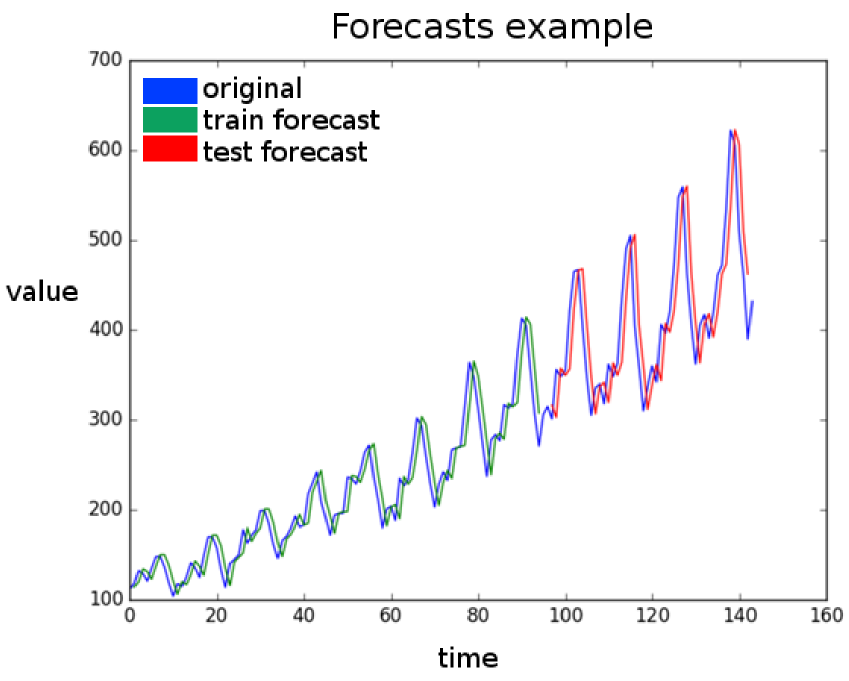


Figure 3.6: Time series forecast example

Time series can be decomposed into 4 components:

- Level: the baseline value of a series

- Trend: could exist and represents linear increasing or decreasing behavior of the series over time
- Seasonality: could exist and represents repeating patterns of behavior in the series
- Noise: could exist and represents variability which cannot be explained by a model

Hence, time series values can be presented in the following way:

$$\text{time series} = \text{level} + \text{trend} + \text{seasonality} + \text{noise}$$

3.3.2.1 ARIMA

A popular and widely used statistical method for time series forecasting is the ARIMA model [34].

ARIMA stands for Autoregressive Integrated Moving Average and it is a generalization of AutoRegressive Moving Average and adds the notion of integration. Features of the model are:

- AR – Autoregression, uses dependent relationship between time series observation and a certain number of lagged observations
- I – Integrated, uses differencing of raw observations (data values are replaced by difference between their value and their previous value)
- MA – Moving Average, uses dependency between an observation and a residual error from moving average model

Parameters of ARIMA model are:

- p – the number of lag observations included in the model
- d – differencing degree (number of times observations are differenced)
- q – moving average window size

Combination of these three concepts makes this a very powerful algorithm. Results are also more intuitive than in case of neural networks, which are usually treated as black boxes. It is also possible to eliminate one of this steps, e.g. ARIMA (1, 0, 1) would perform the same as ARMA, that is, without differencing.

3.3.2.2 Seasonal ARIMA

Seasonal ARIMA [185] adds the notion of seasonality to standard ARIMA, making it even more powerful. It enables not to look only at last N consecutive values, but to observe values from last M periods (e.g. at the same time last day, week or month). This adds 4 new parameters to the model, so now we have

$$\text{ARIMA(p, d, q)(P, D, Q)}_\text{m}$$

where $\text{(P, D, Q)}_\text{m}$ corresponds to seasonal part and m is the number of periods per season. It is always challenging to determine parameters of a machine learning model. Concretely, in the case of ARIMA and Seasonal ARIMA models ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) diagrams are used (figures 3.7 and 3.8, respectively).

These plots can be very useful and informative, but do not provide a concrete answer about the parameters of the model. For that reason numerical methods such as the ones based on Akaike Information Criterion were developed.

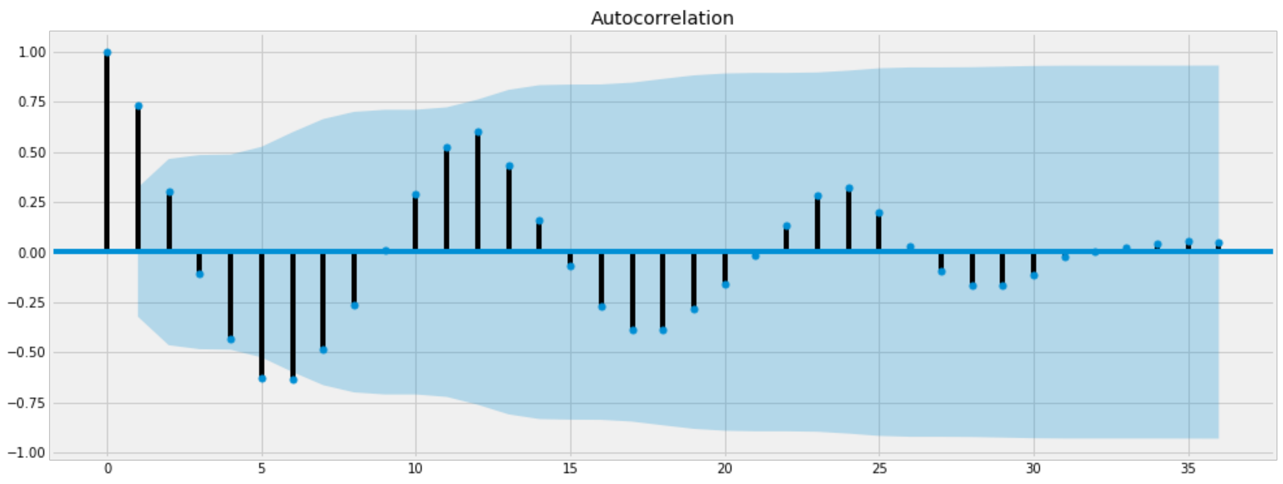


Figure 3.7: Determining lags using ACF

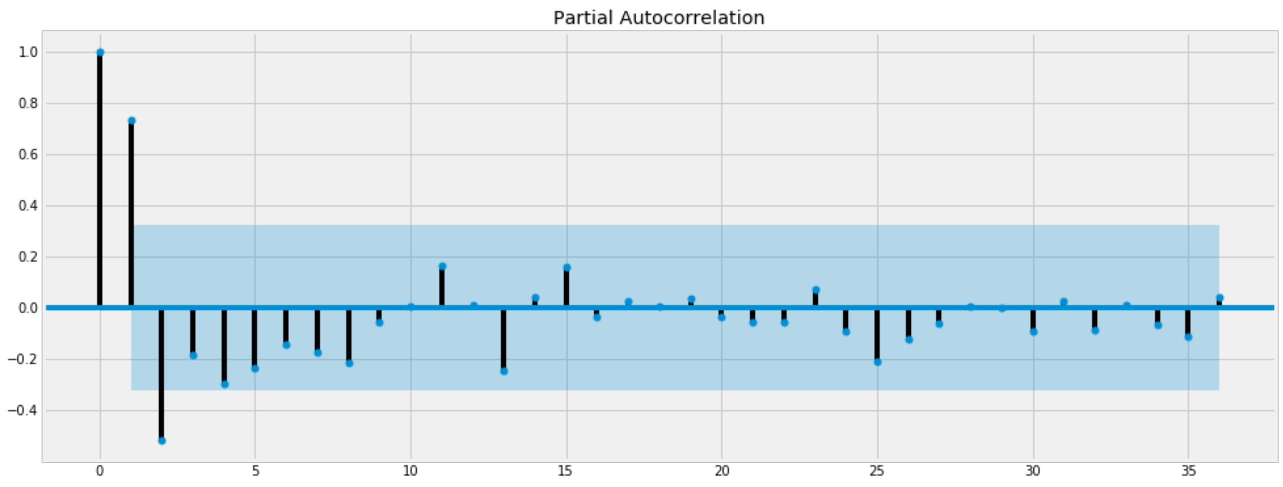


Figure 3.8: Determining lags using PACF

3.3.2.3 VAR

One limitation with previously described models is that they consider only univariate date. We often track multiple parameters at the same time (multivariate) and we would like to be able to forecast values of each of them. VAR (Vector AutoRegression) [100] is one of the algorithms which can be used for such purpose. In case of multivariate time series values of one parameter depend on its previous values, but also on previous values of other parameters.

All variables are treated symmetrically and modeled as if they influence each other equally (variables are treated as "endogenous". VAR model is a generalization of the univariate autoregressive model for forecasting a collection of variables – vector of time series. Parameters are learned for each parameters and predicted based on its previous values and previous values of other parameters. An example in case of two variables and one lag is given with the following formula:

$$y_{1,t} = c_1 + \phi_{11}\, y_{1,t-1} + \phi_{12}\, y_{2,t-1} + e_{1,t}$$
$$y_{2,t} = c_2 + \phi_{21}\, y_{1,t-1} + \phi_{22}\, y_{2,t-1} + e_{2,t}$$

Here $e_{1,t}$ and $e_{2,t}$ represent white noise processes. Coefficients $\phi_{ii,l}$ represent the influence of l-th lag of variable y_i on itself, and coefficient $\phi ij, l$ represents the influence of l-th lag of variable y_j on y_i. If the series is stationary we can fit VAR directly. If the series are non-stationary we first differentiate them to make them stationary and then fit VAR. Fitting procedure follows the principle of least squares.

When using VAR we need to determine:

- Number of variables to forecast (K)
- How many lags should be included (p)

Number of parameters to fit is equal to $1 + pK$ per equation. Hence, it is useful to include only variables which are correlated to each other and as such useful to forecast each other. One of the approaches used to determine these parameters is Akaike Information Criterion, as in the previous case.

An example of forecasting two parameters (consumption and income) using VAR with lag 3 is given on Figure 3.9.

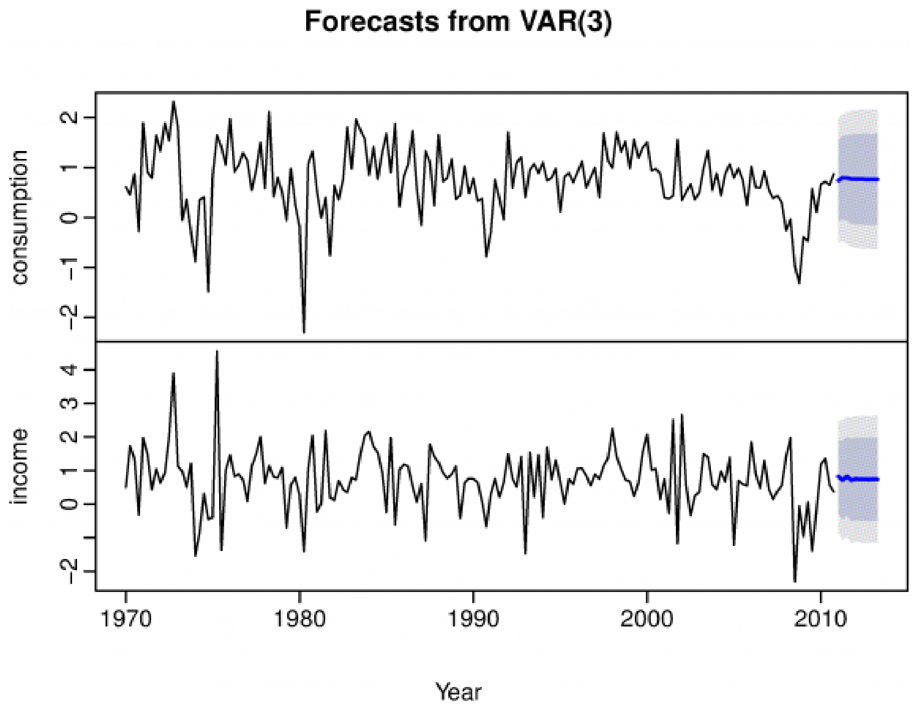


Figure 3.9: Example of VAR usage for prediction of two parameters (source: [100])

3.3.2.4 Neural Network Based Predictions

It is possible to phrase a time series prediction problem as a regression problem [35] and use artificial Neural Networks (Figure 3.10) to train a model which can be used to perform predictions. An example of an artificial (feed forward) neural network is given on Figure 3.10. This neural network has an input layer, a single hidden layer and an output layer. There are 6 inputs, 4 outputs and 9 neurons in the hidden layer. As with regular regression the dataset would be split into train and test datasets. At the moment t we want to take last N observations into account to predict the observation at the moment $t + 1$. In this case N represents the window size and represents a parameter which can be tuned depending on the use case. Sliding window would be used to create training instances out of original observations. For example, in the current moment (t) we might predict the value at the next moment in the sequence ($t + 1$) based on the current value (t), as well as the two prior values ($t - 1$ and $t - 2$). As we observe this as a regression problem the input variables would $t - 2$, $t - 1$ and t, while the output variable is $t + 1$.

Let us consider the following example.

$$x_0,\ x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ x_7,\ x_8$$

In case of our example where window size is 3, we would create following set of instances:

$$x0,\ x1,\ x2,\ \mathbf{x}3$$
$$x1,\ x2,\ x3,\ \mathbf{x}4$$
$$x2,\ x3,\ x4,\ \mathbf{x}5$$
$$x3,\ x4,\ x5,\ \mathbf{x}6$$
$$x4,\ x5,\ x6,\ \mathbf{x}7$$
$$x5,\ x6,\ x7,\ \mathbf{x}8$$
$$x6,\ x7,\ x8,\ \mathbf{x}9$$

Dependent values are bolded. This dataset would be then fed to our neural network which would produce a model for predictions. Window size is one of the parameters which should be considered, while the neural network architecture (number of layers and neurons in each layer) is another thing to have in mind. There are two basic layers (input and output) and an arbitrary (minimum 1) of the hidden layers.

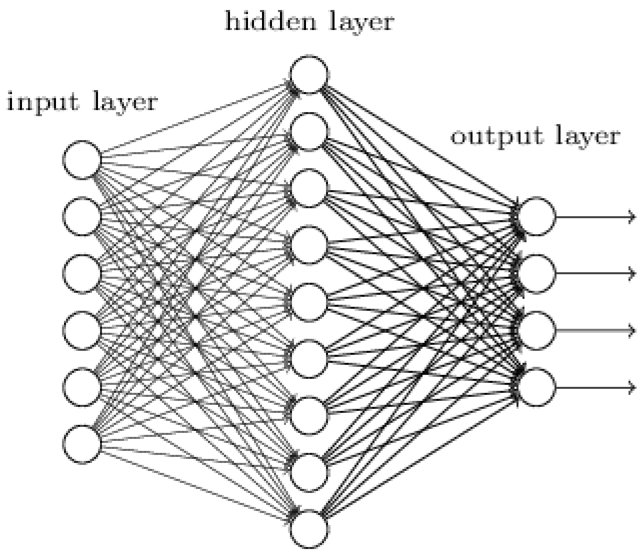


Figure 3.10: Example of an artificial neural network

There are many implementations of neural networks which can be used, such as Theano, Torch, Caffe and TensorFlow.

3.3.2.5 Recurrent Neural Network (RNN)

Recurrent neural network is a class of neural networks where connections between the neurons form a directed cycle (loops). Figure 3.11 shows a part of a recurrent neural network, where loop allows information to be passed from one step of the network to the next one [168].

A recurrent network can be presented as multiple copies of the same network, where each part is passing information to a successor [23]. Unrolled recurrent neural network is given on Figure 3.12.

Plain recurrent neural networks do not work well in case of "long-term dependencies". During backpropagation and gradient calculation two situations can happen -– *vanishing gradients* (due to multiplication of large

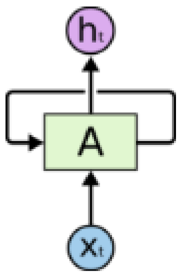


Figure 3.11: Part of a recurrent neural network

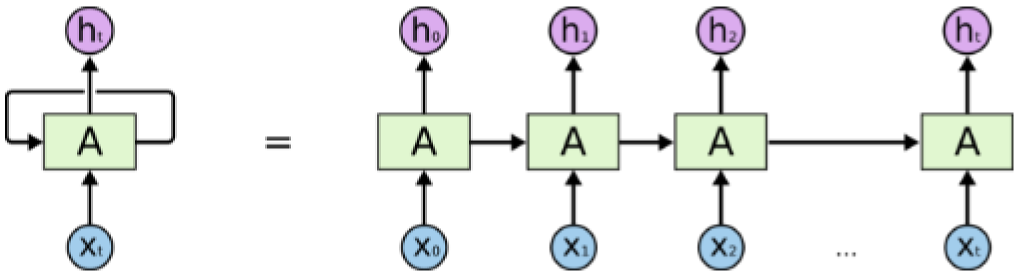


Figure 3.12: *Unrolled* view of recurrent neural networks

number of small values) and *exploding gradients* (due to multiplication of large number of large values). To overcome this a new kind of RNN's was developed –- LSTM's.

3.3.2.6 LTSM

LSTM (Long Short Term Memory) networks represent a special kind of RNN, able to learn long-term dependencies. LSTM's unlike traditional RNN's don't have the vanishing gradient problem and they use backpropagation Through Time. LSTM's are very popular in the area of speech recognition and handwriting recognition. Instead of neurons, LSTM networks have memory blocks that are connected through layers [36]. A block contains gates that manage the block's state and output. There are three types of gates withing a unit:

- Forget Gate – conditionally decides what information to throw away
- Input Gate – conditionally decides which values from the input to use for the update of the memory state
- Output gate – conditionally decides what to deliver as output based on input and the memory of the block

Each unit represents a state machine, where the gates of the units have weights that are learned during training. An illustration of one cell in LSTM network is given on Figure 3.13.

These cells are combined into a LSTM network. An example of such one-layer network is given on Figure 3.14 with gradient flow given in red.

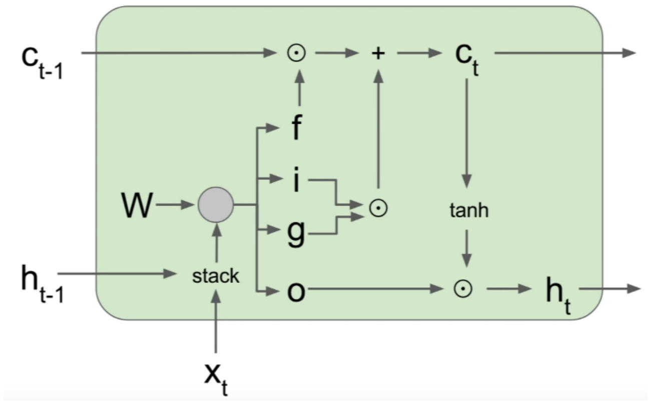


Figure 3.13: LSTM cell

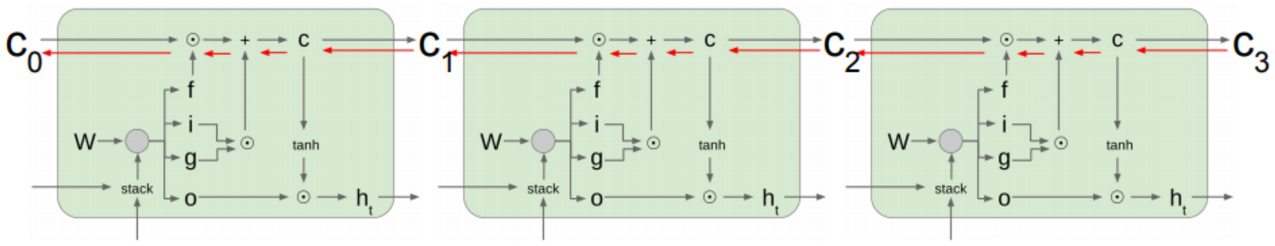


Figure 3.14: LSTM network and gradient flow

3.3.3 Conclusion

We have analyzed different methods for time series forecast. Several methods exist, each having its strengths and weaknesses. Choice of appropriate method depends highly on data being analyzed. When predicting one stationary parameter AR, or ARMA methods might suffice. If the time series is not stationary integration is needed, hence ARIMA would be more appropriate as it adds the step of differentiation, making time series stationary. If seasonality exists Seasonal ARIMA would provide most benefit.

On the other hand, in case we need to forecast more than one parameter at the same time we might use VAR or Recurrent Neural Networks. If we decide to use RNN's it would be best to use Long Short Term Memory networks, to avoid problems such as *vanishing* and *exploding* gradients. Hence, we can conclude that there are enough methods to cover most of use cases and the choice of the concrete method will depend on the concrete use case.

3.4 Cloud Adaptivity

In the following sections, we are discussing research domains and efforts that relate to methodologies and technological approaches for, reactively or proactively, adapting aspects of cloud offerings, based on internal or external triggers with the objective to optimise their operation. Cloud adaptation refers to the process of dynamic selecting and configuring the cloud resources (such as CPU, memory, storage or networking) that are necessary to provide a service with the desired quality attributes. Adaptivity in big data-driven cloud infrastructures manifests primarily in two layers: the first layer refers to changes that can be performed to the processing topology and the virtual resources available. The second layer refers to changes that can be performed in real time to application-specific code on end devices.

3.4.1 Cloud Resource (re)Configuration

Today, many applications are deployed in public or private cloud infrastructures. Virtualization technologies have helped to lower the cost and increase the flexibility of cloud services by enabling the efficient utilization and sharing of computing resources. The objective of this section is to identify recent research about "cloud resource configuration" techniques and technologies that further optimize the benefits of cloud computing. The term "cloud resource configuration" refers to the process of dynamic selection and adaptation of the cloud resources (such as CPU, memory, storage or networking) that are necessary to provide a service with the desired quality attributes. Cloud resource configuration can be performed using open standards and APIs such as OCCI [167], CIMI [44], TOSCA [166] or proprietary but widely used APIs like Amazon Web Services [20].

3.4.1.1 Resource Adaptation Objectives

Jennings et al. [113] in their survey point out the fact that in cloud-computing may exist different types of actors which have different objectives. Cloud Providers provide services (e.g. cloud infrastructure) to cloud users (e.g. DevOps). Cloud users provide services to end users (e.g. a web-based application). In IaaS or PaaS context, cloud providers agree on certain SLAs and Service Level Objectives (SLOs) with cloud users. There are many categories of SLAs [132]. According to the authors, more relevant to resource management are the SLOs that are quantifiable (like those that are related to performance and availability). SLOs sometimes, depending on the definition of the SLA, may be satisfied up to a certain possible degree, when other constraints must be satisfied simultaneously. Cloud providers may pursue additional objectives that are important for their business such as energy use minimization and fault tolerance, in some cases in a prioritized manner depending on the context (e.g. prioritize low energy use when the total workload is not high). Cloud users have SLAs with their customers. In order to satisfy them they agree SLAs with cloud providers but they may have additional objectives (such as minimizing the risk of violating SLAs with end users by using multiple cloud providers).

3.4.1.2 Resource Adaptation Functions

Jennings et al. [113] in a recent survey of over 250 publications about resource management relevant to cloud-based infrastructures recognize four categories of enabling technologies: "Infrastructure Scaling", "Virtual Machine Migration", "Virtualization" and "Equipment Power State Adjustment". In their perspective the resource management functionalities can be distinguished according to the type of actor that is involved in resource configuration activities. End users are only responsible for creating the workload to the cloud application and thus are only indirectly involved in the cloud configuration. Cloud users provide applications or services to the end users by utilizing the infrastructure that cloud providers offer. Each actor type has different objectives and can intervene to the system by using different layers of cloud configuration functionalities. Among the challenges that the authors recognize belong to: (i) the way to achieve predictable performance for cloud applications knowing that the cloud resources are shared between different applications; and (ii) the way to globally manage different types of resources (compute, storage, networking, etc.) in an orchestrated way in order to achieve "Global Manageability of Cloud Resources".

Rajiv et al. [197] summarize the different types of attributes (cores, speed, capacity, etc.) that can be configured and orchestrated in cloud infrastructure resources (CPU, BLOB storage, network, etc.) according to its type (IaaS, SaaS, PaaS) and the operations that can be applied to each attribute (start, stop, restart, etc.). This work not only describes the cloud resources as the ontology in Youseff et al. [252] but it attempts to identify the way a cloud-based infrastructure can be programmed in order to achieve its goals.

3.4.1.3 Horizontal or Vertical Scaling

A cloud-based application execution environment can be scaled horizontally or vertically. Horizontal scaling refers to the ability to add more nodes (also known as scale-out) of the same type (in terms of resource configuration such as CPU and memory) to a cluster. Vertical scaling refers to the ability to assign more resources to a server (also known as scale-up). When the application does not need the resources the opposite actions can be performed (i.e. scale-in or scale-down). If a platform allows both horizontal and vertical scaling then it supports hybrid scaling. Horizontal scaling is feasible or more adequate when the application has the capability to detect new instances of itself in an automatic or semi-automatic manner. If the application stores data it must be able to execute in an efficient way complex replication mechanisms in order to run in additional nodes. In order to achieve horizontal scaling many times it is also required to include external load-balancing services and thus make the deployment of the application more complex. Therefore horizontal scaling is supported only by some types of applications. On the other hand vertical scaling of VMs is more adequate for applications which are not designed for (automatic) horizontal scaling, since resources like CPU, RAM or storage can be dynamically added. Nevertheless, vertical scaling also dictates the need for support by the scaled application. For example, applications that are executed in a Java Virtual Machine (JVM) cannot alter the Java heap size without restarting the JVM when they are vertically scaled. In Figure 3.15 Vaquero et al. [238] depict the available application scaling mechanisms.

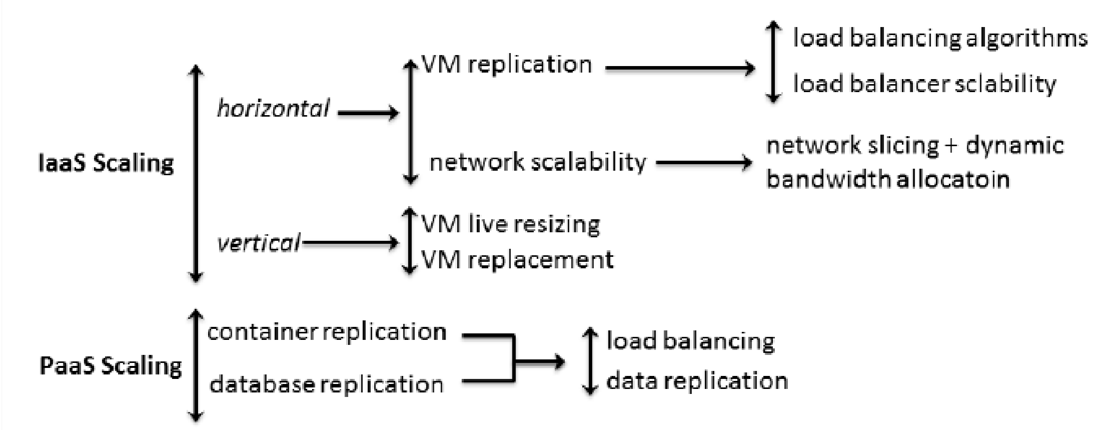


Figure 3.15: Summary of available mechanisms for holistic application scalability (source: [238])

Scaling of the cloud infrastructure can be controlled by a dedicated software. In this case the term autoscaling is used. Chenhao et al. [194] in a recent and extensive review paper identifies many auto-scaling techniques and identifies the type of scaling that each one uses (horizontal, vertical or hybrid). Moreover, he provides a taxonomy of the challenges that the different publications about auto-scaling are tackling and the methods that they use. In addition, live vertical scaling must be supported by the hypervisor and the guest operating system. Turowski et al. [235] compare the vertical scaling capabilities of five guest operating systems (Linux-based and Windows) and four hypervisors (KVM, VMware, Xen, Hyper-V) that can be used in Openstack.

Recently, the concept of combining service autoscaling with Operating System – level virtualization has been more popular in research, as containers provide more flexibility than found on traditional virtual machines. For this reason there have been studies outlining the flexibility and the speed of architectures based on containers.

Kukade et al. [129] propose the adoption of a master-slave architecture in order to implement a service based on containers. The slaves are nodes where containers can be deployed, while the masters are responsible for routing incoming requests to running containers. The architecture is scaled up or down, depending on the relation of the memory load or HTTP requests with the respectively defined thresholds. In a modern platform such as PrEstoCloud, it is necessary to consider the collection of additional metrics, so that more meaningful scaling decisions can be taken.

Jarzab et al. [111] propose a platform design based on the Service Oriented Infrastructure paradigm, which can be used by modern computing infrastructures. It supports adaptable provisioning with lightweight virtualization, which was presented on a complex case study for provisioning JEE middleware on top of the Solaris 10 lightweight virtualization platform (containers). The platform requires the specification of SLA for user applications, the creation of manageability endpoints, and the definition of adaptation policies to provision an application service. These are used by the Adaptation Management module that implements monitoring, analyzing, planning and execution (MAPE) activities in a control loop. In this implementation, the module supports reasoners based on Drools, Jess, PMAC policy languages and uses a rule engine, which supports a scalable pattern-matching algorithm. The rules that are evaluated as true, trigger actions to be performed. In PrEstoCloud, we will consider the use of containers alongside Virtual Machines, in order to provide better control over the execution of the applications.

Hoenisch et al. [99] present a multi-objective optimisation model that aims to assist scaling in a cloud platform. The authors assume that the platform to be scaled is based on VMs, and that containers are used in order to provide flexibility on the allocation of resources of a VM among tasks. The model aims to minimise the total leasing cost of the VMs while at the same time prioritising container deployment on VMs that have already downloaded the data required for container startup, and have free resources. The model includes terms whose value can be adjusted by the user, in order to emphasize the usage of already existing VMs, and to avoid overprovisioning. However the software used to find a solution to the problem (IBM Cplex) is a closed source product, which is a limitation for an open-source software platform.

Marian et al. [204] propose to carry out load balancing on container based setups using a swarm model based on "pheromone" robots. Swarm models utilize a large number of unintelligent actors in order to produce complex global behaviours. In the case outlined, each node is assigned a "pheromone" value p, which is calculated taking into account the number of containers that are scheduled for execution, as well as the number of hosts that can be deployed. The final value describes the probability of migration of a container to ($p < 0$) or away from ($0 < p < 1$) this node. This probability is evaluated on set time intervals by each OpenVZ container, which – when necessary – asks the host to migrate it to another execution host. Finally, after a number of periods the system converges to equal load, providing a decentralized method to establish a balanced architecture. However, in PrEstoCloud we aim not only to provide simple load balancing assuming a single type of container and a single task, but also define richer execution policies, that will be capable of making seemingly "unfair" recommendations.

3.4.1.4 Live Migration

The second significant type of resource adaptation function refers to virtual machine or container live migration as it is explained in the following sections.

Virtual Machine Migration

In a cloud-based environment applications run in isolated Virtual Machines. Many virtualization technologies allow serializing VMs in files that include the current state of the system, stopping them and restarting them in a different Physical Machine (PM). This process is called VM Migration. There are methods to stop, transfer and restart a VM in a different VM, without letting the user executing the application sense any (notable) interruption of it. If this objective is achieved then the process is called VM Live Migration. VM migration and live migration can be used in a cloud environment in order to move VM instances to PMs with more resources (in order to be possible to scale them vertically), to consolidate VMs in less PMs (and thus achieve better resource utilization) or to substitute a PM that has hardware or network connectivity problems with another one without disrupting the services that it runs or losing data. There are several approaches that use live migration either for resource consolidation or to handle increasing workloads [37], [95], [162].

Most approaches for live migration assume that storage (or block devices) is shared between all PMs that belong to a cloud infrastructure and thus their live migration technologies solve only the problem of moving memory and device state between machines. Mashtizadeh et al. [151] describe and compare three different techniques for Live Storage Migration (Snapshotting, Dirty Block Tracking and IO Mirroring). Live storage migration is important for enterprise users because it improves VM mobility, allows the maintenance of storage elements without downtime and enables automatic storage load-balancing. The main live VM migration techniques are "pre-copy" and "post-copy" migration. There are also many optimized versions of them. Kapil et al. , in their survey [118], describe and compare different approaches for live migration which aim to reduce the migration time and the amount of data that is transferred with compression [58], [114], [224] or deduplication [6], [58], [202], [248].

VM (live) migration has the drawback that it consumes resources of the source and destination physical machines and thus degrades the performance of all VMs that run in the involved PMs whether being migrated or not. Approaches like the one of Deshpande et al. [57] try to eliminate the resource pressure during live migration with their agile migration method.

Container Live Migration

Container Live Migration is supported on some platforms (e.g. OpenVZ) [180], however others – including Docker [59] – have not reached this level of maturity yet. At present, Docker offers experimental support for check pointing a running container on the disk, and restoring it to memory using CRIU [51] – however this cannot be done reliably in different environments. Furthermore, currently there are not extensive research results on this field, as the emphasis has been traditionally placed on live migration of VM's. The research work that has been already published emphasizes the case where the destination node has already been decided beforehand, and is known to the system [150], [213], [253]. Wikipedia [247] provides a comparison table concerning the different technologies implementing OS-level virtualization. It can be clearly noticed that live migration off-the-shelf is only supported by a few proprietary technologies, and the only open-source solution (OpenVZ) is available only for Linux containers, on Linux hosts.

Machen et al. [150], present the layered migration framework, as a three-layered architecture, able to model the execution state for every container, which they implement separately on top of the open source KVM virtualization technology and on top of the open source LXC containerization technology. The first layer contains a configured base image of an operating system, the second layer contains the main app deployed on the container at an idle state, and the third layer contains the current state of the execution. They propose the caching of the first two layers in mobile edge clouds (cloud-like infrastructures in close proximity to the users), so that the latency associated with the download of the image can be zeroed. As a result only the final layer is required to be transmitted, which leads to much shorter downtime both in the cases where VMs or containers are used. A similar approach, which focuses on copying only the read-writer layer used by Docker, having copied the rest of the layers in advance, was followed by Cui et al. [54]. The main problem in both cases is the minimization of the data to be transferred during the last step of the migration, so that a stop might be unperceivable.

In this context, Harter et al. [91] proposed a modified Docker implementation, that can lower the time overhead associated with starting a new container. Their architecture modifies the classic setup of Docker which is based on a local disk, in favour of a proprietary Tintri Store NFS solution [231], while maintaining similar performance during service execution. Each container uses a single-layer Ext4 filesystem in lieu of a multi-layer AUFS filesystem which was the original filesystem employed by Docker. This design leverages block-level copy-on-write (rather than file-level which is the case with AUFS) – a feature provided internally by Tintri Store – and eliminates the performance problems identified by the authors as being related to layered filesystems. Since the filesystem has moved to a network-storage setup, there is no longer a need for container data to be transmitted across the network to a local disk, but instead all operations are carried out on the Tintri Store itself and only metadata is communicated. Finally, the authors modified the code regarding container creation, and extended the loopback module API of the Linux kernel, in order to avoid unnecessary disk operations and exploit common cache sharing between containers originating from the same Docker image. The solution proposed or parts of it could be considered for implementation in PrEstoCloud in an open-source form, as part of the centralized processing architecture.

Yu et al. [253] take a different path to container live migration emphasizing the logging-and-replay approach over the pre-copy and post-copy traditional alternatives. Instead of iteratively copying the modified memory regions (pre-copy), or migrating the container using a phased approach (post-copy), the authors recommend to initially copy only a base image reflecting execution state at the time the live migration procedure is initiated. Then, in order to compensate for the activity of the container until the migration actually takes place, the execution events are logged using Revirt [41] and are iteratively sent for playback to the container that will take over. When the log file size drops below a threshold, the source container is frozen, the last log file is transmitted and the destination takes over execution.

3.4.1.5 Decision Making in Cloud Resource Configuration

Decision making in the context of cloud resource (re-)configuration has been supported by many theories and mathematical tools. It can be proactive or reactive. Reactive decision making responds to changes in the cloud infrastructure that have been already realized and detected. Proactive decision making tries to predict the future state of the cloud infrastructure and takes it into consideration.

The review paper of Lorido-Botran et al. [8] identifies and compares five categories of approaches for decision making in auto-scaling: Threshold-based rules, Reinforcement learning (RL), Queuing theory (QT), Control theory (CT) and Time series analysis (TS). Gandhi et al. [79] identify five similar auto-scaling approach categories: Prediction models, Control theoretic techniques, Queueing-based models, Black-box and Grey-box approaches. Black box models use machine learning or statistical methods for decision making in order to overcome the problem of modelling the cloud application using expert knowledge. Grey-box models are hybrid approaches that use models in combination with machine learning [79]. Chenhao et al. [194] support in their review paper that, according to the bibliography, resource estimation in horizontal or vertical auto-scaling can be performed using Rules, Fuzzy-Inference, Application-Profiling, Analytical Modelling, Machine Learning or hybrid methods. Analytical modelling according to the authors includes Queuing Theory and Markov Chains. Machine learning includes Reinforcement Learning and Regression. Regression is applied in auto-scaling techniques that use Time-series Analysis or Control Theory. Based on the above observations, we conclude that they use similar or overlapping categorization methodologies for the decision making approaches. In the next paragraphs, we present some of the most interesting approaches in this domain.

Rule-based decision making can be found in commercial auto-scaling systems like AWS Auto-Scaling service [19] or RightScale [201]. Relative simple rules that can be triggered by a set of performance metrics are easier for the (DevOps) user to understand and set-up. Usually rule-based systems are reactive but there also proactive auto-scaling approaches which use rules. In [148] the authors examine nine approaches that use rules for auto-scaling. Most approaches are reactive but according to the authors those that are proactive combine rules with queuing theory [39] or time-series analysis [130]. Rule-based decision making has been applied for both horizontal and vertical scaling. The most approaches use conditions based on CPU load but some use additional metrics like memory, I/O rates, response time or network performance metrics (bandwidth, delay, and jitter). Lorido-Botran et al. [148] propose the use of dynamic thresholds which are automatically modified as a consequence to SLA violations. Dynamic thresholds are used also by Lim et al. [145]. As stated by Chenhao et al. [194] rule-based decision making requires deep understanding of the application dynamics in order to determine the right actions and triggering conditions, factors that can significantly affect the performance of the auto-scaling [5]. An advanced form of rule-based decision making is the utilization of fuzzy inference [78], [135]. With fuzzy inference numeric thresholds can be substituted by linguistic terms like "low", "medium", "high" which may be easier understood by humans. Fuzzy rules that can be learned dynamically in runtime [109], [133], [251] have been used for auto-scaling in order to overcome the problem of manual designing fuzzy rules.

Queuing theory studies with mathematics waiting lines, or queues. A cloud application which runs on a variable number of servers can be modelled as a queue of requests. With queuing theory we can estimate the necessary resources (e.g. number of servers) that are required to process a workload with specific size. Requests which are not processed can be buffered and thus delayed. For example Ali-Eldin et al. [9] model the cloud infrastructure as a G/G/N stable queue where the variable N denotes the number of servers (VMs) required to process a specific workload. When the workload increases VMs can be added and when it decreases they can be removed. Requests which cannot be processed by the available number of servers are buffered. The number of the buffered requests is used as criteria to add more VMs. There are several approaches that use queuing theory. Lorido-Botran et al. include in their review [148] five approaches. According to the authors they have been used only for horizontal scaling (proactive or reactive). Some other approaches use multiple queues. Urgaonkar et al. [236] use one queue per server in a network of queues in order to model the cloud infrastructure. Zhang et al. [256], Han et al.[67] and Bacigalupo et al. [24] model multi-tier applications with multiple queues. Similarly Chenhao et al. [194] include in their review approaches that abstract the cloud application as a single queue, approaches that use a queue for each server and hybrid methods. As stated by the authors [148] queuing models are not the best option for general purpose auto-scaling systems because they impose a fixed architecture. Any change in the architecture (for example in the pool of available resources) requires solving the queuing model with analytical tools or numerical methods which is a computationally expensive task.

Reinforcement Learning (RL) is a machine learning method. Approaches that use RL learn the most suitable action for each state of the system based on experience which is acquired by an agent that interacts with the system following trial and error methods. With RL the auto-scaling can be performed without providing hard-coded rules. In order to perform RL we must define all possible actions (usually different types of horizontal or vertical scaling), a cost (or reward) function which often is associated with the cost of the cloud resources and all the possible states of the system. Depending on the type of scaling that is supported by the approach the state space can contain multiple combinations of the number of the VMs, the amount of memory, the workload size (in terms of response time), the CPU load and other parameters. Lorido-Botran et al. [148] review eight approaches that use proactive decision making for both horizontal and vertical auto-scaling with RL. All of them support only one type of scaling (horizontal or vertical). According to the authors there are three types of difficulties when designing an auto-scaling method which is based on RL: (i) bad performance during online training (for long time); (ii) computational complexity due to the large number of possible states (curse of dimensionality); and (iii) undesired bad performance when the environment changes. There are approaches that try to overcome these problems with different methods [27], [30], [64], [65], [199], [227].

Control Theory is also a tool that has been used extensively in auto-scaling. Lorido-Botran et al. [148] review twelve approaches that are based on Control Theory. In these approaches some kind of controller automatically adjusts the resources of the cloud infrastructure (e.g. the number of VMs or the memory of the VM) in order to maintain the value of an output variable (e.g. response time or CPU load). Some of these approaches are applicable to horizontal scaling and some to vertical scaling but none to both. There are different kinds of controllers. Proportional Integral Derivate (PID), Proportional Integral (PI) and Integral (I) controllers belong to the category of fixed gain controllers. Fixed gain controllers are simpler to implement and thus preferred by some authors. Zhu et al. [259] use a PID controller, Park et al. [184] use a PI controller and Lim et al. [145] use an I controller to configure the resources of a cloud application. Fixed gain controllers have the disadvantage that they remain fixed during the operation of the cloud application. In order to overcome this limitation some authors use adaptive controllers [8], [9], [25], [183]. A controller in order to properly operate, it requires a formal model (named either transfer function, state-space function or performance model) that associates the input variables with its output variable. PID controllers use linear transfer functions but other types of controllers use non-linear approaches. A controller can be Single-Input Single-Output (SISO) or Multiple-Input Multiple Output (MIMO). Padala et al. [183] use a MIMO controller which allocates the amount of multiple virtualized resources (CPU and disk I/O in multiple physical machines which run VMs) in order to achieve the desired performance metrics (or SLOs). Other performance models that have been used are based on Kalman filters [117], Splines [25], Guasian Process Regression [81] or Fuzzy models [134], [246], [249].

3.4.2 Application Fragmentation & Refactoring

Cloud services characteristics (service properties such as price, availability, response time, etc.) can change at any time during the application execution. Hence, there is a need to support the adaptation of applications in such dynamic conditions, in order to ensure that the cloud services currently provided to deployed applications adhere to the established requirements. Cavalcante et al. [40] developed an autonomous adaptation process for cloud-based applications by replacing a service by an alternative one that fulfils the application needs and describe the adaptation process within the Cloud Integrator, a service-oriented middleware platform for composing, executing, and managing services provided by different cloud platforms. Inzinger et al. [105], [106] proposed a provider-managed, model-based adaptation approach for cloud computing applications, allowing customers to specify application behaviour goals or adaptation rules, thus actively engaging the application refactoring process. There are also recent agent-based efforts that try to fuse adaptivity in cloud resources usage. For example, Comi et al. [50] present an approach based on agent cloning, i.e. a mechanism of agent reproduction allowing providers to substitute an "unsatisfactory" agent acting in a "cloud context" with a clone of an existing agent having a suitable knowledge and a good reputation in the multi-cloud context.

Another stream of work focuses on application migration approaches. Gholami, Mahdi Fahmideh, et al. [82] published a detailed survey of cloud migration approaches. The authors revealed that little work exists that provides a mean to design situation-specific approaches with respect to the characteristics of a migration project. Supulniece et al. [103] described methods used for enterprise application decomposition for cloud migration projects. Their work distinguishes between four decomposition phases: fact extraction, pre-processing, clustering / component classification and post-processing. Methods for fact extraction include static, dynamic and semantic code analysis as well as dynamic SQL analysis. Pre-processing includes similarity evaluation, trace compression, rules, classification, code cleansing and concept assignment. Clustering and component identification is typically done with clustering methods as well as rules. Finally, post-processing refers to evaluation methods using rules, refinement as well as optimisation and layer identification.

A methodological approach for cloud migration has been developed by Jamshidi et al. [110] based on (i) a catalogue of fine-grained service-based cloud architecture migration patterns that target multi-clouds, (ii) a situational migration process framework to guide pattern selection and composition, and (iii) a variability model to structure system migration into a coherent framework. The proposed migration patterns are based on empirical evidence from several migration projects, best practice for cloud architectures and a systematic literature review of existing research. The methodology allows an organization to (i) select appropriate migration patterns, (ii) compose them to define a migration plan, and (iii) extend them based on the identification of new patterns in new contexts. The patterns are at the core of our solution, embedded into a process model, with their selection governed by a variability model.

A cloud migration strategy recommender method is proposed by Bonab et al. [31]. The main rationale behind this method is to provide easy and fast recovery from failed components or replacing the required functionality of the legacy software with the reliable cloud services. In this paper a semi-automated reverse engineering method based on the clustering algorithms is proposed to recommend the best migration-to-cloud strategy. The recommendation is based on four defined metrics: the extent of effort required for reengineering, maintenance costs, achieved availability and the number of cloud services that are used.

Kwon and Tilevich [131] proposed an application refactoring approach based on a recommender tool that computes the coupling metrics for all the classes in a legacy application and then displays the classes that are least tightly coupled. Accessing the functionality represented by these classes from a remote cloud-based service should impose only a limited performance penalty on the refactored application. The approach leverages two recommendation mechanisms: profiling and clustering-based recommenders. (i) The profiling-based recommender engages a static program analysis and runtime monitoring to collect program information. By combining the class coupling metrics collected through both static analysis and runtime monitoring, the recommendation algorithm then suggests a subset of an application that can be transformed to cloud-based services. The profiling-based recommender sorts application classes based on their execution duration and frequencies, so that the programmer can know what classes are computation-intensive and how frequently they are accessed. (ii) The clustering-based recommender clusters classes with similar functionality, thus identifying class clusters whose functionality can be naturally exposed as a cloud-based service. Because the clustering-based recommender groups classes based on their functionality, the programmer can avoid duplicating a functionality in the cloud by selecting candidates for cloud-based service from different clusters.

Hilton et al. [98] developed Cloudifyer, a touchdevelop IDE plugin which refactors touchdevelop scripts in place. First, Cloudifyer retrieves the source of the target app as an Abstract Syntax Tree (AST) stored in JSON format from the touchdevelop script bazaar. It then transforms the AST as needed. Once all the transformations are performed, Cloudifyer completes the refactoring by saving the new AST for the target app.

Vasconcelos et al. [239] presented a novel approach to support organizations in automatically adapting their existing software applications to the cloud. The approach is based on the loosely-coupled implementation of non-intrusive code transformations, called cloud detours, which enable the automatic replacement of local services used by an application with similar or functionally-related services available in the cloud.

In terms of programming framework, we intend to use JPFF in PrEstoCloud. JPPF [112] is an offloading mechanism, which enables the partitioning of cloud application processing tasks to multiple processing nodes at source-code level. It supports the allocation of processing tasks on any Operating system capable of running a JVM (also on Android despite running its own JVM). There have been proposed alternative application offloading frameworks such as CloneCloud [43], MACS [126], MAUI [52], and JADE [193] – some offering more advanced features (such as dynamicity on the offloading decision and automatic application partitioning) – which will be discussed in D5.5.

Chapter 4

Technological Assets

This Chapter provides a technological overview of some tools that will be used as building blocks for the PrEstoCloud platform. We first introduce ProActive, that will be the heart of the control layer of PrEstoCloud. ProActive enables to provision resources in public and private clouds and to execute workflows that lead to dispatching computing tasks on those resources.

Next, we present BtrPlace, a VM placement algorithm that is not bound to any specific VM provisioning and management solution. It also enables the optimization of the placement of VMs on resources materialized as physical server in a private data center. BtrPlace will be extended for PrEstoCloud to the case of federated clouds. We further position BtrPlace with respect to the existing literature and Kubernetes and Kubertvirt.

Then, we focus on the edge-cloud communication layer, and the need to be able to set-up an overlay on demand to interconnect all the resources in the private, public could and up to the edge. We introduce the Software Defined Networking (SDN) and Network Function Virtualization (NFV) paradigms that might be useful to build such an overlay. We also present the networking solutions offered by leading cloud providers (AWS, Azure, etc).

The end of the Chapter is dedicated to the review of various complex event processing engines that available on the market, as well as a competitive comparison between them.

4.1 ProActive

ProActive is an open-source software suite that offers solutions for orchestrating large scale computing tasks. Its foundations rely on the active objects programming abstraction. The ProActive Programming is at the core of the solution and provide the base for an efficient distribution and parallelization. It is the foundation of the high level solutions: ProActive Workflows & Scheduling, ProActive Big Data Automation and ProActive Cloud Automation.

4.1.1 Active Objects

Active objects [138] are the basic units of activity and distribution used for building concurrent applications using ProActive [38]. An active object runs with its own thread. This thread only executes the methods invoked on this active object by other active objects and those of the passive objects of the subsystem that belongs to this active object. With ProActive, the programmer does not have to explicitly manipulate Thread objects, unlike in standard Java. Active objects can be created on any of the hosts involved in the computation. Once an active object is created, its activity (the fact that it runs with its own thread) and its location (local or remote) are perfectly transparent. As a matter of fact, any active object can be manipulated just like if it were a passive

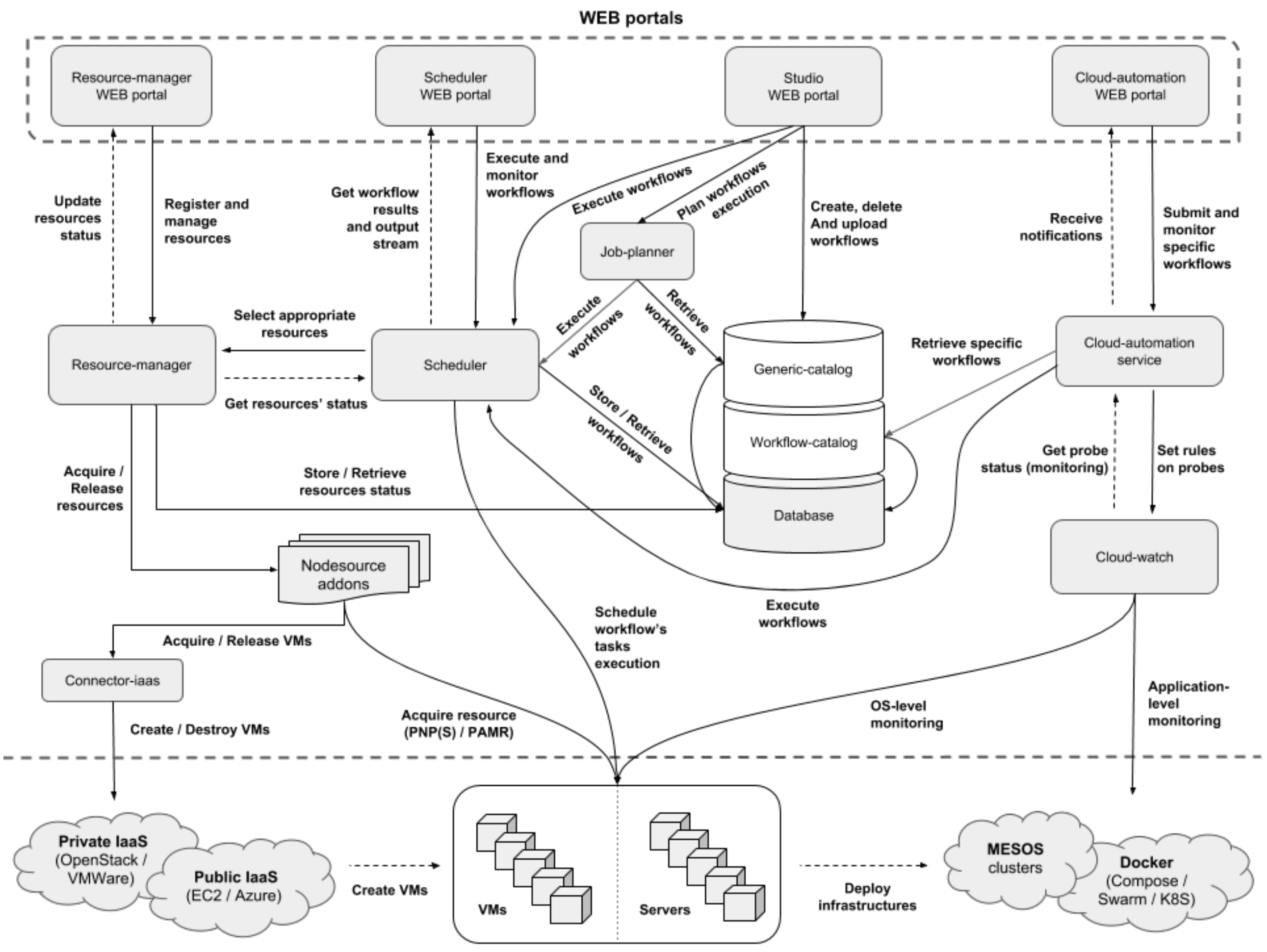

WEB portals
Resource-manager WEB portal
Scheduler WEB portal
Studio WEB portal
Cloud-automation WEB portal
Update resources status
Register and manage resources
Get workflow results and output stream
Execute and monitor workflows
Execute workflows
Plan workflows execution
Create, delete And upload workflows
Receive notifications
Submit and monitor specific workflows
Job-planner
Select appropriate resources
Resource-manager
Scheduler
Get resources' status
Execute workflows
Retrieve workflows
Store / Retrieve workflows
Generic-catalog
Workflow-catalog
Database
Retrieve specific workflows
Cloud-automation service
Get probe status (monitoring)
Set rules on probes
Cloud-watch
Acquire / Release resources
Store / Retrieve resources status
Nodesource addons
Acquire / Release VMs
Connector-iaas
Create / Destroy VMs
Acquire resource (PNP(S) / PAMR)
Schedule workflow's tasks execution
Execute workflows
OS-level monitoring
Application-level monitoring
Private IaaS (OpenStack / VMWare)
Public IaaS (EC2 / Azure)
Create VMs
VMs
Servers
Deploy infrastructures
MESOS clusters
Docker (Compose / Swarm / K8S)

Figure 4.1: ProActive overview

instance of the same class.

4.1.2 Workflows and Scheduling

The ProActive Workflows & Scheduling use active objects to offer a job scheduling solution that allows to distribute and execute jobs and business applications, monitor activity and view jobs results. It optimizes the resources usage, managing heterogeneous platforms on multiple sites.

Every existing big data framework supports some kind of scheduling. Scheduling is usually considered at several level that should not be confused. The first one is operating system scheduling, that will be not addressed in this project, the second one is cluster scheduling, and the last one is workflow scheduling. Cluster scheduling objective is to distribute computing tasks on a set of more or less uniform set of computing resources. In this project, this objective is addressed by the BtrPlace algorithm (see Section 4.2).

Workflow scheduling concern is to organize the application logic computation schedule [254]. The cluster schedulers does not take into account the dependencies between the different tasks of an application to be executed (sequence, parallelism, branches). So workflow schedulers are designed in both Hadoop and “container-oriented” architectures. They take workflows (graph of tasks) as input and operates the underling cluster scheduler. For instance Argo operates Kubernetes, Oozie, Airflow or Azkaban operate Yarn or Chronos operates Mesos [198]. For stream-oriented big data platforms such as Storm, Spark or Flink, things are different since the dataflow is natively expressed as a computing graph, and there is no direct need of a additional workflow scheduling engine.

4.1.3 Cloud Automation

ProActive Cloud Automation product is a extension module to Workflows & Scheduling that allows to use workflows to deploy and manage services and Cloud applications. It allows self-service deployment, as well as automate the delivery of in production services. It control application elasticity with automated scaling-up and down, horizontally and vertically. Thanks to unified cloud management, applications execute on multi-vendor private, public and hybrid clouds.

ProActive Cloud Automation offers monitoring capabilities for resources and applications to collect and track metrics, and triggers. It monitors the status of different resources such as VMWare ESXi, vSphere, vCloud, OpenStack, CloudStack, physical and virtual machines, storage, network and applications. It allows to expose your own metrics with simple APIs - JMX, REST and plain text file - and to set alarms for troubleshooting or trigger automatic actions such as elasticity or disaster recovery plan.

4.2 BtrPlace: Virtual Machine Placement under Constraints

BtrPlace is an open-source VM placement algorithm developed by COMRED, a CNRS team of the I3S laboratory. This section first discusses a state of the art related to VM scheduling and job placement in general. We then develop how BtrPlace works and discuss how it will be used inside PrEstoCloud at a coarse grain.

4.2.1 Context

Infrastructure As A Service (IaaS) clouds provide clients with hardware via Virtual Machines (VMs). To deploy an application in an IaaS cloud, a client installs the application and selects one of a Service Level Agreement (SLA) offered by the provider. The SLA covers, for example, the expected availability and the minimum amount of resources to allocate, and may also cover possible placement constraints, such as anti-colocation between VMs that host a replicated service. Inside the cloud, the VM scheduler deploys the VMs to appropriate physical servers according to the various offered SLAs. When environmental conditions (failures, load spikes, etc.) or the clients' expectations evolve, the VM scheduler reconfigures the deployment accordingly, using actions over the VMs such as the live migration [45].

At the Platform As A Service (PaaS) level, the scheduler deploys software components inside their runtime. For example, a PaaS scheduler might deploy Java War applications inside Tomcat services. Despite the manipulated entities differ from those manipulated at the IaaS level, we observe numerous similarities in terms of consolidation expectations or SLOs typically: the scheduler must place components inside containers with regards to their expectations in terms of performance, security, privacy, etc and according to the components management lifecycle.

The scheduler is the cornerstone of the good functioning of a cloud. The provider bases his offering and the clients base their requirements on the scheduler features. It matters then to exhibit a scheduler with enough features to attract a maximum number of clients, and important consolidation capabilities to reduce the running costs. As a natural consequence, both the research and the industry community investigate on scheduling techniques are features for cloud architectures.

4.2.2 Industrial solutions

In terms of industrial initiatives, three projects propose production ready IaaS schedulers. OpenStack [176] and Apache CloudStack [21] are the leading open-source providers for IaaS clouds. Each propose a VM scheduler that compute a placement for the VMs, satisfying some filtering rules such as anti-affinity. Each scheduler is bundled with some pre-defined rules while the API allows third party developers to develop their own rules when needed. For docker based environments, Google proposes Kubernetes [128], a complete software stack to deploy and supervise docker containers. Kubernetes is also extensible and its micro-service architecture allows to use a custom container scheduler. In terms of PaaS scheduler, Cloudify [48] and Openshift [175] are open source providers to deploy software services on top of an existing IaaS infrastructure.

Such schedulers are interesting because of their maturity. They are used in production by numerous clients over small to big infrastructures. Such solutions are however tightly coupled with their underlying cloud and cannot be used on another infrastructure or platforms. This is strongly reducing their practical interest if we focus multi-cloud environments. They are also limited in terms of features. For example, none of them propose any dynamic scheduling feature. For example, the scheduler cannot revise the current placement when the environment is changing (typically when a load spike occurs or when the client expectations change). This might be problematic in the context of PrestoCloud where we propose an autonomous system to adapt the application configuration and deployment depending on the live conditions.

4.2.3 Academic solutions

The research community proposes numerous architectures and prototypes to ease scheduling, especially at the IaaS level. One of their original motivation being the use of live-migration to perform dynamic scheduling, often applied to green computing [95], [241]. In this context, the scheduler regularly analyses the environment to check if SLO are violated (an observable performance issue for example) or if a better consolidation is possible. If so, the scheduler will re-arrange the placement to reach a new viable placement.

Initially, the schedulers where centralised software components. One single scheduler being used to manage a complete infrastructure. The problem they address is however an instance of the vector packing problem, a NP-hard problem in a strong sense [94]. Accordingly, the scheduler latency necessarily increases exponentially with the size of the infrastructure. With the ever-increasing size of the infrastructure, developers proposed different approach to improve the scheduler scalability. In this context, the scheduler is focusing on placing jobs, potentially having dependencies and proprieties on the nodes. The Borg scheduler of google for example runs multiple independent copies of a scheduler on a shared infrastructure and conciliate conflicts lazily [240]. Sparrow [182] proposes a two-level scheduling algorithm to schedule jobs at a very low latency. This architecture proposes a very low latency that benefits to the users when the jobs to run are very short (typically a few seconds). At the extreme cases, fully distributed schedulers such as DVMS [195] leverages peer-to-peer architecture to reconfigure the VM placement. This approach is very effective at a very large scale but does not help at computing the initial VM placement, nor at fixing a placement when the infrastructure is heavily loaded or when the clients SLOs are very different. Indeed, when a violation is detected on a node, the node contacts a neighbour and tries to fix the violation over these two nodes. If it fails, then it contacts a third neighbour, etc. Accordingly, finding the minimal number of nodes that is sufficient to solve an issue is costly. On the other side, hierarchical or clustered approaches try to solve issue on a decent number of nodes by default. This limits their scalability when the node partition is too high, but it is still appropriate for cluster of hundreds or a few thousand of nodes. Recently, Firmament [83] exhibits a centralised approach that is scalable enough to support large Google-size data centre. Firmament proposes a flexible design based on the min-cut max-flow algorithm and takes the benefits of the hierarchical design of a data centre to speed up the placement algorithm.

All these solutions improved the state of the art related to VM scheduling. On one side, the initial solutions propose dynamic scheduling that is effective when the workload is composed on service VMs running permanently and having varying resource requirements. On the other side, distributed scheduling algorithms decreased the scheduling latency when the scheduler have jobs to run, with an assumed constant resource usage. All these approaches are however limited in terms of SLOs they support, reconfiguration capabilities their flexibility in general to fit different hosting environments such as those we target with PrEstoCloud (edge computing, hybrid clouds, etc.).

BtrPlace is an open-source research prototype that targeted extensibility as a primary feature [93]. BtrPlace uses a Constraint Programming solver [203] to model a VM scheduler and relies on the composability offered by the paradigm to support the addition of new concerns, new placement constraints or new optimisation objectives. BtrPlace uses Choco [102] – an open-source Java constraint solver – internally. Currently BtrPlace addresses performance, security, reliability, networking or energy efficiency concerns. It has been used as a research prototype inside two FP7 European project. In the Fit4Green [74] and the DC4Cities [55] projects, developers with varying level of expertise enhance BtrPlace to make it address energy efficiency through different consolidation policies that fit infrastructure peculiarities [62], [63], [124]. Despite is centralised architecture by default, BtrPlace has been qualified to support infrastructure of thousands of servers while it also embeds several techniques to increase its scalability when needed, such as partitioning techniques to solve sub problems in parallel.

Despite BtrPlace is a research prototype, it is a production ready software. It is released several times per year, documented and tested [228]. Two companies use it officially for production environments. First, the Onyx plaform [229] uses BtrPlace to perform distributed computation. In this context, BtrPlace was tuned to be used as a PaaS scheduler and place peers performing data oriented computation over physical or virtual machines inside public or private clouds. Second, Nutanix is the worldwide leader of hyperconverging systems [165]. The company uses BtrPlace to mitigate hotspots in thousands of private entreprise cloud with a size varying from threes to thousands of servers.

4.2.4 Why BtrPlace

BtrPlace is an extensible and composable placement algorithm. It is designed to provide a core placement algorithm to be customized according to workloads and infrastructures particularities [93]. This approach makes it already rich enough to support most of the placement constraints available in private and public clouds (resource allocation, multi-level affinity and anti-affinity, isolation, counting oriented constraints) without having any hard dependencies for a given platform. BtrPlace is valuable for PrestoCloud to help at developing a scheduler that will be capable to embrace a heterogeneous environment composed of private and public clouds and an edge layer, each having its prerequisite that could accept a workload made of heterogeneous software components. Writing an efficient placement algorithm from scratch is a long and challenging task that requires a high expertise in several domains (combinatorial optimization, technical expertise in cloud technologies). On the other side, starting from a scheduler bundled with a platform like the schedulers in Kubernetes or OpenStack is very risky. First this requires understanding a large codebase, having a significant technical debt, without having the guarantee to being able to successfully meet the scheduler expectations. Second it will make harder to decouple the scheduler from its original platform that does not reflect PrestoCloud objectives. We then consider it is wise to enhance a customizable placement algorithm with a limited codebase and no strong commitment to a certain kind of infrastructure or workload. This approach was applied successfully by either core BtrPlace developers but also external developers. For example, The Onyx startup made BtrPlace suitable to manage orchestrate a distributed computing environment running peers on either VMs, containers or bare-metal [229]. The EU project Fit4Green and DC4Cities succeeded at customizing BtrPlace to address energy-related concerns over a IaaS [63]. [29] customized it to schedule processes in HPC cluster. Finally, Nutanix uses it at a daily basis to mitigate hotspots in thousands of private clouds [164].

4.2.5 Comparison with Kubernetes and Kubevirt

Kubernetes is a Docker container orchestration system initially designed to deploy over a single data center (data center federation is still an ongoing feature). It embeds the required services to be able to monitor, deploy, scale in or out docker containers. Kubevirt is an API add-on for Kubernetes that make it supports VMs. PrestoCloud focuses on edge computing and hybrid cloud computing and does not rely on containers for security and portability issues as some of the application in the use cases cannot run in containers. BtrPlace is a standalone and extensible placement algorithm originally designed for VMs. It is not coupled to any virtualization environment, monitoring or storage systems or even security model while the Kubernetes scheduler is tightly coupled with the services provided by its platform. We then consider it is safer to customize the BtrPlace algorithm to make it fit the environment particularities, rather than trying to modify a massive production grade environment with project specific features. Such decision was considered successfully in the past in either European projects (Fit4Green, DC4Cities), startup (the Onyx startup leverages BtrPlace to perform distributed data processing) or private cloud company like Nutanix.

4.3 Networking

4.3.1 Software-Defined Networking

Network virtualization is a heavy trend in the networking domain. Virtualizing the network refers to the ability to share a physical network infrastructure in-between several clients, e.g. tenants in a data center, each with its private set of networks (possibly with overlapping subnets) that need to be supported simultaneously.

Virtualization often goes hand in hand with softwarization, where hardware networking equipments – switches, routers, firewalls, load balancers, etc – are replaced by virtual machines running on off-the-shelf servers. Once a network function is softwarized, it can be easily updated, deployed where it is needed, and cloned to scale with traffic demand.

The most popular network virtualization technique nowadays is *Software Defined Networking* (SDN). SDN breaks the conventional silo approach, where switches and routers take their decision independently from each other (based on routing or control information exchanged) in a distributed manner. In other words, SDN decouples the *control plane* (where to route the packet) from the *data plane* (how to forward the packet) with switches under the control of a centralized component called a controller.

SDN is a relatively new and promising network architecture, that is highly flexible, and overcomes the limits of the forwarding mechanisms of legacy IP networks. Indeed, SDN-based networks – or *SDN networks*, as we will call it in the remaining of this document –, unlike legacy IP networks, do not only rely on the destination IP address to make forwarding decisions but can also use several other information from the other network layers (e.g. MAC, Network and Transport header of packets) when making forwarding decisions [230]. SDN technologies make also a clear separation between the control plane and the data plane. In SDN the *forwarding devices* (or switches) are considered as dummy devices that only follow the forwarding policies dictated by an external programmable entity: the controller, which implements the control plane. Thus, there is no more need to have specific, multiple vendor network devices such as switches, routers, firewalls etc. And there is no need anymore to run a CLI (Command Line Interface) to configure network devices, and repeat such operations for every concerned device. In SDN networks, any needed change in the forwarding policies is coded only once in the *controller*, and the latter will propagate the forwarding rules to all the SDN equipments [96]. The controller and the switches communicate together using a so-called *southbound interface* such as OpenFlow [230]. In this section, we first describe how SDN networks deal with user data traffic, and then we will briefly explain the main components of SDN networks.

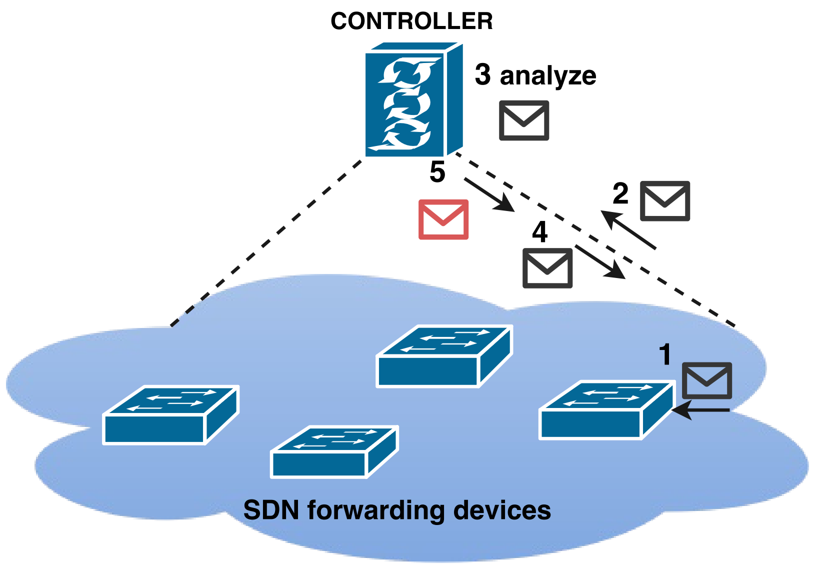


Figure 4.2: SDN data packet treatment process

4.3.1.1 Mode of Action

Pure SDN networks are composed of *SDN switches* that are connected to the *controller*. When a packet arrives at an SDN switch, see Step 1 in Figure 4.2, the switch will first check its list of pre-installed forwarding rules (example rule: incoming packets from port 1 source A to destination B should be sent from port port 2). If the packet header information does not match any installed rule (i.e. there is a *packet miss*), the switch will then forward the packet to the controller using the default SDN rule which matches any packet (Step 2). Upon reception of the data packet on the controller, the controller's network applications will analyze the packet headers and decide the list of *action(s)* to be taken when matching packets arrive at the SDN switches (Step 3). Afterwards, the controller will transmit this packet back to the switch and implement the actions directly on it (Step 4). In addition to that, the controller will transmit a *flow_mod* event to the switch with the list of forwarding rules that need to be installed within the switch (Step 5), so that next upcoming matching packets can be treated directly on the switch without the need to forward them to the controller.

4.3.1.2 Main Components

In this Section, we provide a detailed description of the main components in SDN networks.

SDN Forwarding devices

As explained earlier, SDN forwarding devices, also called *SDN switches* are dummy devices. When an SDN device is integrated into an SDN network, it first notifies the controller of its existence, of its basic configuration, and of the state of its components (ports, links etc). These switches then rely on the controller to receive a set of rules they need to know to process incoming packets. These *forwarding rules*, also sometimes loosely referred to as *flows*, are saved in the physical memory of the switch. SDN switches mainly use the Ternary Content Addressable Memory (TCAM) to store the flow. This hardware memory allows quick rules installation and rapid matching of packets. However, TCAM memory is both expensive and power hungry [189], and, therefore, most physical switches provide limited TCAM memory, which supports between 2 thousand and 25 thousand rules [223], depending on the switch cost. When the TCAM is full, the SDN rules are then placed in the software memory. However, installing the rules in software (i.e. classical RAM) degrades the performance as packet matching will require to use the switch CPU, which, in turn, will increase the delay. We refer to this situation as *slow path*.

Two types of SDN switches exist:

- **hardware switches**, such as Pica8 [190], HP 5412zl [97], and so on;
- **software switches**, such as OpenvSwitch (OvS)[188].

Only hardware switches install the forwarding rules (flows) in the TCAM, but software switches such as OvS can however benefit from the memory cache to boost performance. The functionality of the forwarding device depends on the forwarding rules installed. It can act, for example, as a standard (layer 2) switch, as a router (layer 3), as a firewall (layer 3 and 4), as a load balancer.

Southbound Interface

The *southbound interface* allows the exchange of messages and instructions between the *SDN controller* and the SDN forwarding devices. It is a description of the general format of the network protocol deployed between the controller and the forwarding devices. Multiple standardized southbound interfaces exist, such as OpenFlow [230], ForCES [60], and POF [221]. ForCES [60] allows the separation of the control plane from the data plane without the need to change the current network architecture. Thus the control plane in general should be managed by a third-party firmware.

As for POF [221], it allows, like OpenFlow, the total separation of the control and of the data plane while changing the network architecture to use a controller and forwarding devices. However, unlike OpenFlow, POF does not analyze the packet header on the switch to match incoming packets. It rather uses a generic flow instruction set (FIS) generic key that the switch uses to perform packet matching on the forwarding devices.

Finally, OpenFlow is the most deployed SDN protocol southbound interface. The OpenFlow SDN architecture is an SDN architecture that uses OpenFlow protocol southbound interface to allow communication between the controller and the forwarding switch. Multiple OpenFlow protocol versions co-exist especially (v1.0, 1.3, 1.5). In OpenFlow, an SDN forwarding rule – also called a flow entry – is composed of three parts:

1. Match fields: packet header values to match the incoming packets in addition to the ingress port. To match any possible value for a specific field, a wildcard can be used.
2. Actions: set of instructions to apply to the matching packet (e.g. forward, drop, modify)
3. Counters: used to collect statistics of packet and byte count match for each flow in addition to the timers information.

All OpenFlow protocol versions use the same structure of SDN rules with some action and matching field additions in each version. The basic flow rule in OpenFlow v1.0 matched 12 packet header fields. This increased to 15 fields in OpenFlow v1.3. The usage of multiple field matching instead of destination based matching in the switches allows thus to unite multiple legacy network device functionalities in a single rule. However, the forwarding rule complexity comes at a price of increase in memory space used per rule.

Controller

In SDN, the *controller* is responsible for managing the control plane of all the network. The controller node is composed of (Figure 4.3):

- the Network Operating System (NOS), in other words the controller
- the northbound interface
- and the network applications

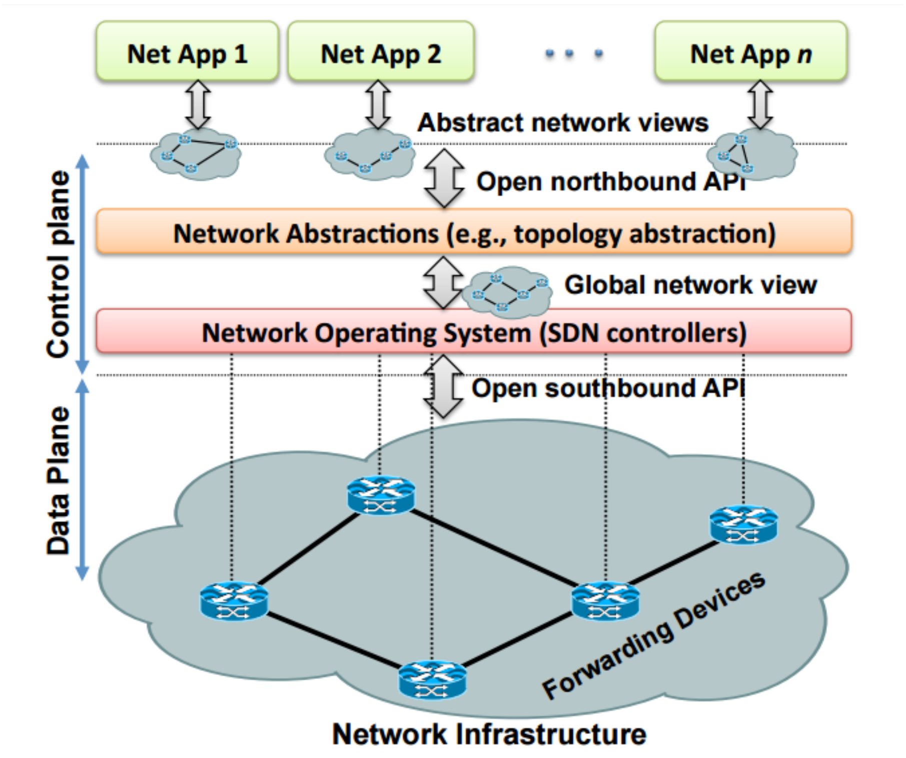


Figure 4.3: SDN network structure (source: [127])

The SDN controller runs on the *Networking Operating System* (NOS), where the NOS is a software platform that runs on commodity servers. It allows to access the server resources and basic services. The SDN controller communicates with the SDN network devices, and creates the global topology view of the network. It also monitors the full state of all the network components regularly. The controller then informs the network applications of the network states using the northbound interface (e.g. a REST API). Then the network applications manage and implement policies in the network devices through the controller interface.

Based on their network configuration requirements and specific needs, an administrator can program new network applications (new network functionalities) in standard programming languages such as Java, C++ and Python. This gives the administrator full control over the network topology and allows the infrastructure to be reactive to network and traffic dynamics.

Two types of SDN controllers exist:

- Centralized controller
- Distributed controller

The centralized controller provides a centralized global view of the network which allows online reactivity to changes in network states, and simplifies the development of sophisticated functions. These controllers are usually designed to keep up with the throughput of datacenters and enterprise networks. However, the centralization of the control plane in a single node reduces network resilience as the centralized controller represents a single point of failure in the network. Multiple centralized SDN controllers exist such as POX [171], NOX [104], Ryu [205], Beacon [69] and Floodlight [107]. In 2013, more than 30 different OpenFlow controller existed, created by different vendors or research groups. These controllers use different programming languages, and different runtime multi-threading techniques. The POX controller is mainly used for prototyping [211]. The NOX controller is not supported anymore. As for the remaining, most known controllers (e.g. Ryu, Beacon and Floodlight), Beacon has the best performance in throughput and latency [211]. When active developement of Beacon stopped, a fork of the project, Floodlight v2.0, was its natural successor.

A distributed controller can be a physically distributed set of controllers, or a centralized cluster of controller nodes. Distributed controllers provide fault tolerance, but require an additional overhead to maintain consistent network states across all the controllers. Hence, when the network state changes there will always be an inconsistency period of time. Multiple distributed controllers exist, e.g. OpenDaylight[173] and ONOS [169]. Both OpenDaylight and ONOS provide similar functionalities with similarities in performance. However, the main difference relies in the main domain focus of each controller. ONOS focuses more on meeting the needs of service providers while OpenDayLight focuses on providing all of the detailed network functions that one needs to be able to integrate any functionality required. OpenDaylight is thus said to be the "Linux of networking" [90].

4.3.2 Network Function Virtualization

Much like Software-Defined Networking (see Section 4.3.1) is the networking community's answer to the dynamicity requirements in data centers, that traditional legacy distributed networking algorithms and devices were not able to cope to, Network Function Virtualization is an effort deeply rooted in the telecommunication industry. In telecom networks, the complexity of operations is handled by the networking infrastructure itself, rather than by the edge devices, which are fairly simplistic, e.g. telephones. Computer networks are, on the other hand, conceived so that the most complex functions occur at the edge of the network, with complex edge-devices (e.g. computers), and relatively simple functions within the network core, when compared to telephone networks. [123]

Network Function Virtualization (NFV) is originally an effort from Telecommunication Service Providers (TSPs) to lower their operating expenses and their capital expenses. Indeed, with such complexity held within the cores of telecommunication networks, TSPs rely on proprietary vendor hardware in order to acquire "blackboxes" that they can plug in into their network to obtain the desired functionality. These are the capital expenses. At the same time, TSPs need their staff to (i) physically go and install the equipment on the many sites used by the TSP, (ii) train their staff to use the equipment appropriately. This leads to operating expenses. The primary goal of NFV is to reduce these costs by avoiding the purchase of new blackboxes. These functions would now be purely software, and deployed on commodity hardware. Not only does this lower TSPs expenses, but it also gives them more flexibility in service deployment. For example, multiple local sites can be aggregated into a larger one, because the software would be able to scale better than the propietary hardware. Moreover, software is more easily updated, which means that TSPs can provide new services more frequently (higher technology turnover). This is particularly important in the context of wireless telecommunications, such as with 4G/LTE, and the upcoming 5G networks. [143], [154]

From this description, it is clear that SDN and NFV, while originating from very different backgrounds, share one main goal: the *softwarization* of the network, enabling it to run on standard, commodity hardware. However, they are also two very distinct concepts. Indeed, the NFV concept brings networks functions from hardware to software (*service/function* abstraction), while SDN achieves a centralized network architecture that is easily programmable (*network abstraction*) [143], [154]. As stated by the Open Network Foundation in [170]: "the NFV concept differs from virtualization concept as used in the SDN architecture. In the SDN architecture, virtualization is the allocation of abstract resources to particular client or applications; in NFV, the goal is to abstract NFs away from dedicated harware, for example to allow them to be hosted on server platforms in cloud data centers".

Nevertheless, SDN and NFV are complementary concepts. The SDN controller could be seen as an NFV if it runs in a virtual machine. The vNFC [120] and OpenNF [80] efforts are the first research steps towards this possibility. On the other hand, a network abstraction model that would also support NFVs can create dynamic *service chains*, where a number of network functions are used one after another, and rely on SDN to steer the traffic between them [53], [89]. These are currently active research areas.

In PrEstoCloud, we will operate a mix of public and private clouds. While public clouds may rely on SDN to offer network virtualization to tenants, the controller(s) and switches remain opaque to the end user that can only inject routing or security rules through an API or GUI. We will thus consider injecting a VNF , i.e. a virtual machine, featuring an SDN capable switch like OvS to steer the traffic from within the IaaS to that VNF, which will enable inter-cloud networking. In the remainder of this Chapter, we will focus on the available technologies to achieve this goal.

4.3.3 Inter-Cloud Networking

In order to create a multi-IaaS network, we need to analyze the networking solutions proposed by the different public cloud providers.

In general all the providers propose solutions for network management, whose common properties include:

1. creation and management of virtual networks, defining IP addresses ranges and subnetworks;
2. Network Address Translation (NAT) management and routing tables definition;
3. connection to the Internet (public addresses) or private connections through VPNs.

We will describe below all the different network services or products offered by the various providers, so as to understand their capabilities and limitations.

The considered providers are the most popular public cloud providers, and are also the ones supported by ActiveEon's ProActive multi-cloud managment platform (see Chapter 2). We did not investigate Microsoft's Hyper-V, as Microsoft is now shifting its whole business to Azure.

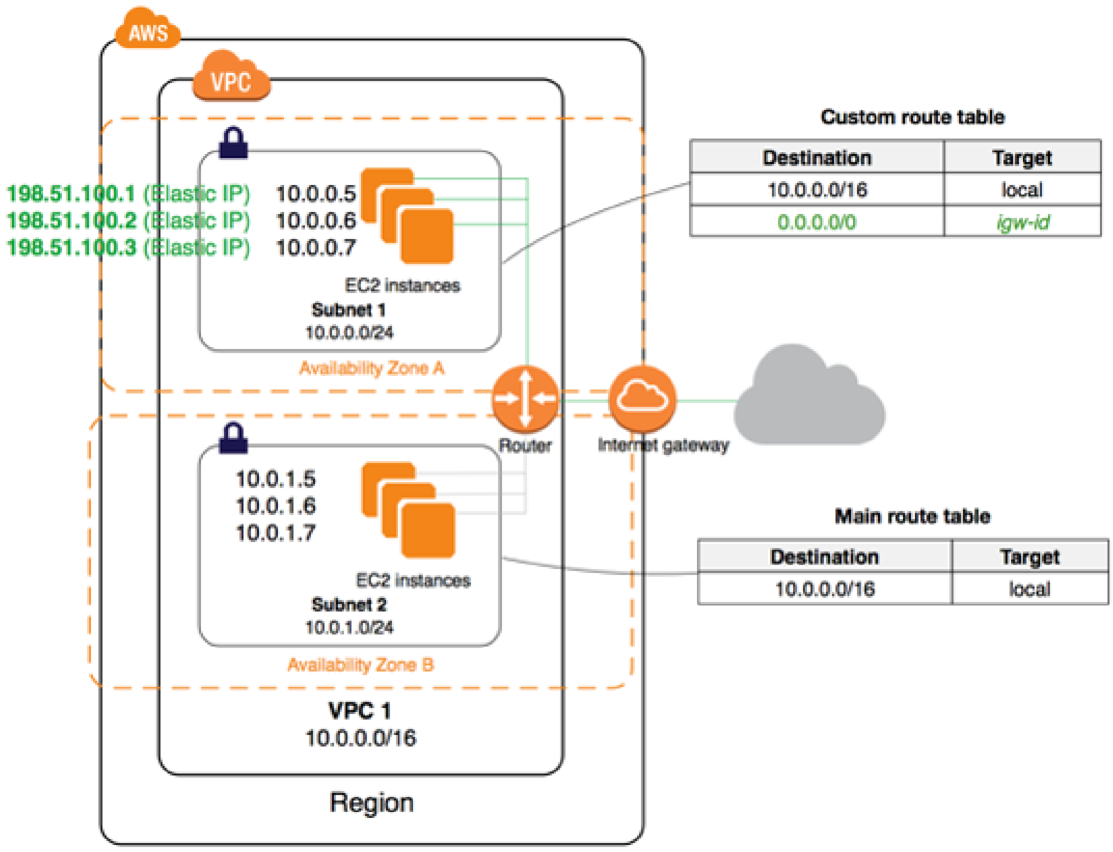
4.3.3.1 Amazon Web Services

Amazon Web Services (AWS) is divided in different isolated data centers, called *regions* [16]. By default, resources in one region cannot communicate with resources in other regions. Across-region communication occurs over the public Internet. Internally, these regions are divided, in turn, into several *availability zones*. These zones are designed to ensure fault tolerance between machines instantiated into different availability zones.

If we need more freedom on our virtual machines placement, e.g. to meet specific delay or throughput constraints, Amazon provides two different options. First, the user has the possibility to require a *Dedicated Host* [12]: a physical machine in which they can instantiate virtual machines. Second, *Placement Groups* [15]: a logical aggregation of VMs in the same availability zone.

Amazon *Virtual Private Cloud* (VPC) [18] is a dedicated virtual network inside an AWS region. A VPC is isolated from the other virtual networks and it can be split in different subnets. An AWS resource (e.g. an EC2 instance) can be instantiated inside a subnet. The different subnets can be configured to be connected to the Internet, through an Internet Gateway, or to remain private networks (Figure 4.4); two subnets inside the same VPC can have different connections to the gateways. Additionally, a VPN gateway can be used to connect the VPC to the user personal infrastructure (i.e. the user's corporate network) (Figure 4.5).

All the EC2 virtual machines can communicate directly inside the same VPC. The VPN gateway can be used to connect two different VPC inside the same AWS region. In case of VPN multi-region connectivity, Amazon proposes several solutions [14]. All the AWS instances connected to the Internet have their own public IP address. This address is bound to the instance, meaning that if the instance fails, the service at the relative IP address will be unreachable. To improve fault tolerance, Amazon provides the *Elastic IP* service, which allocates a public IP address that can be assigned dynamically to one instance. Thus if the assigned instance fails, we



Custom route table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	*igw-id*

Main route table

Destination	Target
10.0.0.0/16	local

Figure 4.4: AWS VPC network architecture: use of an internet gateway to connect to the internet (source: [18])

can easily re-assign this public address to another one. In this case, if a user connects to this IP address, even using the Amazon DNS server, it will reach the backup service.

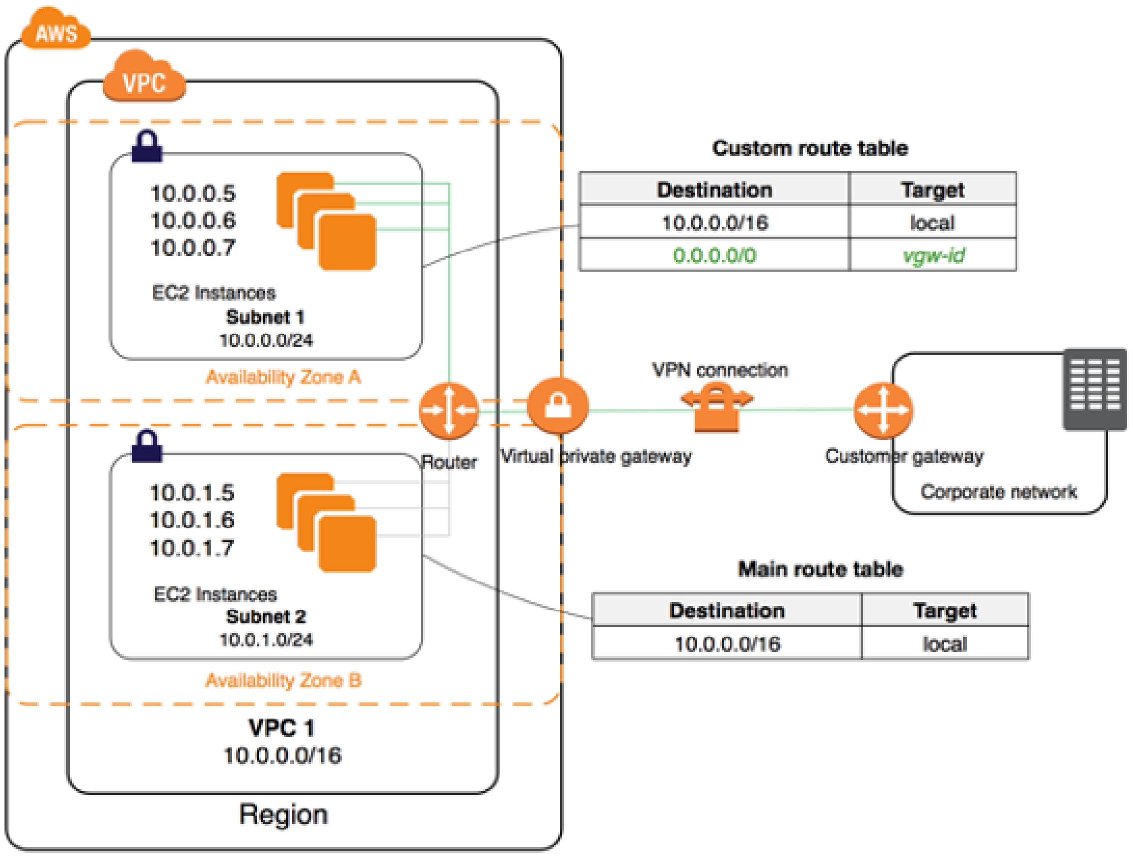
Moreover, Amazon *Direct Connect* [13] provides dedicated and isolated connectivity to the AWS services. Through Direct Connect Partners (such as Internet Service Providers), a direct connection between the VPC and the user's private network can be established, without letting data flowing across the Internet. A specific piece of software, called LOA-CFA [17] must be downloaded by the user in order to complete the Cross Connection. Amazon also provides several locations with which the connection can established, in case the equipment is hosted in the same facility as AWS Direct Connect.

4.3.3.2 Microsoft Azure

Similarly to Amazon Web Services, Microsoft Azure is divided into regions [75] to provide service availability and redundancy. For instance, each region is paired with another one in the same geographic zone to replicate the resources.

The *placement groups* [33] are used to map instances to Affinity Groups. They enable to place the machines in servers located near each other in the same region. Similarly, an *Availability Set* [163] are used to logically ensure redundancy and fault tolerance. Moreover, *scale sets* [88] can be used to manage a group of similarly-configured VMs. This service will help applications that need to scale computation resources.

Azure allows the creation of *virtual networks* (VNets) [159] where to assign virtual machines. To assign a VM to a VNet, we must create a *Network Interface* (NIC). A VM can have multiple NICs in the same VM. A NIC supports both public and private IP addresses. With public IP addresses, we enable the VM to communicate



Destination	Target
10.0.0.0/16	local
0.0.0.0/0	*vgw-id*

Destination	Target
10.0.0.0/16	local

Figure 4.5: AWS VPC network architecture: use of a VPN gateway to connect to a personal infrastructure (source: [18])

with other VMs not connected to the same VNet directly through the Internet. Private IP addresses are used for communications inside the same VNet. A VNet can be divided into subnets [158] to organize the user's network. These subnets can be also used to extend the on-premises network of the user with the VNet.

To make multiple VNets communicate, even if in different regions, a *VNet-to-VNet* connection can be configured [153]. Through a VPN using an IPsec tunnel, multiple VNets can reach each other (Figure 4.7).

The same method can be used to connect a VNet to the user's infrastructure configuring a *Site-to-Site* connection [42] (Figure 4.6). In addition, Azure provides a service similar to AWS Direct Connect called *Express Route*. It allows the user to directly connect to the VNet without passing through the Internet. Express Route partners will enable this connection, reducing delays and increasing throughput.

Moreover, Microsoft specifies the possibility to create a Point-to-Site connection [152] in case we want to connect just one computer to the VNet through a P2S SSTP tunnel. A VNet can be connected to multiple

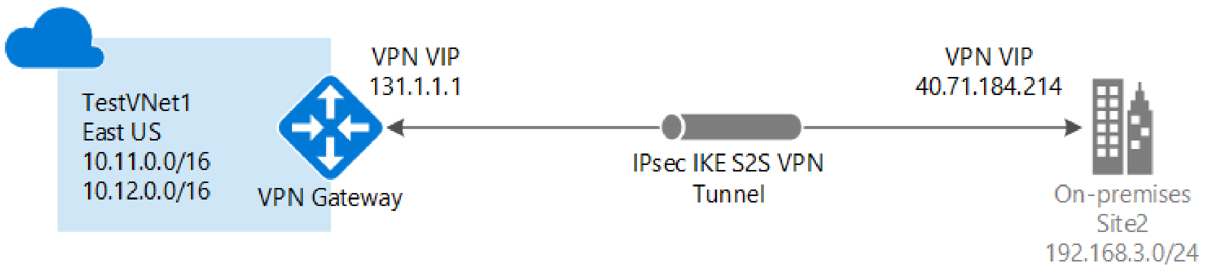


Figure 4.6: Azure networks connection models: Site-to-Site connection model (source: [42])

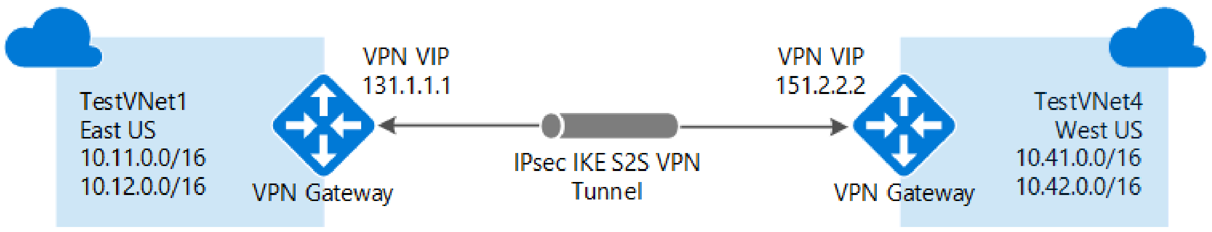


Figure 4.7: Azure networks connection models: VNet-toVNet connection model (source: [152])

"points" through different tunnels.

4.3.3.3 VMware vCloud Air

The service provided by VMware doesn't seem to have a region subdivision; in any case it supports the creation of virtual private networks in his vCloud Air platform through its *Virtual Private Cloud* [243].

vCloud Air *Hybrid DMZ* [242] enables isolated virtual networks in the cloud. When we connect our network to the cloud, H-DMZ allows us to extend our security policies and network functions to the cloud.

vCloud Air *Advanced Networking Services* [244] provides network segmentation, dynamic routing and network interconnection in two different ways:

- point-to-site SSL connection similar to the one proposed by Azure and
- site-to-site connection through IPsec tunnel.

vCloudAir *Direct Connect* [245] offers similar functionalities as AWS Direct Connect and Azure Express Route.

4.3.3.4 OpenStack and CloudStack

Both of these platforms are open source projects, that serve as frameworks to build a public or private cloud service for third-party companies. Both of them provide modules to offer the same functionalities as the competitors.

OpenStack provides the *Neutron* module [177] for network management, enabling virtual networks definitions, *floating IP* (same as AWS Elastic IP) and various networking plug-ins to support several technologies (e.g. OvS, CiscoUCS, Linux Bridge, etc.).

Similarly, CloudStack aims to build a Network as a Service [49], which uses the *Nicira NVP* and *Nuage VSP* modules, together with other several network plug-ins, to manage virtual networks and create shared networks. *Elastic IP* assumes the same function as AWS namesake.

4.3.4 Interconnection Technologies

Interconnecting public clouds together, or interconnecting a public and a private cloud, requires establishing tunnels over the public Internet. This can be achieved thanks to the VPN services provided by cloud providers, e.g. Azure Express Route or Amazon Direct Connect, or by using a software tunnelling solution embedded in virtual machines, in other words, a NFV.

The most popular technologies to establish tunnels are IPsec and TLS (SSL) based. IPsec is considered as the de-facto VPN standard. It works at layer 3 of the protocol stack. IPsec provides confidentiality, integrity protection, data origin authentication, and replay protection by encrypting and signing every message. The protocol is described in many RFCs [66], [77], [119], [121], [122]. Tunnels are established as a combination of so-called (unidirectional) *security associations* (SA). Security policies are then used at the tunnel endpoints to decide which traffic is handled by one of the SAs, and which traffic is forwarded unmodified. Amazon Direct Connect offers IPsec. IPSec is natively supported in the Linux kernel since version 2.6. The most popular versions are LibreSwan [144] and OpenSwan [178].

TLS-based tunnels work above the Transport Layer Security layer, i.e. at layer 4 of the protocol stack. Popular softwares that feature TLS-based tunnels are OpenVPN [179], SoftEther [219], and Openconnect [172]. TLS-based tunnels might offer less performance, when compared to IPSec, due to the kernel level support of IPSec. However, an advantage of TLS-based tunnel is the better compatibility with NAT as compared to IPsec.

When both clouds-to-be-interconnected are behind firewalls, the interconnection might become a problem. In such a scenario, an option is to use NetVirt [161]. NetVirt uses a relay (called a switch) as a meeting point for firewall/NAT traversal, in order to link machines located in the two clouds connect in order to establish the tunnel. Please note that the NetVirt switch is just a rendezvous point, and is no longer required once the tunnel has been established. Using standard and well-known perfect forward secrecy algorithms (e.g. Diffie-Hellman), the NetVirt switch is also unable to decrypt the tunnel.

4.3.5 Towards an on-demand VPN service for PrEstoCloud

In the previous section, we surveyed a number of technologies that will form a toolbox to build network overlays connecting the different subnets deployed by PrEstoCloud. To avoid the vendor lock-in effect and work around any incompatibility between technologies offered by different cloud providers, we envisage deploying dedicated virtual machines acting as gateways (hence VNF) between the different data centers. These VNFs will maintain a full mesh overlay between themselves, enabling compute nodes in different data centers to communicate seamlessly. A number of challenges will pave our way, that we will investigate in detail in Work Package 3 of PrEstoCloud but already sketch here:

- How to segment IP (private) addresses at the overlay level so as to prevent address conflict at the overlay level and accommodate edge nodes in case they need to be directly reachable via the overlay.
- How to ensure that the overlay be permanently on. Resiliency might entail using a cluster of gateway nodes monitoring each other and ready to take over the role of a failed sibling. Researchers in [220] addressed this issue from the cloud provider viewpoint by decoupling the control path (tunnel establishment) and the data path (containing actual sessions' parameters) and we could inspire from some of their ideas.
- Scalability of the VPN gateway service might also be an issue if a host of edge nodes need to be connected to the overlay. We will have to assess the resource consumption (RAM, CPU) consumed by the different solutions when the number of connections increases. A related issue is the stability of the VPN client in the edge node [10]. The exact solution to use might depend on the application requirements in terms of rate.
- Establishing an efficient (resilient, scalable, ...) VPN overlay architecture will be a first step in PrEstoCloud. We further envisage to continuously monitor the amount of bandwidth offered in the overlay so as to expose this information to the application. Measuring the cloud inter data centers has received some attention lately [186], [187]. How to measure bandwidth with a limited intrusiveness, for the running PrEstoCloud application, constitutes a first challenge. A second challenge is how to measure in a cost effective manner as the user pays for the traffic sent from the data center (traffic received is in general free of charge). In such a context, using bandwidth hungry tools like in [187] is not an option, and we will revisit tools invented over a decade ago by the network measurement community, esp. Pathload [108]. While accurate in non virtualized environments, these tools might need to be tuned for the virtualized

case. Indeed, the accuracy of the tool mandates precise timing information that the virtualization layer might bias.

4.4 Complex Events Processing Engines

On the Complex Event/Streaming Analytics market 3 main groups of solution providers or vendors can be identified. First group is pure OSS (Open Source Software) around the Apache foundation. Apache is offering different solutions, which are Apache-Storm, Spark, Flink, Samsa and Apex. The Apache solutions could be seen more as frameworks less than ready to use (with of course customizing aspects) products Second group is semi OSS/CS (Closed Source) from EsperTech, with Esper (OSS) and Esper Enterprise Edition (CS). The third group is that one of commercial products/vendors. Here to be named IBM with IBM Streams and Watson Analytics, Microsoft Azure Stream Analytics, SAP with HANA Smart Data Streaming, Amazon Kinesis, Tibco Business Events and StreamBase and Software AG Apama Streaming Analytics.

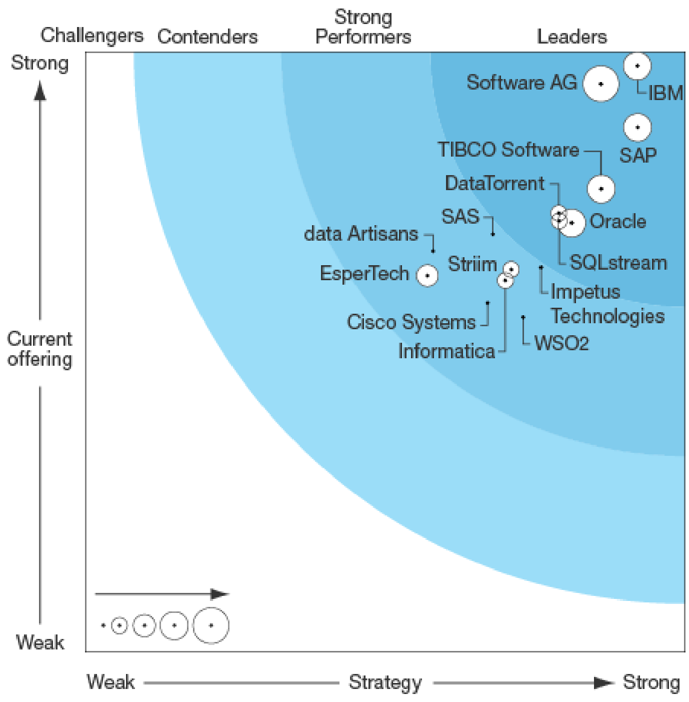


Figure 4.8: The Forrester Wave: Big Data Streaming Analytics, Q1 2016 (source: [85])

Forrester evaluated 15 vendors of commercial big data streaming analytics products in [85]. The results are depicted in Figure 4.8.

In the remainder of this Section, we review in detail two CEP engines: Apama and Siddhi. Then we provide a summarized review of other main competitors as well as a comparative analysis.

4.4.1 Apama

Apama is an event processing platform coming along with an integrated development environment. It also contents a real-time graphical dashboard for monitoring application execution, an Event replay application as well as an integration framework to integrate Apama into event messaging environments, databases and application environments. Apama comprises of three tiers:

1. A runtime engine called the Apama Correlator, which consumes all data sources, executes the applications, looks for event patterns and delivers insights and actions in real time.
2. Development tools, which supports an easy development and testing of Apama applications.
3. A large variety of adapters and plug-ins for connecting to Apama via real-time feeds and static data sources.

Apama applications are defined in any one of three ways or any combination:

- Java: Standard Java.
- Apama Event Programming Language (EPL): EPL is Apama's event-based language which has been designed around the requirements of defining and acting upon event patterns.
- Apama queries: graphically constructed applications built using the Software AG Designer environment.

4.4.1.1 Apama Correlator

The Apama Correlator (see Figure 4.9) is the core execution engine for the Apama platform. It acts as a "container" for Apama applications that can be injected and removed dynamically at run-time without disrupting other running applications. When injected into the Correlator they are divided into:

- Event patterns that define the events and patterns of events that an application is interested in. These are passed to internal units within the Correlator for high-performance execution.
- Application logic that is executed when event patterns are matched, such like a traditional "call-back" or "listener". The application code can perform any operation, including generating output events.

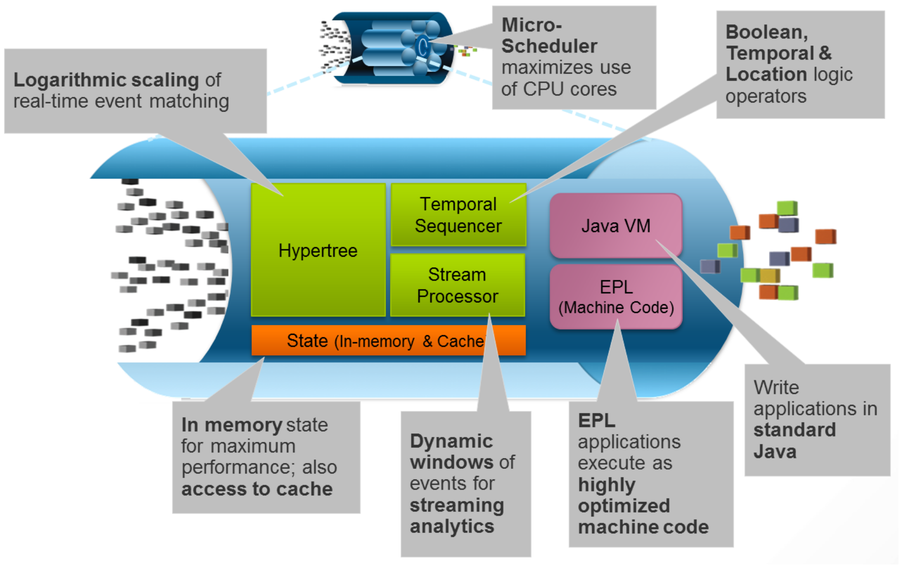

Micro-Scheduler maximizes use of CPU cores
Boolean, Temporal & Location logic operators
Logarithmic scaling of real-time event matching
Hypertree
Temporal Sequencer
Stream Processor
Java VM
EPL (Machine Code)
State (In-memory & Cache)
In memory state for maximum performance; also access to cache
Dynamic windows of events for streaming analytics
EPL applications execute as highly optimized machine code
Write applications in standard Java

Figure 4.9: Apama Correlator

The key elements that run an Apama application are:

- The Apama HyperTree, where events are initially presented to the Correlator and are immediately inspected via a high-performing algorithm. The HyperTree knows whether there is any value in processing a specific event any further (i.e., whether any application is interested in it) and what to do next.

- The temporal sequencer, which builds upon the event matching capabilities of the HyperTree to provide multiple temporal correlations. For example, if an application seeks event A, to be followed by event B within 500 milliseconds, the Correlator only concerns itself with looking for event A and does not waste effort looking for event B yet. Only when event A is detected does the Correlator begin to look for B, and only for a maximum of 500 milliseconds.

- The stream processor which stores and organizes windows of events and orchestrates the execution of real-time analytics over these event windows.

Identifying defined event patterns, the corresponding application is notified by invoking. Any action/application logic written in Java is executed within a built-in Java Virtual Machine (JVM) within the Correlator. Any action or application logic written in the Apama Event Programming Language (EPL) is executed as native machine code, giving a significant performance advantage; benchmarks show that applications execute faster in EPL than in either Java or C++. Correlators do not execute applications in a sequential manner. They can be seen as modular compositions that segment the event monitoring, analysis and action stages into logically related but independent segments. Thousands of these segment instances can operate simultaneously. To do this, Apama includes the notion of multiple parallel executing "contexts". Each context can be considered as a lightweight execution container which allows Apama to scale simply.

4.4.1.2 Monitors

The basic EPL structure is called a "monitor" that consists of two parts:

- A Listener sifts through the streams of all events passed to the Correlator seeking events that match against an event expression. Listeners provide a mechanism in Apama's EPL (or Java) to express and activate an event template. As events are injected into the Correlator, each attribute is examined and compared against all event templates for all active Listeners. Listeners are expressed declaratively, which is more appropriate for complex pattern matching.

- An Action (commonly expressed as an "action block") provides a set of imperative operations to be taken in case the associated listener fires. If the relevant event template specified by the listener is matched, the action goes on to invoke a different action. Note that the invocation of Listeners themselves is performed by action blocks (the "onload" action) as a bootstrap operation within the Correlator.

The example available in Listing 4.1 shows a simple monitor with event type definition, a global variable declaration, an event expression that specifies the pattern to be monitored and an action that operates on an event that matches the specified pattern.

```
package EventData;
//Definition of the event type that the correlator will receive.
//These events represent stock ticks from a market data feed.
event as StockTick {
 string name;
 float price;
}
//A simple monitor follows.
monitor SimpleShareSearch {
 // The following is a global variable for storing the latest
 // StockTick event.
 StockTick newTick;
 // The correlator executes the onload() action when you inject the
 // monitor.
 action onload() {
  on all StockTick(*,*):newTick processTick();
 }
 // The processTick() action logs the received StockTick event.
 action processTick() {
 log "StockTick event received" +
  " name = " + newTick.name +
  " Price = " + newTick.price.toString() at INFO;
 }
}
```

Listing 4.1: Example of an Apama monitor

4.4.1.3 Event streams

Apama EPL allows code authors to express event-driven programs using natural event processing constructs. An EPL program consists of a set of interacting monitors that receive, process and emit events. Monitor instances are self-contained, communicating with other monitor instances via events. An Apama application can, thus, be viewed as a dynamic network of interacting monitor instances communicating via events. It is dynamic because the application creates and destroys monitor instances in response to the external events received; similarly, the monitor instances dynamically subscribe and unsubscribe to particular event patterns or complex event expressions as needed. Thus, at any given instant, the application has only the monitor instances it needs and is only listening for the events of interest at that time. This novel approach makes Apama a highly efficient and responsive tool for complex event processing.

```
from s in all Temperature (sensorId = "T0001")
 within 60.0
 select Out (Last(s.value), mean(s.value), stddev(s.value)):o {
  // define the upper & lower bands as mean +/- 2x StdDev
  if ( o.value > (o.mean + o.stdv *2.0) )
        or ( o.value < (o.mean - o.stdv *2.0) ) then {
   log "Unusual Temperature for T0001";
  }
 }
```

Listing 4.2: Example of an Apama Event stream

The example in Listing 4.2 receives a stream of "Temperature" events where the sensorId is "T0001"; it maintains a 60 second window of these events and then builds an augmented event that includes the last reading, the moving average (mean) and the moving standard deviation calculations – these last two are over the data within the 60-second window. When we get an update to the window we then calculate some simpler upper and lower bands using the formula `Band := Mean +/- 2 x StdDev`. These define confidence bands around the temperature values such that if we received a temperature reading that lies outside these bands it would be 2 x Standard Deviations out of the norm, and we would define that here as unusual. When we see an unusual reading then we log an alert, but could of course react in a different way.

4.4.1.4 Apama queries

Apama queries let business analysts and developers create scalable applications that process events originating from very large populations of real-world entities. Scaling, both vertically (same machine) and horizontally (across multiple machines) is inherent in Apama query applications. Apama queries can be used alongside EPL monitors in the same correlator process, interacting by sending events between them. Incoming events that queries process are partitioned by, for example, customer account numbers, car license plate numbers, devices or some other entity. In a query application, the correlator processes the events in each partition independently of other partitions. Apama is often deployed in situations where requirements can change quickly and it's imperative to ensure that applications are developed quickly and evolved over time to address changing circumstances. Another aspect is often to ensure that the tools be made are accessible to business users, who can take a greater portion of control of the applications over time and who are inevitably the source of the ”business logic” that drives many event processing applications. Apama queries are designed to be easy to develop for both the business analyst and the application developer.

Graphical tools to specify the application design and full round-trip engineering allow both the business analyst and the developer to work on the same queries. At the developer level, an Apama query is defined using the Apama EPL.

Queries comprise:

- Parameters: an optional list of values that can be specified at runtime.
- Inputs: a list of event types which the query will consume; they are organized into windows of a time duration or number of events, and partitioned by the key fields of the event.
- Pattern: an event pattern, ranging from a simple query looking for a single event to complex patterns involving multiple event types.
- Conditions: optional restrictions on the event pattern, including time constraints, conditions on the contents of the events, or for detecting absence of events.
- Aggregates: optional aggregation such as averages, counts or user-defined aggregate operators.
- Actions: action to execute when the query fires – either EPL or sending an event to be output or processed further.

```
query ImprobableWithdrawalLocations {
 parameters{
  float period;
 }
 Inputs {
  Withdrawal(value>500) key cardNumber within period;
 }
 Find Withdrawal as w1 -> Withdrawal as w2
    where w2.country != w1.country {
 log "Suspicious withdrawal: " + w2.toString() at INFO;
 }
}
```

Listing 4.3: Example of an Apama Query

The code in Listing 4.3 provides an example of a query. This query monitors credit card transactions for a large set of credit card holders. The goal is to identify any fraudulent transactions. While this example illustrates query operation, it is not intended to be a realistic application.

4.4.1.5 Plug-ins

It is possible to link third-party libraries to the Correlator as plug-ins. By either coding or auto-generating a plug-in wrapper around the interface of the library, the library’s operations can be exposed to EPL. Plug-ins can be developed in Java, C++ or C. The fragment of EPL below shows how a plug-in library, which has been appropriately built, can be imported and its operations invoked from EPL.

```
import "apama_math" as math
// ...
float a, b;
// ...
a := math.cos(b)
```

Listing 4.4: Example for importing a third-party plug-in in Apama

4.4.1.6 Connectivity

Apama provides a suite of off-the-shelf connectivity and integration strategies plus a number of client software development kits. These allow developers to write custom adapters or software applications that interface existing enterprise applications, event sources and user interface clients to the Correlator. Integration with event data sources is a critical component to the delivery of CEP applications and is both a key component within the Apama architecture and a key focus of Apama's engineering efforts. The correlator can directly connect to a Java Message Service (JMS) bus or to Software AG's Universal Messaging broker. For the closest coupling, connectivity plugins provide a simple abstraction with high performance for in-process connectivity to external data sources. The Integration Adapter Framework (IAF) provides a separately monitorable process that can work with a range of adapters communicating with third-party messaging systems, extract and decode self-describing or schema-formatted messages, and transform them into Apama events. All of the connectivity options are bi-directional, providing both source and sink support. In addition to receipt of disparate events and normalization for processing within the correlator, Apama can generate events that are transformed by adapters into the requisite proprietary representations of the event (e.g., message on an ESB or an order to a trading exchange) as required by third-party messaging systems. Therefore, Apama adapters are the key mechanism by which Apama can trigger actions, converting internal events into a target format that is emitted to a target external service.

4.4.1.7 Connectivity plug-ins

Connectivity plug-ins allow adapters to be run in the same process as the correlator, providing a tightly coupled mechanism to send or receive events from external systems. A simple configuration file specifies the plug-ins used and provides powerful configuration data for plug-ins to use. A number of connectivity plug-ins can be used together to form a chain of plug-ins to separate transports from mapping or decoding functionality. Connectivity plug-ins can be written in Java or C++, and a mixture of plug-in types used within the same chain. Plug-ins handle events in a native, protocol-neutral key-value "map" format (using the standard Java "Map" interface). Sample plug-ins provided include:

- JSON codec to convert Apama events to/ from JSON form
- A simple HTTP client
- A simple HTTP server
- Classifiers for identifying the type of events via configuration
- Mapper to provide rule-based transformation of event fields.

4.4.1.8 Apama's Integration Framework

Adapters can also be hosted in Apama IAF (Integration Adapter Framework). The IAF is a middleware-independent and protocol-neutral adapter framework that is designed to allow easy and straightforward creation of software adapters to interface Apama with middleware buses and other message sources. All events, regardless of source, are converted to the internal format, thus enabling Apama to natively support correlations that span disparate external data formats. Sample plug-ins provided include:

- The Apama Database Connector (ADBC): ADBC connects to standard ODBC and JDBC data sources. Apama applications can use the ADBC adapter to store and retrieve data in standard database formats.
- Web Services Client Adapter: The adapter is a SOAP-based IAF adapter that allows Apama applications to invoke web services.

- File IAF Adapter: The File Adapter reads information from text files and writes them to text files by using Apama events.

4.4.1.9 Software AG Designer

Software AG Designer is the design tool for all Software AG products. Also for support of designing and writing Apama applications and services, Apama provides the comprehensive toolset called Software AG Designer provided as an Eclipse plugin. It is a full-featured development environment that includes a code/syntax editor, a debugger, a profiler, and a graphical editor for building Apama queries. Designer's use of Eclipse delivers many natural benefits to Apama developers, including the abilities to:

- Mix/match development tasks
- Take advantage of integration with third-party tools, such as source code repositories
- Create project definitions that manage source code components
- Incorporate specific features as plug-ins.

It supports a number of perspectives. An Eclipse perspective is a named organization of views, menus and toolbars that can be saved and switched to a unique tab of the organizer for a particular task. Software AG Designer perspectives include:

- A developer perspective and a workbench includes Monitors, Apama queries, Dashboards and Adapters
- A runtime perspective for the development of applications,
- A debugger perspective for debugging EPL applications,
- A runtime perspective for monitoring the execution of running services, such as Correlators and adapters), and applications, such as monitors and
- A profiler perspective for analyzing the execution statistics of running monitors.

4.4.1.10 Apama Streaming Analytics

Apama Streaming Analytics is built on an in-memory architecture that enables real-time processing of extremely fast, large data volumes – orders of magnitude larger than traditional database-based IT architectures. Predictive models developed in any kind of data mining tools, can be loaded in Predictive Model Markup Language (PMML) format. This step takes a fraction of the time and eliminates the need for manual coding, cross-checking and error correction. Apama Streaming Analytics ingests these models rapidly, making them instantly available to the process that they support, as defined by various Apama applications. It probes incoming event data from any device, social media stream or business system with extremely low latency against the imported predictive models for real-time scoring. Streaming data is analyzed and can also be enriched with historic and contextual data-at-rest where necessary, to identify patterns that have happened or are likely to happen. The platform's visualizations and visual analytics for business users support both human-oriented and automated intelligent actions, alerts and notifications. Streaming analytics include (relevant abstract):

- Filtering, correlation, aggregation and pattern detection with time and location constraints
- Enrichment of streaming data with context data
- High-performance messaging for mobile, the Web and the Internet of Things (IoT)
- In-memory architecture

- Operationalization of predictive models
- Support for MQTT and AMQP standards and protocols for easier integration with the IoT
- Deployment on local, server or cloud platforms.

4.4.1.11 Apama on the top of a distrusted system

Apama implements its own scalable processing using multiple system threads, maximizing the use of multi-core CPUs and by linking multiple correlators together across multiple machines. Apama can also establish connectivity to Kafka, so if a customer prefers a Kafka distribution architecture, Apama can be integrated into this architecture.

Its generic architecture is allowing multiple correlators across multiple environments (e.g. separate clouds) to communicate. This enables to communicate with the different clouds via sending message from correlator to correlator directly or via a messaging broker. In this way the Apama architecture can scale across multiple clouds. Apama's use cases includes running streaming analytics at multiple edge nodes.

4.4.1.12 Apama on federated clouds

Apama has a generic architecture that allows multiple correlators across multiple environments (e.g. separate clouds) to communicate. This allows it to communicate with the different clouds via sending message from correlator to correlator directly or via a messaging broker. In this way the Apama architecture can scale across multiple clouds. Apama's use cases include running streaming analytics at multiple edge nodes and then the processed data is then further processed in a centralised streaming analytics system.

4.4.1.13 Apama licensing

Software AG offers two different versions of Apama, the free-to-use Community Edition and the full version. The main restrictions of the Community Edition are the number of correlators (limited to 4) and the correlator's memory (limited to 1GB). For research and demonstration purposes as well as for initial professional exploitation, the Community Edition is expected to be fully sufficient. For an increased scalability, Software AG recommends the full version.

4.4.2 Siddhi

One of recently emerged CEP engines is Siddhi [217]. It is a lightweight, easy-to-use, open source Complex Event Processing server. Siddhi represents a new generation of open source CEP engines, designed to satisfy two challenges for the traditional CEP approaches (like Esper [70]), arising mainly due to an expansion of the real-time data sources (big data):

- latency in the processing has become very critical, so that CEP tasks have to be parallelized (scalability is an issue)
- complexity of the situations to be detected requires networks of CEP engines in order to process the real-time data

The most important advantages of Siddhi, compared to Esper, are:

1. Siddhi engine is massively scalable (required for big data). Esper cannot scale efficiently (cannot be distributed).

2. Siddhi is based on Apache Storm and enables the definition of complex workflows based patterns (request from use cases). Esper is based on the language (EPL) that cannot support workflow-like processing.
3. Siddhi/WSO2 (Web Services Oxygenated) is available under Apache Software License Version 2.0, which is the best possible open source license for the commercialization, if we want do exploit the results. Esper is LGPL and for the commercial license the price is quite high – so exploitation opportunities of Esper-based solutions are less obvious.
4. The community around Siddhi is better organized, so support is stronger

Other advantages are:

- Siddhi identifies the most meaningful events within the event cloud, analyzes their impact, and acts on them in real-time.
- It is built to be extremely high performing with WSO2 Siddhi and massively scalable using Apache Storm.
- The WSO2 CEP is built up on the award-winning, WSO2 Carbon platform, which is based on the OSGi framework enabling better modularity for realizing service oriented architecture (SOA). The WSO2 Carbon framework contains many enhanced features and optional components to customize the behaviour of the server through simple, point-and-click provisioning.
- The CEP can be tightly integrated with WSO2 Data Analytics Server, by adding support for recording and posting processing events with Map-Reduce via Apache Spark, and WSO2 Machine Learner for predictive analytics.

Figure 4.10 describes the structure of the engine.

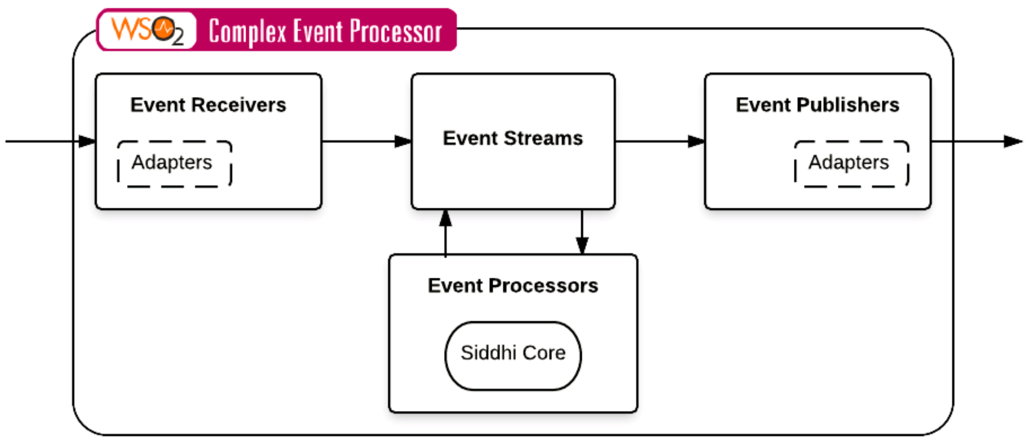


Figure 4.10: WSO2 CEP architecture

The WSO2 CEP architecture consists of the following components:

- **Event receivers**: Event receivers receive events that are coming to the CEP. WSO2 CEP supports the most common adapter implementations by default. For specific use cases, plug custom adapters can be used.
- **Event streams**: Event streams contain unique sets of attributes of specific types that provide a structure based on which the events processed by the relevant event flow are selected. Event streams are stored as stream definitions in the file system via the data bridge stream definition store.

- **Event processors**: Event processor handles actual event processing. It is the core event processing unit of the CEP. It manages different execution plans and processes events based on logic, with the help of different Siddhi queries. Event Processor gets a set of event streams from the Event Stream Manager, processes them using Siddhi engine, and triggers new events on different event streams back to the Event Stream Manager.

- **Event publishers**: Event publishers publish events to external systems and store data to databases for future analysis. Like the event receivers, this component also has different adapter implementations. The most common ones are available by default in the CEP. A user can implement custom adapters for specific use cases.

4.4.2.1 Architecture

At a very high level, Siddhi receives incoming events in "Event Streams" via input handlers, processes them, and notifies the output via callbacks. Here, we use the term Event Streams when the events in a particular Event Stream have a definite schema and when they are logically ordered in time.

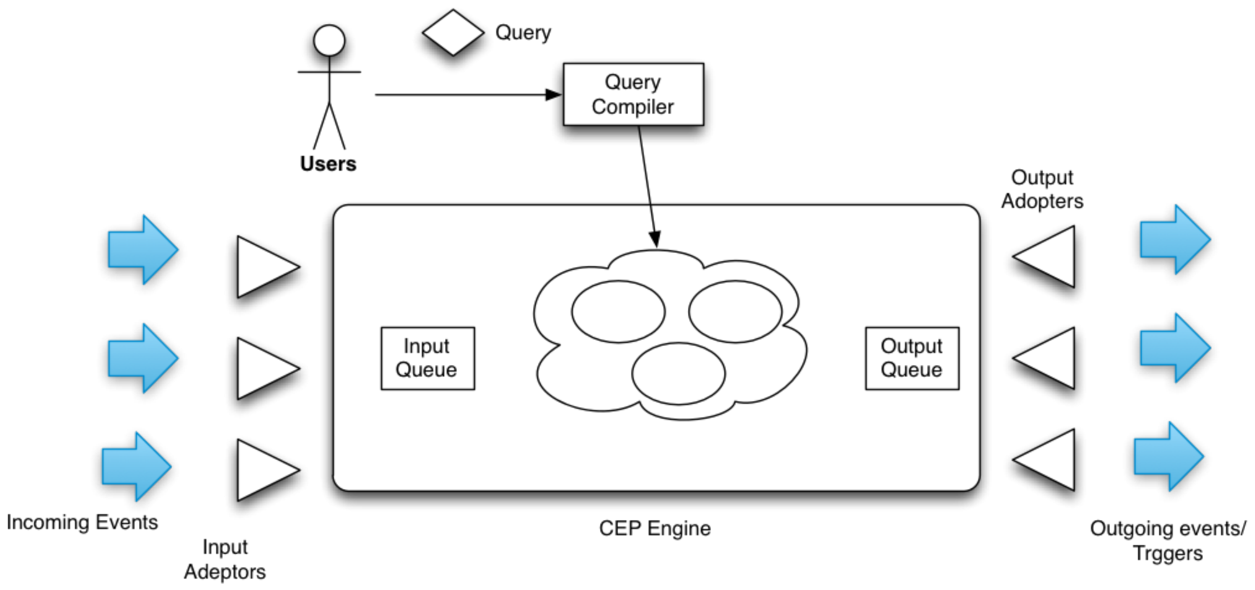


Figure 4.11: High-level architecture of Siddhi

Stream Definition

Each event in Siddhi has a Stream ID representing the Event Stream it belongs to, timestamp representing the event creation time, and an `Object[]` array containing the data attributes of the events.

To process an event stream in Siddhi, we have to first define that stream; e.g.

```
define stream StockQuartStream (symbol string, price float, volume int);
```

When defining streams, we specify its name and its attributes, and each of the attributes is defined as pairs of their name and type in order. Note: in WSO2 CEP, users are not given the option to explicitly define an event stream, whereas in WSO2 CEP 2.x, defining the stream is implicitly done using the input/output mapping of the CEP bucket.

When an event stream is defined, internally, Siddhi creates an input handler, which we can use to send events into the system on the defined event stream. At the same time, we can also add callbacks to event streams, which will receive notifications when events are produced on those event streams.

4.4.2.2 Siddhi Queries

Siddhi supports the following complex event processing queries through its SQL-like query language:

1. filter
2. window
3. join
4. sequence
5. patterns

Siddhi Event Query Language has the following structure:

```
from <incoming stream>[<incoming stream filter>]#<window on the stream>

      insert into <outgoing stream> <outgoing stream attributes>
```

Here, when events arrive from the incoming event streams, they are filtered and only the success events of the filter will flow to the window. These windows, based on their configuration, sustain some of the incoming events for a certain period of time for further processing, like aggregation calculations. Finally, all these events will be projected on the outgoing event streams based on the defined outgoing stream attributes.

Basic Siddhi Queries

Filter Query

The following is a code snippet demonstrating a simple filter query in Siddhi.

```
    from StockQuartStream[symbol == 'FB']
insert into FacebookStockStream price, volume;
```

When a query is defined, it will implicitly define its output stream. Hence, in this case, the above query will implicitly define `FacebookStockStream` to have the price of type float and volume of type `int`.

Window Query

If we want to calculate how many Facebook stocks are traded in the last minute, we can improve the above query by adding a time window.

```
from StockQuartStream[symbol == 'FB']#window.time(1 min)
          insert into FacebookCountStockStream
        price, volume, count(price) stockCount;
```

Similar to windows in WSO2 CEP 2.1.0, Siddhi also supports transforming streams using `#transform(...)`.

Basic Siddhi Query Architecture

The architecture of a basic Siddhi query (having Filter, Transform, and Window) is shown in Figure 4.12.

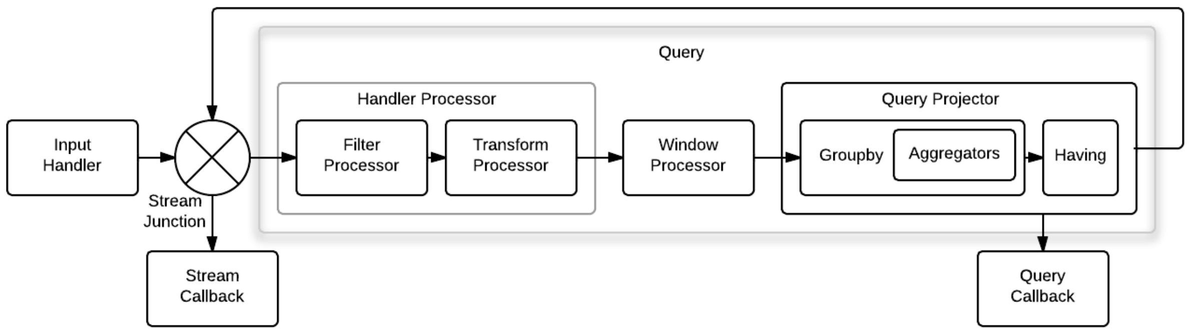


Figure 4.12: Siddhi basic query

Here, the events flow from the "Input Handler" of the incoming "Event Stream" to its respective "Stream Junction". The Stream Junction is responsible to send the events to all components that are registered to that Event Stream. In Siddhi, we can find two main types of Stream Subscribers; Stream Callback – which is used to notify an event occurrence on a particular stream –, and Query Handler Processors – which are responsible for filtering and transforming the events for further processing. Only the event that passes the filter conditions will be outputted from the Query Handler Processor, which will indeed be fed into the Window processor where the events will be stored for time, length or uniqueness-based, or other custom processing. The events are then fed into the Query Projector to perform event attribute level processing such as avg(price), group by and having. Finally, the output of Query Processor will be sent to its registered Query Callback and its output stream's Stream Junction where the event will be fed to all the Queries and Stream Callbacks registered to that Event Stream.

These output streams are implicitly defined by inferring the query, and hence, we don't need to define them explicitly.

4.4.3 Comparison of CEP engines

Having no access to other commercial products we focused on OSS and as Software AG is partner in the consortium on APAMA.

4.4.3.1 Apache Storm

Apache Storm is an open source distributed real-time computation system. Storm can be reliably process unbounded streams of data. Storm is simple, can be used with any programming language. Apache Storm is the oldest and most mature codebase (2011). It has low latency but low throughput event processor. Storm has a very large community with rich resources.

4.4.3.2 Apache Spark Streaming

Apache Spark is an open source stream processing engine. It is strictly a stream processing engine focused on high throughput and reliability. It can be installed and used both On-Premises and in a cloud. Most cons

- Complex to Implement

- Works with external data stores
- Built for batch workloads and manages "streaming" by using many micro-batches
- Cannot process data in real-time
- Large community since 2012 release

4.4.3.3 Apache Kafka Streams API

Apache Kafka Streams is an open source stream processing API for developers building applications based on Kafka messaging. The Input streams from Kafka are transformed by Streams API to output streams. Main cons:

- Not available for use as a cloud/on-premise "service"
- Provided as a Java API for Kafka
- Underlying messaging system must be Kafka
- Streams only
- No straightforward way to do discrete pattern matching
- Architected for throughput, not low latency
- Tooling is for Java code programmers
- No higher level business analyst tooling

	Apama	**Apache Spark Streaming**	**Apache Kafka Streams**	**Apache Storm**
Graphical development environment	Yes	No	No	No
Low Latency	Yes	No	Partial (reduced throughput)	Yes
True real-time (no batching)	Yes	No	Yes (batch size of 1)	N/A
Event-based programming language	Yes	No	No	No
Discrete event pattern detection	Yes	No	No	N/A
High Availability	Yes	Yes	Partial(via Kafka)	N/A
Small footprint (suitable for edge devices)	Yes	No	No	N/A
On-Premise Version	Yes	Yes	N/A (Java library)	N/A
Cloud Version	Yes	Impetus, Databricks (AWS)	N/A (Java library)	N/A

Table 4.1: Comparison between CEP engines

4.4.3.4 Apache Samsa

Originally developed alongside Kafka for LinkedIn. Newer engine (2014) with close ties to Hadoop YARN. Technically superior to Storm and Spark as combining parts of them, but weakest community of all Apache CE Solutions.

4.4.3.5 Apache Apex

Handed to Apache by DataTorrent and OSS since 2016. It seems to be intended to unify stream and batch processing as a platform for building distributed applications on Hadoop. It has a limited community and adoption for native OSS offering.

4.4.3.6 Apache Flink

Flink is OSS since 2015. Flink has solid stream processing capabilities, but as a new project, it has a small community and limited adoption so far.

4.4.3.7 Esper

Esper (non commercial) is an Open Source Complex Event Processing (CEP) -– engine only having separate versions for Java and .NET applications build by EsperTech. OSS version of Esper can easily be accessed and start working with is for free. It has a capable, mature, basic CEP engine but does not offer lowest latency or highest volumes. Esper is customizable in high manner and provides flexibility for building own applications and solutions.

Chapter 5

Conclusion

The main goal of the PrEstoCloud project is to advance the cloud computing virtual infrastructure in order to enable dynamically scalable infrastructure and deployment of distributed big-data stream applications, across multiple federated and hybrid clouds, and up to the edge of the network. PrEstoCloud will primarily target custom-made applications which are composed of micro-services – i.e. small, independent processes communicating with each other by using language-agnostic APIs. Moreover, it targets data-intensive applications, i.e. applications which use a data parallel approach to process large volumes of data, and in particular, applications that engage and process streaming big data streams. In this Deliverable, we have reviewed the state-of-the-art of the technologies and methodologies in the relevant technical fields.

In Chapter 2, we discussed the cloud infrastructure. We reviewed the existing tools and techniques to manage and monitor applications running on federated and hybrid clouds, as well as the challenges posed by the edge computing paradigm. We saw that, while there are many techniques devoted to the management and monitoring of applications running on federated/hybrid clouds, the solutions are usually not inter-operable, due to lack of standards. Moreover, we saw that the current generation or architectures related to edge/IoT devices rely on static pre-configurations of both hardware and software that is unfit for a dynamic environment. Finally, we compared the foreseen PrEstoCloud platform with the last offering in IoT software stacks: Amazon AWS Greengrass.

In Chapter 3, we looked at the anticipation of changes, in order to adapt the infrastructure on which the application is running. We introduced the related challenges: situation awareness, context detection, adaptivity and the capability to predict situations adaptation. We saw that there is currently no approach that goes beyond cloud resource selection and allocation towards recommending cloud and edge resource adaptations based on the dynamically changing processing needs, taking under consideration trade-offs between cost, speed, efficiency and reliability of adaptations. In PrEstoCloud, we plan to investigate new ways to manage the relations between Big Data processing, cloud and edge resources adaptation needs and adaptation strategies. Our approach will extend state of the art through a novel data analytics approach for understanding from the past data different modes of the workloads and predicting which can be the next one based on the current situation. The main idea is to use as many as possible factors that influence the workload and build behavioral models from past data. These models are truly data driven and build in an unsupervised manner (no labeled data a priori needed). The main advantage is that the past data will "explain" what is the usual behavior of the system in a high-dimensional space, which can be dynamically changed as incoming data will be changed. In that way we ensure that the model will be continuously updated to the new situations (resolving the problem of the model drift). Moreover, this model is used in real-time in order to check what the current parameters of the system indicate regarding the workload, i.e. can the current situation be classified as "usual" (with the standard set of actions), or it is unusual and required an additional processing.

Situation-awareness will be supported using machine learning methods that will analyse contextual data and will infer high level contextual information useful for making the adaptation recommendations. With respect to adaptation recommendations, we plan to research and develop two recommenders that will issue recommendations with respect to adapting resources allocation in real-time and deploying fragments of the data-intensive applications across the processing topology. The processing topology is the cloud and edge resources available and capable of executing the application fragments. For example, a private cloud and a set of processing units close to or at the edge, e.g., CCTVs, mobile phones, regional processing units, drones. These two recommenders will be designed and implemented as part of WP5 work and will be reported in terms of deliverables D5.5 and D5.6.

Specifically, we aim to develop the Application Fragmentation & Deployment Recommender that will undertake the responsibility of describing the appropriate fragmentation of cloud applications into smaller parts in order to be efficiently deployed over cloud / edge resources. Moreover, it will associate applications and application fragments with placement constraints and optimization preferences. The input of this mechanism should involve the available VM flavours & edge devices (i.e. resources types) as well as the qualitative, quantitative preferences of the DevOp and/or the Application developer. Based on this input the recommended fragmentation will be serialized in a TOSCA specification that will refer to type-level VM or Edge resources (as hosting nodes), while the specific instances will be decided based on the advanced optimization mechanism of the PrEstoCloud control layer.

The second recommender, called Resources Adaptation Recommender, is expected to engage the run-time operation of our platform since it will provide context-aware, edge and cloud adaptation recommendations that may include changes to the already used resources and reconfigurations with respect to where each application fragment has been hosted. It will receive as input the current processing topology and placement (i.e. resources used and hosting location of each application fragment), the detected situations along with the respective context of the used and the available edge devices. Based on this, it should be able to generate as output the recommendation to reconfigure the processing topology, e.g., to introduce new processing nodes, replicate nodes for failover purposes, remove redundant or underused processing nodes and move application fragments among the available cloud and edge hosting nodes. For example, assume multiple streams coming from a variety of different cameras (i.e. CCTVs, mobile phones). Based on the data volume and velocity of these streams, the PrEstoCloud platform will be able to detect appropriate situations and recommend adaptations that will affect the processing topology. Such adaptations may involve moving away or closer to the edge certain application fragments (e.g. video transcoding, face detection) and/or instructing the use of additional instances of the same application fragments on different virtual hosts. Furthermore, we can imagine the diversity of the considered multiple streams used for adaptation being further augmented by streams that may further enhance a certain multi-cloud application. For example, the consideration of social media streams for analyzing and detecting security incidents could even allow PrEstoCloud to better predict the sudden increase of the current workload (i.e. several people reporting on twitter about a gunfire, will most likely start streaming video through their phones immediately after their textual post).

In addition, we will propose algorithms for devising proactive adaptation actions by taking into account efficiently the outcomes of Big Data analytics and complex event processing. To support predictive behavior, instead of using the same threshold-based alerts for an entire class of cloud resources as most monitoring systems typically use, we will provide highly specialized adaptation recommendations based on monitoring multiple data streams, understanding each individually as well as their relationship to each other, resulting in a highly sensitive system that can provide early recommendations for adaptations to optimise performance. Our work will focus on recommending auto-scaling decisions on a hybrid VM and container system. The approaches presented by OS-level virtualization and VMs are complementary, and we have the intention of combining them in order to reduce the costs associated with starting a VM, and make more efficient use of the allocated resources.

Finally, in Chapter 4, we reviewed a number of tools that can be considered as building blocks for the PrEstoCloud platform. We first introduced ProActive, an open-source cloud broker, able to deploy and monitor applications across clouds, through each cloud's specific API entry points. Then, we introduced BtrPlace, an open-source scheduler able to place VMs under constraints, while optimizing an objective. Despite numerous existing solutions to manage VM in data centers, none take the benefit from hybrid architectures. More importantly, only but BtrPlace is flexible enough to fill this gap, because no other solution supports the possibility to schedule either over an edge cloud, a private cloud, or a public cloud, depending on the Service-Level Objective and the infrastructures capabilities. We then reviewed the solutions related to the communications between the multiple deployment sites. We first reviewed the existing networking solutions available within public and private clouds, and the protocols available for interconnecting different clouds together. Our work in this domain will leverage existing solutions in public cloud infrastructures to create a virtual network connecting the multiple sites where the application is deployed. Our solution will rely on standardized network protocols so that they can easily be integrated into private corporate clouds, or in less powerful environments, such as edge clouds. We then reviewed existing data brokers technologies, on which PrEstoCloud will rely to deploy a publish/subscribe model of relation between the edge devices, the application, and the platform components. Finally, we introduced and reviewed CEP engines, technologies that will be used inside the meta-management layer in order to process the meta-data generated by edge devices and application.

On top of reviewing the state-of-the-art in all of these different fields, this Deliverable will be a helpful reference for establishing a common understanding among the partners of the PrEstoCloud consortium. For technical partners, it is an opportunity to understand each other's field of expertise, in order to better integrate their solutions together, in a coherent manner. For pilot partners, the Deliverable will make clear the existing abstractions regarding the technical disciplines, enabling them to better formulate their needs with respect to the PrEstoCloud architecture and its expected behavor.

Bibliography

[1] G. D. Abowd, M. Ebling, G. Hung, H. Lei, and H. W. Gellersen, "Context-aware computing", *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 22–23, Jul. 2002.

[2] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness", in *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, ser. HUC '99, Springer, 1999, pp. 304–307.

[3] A. Adi and O. Etzion, "Amit - the situation manager", *The VLDB Journal*, vol. 13, no. 2, pp. 177–203, May 2004.

[4] D. Agrawal, S. Das, and A. E. Abbadi, "Big data and cloud computing: New wine or just new bottles?", in *Proceedings of the VLDB Endowment 3, 1-2*, Sep. 2010, 1647–1648.

[5] F. Al-Haidari, M. Sqalli, and K. Salah, "Impact of cpu utilization thresholds and scaling size on autoscaling cloud resources", in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 2, Dec. 2013, pp. 256–261.

[6] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu, "Vmflock: Virtual machine co-migration for the cloud", in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, ser. HPDC '11, ACM, 2011, pp. 159–170.

[7] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. Jayaraman, S. Khan, A Guabtni, and V. Bhatnagar, "An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art", *Computing Journal*, vol. 97, no. 4, pp. 357–377, Apr. 2015.

[8] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures", in *2012 IEEE Network Operations and Management Symposium*, Apr. 2012, pp. 204–212.

[9] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control", in *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*, ser. ScienceCloud '12, ACM, 2012, pp. 31–40.

[10] A. Alshalan, S. Pisharody, and D. Huang, "A survey of mobile vpn technologies", *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1177–1196, 2016.

[11] Amazon CloudWatch, *Developer guide*, http://awsdocs.s3.amazonaws.com/AmazonCloudWatch/latest/acw-dg.pdf.

[12] Amazon Web Services, *Amazon ec2 dedicated hosts*, https://aws.amazon.com/ec2/dedicated-hosts/?nc1=h_ls.

[13] ——, *Aws direct connect*, https://aws.amazon.com/fr/directconnect/.

[14] ——, *Multiple region multi-vpc connectivity*, https://aws.amazon.com/answers/networking/aws-multiple-region-multi-vpc-connectivity/.

[15] ——, *Placement groups*, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html.

[16] ——, *Regions and availability zones*, http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html.

[17] ——, *Step 3: Download the loa-cfa*, http://docs.aws.amazon.com/directconnect/latest/UserGuide/getstarted.html#DedicatedConnection.

[18] ——, *What is amazon vpc?*, http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Introduction.html.

[19] Amazon Web Services (AWS), *Auto scaling*, https://aws.amazon.com/autoscaling/.

[20] ——, *Cloud computing services*, https://aws.amazon.com/.

[21] Apache CloudStack, *Open source cloud computing*, https://cloudstack.apache.org.

[22] Apache Storm, *Concepts*, http://storm.apache.org/releases/1.0.0/Concepts.html.

[23] J. Aungiers, *Lstm neural network for time series prediction*, http://www.jakob-aungiers.com/articles/a/LSTM-Neural-Network-for-Time-Series-Prediction, Dec. 2016.

[24] D. A. Bacigalupo, J. van Hemert, A. Usmani, D. N. Dillenberger, G. B. Wills, and S. A. Jarvis, "Resource management of enterprise cloud systems using layered queuing and historical performance models", in *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW)*, Apr. 2010, pp. 1–8.

[25] L. Baresi, S. Guinea, A. Leva, and G. Quattrocchi, "A discrete-time feedback controller for containerized cloud applications", in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016, ACM, 2016, pp. 217–228.

[26] J. Barr, *Cloudbursting -– hybrid application hosting*, http://aws.typepad.com/aws/2008/08/cloudbursting.html, Aug. 2008.

[27] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud", *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[28] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques", *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161 –180, 2010.

[29] S. Blagodurov, A. Fedorova, E. Vinnik, T. Dwyer, and F. Hermenier, "Multi-objective job placement in clusters", in *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015.

[30] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters", in *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, ser. HotCloud'09, USENIXn, 2009.

[31] B. A. Bonab and O. Bushehrian, "A semi-automated reverse engineering method to recommend the best migration-to-cloud strategy", in *2015 International Symposium on Computer Science and Software Engineering (CSSE)*, Aug. 2015, pp. 1–7.

[32] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things", in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, ACM, 2012, 13–16.

[33] G. Bowerman, *Working with large virtual machine scale sets*, https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-placement-groups.

[34] J. Brownlee, *How to create an arima model for time series forecasting with python*, http://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/, Jan. 2017.

[35] ——, *Time series prediction with deep learning in keras*, http://machinelearningmastery.com/time-series-prediction-with-deep-learning-in-python-with-keras/, Jul. 2016.

[36] ——, *Time series prediction with lstm recurrent neural networks in python with keras*, http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/, Jul. 2016.

[37] R. Bryant, A. Tumanov, O. Irzak, A. Scannell, K. Joshi, M. Hiltunen, A. Lagar-Cavilla, and E. de Lara, "Kaleidoscope: Cloud micro-elasticity via vm state coloring", in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11, ACM, 2011, pp. 273–286.

[38] D Caromel, C. Delbé, M Leyton, *et al.*, "Proactive: An integrated platform for programming and running applications on grids and p2p systems", *Computational Methods in Science and Technology*, vol. 12, no. 1, 2006.

[39] E. Casalicchio and L. Silvestri, "Autonomic management of cloud-based systems: The service provider perspective", in *Computer and Information Sciences III: 27th International Symposium on Computer and Information Sciences*. Springer, 2013, pp. 39–47.

[40] E. Cavalcante, T. Batista, F. Lopes, A. Almeida, A. L. de Moura, N. Rodriguez, G. Alves, F. Delicato, and P. Pires, "Autonomous adaptation of cloud applications", in *Distributed Applications and Interoperable Systems: 13th IFIP WG 6.1 International Conference, DAIS 2013*. Springer, 2013, pp. 175–180.

[41] P. M. Chen, *ReVirt*, https://web.eecs.umich.edu/~pmchen/covirt/software.html.

[42] Cheryl McGuire et al., *Create a site-to-site connection in the azure portal*, https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-howto-site-to-site-resource-manager-portal.

[43] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud", in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11, Salzburg, Austria: ACM, 2011, pp. 301–314.

[44] CIMI, *Cloud infrastructure management interface (cimi) model and restful http-based protocol*, ISO/IEC 19831, ISO Standards Catalogue.

[45] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines", in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05, USENIX Association, 2005, pp. 273–286.

[46] S. Clayman and A. Galis, "INOX: A managed service platform for inter-connected smart objects", in *Proceedings of the workshop on Internet of Things and Service Platforms*, ser. IoTSP '11, ACM.

[47] CloudHarmony, *Transparency for the cloud*, http://cloudharmony.com/.

[48] Cloudify, *Pure-play cloud orchestration*, http://getcloudify.org.

[49] CloudStack Documentation, *Advanced networking guides*, http://docs.cloudstack.apache.org/en/latest/#advanced-networking-guides.

[50] A. Comi, L. Fotia, F. Messina, G. Pappalardo, D. Rosaci, and G. M. L. Sarné, "An evolutionary approach for cloud learning agents in multi-cloud distributed contexts", in *2015 IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Jun. 2015, pp. 99–104.

[51] CRIU, *Checkpoint/restore in userspace*, https://criu.org/Main_Page.

[52] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload", in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10, 2010.

[53] C. Cui, *Network function virtualization: Network operator perspectives on industry progress*, EST White Paper, https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf.

[54] W. Cui, H. Zhan, B. Li, H. Wang, and D. Cao, "Cluster as a service: A container based cluster sharing approach with multi-user support", in *Service-Oriented System Engineering (SOSE), 2016 IEEE Symposium on*, IEEE, 2016, pp. 111–118.

[55] DC4Cities, *Let existing and new data centres become energy adaptive*, EU FP7 2014-2016, http://www.dc4cities.eu/en/.

[56] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems", in *Proceedings of the 9th International Conference on Autonomic Computing*, ser. ICAC '12, ACM, 2012, pp. 191–200.

[57] U. Deshpande, D. Chan, T. Y. Guh, J. Edouard, K. Gopalan, and N. Bila, "Agile live migration of virtual machines", in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 1061–1070.

[58] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines", in *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, ser. HPDC '11, ACM, 2011, pp. 135–146.

[59] Docker, *Build, ship, and run any app, anywhere*, https://www.docker.com/.

[60] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, *Forwarding and Control Element Separation (ForCES) Protocol Specification*, RFC 5810, Mar. 2010.

[61] D. Duplyakin, P. Marshall, K. Keahey, H. Tufo, and A. Alzabarah, "Rebalancing in a multi-cloud environment", in *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, ACM, 2013.

[62] C. Dupont, T. Schulze, G. Giuliani, A. Somov, and F. Hermenier, "An energy aware framework for virtual machine placement in cloud federated data centres", in *2012 Third International Conference on Future Systems: Where Energy, Computing and Communication Meet (e-Energy)*, IEEE, May 2012, pp. 1–10.

[63] C. Dupont, F. Hermenier, T. Schulze, R. Basmadjian, A. Somov, and G. Giuliani, "Plug4Green: A flexible energy-aware VM manager to fit data centre particularities", *Ad Hoc Networks*, pp. 1–16, Nov. 2014.

[64] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck, "From data center resource allocation to control theory and back", in *2010 IEEE 3rd International Conference on Cloud Computing*, Jul. 2010, pp. 410–417.

[65] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: Towards a fully automated workflow", in *7th International Conference on Autonomic and Autonomous Systems, ICAS 2011*, IEEE, May 2011, pp. 67–74.

[66] D. Eastlake 3rd, *Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)*, RFC 4305, Dec. 2005.

[67] "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications", *Future Generation Computer Systems*, vol. 32, pp. 82 –98, 2014.

[68] M. Endsley, *Designing for Situation Awareness: An Approach to User-Centered Design, Second Edition*. CRC Press, 2016, ISBN: 9781420063585.

[69] D. Erickson, "The beacon openflow controller", in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13, 2013, pp. 13–18.

[70] EsperTech, *Event series intelligence*, http://www.espertech.com/.

[71] O. Etzion and P. Niblett, *Event Processing in Action*, 1st ed. Manning Publications.

[72] B. Evans, *The top 5 cloud-computing vendors: #1 microsoft, #2 amazon, #3 ibm, #4 salesforce, #5 sap*, https://www.forbes.com/sites/bobevans1/2017/11/07/the-top-5-cloud-computing-vendors-1-microsoft-2-amazon-3-ibm-4-salesforce-5-sap/#721f4db26f2e, Nov. 2017.

[73] *Event Processing in Action*. Manning Publications, 2010.

[74] Fit4Green, *Federated IT for a sustainable environmental impact*, EU FP7 2009-2011.

[75] I. Foulds and R. Squillace, *Regions and availability for virtual machines in azure*, https://docs.microsoft.com/en-us/azure/virtual-machines/windows/regions-and-availability, Apr. 2017.

[76] U. Franke and J. Brynielsson, "Cyber situational awareness - A systematic review of the literature", *Computers & Security*, vol. 46, pp. 18–31, 2014.

[77] S. Frankel and S. Krishnan, *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap*, RFC 6071, Feb. 2011.

[78] S. Frey, C. Luethje, C. Reich, and N. Clarke, "Cloud qos scaling by fuzzy logic", in *2014 IEEE International Conference on Cloud Engineering*, Mar. 2014, pp. 343–348.

[79] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, model-driven autoscaling for cloud applications", in *11th International Conference on Autonomic Computing (ICAC 14)*, USENIX, 2014, pp. 57–64.

[80] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control", in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14, 2014, pp. 163–174.

[81] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy", in *2011 IEEE 4th International Conference on Cloud Computing*, Jul. 2011, pp. 716–723.

[82] M. F. Gholami, F. Daneshgar, G. Low, and G. Beydoun, "Cloud migration process-a survey, evaluation framework, and open challenges", *Journal of Systems and Software*, vol. 120, no. C, pp. 31–69, Oct. 2016.

[83] I. Gog, M. Schwarzkopf, A. Gleave, R. N. M. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale", in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 99–115.

[84] I. Goiri, J. Guitart, and J. Torres, "Characterizing cloud federation for enhancing providers' profit", in *Proceedings of the 3rd International Conference on Cloud Computing*, ser. CLOUD 2010, IEEE, Jul. 2010, 123–130.

[85] M. Gualtieri, R. Curran, H. Kisker, E. Miller, and M. Izzi, *The forrester wave: Big data streaming analytics, q1 2016*, https://www.forrester.com/report/The+Forrester+Wave+Big+Data+Streaming+Analytics+Q1+2016/-/E-RES129023, Mar. 2016.

[86] Q. Guan and S. Fu, "Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures", in *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, Sep. 2013, pp. 205–214.

[87] T. Guo, U. Sharma, T. Wood, S. Sahu, and P. Shenoy, "Seagull: Intelligent cloud bursting for enterprise applications", in *Proceedings of the 2012 USENIX Annual Technical Conference*, ser. ATC'12, 2012.

[88] Guy Bowerman et al., https://docs.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-overview.

[89] E. Haleplidis, J. Hadi Salim, S. Denazis, and O. Koufopavlou, "Towards a network abstraction model for sdn", *Journal of Network and System Management*, vol. 23, no. 2, pp. 309–327, Apr. 2015.

[90] L. Hardesty, *Onos joins the linux foundation, becoming an opendaylight sibling*, https://www.sdxcentral.com/articles/news/onos-joins-the-linux-foundation-becoming-an-opendaylight-sibling/2015/10/, Oct. 2015.

[91] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast distribution with lazy docker containers", in *14th USENIX Conference on File and Storage Technologies (FAST 16)*, 2016, pp. 181–195.

[92] S. Herker, A. Khan, and X. An, "Survey on survivable virtual network embedding problem and solutions", in *Proceedings of the 9th IEEE International Conference on Networking and Services*, ser. ICNS 2013, 2013, 99–104.

[93] F. Hermenier, J. Lawall, and G. Muller, "Btrplace: A flexible consolidation manager for highly available applications", *IEEE Transactions on Dependable and Secure Computing*, vol. 10, no. 5, pp. 273–286, 2013.

[94] F. Hermenier, S. Demassey, and X. Lorca, "Bin Repacking Scheduling in Virtualized Datacenters", in *17th Conference on Principles and Practice of Constraint Programming*, ser. CP 2011, vol. 6876, Springer, Sep. 2011, pp. 27–41.

[95] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: A consolidation manager for clusters", in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '09, ACM, 2009, pp. 41–50.

[96] Hewlett-Packard, *HP OpenFlow and SDN Technical Overview*, http://h17007.www1.hpe.com/docs/networking/solutions/sdn/devcenter/02_-_HP_OpenFlow_and_SDN_Technical_Overview_TSG_v1_2013-10-01.pdf, Oct. 2013.

[97] ——, *Hp procurve switch 5400zl series quickspecs*, https://www.hpe.com/h20195/v2/getpdf.aspx/c04111646.pdf?ver=12.

[98] M. Hilton, A. Christi, D. Dig, M. Moskal, S. Burckhardt, and N. Tillmann, "Refactoring local to cloud data types for mobile apps", in *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft 2014, ACM, 2014, pp. 83–92.

[99] P. Hoenisch, I. Weber, S. Schulte, L. Zhu, and A. Fekete, "Four-fold auto-scaling on a contemporary deployment platform using docker containers", in *Service-Oriented Computing: 13th International Conference, ICSOC 2015, Goa, India, November 16-19, 2015, Proceedings*. Springer, 2015, pp. 316–323.

[100] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and practice*, Open-Access Textbook https://www.otexts.org/fpp/9/2.

[101] IDERA, *Uptime cloud monitor*, https://www.idera.com/infrastructure-monitoring-as-a-service.

[102] IMT Atlantique, *Choco solver*, http://www.choco-solver.org/.

[103] S. Inese, P. Inese, B. Solvita, G. Janis, M. Egils, and O. Edgars, "Decomposition of enterprise application: A systematic literature review and research outlook", *Information Technology and Management Science*, vol. 18, no. 1, pp. 30–36, 2015.

[104] International Computer Science Institute, UC Berkeley, *The nox controller*, https://github.com/noxrepo/nox, 2012.

[105] C. Inzinger, W. Hummer, B. Satzger, P. Leitner, and S. Dustdar, "Generic event-based monitoring and adaptation methodology for heterogeneous distributed systems", *Software: Practice and Experience*, vol. 44, no. 7, pp. 805–822, 2014.

[106] C. Inzinger, B. Satzger, P. Leitner, W. Hummer, and S. Dustdar, "Model-based adaptation of cloud computing applications", in *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2013)*, 2013, pp. 351–355.

[107] R. Izard, *Floodlight documentation*, https://floodlight.atlassian.net/wiki/, Mar. 2015.

[108] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth", in *In Proceedings of Passive and Active Measurements (PAM) Workshop*, Citeseer, 2002.

[109] P. Jamshidi, C. Pahl, and N. C. Mendonca, "Managing uncertainty in autonomic cloud elasticity controllers", *IEEE Cloud Computing*, vol. 3, no. 3, pp. 50–60, 2016.

[110] P. Jamshidi, C. Pahl, and N. C. Mendonçj, "Pattern-based multi-cloud architecture migration", *Software: Practice and Experience*, 2016.

[111] M. Jarzab and K. Zielinski, "Adaptable service oriented infrastructure provisioning with lightweight containers virtualization technology", *Computing & Informatics*, vol. 34, no. 6, 2015.

[112] Java Parallel Processing Framework, *JPPF*, http://www.jppf.org/.

[113] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges", *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, Jul. 2015.

[114] H. Jin, L. Deng, S. Wu, X. Shi, and X. Pan, "Live virtual machine migration with adaptive, memory compression", in *2009 IEEE International Conference on Cluster Computing and Workshops*, Aug. 2009, pp. 1–10.

[115] G. Jung, N. Gnanasambandam, and T. Mukherjee, "Synchronous parallel processing of big-data analytics services to optimize performance in federated clouds", in *Proceedings of the 5th International Conference on Cloud Computing*, ser. CLOUD'12, IEEE, Jun. 2012, 811–818.

[116] S. Kailasam, N. Gnanasambandam, J. Dharanipragada, and N. Sharma, "Optimizing ordered throughput using autonomic cloud bursting schedulers", *Transactions on Software Engineering*, vol. 39, no. 11, 1564–1581, Nov. 2013.

[117] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters", in *Proceedings of the 6th International Conference on Autonomic Computing*, ser. ICAC '09, ACM, 2009, pp. 117–126.

[118] D. Kapil, E. S. Pilli, and R. C. Joshi, "Live virtual machine migration techniques: Survey and research challenges", in *2013 3rd IEEE International Advance Computing Conference (IACC)*, Feb. 2013, pp. 963–969.

[119] C. Kaufman, *Internet Key Exchange (IKEv2) Protocol*, RFC 4306, Dec. 2005.

[120] R. Kawashima, "Vnfc: A virtual networking function container for sdn-enabled virtual networks", in *2012 Second Symposium on Network Cloud Computing and Applications*, Dec. 2012, pp. 124–129.

[121] S. Kent, *IP Encapsulating Security Payload (ESP)*, RFC 4303, Dec. 2005.

[122] S. Kent and K. Seo, *Security Architecture for the Internet Protocol*, RFC 4301, Dec. 2005.

[123] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley, 1997.

[124] V. Kherbache, Madelaine, and F. Hermenier, "Scheduling live-migrations for fast, adaptable and energy-efficient relocation operations", in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Dec. 2015, pp. 205–216.

[125] P. Kokkinos, T. A. Varvarigou, A. Kretsis, P. Soumplis, and E. A. Varvarigos, "Cost and utilization optimization of amazon ec2 instances", in *2013 IEEE Sixth International Conference on Cloud Computing*, Jun. 2013, pp. 518–525.

[126] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud", in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, 2012, pp. 784–791.

[127] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey", *Proceedings of the IEEE*, vol. 103, pp. 14–76, Jan. 2015.

[128] Kubertenes, *Production-grade container orchestration*, http://kubernetes.io.

[129] P. P. Kukade and G. Kale, "Auto-scaling of micro-services using containerization", *International Journal of Science and Research (IJSR)*, vol. 4, no. 9, 2015.

[130] J. Kupferman, J. Silverman, P. Jara, and J. Browne, *Scaling into the cloud*, CS270 - Advanced Operating Systems, http://www.cs.ucsb.edu/~jbrowne/files/ScalingIntoTheClouds.pdf, 2009.

[131] Y.-W. Kwon and E. Tilevich, "Cloud refactoring: Automated transitioning to cloud-based services", *Automated Software Engg.*, vol. 21, no. 3, pp. 345–372, Sep. 2014.

[132] D. Kyriazis, *Cloud computing service level agreements-exploitation of research results*, Technical report European Commission, http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=2496, 2013.

[133] P. Lama and X. Zhou, "Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee", in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2010, pp. 151–160.

[134] ——, "Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee", in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Aug. 2010, pp. 151–160.

[135] P. Lama and X. Zhou, "Efficient server provisioning with end-to-end delay guarantee on multi-tier clusters", in *2009 17th International Workshop on Quality of Service*, Jul. 2009, pp. 1–9.

[136] T. Laukkarinen, J. Suhonen, and M. Hännikaïnen, "A survey of wireless sensor network abstraction for application development", *International Journal of Distributed Sensor Networks*, 2012.

[137] T. Laukkarinen, J. Suhonen, and M. Hennikeinen, "An embedded cloud design for internet-of-things", *International Journal of Distributed Sensor Networks*, vol. 13, 2013.

[138] G. R. Lavender and D. C. Schmidt, "Active object: An object behavioral pattern for concurrent programming", 1995.

[139] P. Leitner, Z. Rostyslav, A. Gambi, and S. Dustdar, "A framework and middleware for application-level cloud bursting on top of infrastructure-as-a-service clouds", in *Proceedings of the 6th IEEE/ACM International Conference on Utility and Cloud Computing*, ser. UCC 2013, 2013.

[140] F. Li, M. Vogler, M. Claeßens, and S. Dustdar, "Efficient and scalable iot service delivery on cloud", in *Prodeedings of the 6th International Conference on Cloud Computing*, IEEE, 2013, 740–747.

[141] ——, "Towards automated iot application deployment by a cloud-based approach", in *Proceedings of the IEEE 6th International Conference on Service-Oriented Computing and Applications*, 2013, 61–68.

[142] S. Li, L. D. Xu, and S. Zhao, "The internet of things: A survey", *Information Systems Frontiers*, 1–17, 2014.

[143] Y. Li and M. Chen, "Software-defined network function virtualization: A survey", *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[144] Libreswan, *Libreswan vpn software*, https://libreswan.org/.

[145] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage", in *Proceedings of the 7th International Conference on Autonomic Computing*, ser. ICAC '10, ACM, 2010, pp. 1–10.

[146] LogicMonitor, *Why hosted monitoring*, http://www.logicmonitor.com/why-logicmonitor.

[147] V. Loia, G. D'Aniello, A. Gaeta, and F. Orciuoli, "Enforcing situation awareness with granular computing: A systematic overview and new perspectives", *Granular Computing*, vol. 1, no. 2, pp. 127–143, 2016.

[148] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments", *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, Dec. 2014.

[149] M. Luther, Y. Fukazawa, M. Wagner, and S. Kurakake, "Situational reasoning for task-oriented mobile service recommendation", *Knowledge Engineering Review*, vol. 23, no. 1, pp. 7–19, Mar. 2008.

[150] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Migrating running applications across mobile edge clouds: Poster", in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '16, ACM, 2016, pp. 435–436.

[151] A. Mashtizadeh, E. Celebi, T. Garfinkel, and M. Cai, "The design and evolution of live storage migration in vmware esx", in *Proceedings of the 2011 USENIX Annual Technical Conference*, ser. ATC'11, 2011, pp. 14–14.

[152] C. McGuire and R. Squillace, *Configure a point-to-site connection to a vnet using the azure portal*, https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-howto-point-to-site-resource-manager-portal.

[153] C. McGuire and S. Wheeler, *Configure a vnet-to-vnet connection (classic)*, https://docs.microsoft.com/en-us/azure/vpn-gateway/virtual-networks-configure-vnet-to-vnet-connection.

[154] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges", *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[155] Monitis, *All-in-one application monitoring platform*, http://portal.monitis.com/.

[156] R. D. L. Mora, *Cisco iox: An application enablement framework for IoT*, http://blogs.cisco.com/ioe/cisco-iox-an-application-enablement-framework-for-the-internet-of-things/, 2014.

[157] S. Murphy, A. Nafaa, and S. J., "Advanced service delivery to the connected car", in *Proceedings of the 9th International Conference on Wireless and Mobile Computing, Networking and Communications*, IEEE, 2013, 147–153.

[158] D. Murray, S. Wheeler, and R. Squillace, *Virtual network and subnets*, https://docs.microsoft.com/en-us/azure/virtual-machines/windows/network-overview#virtual-network-and-subnets.

[159] ——, *Virtual networks and windows virtual machines in azure*, https://docs.microsoft.com/en-us/azure/virtual-machines/windows/network-overview.

[160] Nagios, *The industry standard in it infrastructure monitoring*, http://www.nagios.com.

[161] netvirt, *An open network virtualization platform*, https://github.com/netvirt/netvirt.

[162] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "Agile: Elastic distributed resource scaling for infrastructure-as-a-service", in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, USENIX, 2013.

[163] C. Nottingham and R. Squillace, *Manage the availability of windows virtual machines in azure*, https://docs.microsoft.com/en-us/azure/virtual-machines/windows/manage-availability.

[164] Nutanix, *A quick explanation of what makes ahv virtual machine scheduling with ads so unique*, https://next.nutanix.com/t5/Nutanix-Connect-Blog/A-Quick-Explanation-of-What-Makes-AHV-Virtual-Machine-Scheduling/ba-p/24062.

[165] ——, *Your entreprise cloud company*, http://www.nutanix.com/.

[166] OASIS, *Topology and orchestration specification for cloud applications version 1.0*, OASIS Standard, http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf.

[167] OCCI, *Open cloud computing interface*, http://occi-wg.org/.

[168] C. Olah, *Understanding lstm networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/, Aug. 2015.

[169] Open Network Operating System, *A new carrier-grade sdn network operating system designed for high availability, performance, scale-out*, http://onosproject.org/.

[170] Open Networking Foundation, *SDN Architecture*, TR 502, https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf.

[171] Open Networking Lab, UC Stanford, *Pox wiki*, https://openflow.stanford.edu/display/ONL/POX+Wiki, 2015.

[172] OpenConnect, *Openconnect vpn client*, http://www.infradead.org/openconnect/.

[173] OpenDaylight, *The OpenDaylight Platform*, https://www.opendaylight.org/.

[174] OpenNebula, *Flexible enterprise cloud made simple*, http://opennebula.org/documentation:rel4.0.

[175] OpenShift, *Paas by RedHat, built on docker and kubernetes*, https://www.openshift.com/.

[176] OpenStack, *Open source software for creating private and public clouds*, http://www.openstack.org.

[177] ——, *Openstack networking ("neutron")*, https://wiki.openstack.org/wiki/Neutron.

[178] Openswan, *Welcome to openswan*, https://www.openswan.org/.

[179] OpenVPN, *Openvpn community software*, https://openvpn.net/index.php/open-source.html.

[180] OpenVZ, *Virtuozzo containers*, https://openvz.org/Main_Page.

[181] Opsview, *Documentation*, docs.opsview.com.

[182] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling", in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13, ACM, 2013, pp. 69–84.

[183] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources", in *Proceedings of the 4th ACM European Conference on Computer Systems*, ser. EuroSys '09, ACM, 2009, pp. 13–26.

[184] S. M. Park and M. Humphrey, "Self-tuning virtual machines for predictable escience", in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2009, pp. 356–363.

[185] PennState Eberly College of Science, *STAT 510 – seasonal ARIMA models*, https://onlinecourses.science.psu.edu/stat510/node/67.

[186] V. Persico, A. Botta, A. Montieri, and A. Pescapè, "A first look at public-cloud inter-datacenter network performance", in *2016 IEEE Global Communications Conference, GLOBECOM 2016, Washington, DC, USA, December 4-8, 2016*, 2016, pp. 1–7.

[187] V. Persico, A. Botta, P. Marchetta, A. Montieri, and A. Pescapè, "On the performance of the wide-area networks interconnecting public-cloud datacenters around the globe", *Computer Networks*, vol. 112, pp. 67–83, 2017.

[188] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch", in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, May 2015, pp. 117–130.

[189] Pica8, *Sdn system performance*, http://www.pica8.com/pica8-deep-dive/sdn-system-performance/.

[190] ——, *White box sdn*, http://www.pica8.com/.

[191] G. Prakash, M Thejaswini, S. Manjula, K. Venugopal, and L. Patnaik, "Energy efficient in-network data processing in sensor networks", *World Academy of Science, Engineering and Technology*, vol. 48, 2008.

[192] M. Psiuk, T. Bujok, and K. Zielinski, "Enterprise service bus monitoring framework for soa systems", *Transactions on Services Computing*, vol. 5, no. 3, 450–466, 2012.

[193] H. Qian and D. Andresen, "Jade: An efficient energy-aware computation offloading system with heterogeneous network interface bonding for ad-hoc networked mobile devices", in *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2014.

[194] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey", *CoRR*, vol. abs/1609.09224, 2016.

[195] F. Quesnel, A. Lèbre, and M. Südholt, "Cooperative and Reactive Scheduling in Large-Scale Virtualized Platforms with DVMS", *Concurrency and Computation: Practice and Experience*, Dec. 2012.

[196] M. R. Rahimi, J. Ren, C. H. Liu, A. V. Vasilakos, and N. Venkatasubramanian, "Mobile cloud computing: A survey, state of art and future directions", *Mobile Networks and Applications*, vol. 19, no. 2, pp. 133–143, 2014.

[197] R. Ranjan, B. Benatallah, S. Dustdar, and M. P. Papazoglou, "Cloud resource orchestration programming: Overview, issues, and directions", *IEEE Internet Computing*, vol. 19, no. 5, pp. 46–56, Sep. 2015.

[198] B. T. Rao and L. S. S. Reddy, "Survey on improved scheduling in hadoop mapreduce in cloud environments", *CoRR*, 2012.

[199] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin, "Vconf: A reinforcement learning approach to virtual machines auto-configuration", in *Proceedings of the 6th International Conference on Autonomic Computing*, ser. ICAC '09, ACM, 2009, pp. 137–146.

[200] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis", in *Proceedings of the Third Symposium on Cloud Computing*, ser. SoCC '12, ACM, 2012, 7:1–7:13.

[201] RightScale, *What is auto scaling?*, http://docs.rightscale.com/faq/What_is_auto-scaling.html.

[202] P. Riteau, C. Morin, and T. Priol, "Shrinker: Improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing", in *Euro-Par 2011 Parallel Processing: 17th International Conference, Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011, Proceedings, Part I*. Springer, 2011, pp. 431–442.

[203] F. Rossi, P. v. Beek, and T. Walsh, *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier, 2006.

[204] M. Rusek, G. Dwornicki, and A. Orłowski, "A decentralized system for load balancing of containerized microservices in the cloud", in *Advances in Systems Science: Proceedings of the International Conference on Systems Science 2016 (ICSS 2016)*. Springer, 2017, pp. 142–152.

[205] Ryu SDN Framework Community, *Build sdn agilely*, https://osrg.github.io/ryu/, 2017.

[206] A. Sehgal, V. Perelman, S. Kuryla, and J. Schonwalder, "Management of resource constrained devices in the internet of things", *Communications Magazine*, vol. 50, no. 12, 144–149, 2012.

[207] A. W. Services, *Greengrass*, https://aws.amazon.com/greengrass/.

[208] ——, *Kinesis*, https://aws.amazon.com/kinesis/.

[209] ——, *Lambda*, https://aws.amazon.com/lambda/.

[210] SHALB, *Server monitoring*, http://shalb.com/en/spae/spae_features/.

[211] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/openflow controllers", in *9th Central Eastern European Software Engineering Conference in Russia*, ACM, 2013.

[212] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das, "Cloudpd: Problem determination and diagnosis in shared dynamic clouds", in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2013, pp. 1–12.

[213] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and virtual machines at scale: A comparative study", in *Proceedings of the 17th International Middleware Conference*, ser. Middleware '16, ACM, 2016.

[214] Q. Z. Sheng and B. Benatallah, "Contextuml: A uml-based modeling language for model-driven development of context-aware web services development", in *Proceedings of the International Conference on Mobile Business*, ser. ICMB '05, IEEE, 2005, pp. 206–212.

[215] Q. Z. Sheng, U. Nambiar, A. P. Sheth, B. Srivastava, Z. Maamr, and S. Elnaffar, "International workshop on context enabled source and service selection, integration and adaptation: Workshop summary", in *Proceedings of the 2008 International Workshop on Context Enabled Source and Service Selection, Integration and Adaptation: Organized with the 17th International World Wide Web Conference (WWW 2008)*, ser. CSSSIA '08, ACM, 2008.

[216] Q. Sheng, J. Yu, and S. Dustdar, *Enabling Context-Aware Web Services: Methods, Architectures, and Technologies*. CRC Press, 2010, ISBN: 9781439809860.

[217] Siddhi, *Complex event processing engine*, https://github.com/wso2/siddhi.

[218] A. Smola, *Introduction to Machine Learning*. Cambridge University Press, 2008.

[219] SoftEther, *Softether vpn open source*, http://www.softether.org/.

[220] J. Son, Y. Xiong, K. Tan, P. Wang, Z. Gan, and S. Moon, "Protego: Cloud-scale multitenant ipsec gateway", in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA: USENIX Association, 2017, pp. 473–485, ISBN: 978-1-931971-38-6. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/son.

[221] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane", in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13, 2013, pp. 127–132.

[222] B. Spinnewyn, S. Latre, and B. Braem, "Fault-tolerant application-placement in heterogeneous cloud-environments", in *Proceedings of the 12th International Conference on Network and Service Management*, IEEE, 2015.

[223] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable ethernet for data centers", in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, ACM, 2012, pp. 49–60.

[224] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines", *SIGPLAN Notices*, vol. 46, no. 7, pp. 111–120, Mar. 2011.

[225] C. Technologies, *CA unified infrastructure management*, http://www.nimsoft.com/solutions/nimsoft-monitor/cloud.

[226] Technopedia, *Definition: Azure fabric controller*, http://www.techopedia.com/definition/26433/azure-fabric-controller.

[227] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani, "A hybrid reinforcement learning approach to autonomic resource allocation", in *Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, ser. ICAC '06, IEEE, 2006, pp. 65–73.

[228] The BtrPlace project, http://www.btrplace.org.

[229] The Onyx Platform, *Distributed computation for the cloud*, http://www.onyxplatform.org/.

[230] The Open Networking Foundation, *OpenFlow Switch Specification*, https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf, Jun. 2012.

[231] Tintri, *All-flash arrays for virtualization and cloud*, https://www.tintri.com/.

[232] A. N. Toosi, R. N. Calheiros, R. K. Thulasiram, and R. Buyya, "Resource provisioning policies to increase iaas provider's profit in a federated cloud environment", in *Proceedings of the 2011 International Conference on High Performance Computing and Communications*, ser. HPCC '11, IEEE, 2011, 279–287.

[233] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers", 2, vol. 28, 2012, 358–367.

[234] H.-L. Truong, A. Manzoor, and S. Dustdar, "On modeling, collecting and utilizing context information for disaster responses in pervasive environments", in *Proceedings of the First International Workshop on Context-aware Software Technology and Applications*, ser. CASTA '09, ACM, 2009, pp. 25–28.

[235] M. Turowski and A. Lenk, "Vertical scaling capability of openstack", in *Service-Oriented Computing - ICSOC 2014 Workshops: WESOA; SeMaPS, RMSOC, KASA, ISC, FOR-MOVES, CCSA and Satellite Events, Paris, France, November 3-6, 2014, Revised Selected Papers*. Springer, 2015, pp. 351–362.

[236] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications", *Transactions on Autonomous and Adaptive Systems*, vol. 3, no. 1, 1:1–1:39, Mar. 2008.

[237] M. Vallaey, *The top 5 cloud-computing vendors: #1 microsoft, #2 amazon, #3 ibm, #4 salesforce, #5 sap*, http://info.bigindustries.be/analogue-cloud/aws-clear-leader-in-gartner-magic-quadrant-1, Jun. 2017.

[238] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud", *SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 45–52, Jan. 2011.

[239] M. Vasconcelos, N. C. Mendonça, and P. H. M. Maia, "Cloud detours: A non-intrusive approach for automatic software adaptation to the cloud", in *Service Oriented and Cloud Computing: 4th European Conference, ESOCC 2015, Taormina, Italy, September 15-17, 2015, Proceedings*. Springer, 2015, pp. 181–195.

[240] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg", in *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[241] A. Verma, P. Ahuja, and A. Neogi, "Pmapper: Power and migration cost aware application placement in virtualized systems", in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, ser. Middleware '08, Springer, 2008, pp. 243–264.

[242] VMWare, *Hybrid dmz reference designs for vcloud air*, http://www.vmware.com/cloud-services/infrastructure/vcloud-air-hybrid-dmz.html.

[243] ——, *Infrastructure as a service with vcloud air*, http://www.vmware.com/cloud-services/infrastructure.html.

[244] ——, *Vcloud air advanced networking services*, http://www.vmware.com/cloud-services/infrastructure/vcloud-air-advanced-networking-services.html.

[245] ——, *Vcloud air direct connect*, http://www.vmware.com/cloud-services/networking/direct-connect.html.

[246] L. Wang, J. Xu, and M. Zhao, "Application-aware cross-layer virtual machine resource management", in *Proceedings of the 9th International Conference on Autonomic Computing*, ser. ICAC '12, ACM, 2012, pp. 13–22.

[247] Wikipedia, *Operating-system-level virtualization*, https://en.wikipedia.org/wiki/Operating-system-level_virtualization.

[248] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe, "Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines", *SIGPLAN Notices*, vol. 46, no. 7, pp. 121–132, Mar. 2011.

[249] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "On the use of fuzzy modeling in virtualized data center management", in *Proceedings of the Fourth International Conference on Autonomic Computing*, ser. ICAC '07, IEEE, 2007.

[250] L. D. Xu, W. He, and S. Li, "Internet of things in industries: A survey", *Transactions on Industrial Informatics*, vol. 10, no. 4, 2233–2243, 2014.

[251] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, and J. Chen, "A cost-aware auto-scaling approach using the workload prediction in service clouds", *Information Systems Frontiers*, vol. 16, no. 1, pp. 7–18, 2014.

[252] L. Youseff, M. Butrico, and D. D. Silva, "Towards a unified ontology of cloud computing", in *Proceedings of the IEEE Grid Computing Environments Workshop (GCE08)*, 2008.

[253] C. Yu and F. Huan, "Live migration of docker containers through logging and replay", 2015.

[254] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow scheduling algorithms for grid computing", in *Metaheuristics for scheduling in distributed computing environments*, 2008.

[255] H. Yuan, S. W. Choi, and S. D. Kim, "A practical monitoring framework for esb-based services", in *Congress on Services, Part II*, IEEE, 2008, 49–56.

[256] Q. Zhang, L. Cherkasova, and E. Smirni, "A regression-based analytic model for dynamic resource provisioning of multi-tier applications", in *Proceedings of the Fourth International Conference on Autonomic Computing*, ser. ICAC '07, IEEE, 2007.

[257] Q. Zhang, Q. Zhu, M. Zhani, and R. Boutaba, "Dynamic service placement in geographically distributed clouds", ser. ICDCS 2012, IEEE, Jun. 2012, 526–535.

[258] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "A context sensitive offloading scheme for mobile cloud computing service", in *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*, ser. CLOUD '15, IEEE, 2015, pp. 869–876.

[259] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments", in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10, ACM, 2010, pp. 304–307.