Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements

State of the Art Analysis

Deliverable 1.1

Deliverable:	D1.1
Title:	State of the Art Analysis
Editor(s):	Giuliano Casale (IMP), Tatiana Ustinova (IMP)
Contributor(s):	Danilo Ardagna (PMI), Matej Artac (XLAB), Simona Bernardi (ZAR), Marcello Bersani (PMI), Giuliano Casale (IMP), Pablo Domínguez-Mayo (ZAR), Ioan Dragan (IEAT), Mădălina Eraşcu (IEAT), Marc Gil (PRO), Gabriel Iuhasz (IEAT), Pooyan Jamshidi (IMP), Cristophe Joubert (PRO), José Merseguer (ZAR), Daniel Pop (IEAT), Alberto Romeu (PRO), Matteo Rossi (PMI), Tatiana Ustinova (IMP), Darren Whigham (FLEXI).
Reviewers:	Youssef Ridene (NETF), Ilias Spais (ATC)
Type (R/P/DEC):	Report
Version:	1.0
Date:	31-July-2015
Status:	Final version
Dissemination level:	Public
Download page:	http://www.dice-h2020.eu/deliverables/
Copyright:	Copyright © 2015, DICE consortium – All rights reserved

DICE partners

ATC:	Athens Technology Centre
FLEXI:	Flexiant Limited
IEAT:	Institutul E Austria Timisoara
IMP:	Imperial College of Science, Technology & Medicine
NETF:	Netfective Technology SA
PMI:	Politecnico di Milano
PRO:	Prodevelop SL
XLAB:	XLAB razvoj programske opreme in svetovanje d.o.o.
ZAR:	Universidad De Zaragoza

The DICE project (February 2015-January 2018) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644869

Executive summary

The aim of this deliverable is to review the state-of-the-art in techniques used in the developing of Big Data applications and related technology offerings that are available on the market. After positioning DICE in the scope of DevOps and Big Data, the deliverable provides background on Big Data and related software engineering trends, such as the emergence of the Lamba architecture style.

We then overviewthe state of the art on designing functional and non-functional properties in enterprise software systems, highlighting gaps towards achieving these goals for data-intensive applications. In particular, we survey model-driven engineering (MDE) methods, which are the most popular to combine software design with quality analysis techniques based on formal models for performance, reliability and verification. We extensively discuss existing UML profiles relevant to software quality assessment and highlight their gaps in relation to modelling data intensive applications. Editors and modelling tools that can process such UML profiles are also surveyed and compared.

We then overview the problem of deploying, monitoring and testing an enterprise cloud application, and review existing technologies and open source tools in this area. It is found that some areas, such as non-functional testing, are fairly under-developed in Big Data and thus offer an opportunity for innovation.

In the last part of the deliverable we summarize some relevant Big Data technologies (e.g., Hadoop/MapReduce, Spark, Storm, etc.) and related commercial and open source implementations. For each technology, we highlight quality metrics that may be considered by the DICE monitoring, prediction and analysis tools.

Table of contents

List of figures

List of tables

Introduction

Big Data has recently emerged as a major trend in the ICT industry. However, the heterogeneity of the technologies and software development method in use is still a challenge for researchers and practitioners. The DICE project aims at developing a novel UML profile and tools that will help software designers reasoning about reliability, safety and efficiency of Big Data applications. Since the DICE methodology will cover quality assessment, architecture enhancement, continuous testing and agile delivery, there is a large span of technologies and tools that may act as baselines to this research effort. The goal of this deliverable is therefore to support this investigation with an analysis on the scientific and technical state of the art in the area of software engineering for Big Data. It is important to note that the scope of the DICE project embraces the phases of initial design, development and testing, but not operation of production systems, therefore this survey does not cover service management. That is, the aim of the project is to put the developer in conditions to deliver an application to market in a short period of time, focusing on development and pre-production testing.

The document is organised in four chapters which cover the following topics:

- Chapter A provides basic concepts, introducing definitions and key architectural styles that are central to Big Data applications.
- Chapter B describes the state of the art in functional development of Big Data applications, covering in particular aspects related to Model-Driven Engineering (MDE) and to the DevOps paradigm, which aims at applying agile concepts across the whole service delivery chain.
- Chapter C continues the survey by reviewing current quality engineering methods used in model-driven engineering and DevOps. In particular, we describe baselines such as UML MARTE and UML DAM that are central to the DICE UML profile.
- While the previous are oriented at reviewing research efforts and relevant standards, Chapter D covers technological aspects of Big Data, surveying relevant technologies in real-time and batch data processing.

Out of the present investigation, our main achievement is the identification of the most relevant baselines to the DICE work and of open problems in the area touched by the project. The reader is invited to consult deliverable D1.2 – ‘Requirement specification’ for a detailed requirement analysis that used the output of this survey to define the DICE goals.

A. Data-Intensive Applications

A.1. Definitions and terminology.

DICE focuses on design, analysis and testing of *Data-Intensive Applications* (DIAs). Our notion of DIA is any application that is able to handle *Big Data*, performing computations at a speed that is in line with the *volume*, *variety* and *velocity* of such data. The National Institute of Standards and Technology (NIST) [1] defines Big Data as follows:

> *'**Big Data** consists of extensive datasets primarily in the characteristics of **volume, variety, velocity**, and/or **variability** that require a **scalable** architecture for efficient storage, manipulation, and analysis'*

This definition concerns, on the one hand, the characteristic to be owned by datasets, which are:

1. **Volume**, meaning that Big Data datasets are larger than those commonly handled by conventional databases.
2. **Variety**, data can possibly come from different sources and in diverse forms.
3. **Velocity**, referring to the fact that huge amounts of data have to be analysed in a short time.
4. **Variability**, which refers to changes in other characteristics, as meaning, veracity, etc.

On the other hand, the reported definition contains also a reference to the qualities of the architectures needed to cope with Big Data. Such architecture must be carefully designed to be **scalable** and **efficient**, essentially referring to a high degree of parallelism both in terms of storage and processing.

Historically, ever-growing datasets were analysed by increasingly powerful monolithic systems (**vertical scaling**). This strategy recently clashed against physical limitations and required to look for a different approach. Industry and academia started relying on **horizontal scaling** techniques for parallel distribution of both storage (Petabytes and even Exabytes) and processing capacity (hundreds or thousands of nodes). The necessity of new models and tools suitable to seamlessly provide this **parallelisation** brought to the emergence of sophisticated, reliable and efficient frameworks. Such frameworks, which include, for example, Hadoop, Spark and Storm (see Chapter D), are now commonly referred to as Big Data platforms. The goal of DICE is to realise a model-driven approach that can support the development of applications harnessing the capabilities of these platforms.

A.2. Importance of Big Data for the Business

While business considerations are somewhat orthogonal to the investigation in this deliverable, we include a section to contextualise the importance of DIAs in today's European economy. The European commission has stated on several occasions that 'Big Data is the new oil'. Indeed, data are not only part of almost all economic and social activities of our society, but have managed to be viewed as an essential resource for all sectors, organisations, countries and regions. But why this is a reality? It is expected that by 2020 there will be more than 16 zettabytes of useful data (16 Trillion GB) [3]. Data are not only part of almost all economic and social activities of our society like velocity, variety and socioeconomic value, flags a paradigm shift towards a data-driven socioeconomic mode which suggests a growth of 236% per year from 2013 to 2020. Thus, data blast is indeed a reality that Europe must both face and endeavour in an organised, forceful, user-centric and goal-oriented approach. It is obvious that data exploitation can be the leading spear of innovation, drive cutting-edge technologies, increase competitiveness and create social and financial impact.

Table 1 (source: Big Data Value Association [4]) provides some examples of how Big Data will impact different sectors:

Table 1: Examples of Big Data impact across various sectors.

Sectors/Domains	Big Data Value	Source
Public administration	EUR 150 billion to EUR 300 billion in new value (Considering EU 23 larger governments)	OECD [5], 2013
Healthcare & Social Care	EUR 90 billion considering only the reduction of national healthcare expenditure in the EU	McKinsey Global Institute [6], 2011
Utilities	Reduce CO2 emissions by more than 2 gigatonnes, equivalent to EUR 79 billion (Global figure)	OECD [5], 2013
Transport and logistics	USD 500 billion in value worldwide in the form of time and fuel savings, or 380 megatonnes of CO2 emissions saved	OECD [7], 2013
Retail & Trade	60% potential increase in retailers' operating margins possible with Big Data	McKinsey Global Institute [6], 2011
Geospatial	USD 800 billion in revenue to service providers and value to consumer and business end users	McKinsey Global Institute [6], 2011
Applications & Services	USD 51 billion worldwide directly associated to Big Data market (Services and applications)	Various [8]

Based on the previous examples, data penetration in several sectors of our society is indeed a reality. However, can this influence the business section? Is there a market that shows how important Big Data are for businesses? According to several studies (e.g. International Data Corporation (IDC) [9]) the Big Data market is growing six times faster than the overall ICT market. In addition to, the compound annual growth rate (CAGR) of the Big Data market over the period 2013 – 2017 will be around 27%, reaching an overall total of $50 billion. Furthermore, as identified by demosEUROPA [10], 'Overall, by 2020, big & open data can improve the European GDP by 1.9%, an equivalent of one full year of economic growth in the EU'. At the same time another study concludes that the increased adoption of Big Data will have positive impact on employment, and is expected to result in 3.75 million jobs in the EU by 2017 [11]. Thus, Big Data market has the power to influence the business section. But in which way? Is this influence positive?

According to McKinsey institute, 'The effective use of Big Data offers the benefits to transform economies, and delivering a new wave of productive growth'. IDC confirms that Big Data adoption in Europe is accelerating [12] and subsequently large companies and SMEs has recognised Big Data market value proposition to revise existing socio-economic and business models according to its offerings. Even though (according to IDC [13]) 30% of Western European companies will adopt Big Data by the end of 2015, there is a percentage of 70% of business actors that are eager to obtain new tools and assets to propel them into the data-driven economy.

A.3. Roles in Big Data Systems

In [2], NIST identifies five main roles in Big Data systems:

- ***System Orchestrator***: provides the definition of the system requirements to be fulfilled; examples of requirements are data, architecture, resources, and business requirements. The system

orchestrator defines monitoring or auditing activities as well to verify the fulfilment of the requirements.

- ***Data Provider***: this role makes available sources of data and information to be fed into the Big Data system. Data provider's activities range from Data Collection (e.g. from sensors) and *Persistence* (according to several hosting models, Cloud inclusive) to Access *Rights Management* and *Data Availability Publication* (via dictionaries and catalogues).
- ***Big Data Application Provider***: realises the data life-cycle performing all the necessary operations and transformations on the input data to fulfil the requirements defined by the System Orchestrator. Security and Privacy are primary concerns for the Big Data Application Provider as well.
- ***Big Data Framework Provider***: provides resources and services to the Big Data Application Provider for the creation of Data-Intensive applications. In particular, this role has to supply the infrastructure, the data management layer and the processing framework.
- ***Data Consumer***: this role describes the final user of a Big Data system. They receive and benefit from the data manipulated by the application. Their activities concern research, retrieval and download of the data as well as local analysis, visualisation and reporting.

A.4. Technology Overview

Two important technological ingredients of a DIA which make them different from canonical enterprise applications are their emphasis on data storage and data computation mechanisms. In the following subsections we provide a very short overview of the storage and computation mechanisms available for DIAs, which will also be described in more details in the next chapters.

A.4.1 Distributed computations

In general, the expression '**distributed computing'** refers to a software system where various components, located on different machines, collaborate to achieve a common goal. Over the years many frameworks for distributed computing have been presented; however, they had the disadvantage of being too focused on distributed execution and little on parallel data management capabilities. With the advent of Big Data, it was necessary to develop a new paradigm able to interact effectively with the distributed and variable data. MapReduce is the first and most famous expression of this paradigm. Its open source implementation, Hadoop [14], includes functions for reliability, security, workflow integration and governance. Nonetheless, other solutions have been developed aiming at overcoming the original MapReduce shortcomings. These solutions seek to respond to different needs and often they have to coexist in the same environment. For this reason, **resource management systems** have been recently developed for such solutions. These systems are designed to handle the available computing resources assigning them according to specific policies. In this way, multi-tenant, multi processing models (other than MapReduce) and high availability environments can be created. The most known resource management systems are **YARN** (see Section D.2) and **Mesos** [15].

A.4.2 Distributed Storage and NoSQL solutions

In the classical database terminology, the basic unit of information is called **Data Record** and corresponds to a single observation of a particular event associated with the system to be analysed. Data records in classical database theory represent a piece of structured information and can be thought of as a row of a relational database table. In the Big Data context information are not necessarily structured. They can also be **unstructured** (images, videos) or **semi-structured** (text, documents). Thus, a different data model for storing such data became necessary. **Non-relational databases**, also known as **NoSQL** (Not only SQL, see Section D.4) have been developed for addressing such new needs offering, at the same time, a large degree of scalability in terms of horizontal as well as vertical scaling.

The use of NoSQL solutions in the context of Big Data is preponderant and, in many respects, it made Big Data possible; nonetheless, it is necessary to warn the reader from falling into the easy association 'NoSQL = Big Data' because recent years have witnessed the appearance of highly scalable storage systems based on the relational model.

Moreover, due to the size of datasets and the need for parallelism in the computation, high performing **file systems** such as Hadoop Distributed File System (**HDFS, see section D.2)** have been developed. The main characteristics of these file systems are to be 1) **distributed**, i.e. files and dataset are seamlessly distributed over several nodes. In this way different nodes can access and analyse data at the same time on the same dataset; 2) **multi-structured**, meaning that a variety of data types are supported. Usually, the information is stored at block level; 3) **replicated**, files and blocks are replicated several times for reliability/recovery and to enhance data locality.

A.4.3 Streams and Big Data in motion

In the previous paragraphs, we tried to outline Big Data problems and approaches leaving aside the much of the discussion on the **latency** in data management. We introduced distributed storage solutions and the reader might presume that the only way to deal with Big Data is to collect them into huge datasets (**data at rest**) and analyse them 'a posteriori', periodically or as a response to an event. This is not always the case, though. Data can enter the system from different sources and examined in (almost) real-time (**data in motion**). In this scenario, the data flows are referred to as data streams and they are mainly characterised by velocity and variability. Many solutions for data streaming were devised used long before the concept of Big Data came up. Nevertheless, the parallelisation, resource management and reliability supplied by modern Big Data framework imposed a reconsideration and redefinition of many basic elements in data streaming. As evidence of this, in the last couple of years we have witnessed the successful application of concepts typically used for *data at rest* (Maps, Reduces, direct acyclic graphs (DAGs), Batches) to *data in motion* scenarios. Spark is an example of framework aiming at unifying both scenarios under the same programming paradigm (see Chapter D).

A.5. Architectural Styles for DIAs

The particular focus of DIA on Big Data makes them fairly different from traditional enterprise application. Therefore, the research and technical community have investigated in recent years novel architectural styles to support DIAs. In particular, the data velocity of Big Data applications is application-specific and may range from a few to millions of data items per second. Different Big Data applications enforce different quality of service constraints regarding response time. For example, a reactive use case with high-volume data streams may require an answer in a real-time (milliseconds) fashion.

A.5.1 Lambda Architecture

The Lambda Architecture introduced by Marz [16] is an advanced architectural style to overcome the challenges in general for Big Data applications and more specifically on real-time stream processing. The architectural style decomposes the problem into three key layers: (i) the **batch layer** focuses on fault tolerance and optimises for precise results (ii) the **speed layer** is optimised for real-time response-times and only consider the most recent data and (iii) the **serving layer** provides low latency views to the results of the batch layer. Note that some other layers will be introduced later for coordination and orchestration purposes.

Lambda Architecture enables real-time responses to query over Petabytes of data. Such a query on traditional architectures imposes unreasonably high latency. The Lambda Architecture divides this problem into three layers. The batch layer pre-computes the query function based on the full data set and updates the serving layer. This operation involves high latency and by the time the view of the pre-computed query is

finished it is already outdated. The speed layer only operates on the most recent data in order to provide views for the missing time span of the batch layer. The principal structure of an application based on Lambda Architecture is shown in Figure 1.

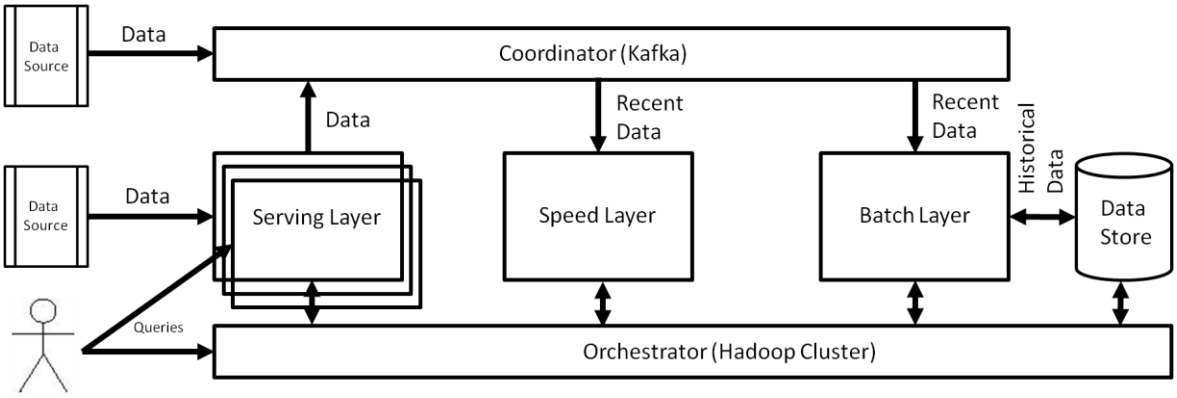

Figure 1. Lambda Architecture.

In more details, the Lambda Architecture [17] may be divided into five components: (i) *orchestration* (ii) *coordination* (iii) *batch layer* (iv) *speed layer* and (v) *serving layer*. Figure 1 illustrates the components of the Lambda Architecture and the data flow through the system. The orchestration component manages resource allocation, provides scheduling functionality to the other key layers. The coordination component facilitates the entry point for data and offers high-level services to synchronise processes within different components. The batch layer facilitates a micro-batch processing system with high reliability and fault tolerance. The speed layer embeds a real-time stream processing system with reasonable response times. The service layer collects results and performance data from both the batch and the speed layer in order to measure the key performance indicators of this architecture.

The *batch layer* maintains the master data set that includes all data to pre-compute the necessary results. The data storage system has to provide the following requirements: (i) efficient writes for new data (ii) scalability to cope with the increasing need to store more data and (iii) support for parallel processing and the ability to partition the data. Since the data set is continuously growing the latency to pre-compute the batch views on the whole data set becomes increasingly expensive and the time span to catch up with the speed layer will increase. The batch layer, therefore, processes new incoming data. The *serving layer* provides fast access to the pre-computed results of the batch layer. These results are outdated because of the high latency of the batch layer. Therefore the write speed to the serving layer is less important than the read performance. The serving layer has to provide low latency random reads in order to answer queries efficiently. The *speed layer* is responsible for providing results to the most recent data and has to fulfil certain latency constraints based on the application. This implies that it is not technically possible to compute results based on the full master data set. Instead incremental computation is applied to the most recent data in combination with persistent state. The speed layer is more complicated than the batch layer, but any error is eventually compensated by the batch layer.

A.5.1.1. Coordination Layer

The coordination layer is the entry point of data into the system. A typical scenario involves reading from an input source and delivering the messages to either the batch or speed layer. Potential *design patterns* for realising coordination layer:

- Persistent Messaging
- In-Memory Messaging

A.5.1.2. Batch Layer

The objective of the batch layer is to compute precise results that eventually replace the possible inaccurate results of the speed layer. In case of node failures the processing may be delayed in order to guarantee this promise. There are two possible strategies to implement the batch layer: (i) a re-computation algorithm that periodically computes the results over the full master set or (ii) an incremental algorithm that processes new data when it is introduced into the system. Potential *design patterns* for realising batch layer:

- Micro-batch processing
- Replay Mechanism
- Precise recovery

A.5.1.3. Speed Layer

The main objective of the speed layer is to deliver results in real-time and therefore optimise the latency and throughput to process incoming data. In comparison to the batch layer, the speed layer does not guarantee the same high availability and fault tolerance as the batch layer. In case of node or processing failures the speed layer drops messages to rapidly process the new incoming data instead of replaying all messages to the last processed state. This design leads to inaccurate results in case of node failure, but these results are eventually replaced by the results of the batch layer.

A.5.1.4. Orchestration

The Lambda Architecture embeds highly distributed services on each layer with the disadvantage of difficult synchronisation and resource management challenges. In order to mitigate these challenges an orchestration layer is introduced. In a typical solution, this layer leverages the Apache Hadoop YARN framework [18], built-in Storm service [19] for resource allocation and Apache ZooKeeper [20] and Kafka [21] for coordination.

A.5.1.5. Serving Layer

The service layer provides a set of tools to support near real-time process monitoring, localised performance measurements and result collection. This layer can access the results of the batch layer computations with low latency. The serving layer would index views of the batch layer and provide the necessary interfaces to access the pre-computed data with low latency queries.

A.5.1.6. Academic and industrial positioning

Fan and Bifet [22] suggest the Lambda Architecture as one solution to the future challenges of Big Data with regard to data mining. Robak et al. [23] explored the application of Big Data and linked data concepts in supply chain management and listed the Lambda Architecture as one possible solution to deal with high volume of data in this field. They argue that such architecture could provide lower latency to react in critical situations. Bär [24] presents a possible implementation of the Lambda Architecture with no merging based on open source software components. He evaluates the capabilities of this architecture on the SRBench Benchmark [25] and DEBS Grand Challenge 2014 [26] task with varying data frequency rates on an unreliable infrastructure.

Google develops a new data processing service provided on top of their cloud platform called Google Cloud Dataflow [27]. It is similar to the Lambda Architecture by allowing clients to process data with stream and batch processing. Google framework and Spark (see Section D.2.3) rely on parallel collections of any size that are distributed across multiple machines in order to provide scalability.

Amazon maintains a set of loosely coupled, but well integrated tools that provide the means to realise Lambda Architecture. Kinesis [28] is a fully managed real-time stream processing service that uses a messaging concept similar to Kafka [21]. Kinesis integrates Amazon S3 (see Section D.6.2) and Elastic

MapReduce [29]. Amazon provides the necessary tools to build a batch and speed layer, but the integration of both layers and its challenges are not offered as a holistic solution yet. Lambdoop [30] is an industry implementation framework that facilitates application development based on Lambda Architecture.

A.5.1.7. Oryx: an example Lambda Architecture

Throughout the DICE project, we will use Oryx [31] as a reference technology to illustrate the applicability of some of our technical results to Big Data applications.

Oryx is an open source framework for building Big Data applications, but also includes ready to be deployed out-of-the-box example application, which can be used as it is or as a basis for developing a custom application. The intended application of Oryx framework is predictive analytics in real time based on the construction of models from the incoming streaming data. The models used in analysis are built using Machine Learning (ML) algorithms. The target areas for the application - among others - are business, health, education and weather forecasts. The design of Oryx is based on the Lambda Architecture.

The first version of Oryx (Oryx 1) [32] was released in 2013. Its architecture consisted of two layers: Computation Layer - where models based on incoming streaming data were built and evaluated based on the requests from a client, and Serving Layer - a medium for accepting requests from a client, transferring them to the Computation Layer for evaluation and returning result to the client. The Computation Layer runs both Batch process, which is an 'offline' process, meaning that it does not operate in real time, rather - several times a day, and model update and evaluation process, which is quick (process time measured in seconds). All parameters are controlled via configuration files. The high-level architecture of Oryx 1 is presented in the Figure 2.

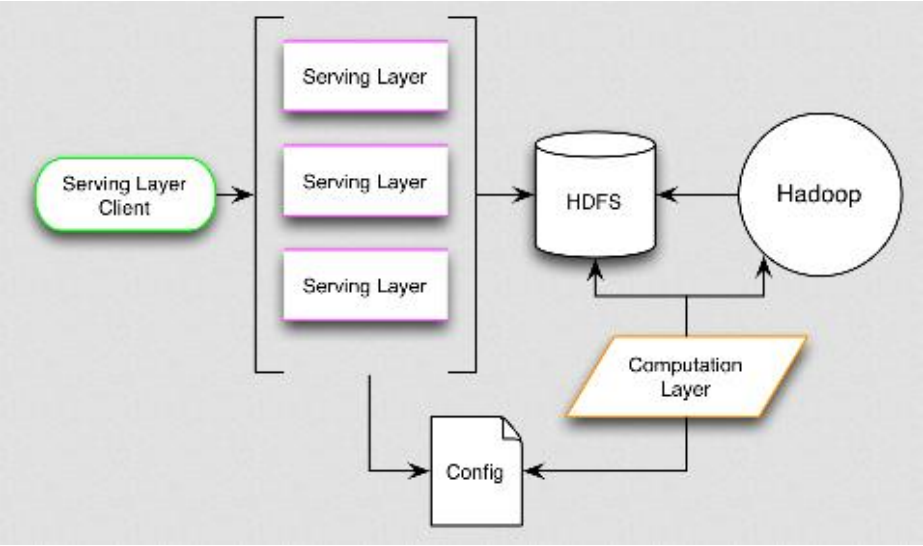

Figure 2. Architecture of Oryx 1 [32].

The high-level architecture of Oryx 2 is presented in the Figure 3.

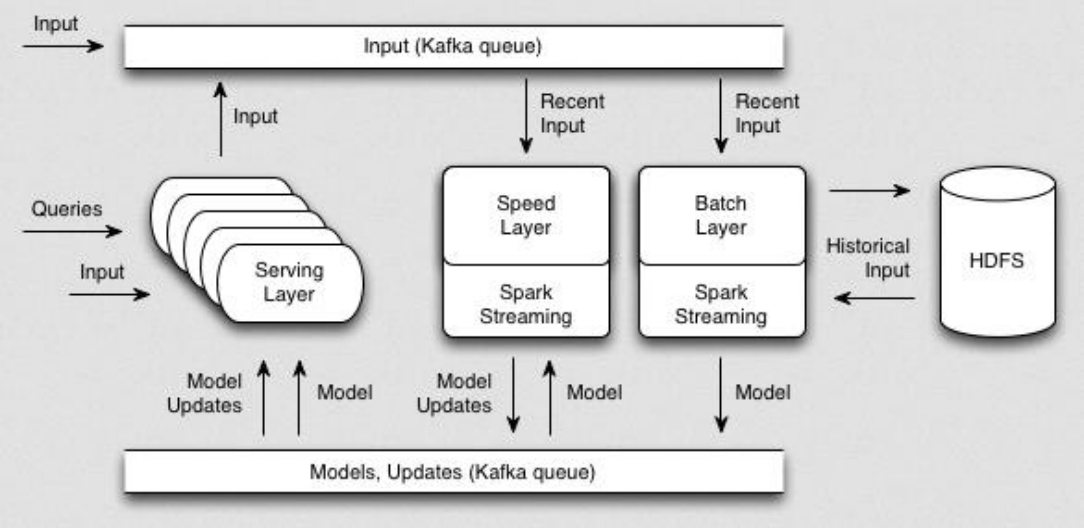

Figure 3. Architecture of Oryx 2 [31].

Both Oryx 1 and Oryx 2 employ machine learning algorithms to construct prediction models. Brief description of these algorithms is given in the Table 2.

Table 2: Machine learning algorithms used in Oryx 1 and Oryx 2.

Collaborative filtering	A technique used by recommender systems (engines) to suggest various items such as movies, music, news, books, research articles, search queries and so on to the client. Collaborative filtering implemented in Oryx uses matrix factorisation-based approach based on a variant of Alternating Least Squares (ALS) [34].
Classification and Regression	Classification can be used, for example, to separate e-mails into 'spam' and 'non-spam'. Regression is used to predict a specific numeric value (e.g. temperature on a given day or salary in certain year). Oryx employs random decision forests algorithms to solve classification and regression problems. Classification and regression belong to the supervised learning category of ML algorithms, which means that they require some initial data sets to be 'trained on'.
Clustering	Clustering is similar to classification in the sense that an object is assigned to a specific category (group, class), but clustering is an unsupervised learning method. It does not require an initial set to create classes to which subsequent incoming data is then compared, but rather tries to create groups (classes) from incoming data by looking for some common features in it. Oryx implements scalable k-means++ [35] for clustering.

A.5.2 Other Architectures

Although the Lambda Architecture has gained consensus in recent years, it has some limitations. First, maintaining code that needs to produce the same result in two complex distributed systems is expensive. Programming in distributed frameworks like Storm [19] and Hadoop [14] is complex. One proposed approach to fixing this is to have a language or framework that abstracts over both the real-time and batch framework. The developer writes code using this higher level framework and then it is translated into low-level stream processing and/or MapReduce code. However, this entails the operational burden of running and debugging two systems which is going to be very high.

Therefore, alternative architectures have been proposed. For example, the Kappa Architecture has been proposed by LinkedIn [36] exploiting stream processing, that is:

1. Use messaging system like Kafka [21] that will let you retain the full history of the data you want to be able to reprocess.
2. When one wants to do the reprocessing, start a second instance of the job that starts processing from the beginning of the data, but put this output data to a new table.
3. When the second job has been finished, switch the query to read from the new table.
4. Stop the old version of the job, and delete the old output table.

This new architectural style of the Kappa Architecture is depicted in Figure 4.

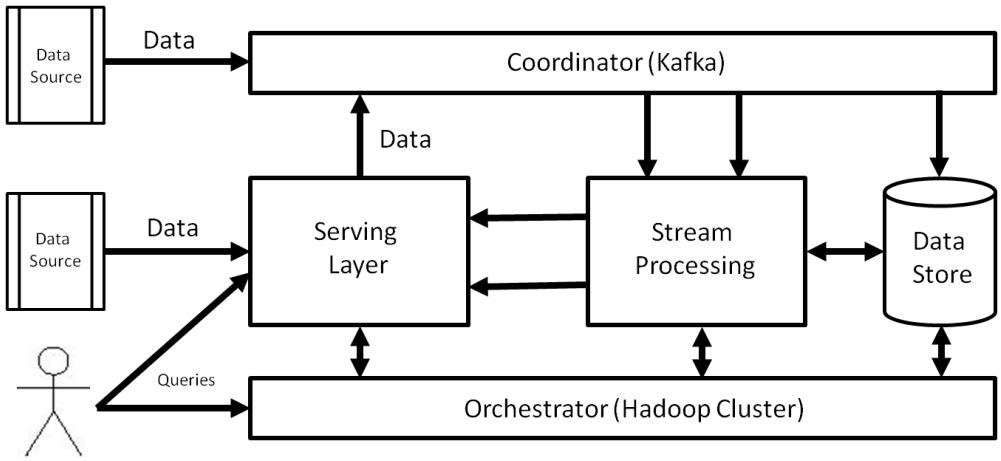

Figure 4. Kappa Architecture.

Another recent proposal is the Liquid Architecture [37], which has two layers: a messaging layer based on Apache Kafka [21], and a processing layer based on Apache Samza. The processing layer (i) executes jobs for different back-end systems according to a stateful stream processing model; (ii) guarantees service levels through resource isolation; (iii) provides low latency results; and (iv) enables incremental data processing. A messaging layer ports the processing layer. Figure 5 shows architectural style of the Liquid Architecture.

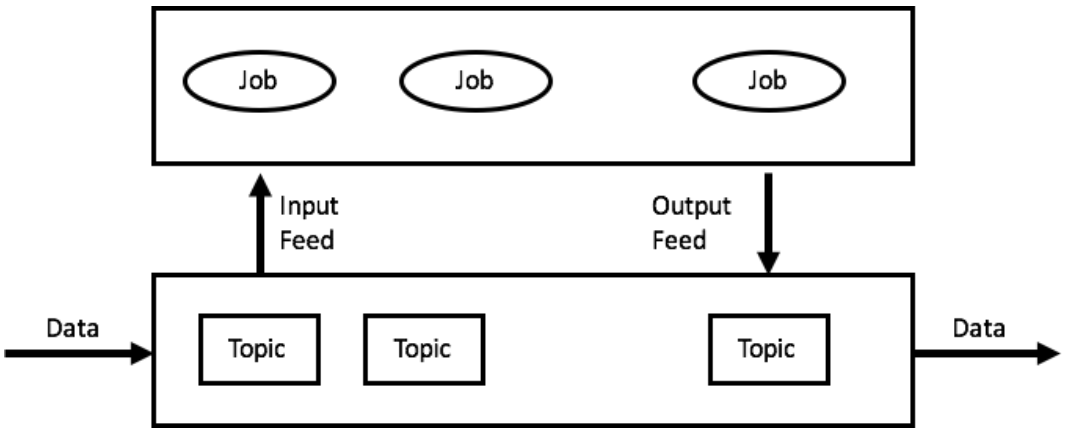

Figure 5. Liquid Architecture [38].

B. DevOps

B.1. Introduction.

DevOps is a hybrid software development and operation paradigm that predicates the intermixed co-operation, collaboration and communication between Dev-Teams (i.e. development teams) and Ops-Teams (i.e. Operations Teams). The fundamental axiom in the software operational paradigm is that Dev and Ops teams should work together since the one cannot deliver quality without efforts and insights of the other [39]. In recent times DevOps started playing a major role in the design, development and continuous integration of industrial-strength software [40].

In the DICE project DevOps is addressed in a novel way through a Model-Driven Engineering (MDE) approach. In fact, we argue that exploiting models and MDE both during the design and the operation time to reason on the applications under consideration and to evolve it to accommodate the required qualities is a good way to accomplish the principles suggested by DevOps.

Thus, in the rest of this chapter we focus on reviewing the methods, practices and tools that can be applicable or adaptable, in a DevOps context, to support part of the life-cycle of Data-Intensive Applications. More specifically:

- In Section B.2 we provide a short introduction to the general DevOps philosophy and practices.
- In Section B.3 we provide an overview of model-driven development and of those model-driven approaches that appear to be closer to the DICE objectives.
- In Section B.4 we present the most well-known modelling tools available, both open source and commercial.
- In Section B.5 we provide an overview of model-to-model transformation approaches. These are very important to automate the development steps in a MDE approach.
- In Section B.6 we review the standard TOSCA deployment approach and highlight the potential of this approach for DICE, and in Section B.7 we review other deployment and management tools well known in the market.
- In Section B.8 we focus on the approaches for continuous integration.
- In Section B.9 we review the approaches to keep track of versions of software and models as this is a very important issue for continuous integration and operation approaches.
- Finally, in Section B.10 we summarize the main strengths and weaknesses of the reviewed approaches and tools with respect to the DICE context.

In each section we present a short overview of the state of the art and highlight the relevance of the specific area being surveyed with respect to the DICE objectives.

While reviewing specific tools, we provide information on the following aspects:

1. The general dimension of the tool (website, licensing, present version, communities, FP7 projects [41] or companies that produce the tool, number of downloads, number of users etc).
2. The formalisms/standards supported by the tool (if any).
3. The features supported by the tool for a specific formalism.
4. The compatibility/integration with tools of different application domain.

B.2. Short introduction to DevOps

The DevOps philosophy revolves around four key values, explained in Figure 6. These are:

- The importance of developing and testing complex software in an environment that is as close as possible to the production one. The aim of this principle is to ensure that developers keep in mind and experience with the operation issues like performance, availability, stability of the service during development.
- The utility of deploying software as far as possible without losing the control of the services in operation and their qualities.
- The importance of continuously checking the quality of the software product and the associated services not only in the specific verification and validation phases but through the entire life-cycle of the services themselves.
- The need for a continuous collaboration between developers and operators with the aim of overcoming the typical frictions between the first ones, focusing on innovating products and the second ones focusing on guaranteeing the stability of the resulting services.

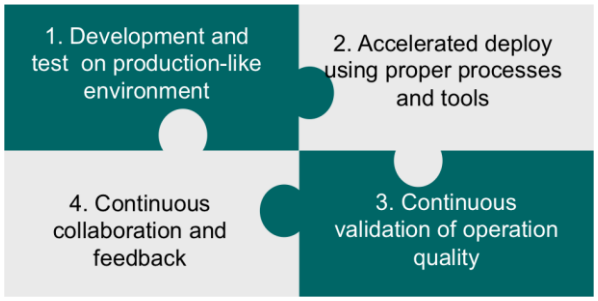

Figure 6. DevOps key values [42].

An essential part of the benefits connected to DevOps stems from the intrinsic collaboration predicated above and represents the possibility of continuously deliver and incrementally manage operating software using organisational and technical integration methodologies. In the scope of said goals, DevOps practitioners developed a line of tools to increase the speed at which Dev and Ops can cooperate, collocate their work and cohesively coordinate their efforts. Figure 7 shows a practical overview of the activities and tools behind the DevOps philosophy. In summary, a series of typical activities entail DevOps:

1. Industrial-Strength Version Control.
2. Continuous integration of several working-copies into a mainline several times per day.
3. Release automation, i.e., the automated (re-)deployment of improved products.
4. Infrastructure automation, i.e., providing automatically adjustable infrastructures.
5. Application management, i.e., the orchestration of services to guarantee optimal service-levels.

The above activities are usually supported by a series of tools that are proliferating as part of the DevOps movement. Widely known examples are:

1. Git (see Section **Error! Reference source not found.**), Ant [43] or Hudson (see Section B.8.2) (*Continuous Integration*).
2. Capistrano [44] or RPM Package Manager [45] (*Continuous Deployment*).
3. Cobbler [46] or Crowbar [47] (*Infrastructure automation*).
4. Puppet [48] and Chef [49] (*Configuration Management*).
5. Nagios [50] and Sensu [51] (*Run-Time Monitoring*).

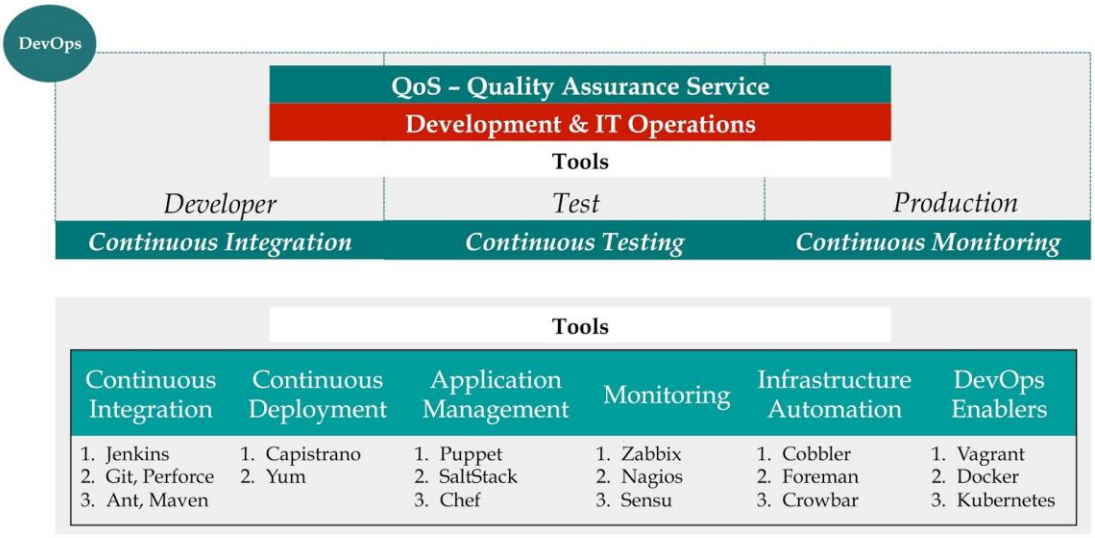

Figure 7. An Overview of the Activities and Tools behind DevOps.

Finally, the organisational integration to be enforced within DevOps typically uses simple combined teams (e.g., dev + ops) or follows other more strict organisational patterns typical in DevOps success stories, many of which are currently under investigation in the Software Engineering Institute [52] to assess their operational effectiveness. For example, Figure 8 shows an example of organisations structure typical in organisations that embrace DevOps. More in particular, Figure 8 shows a balanced blend between Dev- and Ops- people across a product portfolio.

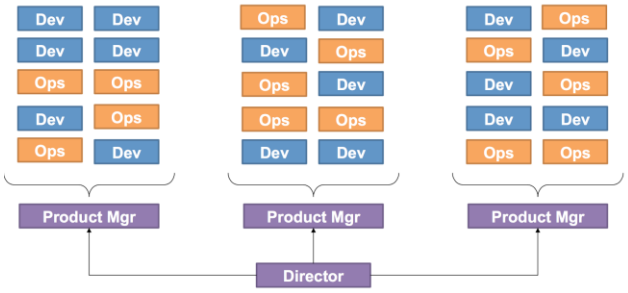

Figure 8. Typical Organisational Structures behind DevOps, an example [42].

In summary, DevOps is a movement that helps to bridge the cultural gap between development and operations [53]. Its goal is to enable each department to be aware of the perspective of the other and push them to change the dynamics in which they interact [54].

DevOps provides patterns to foster collaboration among project stakeholders and addresses shared goals and incentives as well as shared processes and tools [55]. Therefore, the concept of 'sharing' is at the very core of DevOps: sharing ideas, goals, issues, processes and tools. In addition, DevOps incites Devs and Ops teams to share their skills and experiences with each others, which leads to a one team approach where individuals have at least a basic understanding of others domains [55].

DevOps tries to extend Agile practices to operations by eliminating the wall between development and operations and to address the structural conflict between them: both teams work together to deliver application changes to the user at a high frequency and quality.

Although improving communication between developers and operations teams contributes to solve critical issues, it is only a portion of the wider equation; integrating the right tools is important for DevOps. Major parts of the releasing process should be automated along the delivery process in order to facilitate collaborative change [54].

Automation has many benefits: it ensures that the software is built the same way each time and makes parts of the process transparent for the whole team; thus software deployment to different target environments is made in the same manner [55].

Automation includes many steps (preparing the build, checking quality of code, launching build, running all tests, packaging, deploying and staging the artefacts) and necessitates scripts (for building, testing, deploying, configuring application, and configuring infrastructure). In his book, Michael Hüttermann explores concrete patterns for automatic releasing with appropriate tools [55] (Chapters 8 and 9).

By combining different approaches of DevOps and applying well-known DevOps practices, IT performance is strongly improved, which contribute to organisational performance as described by PuppetLabs report [56].

In the scope of DICE, DevOps methodologies and tools represent a set of reference materials as well as a potential target for further integrated support.

On the one hand, DevOps methodologies and tools need to be taken into account as organisational and technical concerns which are the key to enabling continuous delivery of Data-Intensive Applications (DIAs) by design and which govern the feedback loop between runtime and design time that is intrinsic to the DICE approach.

On the other hand, DevOps and connected tools/methods are themselves predicating on the usage of DIAs to further the understanding of the application lifecycle for further improvement and increased organisational/technical agility. In this regard, DICE may be a valuable tool to study where DIAs may play a role in finding valuable business intelligence to speed up application development and deployment. Moreover, testing within the DICE project should make massive and careful reference to DevOps methods and tools in order to establish guidelines and test-cases according to which DIAs may be designed and tested for ‘DevOps-Readiness’. This may include studying the best-fit organisational and socio-technical patterns of integration between DIAs and typical DevOps toolchains.

In addition, this may also entail studying empirically in which circumstances the DICE model-driven assumptions fail to meet DevOps expectations (e.g. continuous integration). Finally, from a methodological point of view, DIAs’ specification and monitoring should take into account DevOps dynamics and tools to enable DIAs monitoring in sight of their continuous improvement.

B.3. Functional modelling

B.3.1 Overview

From a functional perspective focused around modelling, the DICE approach to define, specify and analyse DIAs may inherit results from a number of technological baselines. In particular, we see the following baselines as relevant to the DICE project: (1) the ‘Model-Driven Architecture’ (MDA) standard [57], i.e., the Object Management Group (OMG) [58] standard for Model-Driven Engineering (MDE) [59], (2) the MODAClouds EU project [60] and related key results (e.g. MODACloudML [61]); (3) the REMICS EU project [62] and key results therein (e.g. Model-Driven cloud-migration techniques); (4) the Artist EU project [63] and key results therein that share a similar purpose to REMICS; (5) the JUNIPER EU project

[64] and key results therein share DICE goals and aims at an even lower level of abstraction; (6) in general, Unified Modelling Language (UML) and its profiles called MARTE (Modelling and Analysis of Real-Time and Embedded systems) and DAM (profile for Dependability Analysis and Modelling) (see Section B.3.3). These last ones are of particular interest to DICE as they allow users to model performance and other quality characteristics of applications. The rest of this section elaborates said technologies in more detail, with a hint as to their possible role in DICE.

B.3.2 MDE and domain-specific model-driven approaches

MDE techniques [59] and MDA in particular [57] define the typical abstraction layers for the purpose of engineering software systems using a model-centric perspective. The fundamental axiom behind this engineering paradigm is that any engineering endeavour shall be guided by at least three compounding and interoperating perspectives, namely: (c) Computational-Independent perspective; (b) a Platform-Independent perspective; (c) a Platform-Specific perspective. Using these three perspectives, one or more models can be specified to properly and systematically specify a system-to-be.

1. At the *Computational-independent level*, business-critical details are defined as such that intended business scenarios and systems goals may become apparent and explicit. Typically this perspective is consistent with requirements engineering activities such as stakeholder identification and scenario analysis.
2. At the *Platform-independent level*, architectural, quality and design issues are specified using one or more *Architecture Viewpoints*. The specification at this level typically uses model transformation technologies to support consistency and analysis across multiple Views and Viewpoints.
3. At the *Platform-Specific level*, design decisions are realised into well-formed designs, e.g., reflecting appropriate selection of design patterns, usable technological platforms and middleware (e.g. CORBA [65]).

Within DICE, MDE and MDA play a key role in providing a fundamental specification baseline. More specifically, the DICE profile inherits the MDA separation of concerns and logical decomposition, as well as a model-centric approach featuring multiple model transformations both for model consistency (and eventually technological deployment) and model analysis.

MDE and MDA have been adopted and specialised for various domains in many research projects. Among the others, MODACloudML [61] provides a Domain-Specific Modelling Language (DSML) along with a runtime environment in order to allow, on the one hand, to model the provisioning and deployment of multi-cloud applications, and on the other to automate the deployment and to facilitate their runtime management (in terms of adaptation or reconfiguration actions).

In this way MODACloudsML supports DevOps in achieving better delivery life-cycle by integrating in a single framework both development and operation activities.

MODACloudML supports both the Infrastructure as a Service (IaaS) and the Platform as a Service (PaaS) levels, even if it mainly focuses on the former. From the modelling perspective, MODACloudML allows the application specification at three levels of abstraction, which aim at following the general MDA paradigm: the Cloud-enabled Computation Independent Model (CCIM) to model an application and its data from a high-level business perspective, (ii) the Cloud-Provider Independent Model (CPIM) to characterise cloud concerns related to the application in a cloud-agnostic way, and (iii) the Cloud-Provider Specific Model (CPSM) to model the deployment and provisioning activities on a specific cloud.

At the CCIM level, an application is described as a set of high level services following a Service Oriented Architecture. At this level of abstraction these models involve three main concepts: a set of services, an orchestration, and a set of usage models. At the CPIM level MODACloudML proposes a new approach to describe the deployment, provisioning and data models of multi-cloud systems in a provider-agnostic way, supporting both IaaS and PaaS solutions. At the CPSM level, the design alternatives and deployment models as well as the data models are refined to include provider-specific concerns and technologies.

MODACloudML is also inspired by component-based approaches, which facilitates separation of concerns and reusability. In this respect, deployment models can be regarded as assemblies of components exposing ports and bindings between these ports. In addition, MODACloudML implements the type-instance pattern, which also facilitates reusability and abstraction.

Even if it is not specifically focused on DIAs, MODACloudML can become one of the foundational approaches behind the definition and further elaboration of the DICE profile for at least three reasons:

1. MODACloudML follows already the logical decomposition and model-driven engineering abstraction behind typical MDE-inspired techniques, e.g., with the division and arrangement in three tiers of modelling and analysis.
2. MODACloudML sets to describe Cloud-based applications - it is in fact a fundamental assumption behind DIAs that their very nature resides in the cloud. Hence, MODACloudML is a reasonable technology upon which to draw inspiration both in terms of modelling/deployment and analysis.
3. The MODAClouds project already provides infrastructure and usable technologies/tools for the specification, analysis and deployment of MODACloudML models, e.g., allowing the DICE profile to be quickly developed in prototypical form and further refined by means of MODAClouds-based technology.

Other EU projects that could offer a fundamental inspiration for DICE are REMICS [62] and Artist [63]. They look at ways in which model-driven techniques can be used to: (a) accelerate the adoption of cloud-based technology, possibly migrating legacy assets; (b) use models to drive the continuous improvement of cloud-intensive applications such as DIAs; (c) use model-centric perspectives to evaluate cloud assets and compare multiple cloud-vendors to compute solutions best-fitting with Service-Level Agreements (SLAs) and stakeholder/customer concerns. Quoting from REMICS website [62], the project's purpose is to provide a model driven methodology and tools which significantly improve the baseline Architecture-Driven Modernisation (ADM) [66] concept. In a similar vein, DICE could work jointly with efforts inherited from REMICS by specifying constructs, modelling and methodological notations to enable the systematic migration of legacy Big Data analytics into well-formed and quality-aware DIAs, enabled for continuous evolution.

Finally, the JUNIPER EU project [64] intends to construct the platform from real-time technologies, using real-time analysis, design and development principles, so that appropriate guarantees can then be given with respect to Big Data processing times, performance and similar quality attributes. In its current version, JUNIPER offers a programming model and a series of APIs to speed up the development of performance-aware Big Data applications.

On one hand, the JUNIPER programming model aims to conceptualise the development process of data-centric applications in a way that is general enough to cope with every need (in scope of the JUNIPER scenarios) but also allows common data processing patterns to be abstracted, modelled, and optimally deployed. The programming model proposed in JUNIPER is not a replacement of any of existing parallel processing frameworks and programming models, such as Hadoop/MapReduce [14], Message-Passing

Interface etc. Rather, the programming model shares the purpose of DICE, i.e., that of integrating said technologies to accelerate and further support their adoption during design and development.

On the other hand, as part of the JUNIPER baseline, the EU project offers Java improvements and primitives that tackle some missing feature like flexible parallelism, locality and hardware architecture discovery. More in particular, JUNIPER Application Programming Interfaces (APIs) characterise the host architecture in terms of accepted patterns and assist the development of reactionary software to exploit the hardware. When building software for Big Data systems, their unique architectures result in interesting challenges. The JUNIPER project identifies two main levels at which issues are observed: the cluster level and the node level. While at the cluster level, Big Data systems are deployed 'in the cloud', at the node level, the Big Data problem is to be decomposed into a 'normal data' problem.

From the DICE perspective, besides inheriting JUNIPER concepts and definitions as outlined above, DICE may use JUNIPER results as a baseline for at least two purposes: (a) inspire the DICE model-driven engineering of Big Data applications following patterns and approaches previously introduced and studied within JUNIPER; (b) test the DICE approaches and methodologies against scenarios previously envisioned as part of the JUNIPER challenges. The innovative aspects behind DICE appear clear from at least two standpoints: (a) ***Speed*** - a primary DICE goal is the speed-up of development & deployment times for Data-Intensive Applications, i.e., applications using multiple Big Data programming models at once, of which JUNIPER is only one possible alternative; (b) ***Quality-Awareness*** - another primary DICE goal is to provide facilities to extensively model and verify the assessment of quality aspects across Big Data applications, where JUNIPER merely provides constructs for provisioning of quality.

B.3.3 UML, MARTE and DAM

UML is a General Purpose Modelling Language (GPML). Therefore, it can be used to model a wide range of systems. Quoting the document [67], *'UML is a language with a very broad scope that covers a large and diverse set of application domains. Not all of its modelling capabilities are necessarily useful in all domains or applications'*. Conversely, Domain-Specific Modelling Languages (DSML) and Domain-Specific Languages (DSL) are conceived for addressing the needs of specific application domains. A DSL captures the semantics of the domain and offers the syntax needed for modelling the concepts in such domain. By using a DSL the designer does not need to learn concepts completely irrelevant for the domain they are addressing. In this regard, UML offers a solution, the so-called UML profiling mechanism [67]. Profiling opens the possibility of creating DSLs by extending or restricting UML. A Profile is then an adaptation of UML to fit a specific domain. Examples of UML Profiles are MARTE [68] and DAM [69]. MARTE (Modelling and Analysis of Real-Time and Embedded systems) provides support for the specification, design, quantitative evaluation, and verification & validation of software systems. DAM (Dependability Analysis and Modelling Profile) provides support for the dependability modelling and analysis of software systems. A UML model annotated with the MARTE and/or DAM Profile will be called a UML-MARTE or UML-DAM model.

Summarising, some of the benefits of using the MARTE and DAM Profiles are [68]:

- Providing a common way of modelling both hardware and software aspects of a system to improve communication between developers.
- Enabling interoperability between development tools used for specification, design, verification, code generation, etc.
- Fostering the construction of models that may be used to make quantitative predictions regarding real-time and embedded features of systems taking into account both hardware and software characteristics.

However, neither MARTE nor DAM has a direct support for expressing data location, data properties such as volume or transfer rates or operations that move data. Hence, addressing such lack is one of the objectives of the DICE project.

B.3.3.1. Domain-Specific Modelling with UML

UML offers different diagrams for the modelling of the structural, behavioural and distribution views of a system. For example, the object diagram and the class diagram describe the structure of a system. The state machine diagram, interaction diagrams, activity diagram and use cases are used for the modelling of the system dynamic and behaviour. The component and deployment diagrams describe system distribution. A UML diagram is made of elements, for example, a class diagram is made of classes and relationships among them, such as associations, inheritance or dependencies. The UML package diagram is useful for organising UML diagrams and/or UML elements. The UML model of a system is made of a set of UML diagrams. A UML model has to conform to the UML *meta-model*. A meta-model is a set of related meta-classes. A meta-class is the abstraction of a set of UML elements. For example, in a UML class diagram, each association belongs to the Relationship meta-class of UML since associations share characteristics with other relationships. This meta-model feature is an interesting characteristic of UML since the profiling mechanism builds on it. The UML Profiles package contains mechanisms that allow meta-classes from existing meta-models to be extended to adapt them for different purposes. This includes the ability to tailor the UML meta-model for different platforms (such as J2EE [70] or .NET [71]) or domains (such as real-time, business process modelling or DIA). A UML Profile is made of a set of stereotypes, a set of tags and a set of related constraints. A *stereotype* is just a name that will be attached to certain elements of a UML diagram. Stereotypes have *tags*, we can see them as the attributes added by the stereotype. A constraint can be attached to a stereotype definition. It is expressed in natural language or in the Object Constraint Language (OCL) [72] and describes restrictions for the stereotype, e.g. for expressing subsets of values for the stereotype. Figure 9 goes deeper into the Profile definition. A Profile is a specialisation of the UML concept of Package, which means that a Profile is a set of modelling elements, in fact the stereotypes, tags and *constraints*.

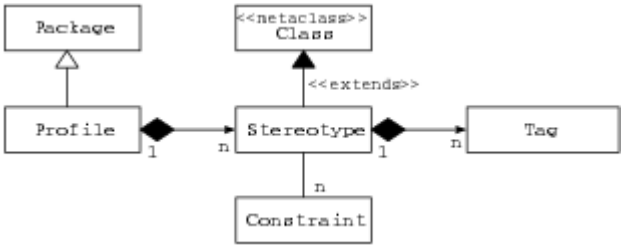

Figure 9. Sketch of UML Profile definition.

Therefore, stereotypes are the cornerstone concept in a Profile. At model-specification level stereotypes are applied to concrete UML elements, which is possible since they extend concrete meta-classes of UML, as we can see in Figure 9.

B.3.3.2. Modelling with MARTE

The MARTE profile consists of three main parts: MARTE Foundations, MARTE Design Model and MARTE Analysis Model.

First, the MARTE Foundations define the basic behaviour concepts for the Real-Time and Embedded Systems (RTES) domain, such as a causality model, a common framework for annotating models with quantitative and qualitative Non-Functional Properties (NFP, say performance, reliability and safety), the modelling of time, the modelling of resources and their allocation concerns.

Second, the MARTE Design Model addresses the modelling of RTES. It introduces a general component model suitable for RTES, high-level concepts for designing qualitative and quantitative concerns (e.g., concurrency and synchronisation) and a detailed resource modelling dedicated to software and hardware. Third, the MARTE Analysis Model enables the analysis features of the system. It consists of a Generic Quantitative Analysis and Modelling (GQAM) profile which includes common concepts to any kind of quantitative analysis (such as workload, scenario, resource, etc.), a Schedulability Analysis and Modelling (SAM) profile and a Performance Analysis and Modelling (PAM) profile. SAM and PAM are specialisation of the GQAM profile. It is important to note that the DAM profile also specialises the GQAM profile.

In addition, MARTE contains some annexes: the Value Specification Language (VSL) profile defines an expression language for specifying the values of constraints, properties, and stereotype attributes, particularly related to NFP; the MARTE Library defines primitive data types, a set of predefined NFP types and units of measures.

B.3.3.2.1. Specification of NFP

The MARTE NFP profile was designed to address with the following specification needs: how to specify NFPs, how to attach NFPs to UML model elements, how to define relationships between different NFPs and how to express constraints on or between NFPs. The NFP profile imports the VSL profile. The VSL profile defines a set of language constructs and a textual grammar supporting extended values, expressions and data types used particularly for specifying NFPs. The NFP package realises the following requirements for NFPs:

- NFP *Qualitative or Quantitative nature*. A quantitative property may be characterised by a set of measures expressed in terms of magnitude and unit. A qualitative property may be denoted by a label (e.g. periodic, sporadic and bursty for an event arrival patterns) representing an abstract characterisation with a certain meaning for designers or tools.
- *Qualifiers* can characterise the precision, accuracy, statistical measure (e.g. mean, maximum, minimum and variance) or source (showing how a given value was obtained, e.g. required, measured, estimated, simulated or calculated).
- *Variables and Expressions*, beside concrete values, raise the level of abstractions of the specified properties, allowing the derivation of ones from the others.
- *Trade-off between usability and flexibility*. Usability suggests the merit of declaring a set of standard property types and their available operations for a certain domain, while flexibility allows for user-defined properties.

VSL defines the following data types: *Bounded Subtype*, *IntervalType*, *CollectionType*, *TupleType* and *ChoiceType*. VSL also defines four kinds of composite value specifications: collection, interval, tuple, and choice. The value specifications defined in VSL can be attached to stereotype attributes or used in constraints.

B.3.3.2.2. MARTE::GQAM framework

The fundamental concepts shared by different quantitative analysis domains are joined in a single package called Generic Quantitative Analysis Model (GQAM). This package is further specialised for schedulability (SAM), performance (PAM) and dependability (DAM) analyses. The core GQAM concepts describe how the system uses resources over time. It contains three main categories of concepts: resources, behaviour and workloads. In the following, we detail the resource concepts. A resource contains common features such as scheduling discipline, multiplicity, services. The following types of resources are important in GQAM:

- *ExecutionHost*: a processor or other computing device on which are running processes.
- *CommunicationsHost*: a communication network or a bus connecting processing nodes and/or devices.
- *SchedulableResource*: a software resource managed by the operating system, like a process or thread pool.
- *CommunicationChannel*: logical channel that conveys messages.

In order to be executed, a *SchedulableResource* must be allocated to an *ExecutionHost* and a *CommunicationChannel* to a *CommunicationHost*. Services are provided by resources and by subsystems. A subsystem service associated with an interface operation provided by a component may be identified as a *RequestedService*, which in turn is a subtype of Step, and may be refined by a *BehaviourScenario*.

B.3.3.3. Modelling with DAM

The DAM profile addresses the dependability modelling of RTES with UML. According to the Figure 10 DAM builds on MARTE and consists of a library and a set of extensions. The latter are the stereotypes of the profile and most of them specialise the ones of MARTE. In the following we describe the main elements of the DAM profile.

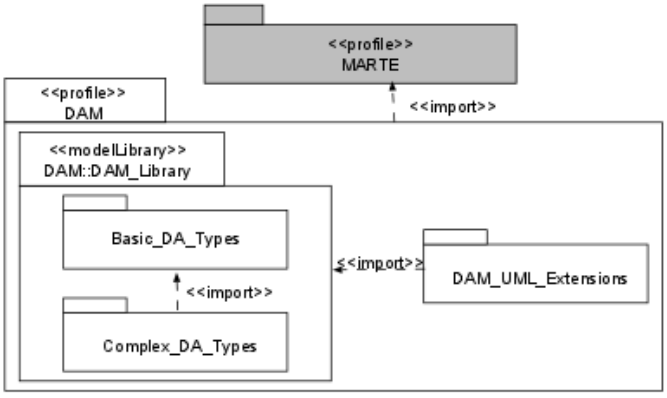

Figure 10. DAM profile overview.

B.3.3.3.1. DAM Library

The DAM library contains basic and complex dependability types. The following MARTE packages have been used to define the DAM library:

- The NFPs MARTE sub-profile for the definition of new basic dependability types.
- The VSL MARTE sub-profile for the definition of complex dependability types.
- The MARTE library, where the Basic NFPs types are imported in order to reuse them (both in the definition of complex and basic dependability types).

B.3.3.3.2. Basic Dependability Types

DAM defines the following basic dependability types:

- Simple enumeration types enable to characterise the threats according to the classification in [73] and [74].
- Data types are new NFP types obtained by specialising the NFP CommonType and NFP Real types of the MARTE library. These new types inherit from their supertypes the following properties:
  - *expr*: An expression in MARTE VSL.
  - *source*: The origin of the specification. It can be estimated (e.g., a metric to be estimated), required (e.g., a requirement to be satisfied), calculated or measured.

- *statQ*: The type of statistical measure (e.g., maximum, minimum, mean).
- *dir*: The type of the quality order relation in the allowed value domain of the NFP, for comparative analysis purposes.

B.3.3.3.3. Complex Dependability Types

Complex dependability types are MARTE tupleTypes characterised by basic NFPs, from the MARTE library, and/or basic dependability types. They enable to characterise both from a qualitative and a quantitative point of views, the threats (i.e., faults, errors, failures and hazards) and the mitigation solutions (i.e., recovery and repair strategies). As for stereotypes, a complex dependability type for DAM is prefixed by 'Da'. Figure 11 depicts these types.

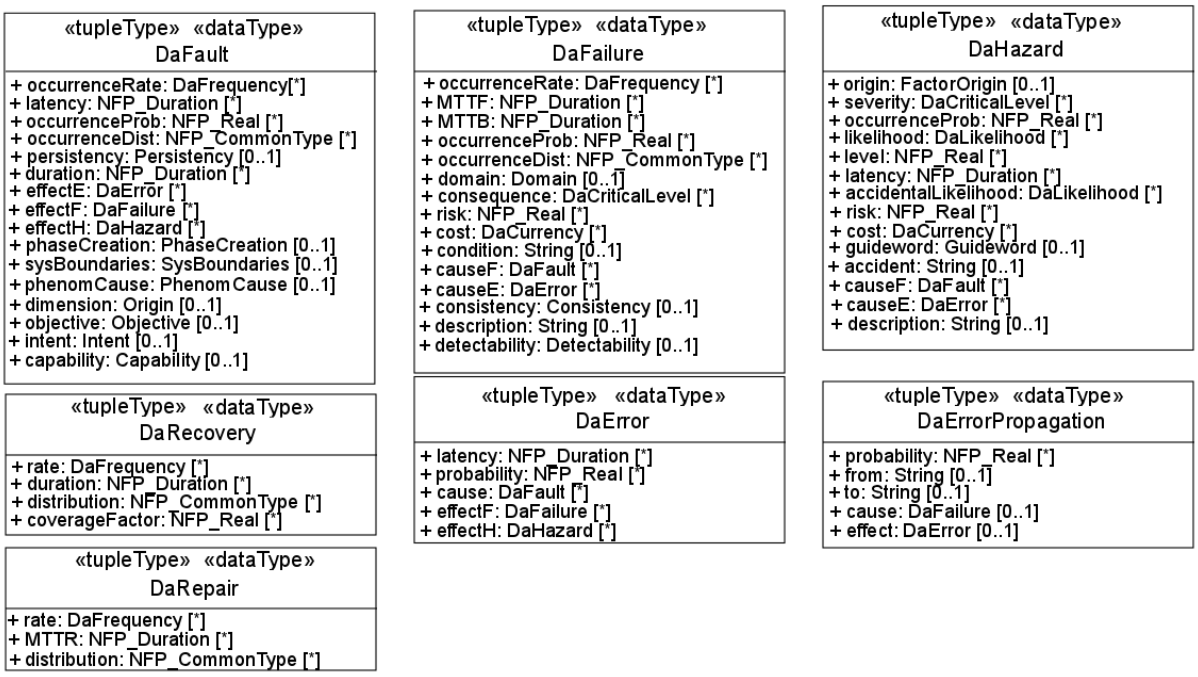

Figure 11. DAM types.

B.3.3.3.4. DAM UML extensions

The DAM extensions provide the domain expert with a set of stereotypes to be applied at model specification level, i.e., the stereotypes necessary to represent the dependability system view in a concrete UML model. DAM aims at providing a small yet sufficient set of stereotypes to be actually used in practical modelling situations. The DAM stereotypes are: *DaComponent, DaConnector, DaService, DaServiceRequest, DaStep, DaErrorPropRelation, DaFaultGenerator, DaReplacementStep, DaReallocationStep, DaActivationStep, DaAgentGroup, DaController, DaVariant, DaAdjudicator, DaSpare* and *DaRedundantStructure*. For a complete description of the tags for each DAM stereotyped refer to [75].

B.3.3.4. Access Control Modelling with UML

Over the years a number of UML profiles have been defined to represent security-related properties of systems. Since the focus of the DICE project for what concerns issues of data protection and privacy rests on problems related to giving access to data and information only to components that have the appropriate rights for that. In this brief section we analyse several approaches for the modelling of access rights through

UML concepts. These approaches could offer the elements through which the DICE profile includes the concepts that are suitable for the modelling of security issues in DIAs.

SecureUML [76] is a UML profile that can be combined with domain specific languages, to provide the latter with the concepts necessary to describe the rights to perform certain actions (e.g., read/write data). In particular, it captures the concepts underlying Role Based Access Control (RBAC) [77] through suitable stereotypes. In particular, these stereotypes allow users to define what resources need to be protected, which users can access them, and what actions the users can perform on the resources.

SecAM [78] is a UML profile based on MARTE and DAM. SecAM addresses access control as well as cryptography, resilience and security mechanisms. The access control package supports the specification of different access control policies: Mandatory Access Control (MAC), Discretionary Access Control (DAC) and Role-Based Access Control (RBAC). SecAM addresses confidentiality, integrity and authorisation issues. SecAM proposes a small set of stereotypes that include: a) who is accessing to the system (SecaSubject stereotype); b) which objects should be protected for unauthorized access or modification (SecaObject stereotype) and c) which operations a user wants to perform on the available objects (SecaOperation).

UMLsec [79] enables to specify security relevant information during development of security-critical systems. In particular, it considers RBAC as access control policy and provides a tool-support [80], implemented as Eclipse plugin, for formal security verification.

B.4. Modelling tools

This section overviews the most relevant UML Computer-Aided Software Engineering (CASE) Tools for making a comparative analysis. The goal is to select an appropriate UML modelling tool for the DICE IDE. The DICE vision imposes some constraints for the modelling framework we are developing. First, any tool to be integrated in the IDE needs to be open source and free of charge. Second, the UML modelling tool needs to import/export XML Meta-data Interchange (XMI) [81] for MARTE Profile annotated models, so the tool needs to support UML2 and MDA. Third, the UML modelling tool should support the largest number of UML diagrams. In the following, we list the UML tools we considered for our comparison. More information about UML tools can be found here:

- http://modeling-languages.com/uml-tools/
- http://software-talk.org/blog/2014/05/comparison-of-free-uml-tools/

Enterprise Architect (EA) [82] is a commercial software, which means that it's not suitable for DICE IDE. However, it is an exceptional tool with a rich set of features. So we use EA as a baseline in our analysis since it defines a comprehensive set of requirements that will be used for comparison with free UML modelling tools.

Papyrus [83], UML2 Modeller is an open source tool. It is based on the Eclipse framework and it is licensed under Eclipse Public License (EPL) [84]. Papyrus primary goal is to implement the complete standard specification of UML2. Papyrus also provides an extensive support for UML profiles. It includes hence all the facilities for defining and applying UML profiles. The MARTE profile is available in Papyrus. An inconvenience we discovered is that the learning curve could be higher than for the other tools. However, this tool is well-documented [83], [85]-[92].

Modelio [93] is an Open Source UML modelling tool developed by Modeliosoft [94]. It supports UML 2.0 and Business Process Model and Notation (BPMN) [95] standards. The core Modelio software was

released under the GPLv3 (a viral license which is not suitable for DICE), although there is a commercial version too. Modelio allows the import/export of UML models from/to other tools via UML XMI format. Modelio enables the use of the MARTE profile [96].

MOdeling Software KIT (MOSKitt) [97] is a free CASE tool, built on Eclipse which is being developed by the Valencian Regional Ministry of Infrastructure and Transport, namely through technology provider Prodevelop, to support the gvMétrica methodology (adapting Métrica III to its specific needs). It supports UML2, BPMN [95] standards and database diagrams. MOSKitt is released under EPL [84]. MOSKitt gives a framework to build Model-to-Model (M2M) transformations based on Eclipse standards, as well as import capabilities for external UML XMI [81] format models.

ArgoUML [98] is an open source and free UML modelling tool distributed under the EPL 1.0 [84]. ArgoUML is a Java-based application that is available in ten languages. ArgoUML also provides code generation for Java [99], C++ [100], C# [101], PHP4 and PHP5 [102]. It also enables reverse engineering from Java. External modules have been developed to complement ArgoUML in specific areas. They provide generation of database schemas or code in other languages like Ruby [103] or Delphi [104]. However, ArgoUML offers support only for UML 1.4 diagrams, which is not enough for DICE.

StarUML [105], [106] is a UML tool licensed under a modified version of GNU General Public License (GPL) [107] until 2014. A rewritten version (StarUML 2) was released in 2015 under a proprietary license. StarUML 2 is compatible with UML 2.x standard and supports totally 11 kinds of UML diagrams: Class, Object, Use Case, Component, Deployment, Composite Structure, Sequence, Communication, Statechart, Activity and Profile Diagram. StarUML 2 stores models in a very simple JSON format.

UML Designer by Obeo [108] supports UML 2.5 models. It uses the standard UML2 meta-model provided by the Eclipse Foundation [109]. Obeo is a free tool (Open Source with EPL license [84]) for prototypes and starter projects but it is necessary to pay a fee for Small/Medium/Large or critical projects. UML Designer is based on Sirius [110]. It provides an easy way to combine UML with domain specific modelling. UML Designer includes a MARTE Designer too (Beta release) [111] which is a graphical tool to edit and visualise MARTE models.

MagicDraw [112] is a commercial UML modelling tool with a free educational edition. It's written in Java, supports UML2 [109], SysML [113], BPMN [95] and UPDM [114]. It provides MDA and code engineering mechanism (support for J2EE [70], C# [101], C++ [100], CORBA IDL programming languages [65], .NET [71], XML Schema [115], WSDL [116]), as well as database schema modelling, DDL generation and reverse engineering facilities. It supports large projects and is used at Netfective Technology as the modelling component of the BluAge product [117].

IBM® Rational® Software Architect [118] is a comprehensive design, modelling and development tool for end-to-end software delivery. It uses the Unified Modelling Language (UML) for designing enterprise Java® applications and web services. Rational Software Architect is built on the Eclipse open source software framework and is extensible with a variety of Eclipse plug-ins. You can also enhance functionality for your specific requirements with separately purchased Rational extensions.

B.4.1 Analysis

Table 3 provides summary of the tools reviewed in this section.

Table 3: UML CASE Tools summary.

Name	Creator	Platform/OS	Open source	Software license	Programming language used
Enterprise Architect [82]	Sparx Systems [119]	Windows (Supports Linux & Mac installation)	No	Commercial	C++ [100]
Papyrus [83]	CEA [120], Atos [121]	Windows, Linux (Java)	Yes	EPL [84]	Java [99]
Modelio [93]	Modeliosoft [94]	Windows, Linux, Mac	Yes	GPL and Commercial	Java
MOSKitt [97]	Conselleria de Infraestructuras, Territorio y Medio Ambiente [122]	Windows, Linux (Java), Mac	Yes	EPL	Java
ArgoUML [98]	Tigris.org [123]	Cross-platform (Java)	Yes	EPL	Java
UML designer [108]	Obeo Model Driven Company [124]	Windows, Linux, Macs	Yes	EPL	Java
MagicDraw [112]	No Magic, Inc. [125]	Windows, Linux, Mac	No	Commercial	Java
Rational Software Architect [118]	IBM [126]	Windows, Linux, Mac	No	Commercial	Java

B.4.2 MARTE profile feature (import *.XMI)

As MARTE [68] is the most well-known UML profile for expressing quality characteristics of software systems, we analyse the tools with respect to their capability of supporting such profile.

MARTE has been defined via a UML2 profile. Thus, UML tools able to support MARTE have to import at least XMI with version of UML2. Currently, three open source tools are available for system modelling using the MARTE profile: Modelio [93], Papyrus UML [83] and MARTE Designer (Obeo, but it is still in Beta Status) [111]. ArgoUML [98] cannot import UML2, therefore it cannot support the MARTE Profile. Table 4 provides an overview of all tools (open source and commercial) and of their level of support for MARTE.

Table 4: CASE tools.

Name	UML2	MDA [57]	XMI [81]	Templates	Languages generated	Can be integrated with	MARTE [68]	User Manual	Forum community *(30/03/2015)*
Enterprise Architect [82]	Yes	Yes	Yes	Supports MDA templates and Code Generation templates	ActionScript [127], C, C# [101], C++ [100], Delphi [104], Java, PHP [102], Python [128], Visual Basic [129], Visual Basic .NET [130], DDL [131], EJB, XML Schema [115], Ada [132], VHDL [133], Verilog [134], WSDL [116], BPEL [135], Corba IDL [65]	Eclipse & Visual Studio [136]	Yes	5 / 5 [139]	Post: 108641 Topics: **28604** Users: 153438 [146]
Modelio [93]	Yes	Yes	Yes	Yes	Java, C++, C#, XSD, WSDL, SQL	Eclipse, EMF	Yes	4 / 5 [140]	Post: **2966** Topics: **639** Users: **968** [147]
Papyrus [83]	Yes	Unknown	Yes	Unknown	Ada 2005, C/C++, Java add ins	Eclipse	Yes	2 / 5 [141]	(Eclipse Forum)
MOSKitt [97]	Yes	Yes	Yes	M2M and M2T generation	HTML, CSS, Java	Eclipse	No	[142]	[148]
ArgoUML [98]	No	Yes	Yes	Unknown	C++, C#, Java, PHP4, PHP5, Ruby [103]	AndroMDA [137]	No	3 / 5 [143]	Post: **1457** Topics: **453** Users: **254** [149]

UML Designer [108]	Yes	Yes	No	Yes	Java	Eclipse	No	2 / 5 [144]	Users: **977** [150]
MagicDraw [112]	Yes	Yes	Yes	Yes	Java, C++, C#, CIL, CORBA IDL, DDL, EJB, XML Schema, WSDL	Eclipse EMF, NetBeans [138]	Yes	3/5 [145]	Post: **9558** Topics: **3099** Users: **1295** [151]
Rational Software Architect [118]	Yes	Yes	Yes	Yes	Java	Eclipse	No	-	-

B.4.3 UML diagrams supported

Even though UML 2.0 has been standardised long ago, not all tools support all its diagrams. Table 5 shows the diagrams that are supported by each tool.

Table 5: UML 2.0 Diagrams supported by modelling tools.

	EA [82]	**Papyrus [83]**	**Modelio [93]**	**MOSKitt [97]**	**UML Designer [108]**	**ArgoUML [98]**	**MagicDraw [112]**	**RSA [118]**
Structural UML diagrams								
Class diagram	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Component diagram	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Composite structure diagram	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Deployment diagram	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Object diagram	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Package diagram	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Profile diagram	Yes	Yes		Yes	Yes	Yes	Yes	Yes
Behavioral UML diagrams								
Activity diagram	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Communication diagram	Yes	Yes	Yes	No	No	Yes	Yes	Yes
Interaction overview diagram	Yes	Yes	Yes	No	No	Yes	Yes	Yes
Sequence diagram	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
State diagram	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Timing diagram	Yes	Yes	No	No	No		Yes	Yes
Use case diagram	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Other diagrams								
BPMN [95] diagram	Yes	Yes	Yes	Yes	No	No	Yes	Yes
SysML [113] diagram	Yes	Yes	Yes	No	Yes	No	Yes	Yes
Database diagram				Yes	No	No	Yes	

B.4.4 Summary

In summary the tools that appear to be suitable to be used in the DICE context are Modelio [93] and Papyrus [83] since they have the appropriate licenses and support the UML Profile mechanism. However, the Modelio plug-in for MARTE is not supported anymore and it does not work with the last version of Modelio designer.

The last version of Papyrus (1.1) was released in June 2015 with Eclipse Mars [152]. This version brings new features and improvements but also tackles some known issues related to performance and user-friendliness. In particular, in a well-filled containment tree, especially a developed tree, the graphical user interface works smoothly as long as the size of the model does not becomes excessively large, i.e., more than 2000 visible nodes.

B.5. Tools for model to model transformations

The purpose of this section is to evaluate tools that transform software models, described using non-formal or semi-formal languages, into formal models. The goal of the target formal model is to perform analyses of the non-functional properties of the software system, such as performance, reliability, availability, safety or privacy. To this end, we have reused the evaluation framework proposed by [153]. This work also reviewed some tools. Here we have added to the original evaluation some new tools that are of interest to DICE. The evaluation framework proposed in [153] is summarized in the Table 6:

Table 6: Summary of the evaluation framework for model-to-model transformation tools [153].

#	Characteristic	Description/question
1	model specification	Does the tool support specification of systems as graphical models? *{Yes/No}*
2	graphical notation for model transformation	Does the tool support graphical specification of transformation? *{Yes/No}*
3	lexical notation for model transformation	Does the tool support lexical specification of transformation? *{Yes/No}*
4	model-to-model transformation support	Does the tool support model-to-model transformation? (e.g., from one UML model to another?) *{Yes/No}*
5	model-to-text transformation support	Does the tool support model-to-text transformation, such as generation of source code? *{Yes/No}*
6	support for model analysis	Is there any support for model analysis? *{Yes/No}*
7	support for Quality of Service (QoS) management	Is there any support for managing QoS during model specification and transformation? *{Yes/No}*
8	meta-model-based	Is the tool based on explicit descriptions of the meta-models of source and target model? *{Yes/No}*
9	MOF[1] integration	Is the tool integrated with a MOF[1] (or other meta-model-based repository)? *{Yes/No}*
10	XMI integration	Is the tool integrated with XMI[1]? {Yes/No} which version(s) of XMI is supported? *{list of versions}*
11	based on UML	Is the tool based on UML models as source and/or target models for transformation? *{Yes/No}*
12	UML specification	Does the tool provide support for UML modelling *{Yes/No}*
13	UML tool integration	Can the tool be integrated with existing UML tools? Either directly, as active plug-ins in UML tools, or indirectly through model exchange via, e.g., XMI? *{Yes/No}*or*{names of the set of techniques}*
14	iterative and incremental transformation support	Does the tool handle reapplication of transformation after model updates? *{Yes/No}*
15	Bidirectional transformation	Does the tool support bidirectional transformations? *{Yes/No}*
16	traceability	Does the tool handle traceability of transformations, i.e., can it maintain traces of the source and targets of a transformation? *{Yes/No}*
17	DSM language support	Is there support for defining domain-specific modelling languages (e.g., UML profiling) and DSM transformations? *{Yes/No}*

[1] **MOF** - Meta Object Facility. **XMI** - XML Metadata Interchange [81]

Once presented the evaluation framework we apply it to the tools of interest.

B.5.1 Palladio Component Model

The Palladio Component Model (PCM) [154] captures the software architecture with respect to static structure, behaviour, deployment/allocation, resource environment/execution environment, and usage profile. In the PCM software is described in terms of components, connectors, interfaces, individual service behaviour models (so-called Service Effect Specifications, SEFF), servers, middleware, virtual machines, network, the allocation of components and servers, models of the user interaction with the system etc. Overall, the PCM captures multiple views of software systems including elements which affect the extra-functional properties (e.g. performance, reliability etc.) of software systems [154]. Table 7 presents evaluation framework from Table 6 applied to Palladio Component Model.

Table 7: Evaluation framework from Table 6 applied to Palladio Component Model.

1	Yes, tool called 'PCM-Bench', which enables software developers to create instances of the PCM meta-model
2	No
3	No info given
4	Model-to-model transformation from a PCM instance to an SRE instance are performed with Java
5	Yes, model-to-text transformation based on the openArchitectureWare (oAW) framework generates code skeletons from PCM model instances. The implementation uses either Plain Old Java Objects (POJOs) or Enterprise Java Beans (EJBs) ready for deployment on a J2EE [70] application server
6	Model validation by checking OCL [72]constraints
7	Yes
8	Yes, the PCM is a meta-model designed to describe component-based software architectures in order to analyse performance properties
9	Yes
10	Model instances can be serialised to XMI-files
11	Yes
12	No
13	No
14	Yes
15	No
16	No
17	No

B.5.2 VIATRA2

The main objective of the VIATRA2 [155]-[157](VIsual Automated model TRAnsformations) framework is to provide a general-purpose support for the entire lifecycle of engineering model transformations including the specification, design, execution, validation and maintenance of transformations within and between various modelling languages and domains.

Table 8 presents evaluation framework from Table 6 applied to VIATRA2 model transformation tool.

Table 8: Evaluation framework from Table 6 applied to VIATRA2 framework.

1	Yes. Models and meta-models are all stored uniformly in the VPM model space, which provides a very flexible and general way for capturing languages and models on different meta-levels and from various domains (or technological spaces).
2	No
3	Yes, VTCL transformation language [155]
4	Yes, Intra model transformations and Inter model transformation, both of these transformation categories are supported by the transformation language of the VIATRA2 framework.
5	Yes, VIATRA2 supports mode-to-code generation in different ways
6	[155]
7	Yes
8	Yes, standard metamodelling paradigms are integrated into VIATRA2 by import plugins.
9	Yes
10	Yes
11	Yes
12	Yes
13	Yes
14	Yes
15	No
16	Yes
17	Yes

B.5.3 UML transformation tool

UML Transformation tool (UMT) [158] is a tool to support model transformation and code generation based on UML models in the form of XMI [81]. This is a generic tool, so it does not provide models in a concrete formalism, but the environment for obtaining them. Table 9 presents evaluation framework from Table 6 applied to UML transformation tool.

Table 9: Evaluation framework from Table 6 applied to UML transformation tool.

1	No. There is no support for specifying models in UMT. It relies entirely on imported models from UML tools
2	No. There is no graphical notation for model transformation
3	Yes. UMT uses XSLT and Java as transformation languages, with possibility of extending to support other languages
4	No
5	Yes. Model-to-text transformation is the main functional domain for UMT

6	No. There is no support for model analysis, except for very simple support for checking of a model's conformance to simple profiles
7	No. There is no support for management of QoS
8	No. UMT only targets the UML meta-model and is not flexible with respect to changing this
9	No. There is no integration with MOF
10	Yes. UMT imports UML/XMI files from different UML tools
11	Yes
12	No. There is no support for specifying UML models. UMT relies wholly on model input from external UML tools
13	No. There is no direct UML tool integration. Integration is indirect through XMI
14	There is lightweight support for regenerating code without overwriting previously generated and modified code
15	No. There is no direct support for bidirectional transformation. However, there is some support for reverse engineering of code to XMI models
16	No
17	The tool does not provide support for defining DSM languages. It provides support for transformations of DSM languages. E.g., transforming one DSM-based model to another DSM-based mode

B.5.4 Other tools

The following tools have poor information and we could not fill the evaluation framework. So we provide a brief description of them.

B.5.4.1. CARiSMA

The CARiSMA [80] core is independent from any particular modelling language. It is just based on the Eclipse Modelling Framework (EMF). CARiSMA enables compliance analyses, risk analyses, and security analyses of software models. A flexible architecture makes CARiSMA extensible for new languages and allows users to implement their own compliance, risk, or security checks.

B.5.4.2. UPUPA (fUML and Profiles for Performance Analysis)

Upupa [159] considers non-functional properties of a software system early in the development process. UPUPA develops a model-based analysis framework based on the Foundational Subset for Executable UML Models fUML [160] for enabling the implementation of model-based analysis tools. This framework enables to carry out model-based analysis of non-functional properties of a software system based on runtime information in the form of traces obtained by executing UML models using the fUML virtual machine. Therefore, the framework integrates UML profile applications with execution traces to enable the consideration of additional information captured in profile applications in the model-based analysis as required for instance in performance analysis.

B.5.4.3. QVT and Related Technologies

QVT stands for Query-View-Transformations [161], a generic, all-purpose model transformation and manipulation language standard that goes along with Model-Driven Architecture (MDA) standard [57] to support its purposes and intent. QVT is an OMG [58] standard and, in essence, it defines a standard way in which model transformation shall take place, using standard elements and operational transformation behaviour. QVT defines standard views in which model information can be presented and manipulated, e.g. in order to migrate a Platform-Independent Specification into a Platform-Specific one. The most widely

known technology related to QVT resides within the eclipse foundation model-manipulation environment, i.e. AMMA [162]. AMMA features Atlas Transformation Language (ATL) [163], which is a QVT-like model transformation language, with its own abstract syntax and environment. The transformation is itself a model conforming to a specific meta-model. This, for example, permits the creation of higher order transformations, i.e., transformations that produce ATL transformations.

B.5.5 Summary

DICE will need to implement Model-to-Model (M2M) transformations extensively. First at UML level, from the DPIM (DICE Platform Independent Model) to the DTSM (DICE Technology Specific Model), and from the latter to the DDSM (DICE Deployment Specific Model). Second, on the DDSM an M2M transformation will be useful for yielding the TOSCA deployment (see Section B.6). Third, each UML diagram in the DIA design needs to be translated into the corresponding target formal model, which comprises a combination of formalisms and UML diagrams at different abstraction levels (DPIM, DTSM and DDSM). The third kind of M2M transformation needs to take into account the DICE stereotypes and tags for parameterising the formal models and for extracting the quality requirements expressed as SLAs. However, from the analysis of tools we carried out in this section we found the following conclusions. First, there is no tool that can be reused for M2M transformations in the context of DICE. Second, the framework in [153] should guide the development of the M2M DICE transformation tool, which means to try to develop a tool able to answer 'Yes' to as many as possible of the framework questions.

B.6. Deployment modelling with TOSCA

TOSCA stands for ''Topology and Orchestration Specification for Cloud Applications'' [164]. In the hands of TOSCA lies the state of the art for deployment solutions that are both technology independent and multi-compliant. This intrinsic characteristic stems from the joined interplay within which TOSCA was originally specified, i.e., the OASIS standardisation effort. Within the OASIS TOSCA Technical Committee (TC) big industrial players (e.g., IBM, Huawei, Ericsson) defined the essential elements for the purpose of providing easily deployable specifications for cloud applications in all its aspects, including, but not limited to, Network Function Virtualisation, Infrastructure Monitoring and similar. Essentially, quoting from the TOSCA specification 1.0 [164], ''TOSCA [...] uses the concept of *service templates* to describe *cloud workloads* as a *topology template*, [...]. TOSCA further provides a type system of node types to describe the possible building blocks for constructing a service template, as well as relationship type to describe possible kinds of relations''. Figure 12 outlines the essential concepts within TOSCA and their respective relation:

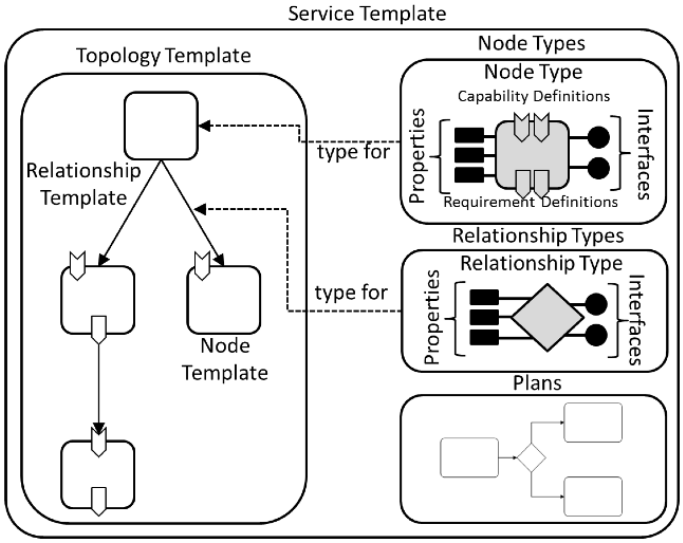

Figure 12. Main TOSCA concepts and their relations [164].

Currently, the working group around TOSCA is focusing on the following key activities:

(a) ***Interoperation and Industrial Adoption*** - this activity is pursued by working on two fronts: (1) from a more ‘‘soft’’ perspective, the group has enacted a fine-grained demoing strategy involving conferences and practitioner events (e.g. Open Source CONvention OSCON 2015 [165]); (2) from a more technical side, the group is working on providing additional constructs needed within industrial practice and ad-hoc node specifications (e.g., ‘compute’ and ‘store’ nodes, etc.). Most of the above activities are carried out by the TOSCA-TC SubCommittee (SC) on interoperability [166]. Currently the SC is concentrating on conducting adoption, interoperation and validation experiments with industry partners.

(b) ***Network Function Virtualisation (NFV)*** - this activity is pursued by defining concepts and relations connected to Software-Defined Networking and ad-hoc TOSCA constructs that may be compatible with said technology. Also, in the scope of these activities the TOSCA-TC is working to define ad-hoc design patterns (e.g. ad-hoc required and provided properties within topology templates) that match the reasoning and logical assumptions behind NFV.

(c) ***Language Simplification*** - this activity is pursued by incrementally refining a simplified TOSCA re-interpretation in YAML (YAML Ain’t Mark-up Language) [167] - a human-readable mark-up language capable of sensibly reducing the learning curve behind TOSCA. The YAML specification is currently at revision 15 and has not reached stability yet.

(d) ***TOSCA Marketing and Dissemination*** - in the scope of dissemination TOSCA has defined a specific sub-committee was entrusted with the investigation of containment technologies (e.g. Docker, see Section B.7.4) and their relation to TOSCA, with particular focus on how such technology can be used to improve TOSCA notations and their value proposition. TOSCA defines its own Cloud Service ARchive (CSAR) containment technology in an abstract way, e.g. to encompass Docker or similar technological solutions. In this regard, TOSCA can be said to profile current containment technology. This group is interoperating strongly with an ad-hoc containment group taking care of the CSAR area of TOSCA.

In the scope of the definition of the DICE profile, TOSCA and TOSCA-ready specifications play a key role in enabling designers at the DDSM level to realise seamless auto-generation of TOSCA-ready templates. These templates can be in turn deployed on TOSCA-enabled orchestration engines (see Section B.7).

B.7. Deployment tools

B.7.1 Overview

Setting up virtual machines and bare-metal machines, management of network connectivity, installing and configuring software are inevitable parts of operating with computation centres and clouds. Performing these tasks manually is an involved and error-prone activity, which also scales badly. Over the decades the IT industry advanced far enough to enable most if not all of these activities to be performed automatically through scripting.

Scripting has an advantage of making all processes repeatable. However, scripted solutions are only applicable to the platforms (operating systems, kernel versions, installed libraries etc.) that they have originally been built for. To abstract specifics of operating systems and their distributions, tools exist, which offer the configuration described in a Domain-Specific Language (DSL). This frees the user to focus on describing the nodes leaving the distribution-specific operations to the tool’s drivers (also called *providers* in some of the tools).

B.7.2 Configuration management

The configuration management tools embody the automation, normally focusing on management of existing nodes. They run stand-alone on the node that they need to configure or use a central service to obtain the instructions on the node configuration. The supported DSL enables expressing common operations (e.g. install a package, create a file, run a programme) in a simple way, while at the same time it is powerful enough to build and maintain complex applications running across one or more clusters. The popular representatives are the following:

- Puppet – configuration management solution [48]
- Chef – automation tool [49]
- SaltStack [168]
- Ansible – automation tool for application deployment [169]
- Vagrant – tool for configuring development environments [170]

These tools have reached maturity, and they also have a large community of developers providing support and development of the base tools. The community of those supplying the recipes is even wider, thus the public repositories contain at least one Chef recipe for each of the core DICE building blocks, which are freely available for installing in private data centres.

B.7.3 Orchestration

The orchestration in computing [171] represents the next level of the automation. It manages and connects services into applications and building workflows. It employs automation provided by configuration tools, but it runs tasks in a particular order or at particular events, e.g., when a database engine starts and becomes available, it configures all the depending services and runs them. This is important for applications, which normally run in multiple tiers. The orchestration tools have a notion of the application's topology. They also play an important role when scaling parts of the application up or down, and when healing the application after faults.

DICE deployment tools act very much like orchestration tools, receiving a TOSCA model at the input and deploying the application based on the contents in TOSCA. Here are some of the more prominent and popular representatives dealing with orchestration.

B.7.3.1. Ubuntu Juju

Juju is the offering for orchestration by Canonical [172].. It introduces and uses Juju charms [173] to represent and handle individual nodes and services. It provides a rich command-line interface, mostly mirrored by the functionality in its web interface console. The charms are structured bundles of files, composed of YAML files [167] to describe the charms and specify actions, scripts and other files required in the charm. The automation in the charms revolves around *hooks* and *actions*. Hooks define what happens when a service installs, starts, stops, changes configuration or gets upgraded. Additional hooks handle joining, changing, departing and breaking of relations with other charms. Actions further extend the functionality of a charm's instance.

A bundle is a YAML file describing the whole application, listing all the charms, their relationships and configuration values.

Charms are available for the core Big Data services, which Canonical openly promotes [174]. A growing number of other data-related charms exist for other services as well.

B.7.3.2. Cloudify

Cloudify is a cloud application orchestrator [175] supporting a wide variety of platforms and infrastructures. It accepts blueprints formatted in a TOSCA-compliant YAML [167].

Because of its reliance on Chef, Puppet and other configuration tools, the Big Data building blocks are well supported. The author's own words are that their original goal was 'to make Big Data deployments a first-class citizen within Cloudify' [176]. Cloudify seems as a consolidated technology with strong points in modelling and deployment of cloud specs. These might come in handy during DICE, especially in terms of their usage as a case-study or TOSCA usage scenario.

B.7.3.3. Alien4Cloud

Application LIfecycle ENablement for Cloud [177] aims to help in designing applications and collaboration during the design by employing the TOSCA YAML documents for describing applications. They offer a Graphical User Interface (GUI) for managing blueprints (called topologies) and monitoring deployment progress. One of the core functionalities behind this technology is, quoting from the home site: 'Create or reuse portable TOSCA blueprints and components. Leverage your existing shell, Chef or Puppet scripts' [177]. These features suggest that Alien4Cloud may easily be integrated in the various methodological and technological phases intended in the definition and application of the DICE profile. The tool relies on third party tools such as Cloudify [175] for the actual deployment. The project is in the early stages of the initial releases.

B.7.3.4. Apache Brooklyn

The Apache Brooklyn [178] is another project using blueprints for describing applications. Its aim at the orchestration is to enable also the application's runtime management with the ability to express SLA-like policies and associated actions. The policies rely on the metrics from the monitoring services, which Brooklyn supports. The actions can only be expressed in Java Virtual Machine (JVM). Currently Brooklyn supports OASIS CAMP [179], but they also plan to support TOSCA.

B.7.3.5. Flexiant Cloud Orchestrator

Flexiant Cloud Orchestrator (FCO) [180] is a world-leading Cloud Orchestration Software solution. FCO provides service providers the ability to design, create and manage their own virtual public, private or hybrid cloud solutions.

With Flexiant Cloud Orchestrator, a data centre operator can manage an entire cloud solution, from hardware, network and storage management through to metering, billing & customer/end-user self-service. The FCO uses Chef recipes [49] for managing the configurations, and therefore supply the Big Data building blocks.

B.7.3.6. Rundeck

Rundeck [181] is an open source software that is designed to automate operational procedures in cloud environments. Rundeck allows tasks to be run on any number of nodes/Virtual Machines (VMs) using a GUI or command line interface. It does this by working in tandem with solutions such as Chef [49] or Puppet [48] and acts as a command and control portal that lets users execute commands using features like node filtering and parallel execution.

B.7.3.7. CAMF

CAMF [182] focuses on three distinct management operations, particularly application description, application deployment and application monitoring. To this end, it adopts the OASIS TOSCA open specification for blueprinting and packaging Cloud Applications. Being the part of the Eclipse Software

Foundation, part of the CAMF code will be made freely available and open source under Eclipse Public License v1.0 [84].

B.7.3.8. CELAR

A relevant background is also the CELAR project [183] and the related tool-support within Eclipse, i.e., c-Eclipse. The CELAR project is an initiative specific for multi-cloud elasticity provisioning. In realising said elasticity provisioning services, CELAR and connected tool-bases are working to implement and gradually extend a deployment engine featuring specific TOSCA templates. As part of the Eclipse ecosystem the complete source code of c-Eclipse is made available under the terms of the Eclipse Public License v1.0 [84]. Similarly, a part of the CELAR project code responsible for automated deployment will also be made freely available as well.

B.7.3.9. Open-TOSCA

Open-TOSCA is an open source initiative from the university of Stuttgart to develop open source TOSCA modelling/reasoning and orchestration technologies including support for modelling via the Winery modelling technology [184] as well as TOSCA containment modelling via an ad-hoc OpenTOSCA Container and instantiation via the VinoThek self-service instantiation portal [185]. Because it is composed of a set of technologies, Open-TOSCA does not have a clear and homogeneous open source licensing model as a single product. Rather, individual licensing has to be evaluated for the single modules it is made of.

B.7.3.10. Tools analysis

Table 10 provides comparative summary of deployment orchestration tools described in detail in the sections B.7.3.1-B.7.3.9.

Table 10: Comparative summary of deployment orchestration tools.

Tool name	License	Input format	Configuration support	Native cloud support	First release	Latest release
Ubuntu Juju [172]	AGPL [186]	Command line, YAML [167]	Juju Charms [173] - hooks and actions run executables in target environment	OpenStack, AWS [192], ...	at least 2 years ago	May 2015 (recent)
Cloudify [175]	Apache License License 2.0 [187], UI commercial (currently free)	TOSCA YAML	Scripts, Chef [49], Puppet [48] , SaltStack [168], OpenStack API [188], CloudStack [189], custom plugins	OpenStack, SoftLayer [193], Apache CloudStack, VMware vSphere [194] and vCloudAir [195]; plug-ins for AWS,...	February 2012 (3 years ago)	December 2014 (4 months ago)
Alien4Cloud [177]	Apache License 2.0	interactive GUI; YAML internally (via Cloudify [175])	see Cloudify	see Cloudify	Q1 2015 (in progress)	
Apache Brooklyn	Apache License 2.0	CAMP-compliant YAML[167] + JVM plug-	Chef, SaltStack, scripts	'many supported', leverages Apache jclouds [196]	January 2013 (2 years ago)	December 2014 (4 months ago) - version

[178]		ins, RESTful API				0.7.0
Flexiant Cloud Orchestrator [180]	FCO custom license	GUI, SOAP/REST API	Chef, FCO Blueprints [190], FCO Triggers [191]	FCO -Hypervisors supported Virtuozzo Xen 4 KVM VMware vSphere Hyper-V 2012 [197]	2007	April 2015 (recent)
Rundeck [181]	Apache License 2.0	GUI, Command Line	Chef, Puppet, Jenkins (see Section B.8.2)	Using configuration frameworks any cloud platform can use.	2010	April 22 2015 V2.5
CAMF [182]	Eclipse Public License 1.0 [84]	TOSCA XML (GUI in Eclipse)	Chef	'many supported', leverages Apache Java Multi-Cloud Toolkit (jclouds) [196]	under development, started 2013	not released yet
CELAR [183]	Apache License 1.0	TOSCA	Unknown [183]	CELAR Server-based (for the moment) but environment is under definition	under development, started 2013	not released yet
Open-TOSCA [184], [185]	No license	TOSCA	Implementation Artefacts Engine supporting plugins	OpenStack, AWS	April 2015 (1 month ago)	April 2015 (1 month ago)

B.7.4 Virtualisation and containers

Orchestration tools use description of services and applications to be deployed at a high level, but they need to handle the actual deployment and configuration also at the low level. The possibilities of where and how to instantiate the needed building blocks are only limited by the target environment's support and the drivers included in the orchestration and configuration tools. Currently the most common means of provisioning and deploying resources is to create and use virtual machines in an IaaS environment. Virtualisation enablers are the hypervisors such as KVM, Xen or VMWare [197]. Newer hypervisors aim to offer light-weight, simplified or faster execution from the traditional ones. A peer H2020 project MIKELANGELO [198] is going to produce a faster, lightweight software stack for virtualisation under widely adopted and supported OpenStack [188].

The increasingly popular alternative to virtualisation is using containers. The containers rely upon the Unix container technology which has been available for a long time, but have only recently received a good support for management and portability of the applications within containers. Docker [199] and LXC [200] are only two of the representatives. Docker in particular offers to package a service or an application in a container, creating a lower footprint on the overall execution environment and increasing the portability of the application.

The concept opposite to the virtualisation and containers is using the whole bare-metal computer as a computational unit (MaaS - Metal as a Service). This is possible through standards such as Intelligent Platform Management Interface (IPMI) [201].

In DICE we aim to support the ability to deploy the building blocks first in the more popular environment such as the virtualisation in the IaaS. For the more advanced releases, we will also look into offering an ability to deploy parts of the applications using Docker containers and the OS^{V} [202] virtualisation.

B.7.5 Summary

In DICE we are aiming to support standards, and the OASIS TOSCA is an important one. We will base our solution on one of the more stable and powerful TOSCA-compliant orchestration tool. At the time of the analysis, Cloudify [175] is a tool which has been available for several years, while its support and development are still strong. Its command-line and RESTful interface is perfect for our use, where the users never need to see the orchestrator directly. Also its reliance on jclouds [196] as the abstraction of the IaaS and the extendibility enable its potential use in a variety of cloud providers, including the Flexiant Cloud Orchestrator [180] as the dedicated testbed in the DICE project. The choice is further enforced because another solution - Alien4Cloud [177] - uses it as a basis. Alternatively, Apache Brooklyn [178] promises to add support for TOSCA in the upcoming months, making it also worth considering.

For the low-level configuration management, we plan to use Chef [49]. The competition at this level is high, but Chef stands out because the DICE developers have a higher familiarity with Chef, and there is a potential for reusing pre-existing cookbooks and recipes.

B.8. Continuous integration tools

The software engineering practice where the developers merge their development changes into the shared mainline daily or even more frequently has been named Continuous Integration (CI) [203], [204]. The practice enables both a higher rate of software releases as well as a greater confidence in the quality of the produced code. The latter also depends on a well-built development and testing environment, and on the developers adhering to the test-driven approaches of the development. The overall idea is that the developers perform code and project validation with every change, working towards the code which does what is expected from it, while at the same time it does not break any other parts of the application. By often integrating changes introduced concurrently by different members of the team, the developers find and resolve conflicts while they are smaller and easier to resolve. continuous integration is therefore an important element of the Application Lifecycle Management and thus crucial to the DevOps ecosystem.

The code validation takes many forms, from preparing and running unit tests at the small scale to integration tests to test multiple modules, services and even systems. This can easily include assessment of the quality of each deployment, giving both a binary response (pass/fail) as well as softer ones to help drive the development and the improvements. In DICE we have set out to offer the tools to perform such evaluation, and while it will be possible to execute them manually, the native way of using them in DICE will be via the continuous integration.

The tools implementing the continuous integration handle one or more jobs, each of which can be triggered manually, periodically or by some service or another job. The jobs typically follow a pattern of obtaining the latest code changes from a Version Control System (VCS) such as Git (see Section B.9.1.3) or Subversion (SVN) (see Section B.9.1.2), saving them in their local space. Then they attempt to build the project and save the output of the successful builds (binaries, executables, libraries etc.) in an artefact repository. Finally, they run any verification provided by the developers with the code, indicating whether an application functions as expected or fails at any point.

The process pipelinedescribed above issues and indication when the project baseline breaks. Certain early criticisms of the continuous integration approach [205] pointed out that this indication only happens after the fact, halting the development process for everyone involved. The side effect of this is that the developers approach commits under stress and fear of breaking the build, while at the extreme case the developers start to ignore the failure notifications from the continuous integration tools. These problems can largely be avoided by employing code review steps where the tools verify the merged commit before the actual merge.

The continuous integration jobs can also perform fully custom actions and steps, so any project team is free to follow their own patterns and use their own tools when building and testing their application.

Here we review the tools with active development and recent latest releases. A more comprehensive overview is available at the ThoughtWorks website [206], but the information there is partially outdated.

B.8.1 TeamCity

TeamCity is a part of the JetBrains' commercial offering in their Teamware suite [207]. It natively supports Java, .NET [71], Ruby [103] and XCode [208] languages and environments. Many other languages are supported via plug-ins.

TeamCity supports a remote run feature which performs a check of a build before it is committed into the baseline branch. The feature works from a TeamCity plug-in in Eclipse or other IDE and does not involve the branches in VCS.

The tool also natively supports many build agents, code coverage tools and a code change inspector for the Java code projects. Build agents can run distributed in various locations and host environments to address load distribution and specific requirements of multi-platform builds.

B.8.2 Hudson / Jenkins

Jenkins [209] is a highly popular open source CI tool. It is a fork of the Hudson [210] tool, and both still exist and are actively developed, although Jenkins reportedly has a larger developer community and a higher installation base [211]. Originally the tool had a high level of support mostly for the Java-centric projects. However, currently a wide selection of plug-ins provide support for other types of projects as well (e.g. Python [128], Ruby [103] etc.). The plug-ins can extend the default functionality for the majority of aspects of the CI jobs, including the build steps and the means of sending job result notifications. It is possible to publish the plug-ins in a Jenkins plug-in directory.

Both Hudson and Jenkins are created in Java [99]. Therefore, they can be installed in any operating system or Linux distribution. Each installation can be a fully-featured installation or a small slave agent installation used in a distributed build agent set-up.

B.8.3 Atlassian Bamboo

Atlassian's Bamboo [212] is a commercial offering which works as a standalone service, hosted in the Atlassian's cloud or installed in the customer-controlled environment. It natively includes integration with the JIRA issue tracking service [213], and the Stash code versioning management system [214]. Bamboo emphasises the support for continuous delivery, providing releases in multiple environments. The commercial nature of the project is also evident in solutions easing the continuous integration management with grouping jobs in chained stages, simplified management of distributed version control systems' branching, as well as general support for migrating from popular open source tools such as Jenkins [209]. Nevertheless, it offers RESTful API to support custom add-ons.

B.8.4 Go

An open source continuous integration and delivery tool from ThoughtWorks called Go [215] focuses on handling complex pipelines, chaining jobs according to their dependencies in a directed graph with possible fan-out and fan-in. Naturally it also supports integration with other ThoughtWork projects such as the agile project management tool Mingle [216].

B.8.5 Strider CD

The Strider CD [217] is a relatively recent solution for continuous integration and Continuous Development. It is built on lightweight technology such as node.js [218] and promises a high level of customisability by supporting the plug-ins. The authors focus on improved user experience and automation. VCS branches can have different jobs attached to them.

B.8.6 BuildBot

The BuildBot [219] is a framework for continuous integration emphasising the flexibility and providing the tools for complex projects which mix technologies and languages. The jobs are configured using Python scripts [128]. This means they can be simple, but, if needed, provide the ability to dynamically configure builds and jobs. In the complex software projects it is possible to use a concept of source stamping to include dependencies from various VCS projects, and the built-in versioning system helps manage the dependencies.

B.8.7 CircleCI

The CircleCI [220] is a hosted environment for continuous integration. It advertises the speed of the job execution. This is achieved by the timing of the builds and their subsequent distribution into parallel builds based on this timing. Each job gets its own Docker environment [199], so each build is clean and independent of any previous builds. Natively it also provides support for virtual graphical frame buffer using Xvbf [221] in Linux. Table 11 provides comparative summary of continuous integration tools described in details in the sections B.8.1-B.8.7.

Table 11: Comparative summary of continuous integration tools.

Tool name	License	Eclipse plug-in	Supports custom plug-ins	Job control	First release	Latest release
TeamCity [207]	commercial with a limited free license	Yes	Yes	Web GUI, RESTful	July 2006 (9 years ago)	December 2014 (6 months ago)
Hudson [210]	Eclipse EPL license [84]	Yes	Yes	Web GUI, RESTful	Summer 2004 (11 years ago)	January 2015 (4 months ago)
Jenkins [209]	MIT License [222]	Yes	Yes	Web GUI, RESTful	February 2011 (4 years ago) - forked from Hudson	May 2015 (recent)
Atlassian Bamboo [212]	commercial	Yes	Yes	Web GUI, RESTful	February 2007 (8 years ago)	November 2011 (6 months ago)
CircleCI [220]	commercial, hosted only	No	No	Web GUI, RESTful	unknown	recent
ThougtWorks Go [215]	Apache License 2.0 [187]	N/A	Yes	Web GUI, RESTful	7 years ago	April 2015 (1 month ago)

Strider CD [217]	BSD License [223]	No	Yes	Web GUI, RESTful	July 2013 (2 years ago)	March 2015 (3 months ago)
BuildBot [219]	GPL 2	No	Yes	Web GUI, RESTful	March 2006 (9 years ago)	April 2015 (2 months ago)

B.8.8 Summary

DICE DevOps support tools naturally aim to support the continuous integration and Continuous Deployment. The tools used will represent the glue between the development and simulation work on the one side, and automated deployment and application's execution on the other side. The selection is wide and strong, but DICE needs to offer an open source solution. Of the ones that comply with this requirement and have also a strong community, active support and a high level of adoption, Jenkins [209] is certainly at the top. It is also the solution favoured by many DICE partners for their own internal or collaborative projects. As a good alternative, BuildBot [219] also offers a good support for highly customised solutions.

The commercial solutions will likely serve as an inspiration in terms of the features to consider and support in the DICE continuous integration. For instance, the CircleCI's ability to provide headless testing of graphical interfaces (as required by the Selenium library (see Section C.11.2.3) for testing the web GUI applications) may become useful for providing quality tests.

B.9. Versioning of software engineering artefacts.

The purpose of versioning is to map a complex system of software components to a commonly and easily understood name or number. This helps the users understand which functionality to expect from a certain component's version. Considering that few systems operate on their own, the versioning also helps in defining which components are compatible and possible to co-operate. The actual version assignments are ultimately the responsibility of the developers and their project leaders. No system can perform version assignment in a fully automated way. It is also highly dependent on the purpose and type of software being versioned [224]. However, it is a common and recommended practice to use a system which automatically assigns revisions to each change.

DevOps methodology advocates DevOps teams to version everything in their environment: application code, infrastructure, configuration, data, and internal system artefacts. The major aspect of the systems providing the versioning control is that they provide history of changes in the code and the ability for the collaborating teams to obtain a consistent view of the whole project at any time. In combination with continuous integration it is possible to also always be able to obtain a tested and stable version from the change history.

B.9.1 Classical versioning tools

The versioning control systems - more accurately named revision control systems or source control systems [225] - are roughly divided into two categories: centralised systems (represented by the CVS [226] and the Subversion [227]) and distributed systems (Git [228] and Mercurial [231]).

B.9.1.1. CVS

The Concurrent Versions System [226] is one of the earlier representatives of the centralised source control system. It serves the basic purpose of keeping the change history, but its method of checking in single files makes it unsuitable for modern DevOps.

B.9.1.2. Subversion

Subversion or SVN [227] is a software versioning tool from Apache distributed as free software under the Apache License 2.0 [187]. It is mainly used for revision control of source code but also for any kind of

files, such as documentation in binary format, archives, links, etc. It is a popular system, one of the advantages being that its use is not overly complex. Revisions compose changes of one or more files in the project, and the Subversion assigns each revision a unique non-decreasing number. The users can also assign tags of arbitrary unique names to particular revisions, for instance to mark stable versions.

B.9.1.3. Git

Git [228] differs from SVN being a distributed control version system. It is used in very large projects thanks to easier branch operations, a full history tree available offline, and a distributed peer-to-peer model. Currently, it is one of the most widely used version control systems for software development.

Each Git revision receives a commit ID, an alphanumeric string the length of 40 characters which contains a hexadecimal representation of the SHA1 [229] digest of the revision. This commit ID reliably marks the revision and its history, which is crucial for the distributed nature of the whole system. However, the ID cannot work well as a part of the version number like the Subversion's revision number could. Therefore, the developers need to keep their own custom revision numbers, e.g. by storing it in a file in the project and having it auto-increment using a repository hook [230].

B.9.1.4. Mercurial

Mercurial [231] is also a distributed revision control tool like Git [228]. It is also used in major developments like OpenOffice.org [232]. Its main difference - with respect to other revision control systems - is that Mercurial is primarily implemented in Python [128], as opposed to C, and is easier to learn than other distributed versioning system.

B.9.2 DICE needs with respect to versioning

An important issue in DICE is the ability to associate to each UML model element a unique version number, which is consistently propagated through the DICE toolchain. For example, if one splits a component in two, the monitoring and enhancement tools should detect that the monitoring data acquired before this change is probably no longer representative of the application at hand.

It is generally best if any DIA component has a version number stored in some component configuration so that it is available to all the tools used in the chain. The continuous integration tool could increment the version number upon success, but usually a build number of a kind comes from the Git [228] or any other version control system. But otherwise it should be up to the human users to say which version they want the current build to have.

Since history of versions is important and results of the current version build depend on previous version builds due to the enhancement tools, the DICE tools - with the help of the continuous integration tool - could always preserve the latest results and the best results in any version. But since each result has to be bound to a specific commit to the version control system, the best results can only serve as a kind of a reference, while only the latest results actually matter. Existing tools to help automatically increase version numbers depend on the principal language used in the application. For example:

- Versions plug-in for Maven [233] helps manage the versions of the artefacts in a Project's Object Model (POM);
- Python-versioneer [234] provides a number of formats and styles for generating the new version number from any past version numbers used in the VCS;
- Dist::Zilla [235] manages the version increments for the libraries written in the Perl.

B.10. Discussion

In this chapter we gave an overview of a large set of approaches and tools that are related both to development and operation and, in most cases, have been developed in a completely independent way by research and practitioners.

Referring to Figure 13 that shows the DICE high level vision, this state of the art analysis has been drawn within the context of DevOps (Section B.2) that is the movement within which DICE aims at operating. The DICE methodology (on the left hand side of the figure) will be defined keeping the DevOps principles in mind. Moreover, it will be based on Model-Driven Engineering (Section B.3) and will take its roots from the existing model-driven approaches focusing on cloud and DIA applications. The IDE (the box in the Figure 13 that encloses four different tools) will be developed starting with modelling tools such as Papyrus [83] and MOSKitt [97]. These tools allow us to let DICE users exploiting the DICE profile - which will be built as part of the MARTE profile [68] - model DIAs and interact through proper connectors using tools that support simulation, optimisation, creation of deployment recipes and analysis of testing results. Model to model transformations (Section B.4) will be used to support the generation of different views on a DIA model. Such views can either be used to support the transition from high level design of a DIA to the selection of Data-Intensive technologies and to the deployment on the cloud, or they can be used to transform models in a way that is suitable for specific analyses and simulation activities. TOSCA (Section B.5) will be used as an output format representing the deployable model of a DIA. Such format will be provided as input to the deployment and management tool that will be based on those reviewed in Section B.6. Finally, the continuous integration tools (Section B.7) will be used as the basis to support the DICE integration phase, keeping in mind that in our approach the emphasis is not only on integrating, building and deploying code but also on managing and continuously evolving model. Of course, models' and components' evolution results in the need for keeping track of different versions and of the relationships between the various components versions. An analysis of the literature in this field is provided in Section B.9 and is strictly correlated to the continuous integration approaches of Section B.8.

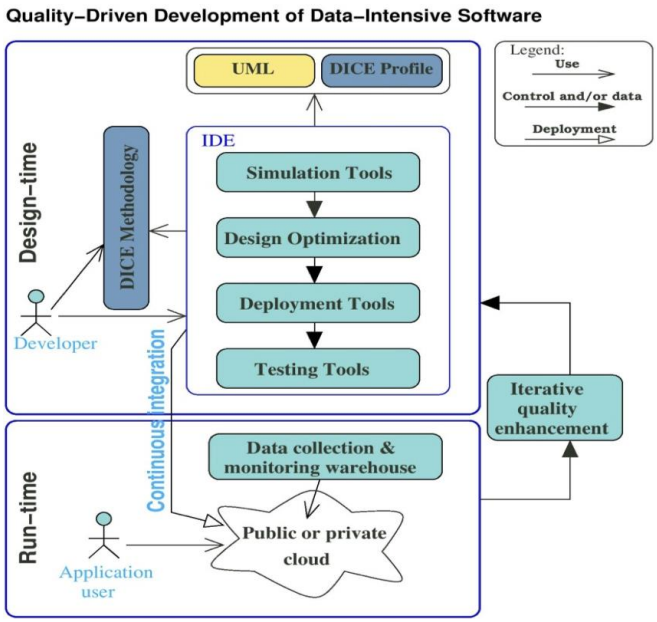

Figure 13. DICE high level vision [237].

C. Quality Assurance

C.1. Non-Functional Properties in DevOps

In the previous chapter we have examined approaches to design and implementation of DIA and the underpinning development toolchain functional properties. In this chapter we investigate non-functional properties. In the DICE vision non-functional characteristics of the software follow the definition of the ISO/IEC standards and may be summarised at a high-level as follows:

- **Reliability**: The capability of a software product to maintain a specified level of performance, including Availability and Fault tolerance.
- **Performance**: The capability of a software product to provide appropriate performance, relative to the amount of resources used, as described by time behaviour and resource utilisation. **The terms performance and efficiency are considered synonyms throughout.**
- **Safety**: The capability of a software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment.

The project will interface the DICE profile to existing reliability and performance tools, which will be enhanced to be able to analyse behaviour related to data. Further, the project will also encompass verification and trace checking capabilities to support safety analysis.

In this chapter we survey tools and methods for quality analysis. We emphasise two dimensions:

- **Design tools and methods**. The annotation and analysis of quality requirements in UML [67].
- **Enhancement tools and methods**. The combination of monitoring, testing and feedback analysis techniques to improve the design and quality analysis of application prototypes.

In more details, the contents of this chapter are as follows:

- First, we introduce definitions of reliability, performance, safety properties and metrics that are relevant to the analysis. **Service-Level Agreements (SLAs)** are readily determined by predicating on these quantities. We illustrate various notions of predicates in the safety properties sections, which can be readily used also to qualify reliability and performance constraints arising in SLAs (e.g., the 95th percentile of end-to-end response times should be smaller than a certain amount). **SLAs can be directly annotated in the UML models**, thus we do not look at other forms of specification (e.g. Web Services (WS) standards [238]).
- Next, we overview **UML annotations** to specify reliability, performance, and safety properties. We emphasise two existing profiles as baselines: UML MARTE [68] and UML DAM [69]. We also overview UML extensions for access control specification and techniques to optimise deployment plans generated from such UML specifications.
- Finally, we introduce Model-Driven **Testing** (MDT) tools and complementary monitoring tools to retrieve information about application behaviour during test, followed by methods to analyse the **monitoring** data acquired, in particular trace checking, feedback analysis, and anomaly detection via machine learning.

In this section, we review the definition of several metrics that will be examined as part of DICE performance, reliability and safety predictions. Performance and reliability are the focus of simulation and optimisation tools, whereas safety predictions will be performed by verifying properties through model checking.

C.2. Performance Metrics

In this section, we review some basic definitions concerning performance prediction metrics. These are standard definitions that are usually used in the context of queueing models, but are also applicable to performance predictions obtained with other formalisms, such as Stochastic Petri Nets (see Section C.5.2.1). We give definitions for the basic case where requests are considered of a single type (i.e. a single class model). The generalisation for multiple types is simple and in most cases requires only adding an index to each metric to indicate the class of requests it refers to [239].

Considering an abstract system where *T* is the length of time we observe a system, *A* is the number of arrivals into the system, and *C* is the number of completions (users leaving the system) we can define the commonly used measures [239]:

- **Arrival rate**: $A = T$
- **Throughput**: $X = C = T$ (simply the rate of request completions)

In a system with a single resource we can measure *Bk*, and denote *Ck* the number of arrivals at the resource. If *Tk* is the time the resource *k* was observed to be busy, we can then define two additional measures:

- **Utilisation**, $Uk = Bk = Tk$ (this is normally given as a percentage, e.g. 30% server utilisation)
- **Service time** per request, $Sk = Bk = Ck$

From these measures we can derive the Utilisation Law as $U = XS$. The above quantities can be made specific to a given resource, for example *Uk* stands for the utilisation of resource k.

One of the most useful fundamental laws is Little's Law [239] which states that *N*, the average number of customers in a system, is equal to the product of *X*, the throughput of the system, and *R*, the average **response time** a customer stays in the system. Formally this gives: $N = XR$. The formula is unchanged if one considers a *closed* model, which has a fixed population *N* of customers. Another fundamental law of queueing systems is the Forced Flow Law [239] which, informally, states that the throughputs in all parts of the system must be proportional to one another. Formally the Forced Flow Law is given by: $Xk = Vk\ X$, where *Xk* is the throughput at resource k, *Vk* is the **visit count** of resource *k*, i.e., the mean number of times users hit this resource. Combining both Little's Law and the Forced Flow Law allows for a wide range of scenarios to be analysed and solved for a particular desired quantity. For example, it is possible to compute utilisation at a server *k* in a distributed network directly as [239] $Uk=X\ Dk$ where $Dk=VkSk$ is called the **service demand** at server k.

C.3. Reliability Metrics

The area of reliability prediction is established and focuses on determining the value for a number of standard metrics, which we review below.

The execution time or calendar time is appropriate to define the **reliability** as $R(t)=Prob(\boldsymbol{\tau}>t)$ that is, reliability at time *t* is the probability that the time to failure **τ** is greater than *t* or, the probability that the system is functioning correctly during the time interval *(0,t]*.

Considering that $F(t)= 1-R(t)$ (i.e., **unreliability**) is a probability distribution function, we can calculate the expectation of the random variable **τ** as $\int_0^\infty t\ dF(t) = \int_0^\infty R(t)dt$. This is called Mean Time to Failure (**MTTF)** [240] and represents the expected time until the next failure will be observed.

The **failure rate** (called also rate of occurrence of failures) represents the probability that a component fails between *(t,dt)*, assuming that it has survived until the instant *t*, and is defined as a function of *R(t)*: $h(t) = -\frac{1}{R(t)}\frac{dR(t)}{dt}$. The **cumulative failure** function denotes the average cumulative failures associated with each point in time, *E[N(t)]*.

Maintainability is measured by the probability that the **time to repair** (***θ***) falls into the interval *(0,t]* [240] *M(t) = Prob { θ ≤ t }*

Similarly, we can calculate the expectation of the random variable ***θ*** as $\int_0^\infty t\, dM(t)$, that is called **MTTR** (Mean Time To Repair), and the **repair rate** as $\frac{dM(t)}{dt}\frac{1}{1-M(t)}$.

A key reliability measure for systems that can be repaired or restored is the **MTBF** (Mean Time Between Failures) [240], that is the expected time between two successive failures of a system. The system/service **reliability on-demand** is the probability of success of the service when requested. When the average time to complete a service is known, then it might be possible to convert between MTBF and reliability on-demand.

Availability is defined as the probability that the system is functioning correctly at a given instant *A(t)=Prob{state=UP,time=t}*. In particular, the **steady state availability** can be expressed as function of MTTF and MTTR (or MTBF):

$$Availability_\infty = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

C.4. Safety Properties

Safety properties vary from system to system and require formalisation in terms of logical expressions that can be verified via formal methods. The general idea of formal verification is the following: given a formal model of the system to be analysed (let us call this model S), and given a property (a requirement), also formalised in some suitable way (let us call this property R), we check whether R holds for S or not (in symbols, we write this as S |= R).

There are many possible formalisms for describing S and R; S and R themselves might be described using different formalisms [241], [242].

A (non exhaustive) list of formalisms for describing the system S are:

- Finite State (Büchi) Automata
- Timed Automata
- Probabilistic Automata (in the broad sense of the term, including Discrete-time Markov Chains, Continuous-Time Markov Chains, Markov Decision Processes, etc.)
- Temporal logics (of many different kinds: with/without metric on time, discrete/continuous-time, etc.).

Typically (though not always) the requirement R is described through a logic. Examples of logics that can be used are the following:

- Linear Temporal Logic (LTL)
- Metric Temporal Logic (MTL), and the similar TRIO
- Computation Tree Logic (CTL)
- Probabilistic CTL (PCTL)
- Continuous Stochastic Logic (CSL)

The kinds of properties that can be expressed and verified depend on the nature of the formalisms used to describe S and R. Some categories of properties that can be expressed are the following:

- **Ordering properties**

For example, a safety statement may require that: 'Every occurrence of *b* must be followed eventually by *a*'

  - A possible (though not the only one) formalisation of this property in LTL is the following:
    $G(b \rightarrow F(a))$

- **Metric properties**

Example: 'Every occurrence of *b* must be followed eventually by *a* within 10 time units'

  - Possible formalisation of this property in MTL:
    $G(b \rightarrow F_{[0,10]}(a))$

- **Invariants/safety properties**

Example: 'the system will never enter state *critical*'

  - Possible formalisation in LTL:
    $G(\neg critical)$

- **Branching-time properties** (i.e., properties of executions)

Example: 'Any time the system is in state *critical*, it can eventually recover to state *normal*'

  - Possible formalisation in CTL:
    $AG(critical \rightarrow EF(normal))$

- **Probabilistic properties**

Example 'The probability of eventually reaching state critical from state *normal* without passing from state *warning* is less than 0.1'

  - Possible formalisation in PCTL:
    $normal \rightarrow P_{\leq 0.1}\ (\neg warning\ U\ critical)$

- **Real-time probabilistic properties**

Example 'The probability of eventually reaching state critical from state *normal* without passing from state *warning* and within 5 steps is less than 0.1'

  - Possible formalisation in PCTL:
    $normal \rightarrow P_{\leq 0.1}\ (\neg warning\ U^{\leq 5}\ critical)$

Over the years, several works have identified 'patterns' with which requirements have been defined by users in various domains. Examples of these works are [243] for LTL specifications, [244] for real-time patterns, [245] for probabilistic properties, and [246] for properties of service-based applications. These patterns might be the basis for the definition of a user-friendly interface through which DICE users can introduce the properties to check safety without having to delve into the finer details of expressing them in logic terms.

C.5. Quantitative analysis for assessment of performance, reliability and safety

Research in reliability and performance analysis has led to a variety of modelling techniques, each focusing on particular levels of abstraction and/or system characteristics. Such modelling techniques are used for predictions of reliability/performance properties, often rely on stochastic assumptions, and they help in answering questions that are more pressing to the engineers (e.g. is the system scalable? Will the system provide a specified response time? Is the system able to tolerate N simultaneous equipment/component failures? When a shutdown occurs, how long does it take to recover the system? Is the system able to provide the service during a given time period? etc.).

It is worth noting that, depending on the system characteristics and on the type of measure to be evaluated, the analysis can be carried out under steady state or transient assumptions [247], [248]. Steady state analysis assumes that the system has reached a stable state, so the estimated measures (e.g., mean response time of the service, steady state availability) are not influenced by the considered system initial state. On the contrary, transient analysis assumes that the system has not yet reached a stable state and the estimated measures depend on the choice of the initial state. The latter type of analysis is much more challenging and more realistic when the system may take long time to reach a stable state. It should be also carried out when the measures of interest are functions of time, e.g., the reliability function (R=Prob{ Time To Failure > t}) and instantaneous availability (A=Prob{'System is up at time t'}).

Herein, we review the main modelling and analysis techniques according to the following classification: combinatorial, state-based and Monte Carlo simulation. Combinatorial techniques do not enumerate all possible system states to obtain a solution, they are simpler than the state-based ones. However, they do not easily capture stochastic dependences. Hence they can be used for a quick, rough-cut analysis. Steady-state techniques are more comprehensive than combinatorial ones, since they allow to model explicitly complex relationships, including sequencing information. Monte Carlo simulation techniques are alternative solutions to state-based ones when the state space of the modelled system is too large to be analysed as a whole.

C.5.1 Combinatorial techniques

Reliability Block Diagrams (RBD) [249] and Fault Trees (FT) [250], used for reliability and availability analysis, belong to this category.

RBD [249] is a graphical representation of the system components and connectors. An RBD consists of blocks and lines: the blocks represent system components and the lines describe the connections between components. If any path through the system is successful, then the system succeeds, otherwise it fails. It is assumed that connectors have reliability equal to one (i.e. do not fail) and that both the system and its components are in one of two states: either up or down. Hence, in an RBD, each component can be seen as a switch that is closed when the component is up and open when the component is down. The system will be up only when a path exists between the input and output nodes. Failure/repair times of components are assumed independent random variables.

Fault Tree (FT) [250] is an acyclic graph with internal nodes that are logic gates (e.g. AND, OR, K-of-M), external nodes that represent component/sub-system faults (basic events, undeveloped events) and, possibly, transfer gates. It is a top-down technique, i.e. the FT construction begins by considering a system failure mode - the 'top event' - and terminates when either the basic events which provoke such a failure are all identified or the desired level of detail is reached. FT can be used to compute the probability of occurrence of the top event, considering the system mission time, and may correspond to the system

unreliability or unavailability computed at a given time instance[1]. Several variants of FT have been proposed to enhance the modelling capabilities, such as Dynamic Fault Trees [251] that include special purpose gates capturing sequence dependencies (functional dependency, spare and priority-AND gates).

C.5.2 State-based techniques

Markov models belong to this category: they have been extensively used for reliability, performance and performability analysis [252], [253]. The simplest Markov model is a Markov chain, which is a Markov process with a discrete state-space. For reliability and performance analysis Continuous-Time Markov Chain (CTMC), i.e. a Markov chain with time taking non-negative real values, is the reference model, and many solution algorithms exist for it [254]. One of the main drawback of CTMC numerical solution is the so-called 'state-explosion' problem, that is for complex systems with large number of components, the number of states can grow in a prohibitively manner. So, techniques aimed at reducing the size of the model have been proposed, such as state truncation methods [255], hierarchical model solutions [256], state lumping techniques [257] and aggregation techniques [258]. The latter do not produce exact solutions, only approximate values. Other types of techniques use special data structures to reduce the space requirements, such as binary and multi-valued decision diagrams [259]. Another problem that may occur when a Markov model is characterised by transition rates with very different order of magnitude is the 'stiffness' (e.g. failure rates and repair rates, failure rates and job arrival rates etc.) that may be avoided by using aggregation techniques [258] or special solvers [260].

Markov models represent the basis of higher abstraction level modelling formalisms, like Stochastic Petri Nets (SPN) [261] and Queueing Networks (QN) [239].

C.5.2.1. Stochastic Petri Nets

Stochastic Petri Nets (SPN) [261] are usually solved by deriving the underlying CTMC, which is isomorphic to its reachability graph, and then by using the wide variety of aforementioned techniques. For large complex models alternative techniques can be used to avoid the space-explosion problem, such as discrete event simulation [262] and bounding techniques [263]. SPN have been extended to include: arcs with multiplicity, immediate transitions, transition priorities, inhibitor arcs. The most popular class of SPN including such extensions is Generalised Stochastic Petri Nets (GSPN) [264]. Other extensions allow more flexible firing rules including guards and enabling functions such as Stochastic Activity Networks (SAN) [265] or Stochastic Well-Formed Nets (SWN) [266]. On the other hand, there have been several proposals on relaxing the 'exponential distribution transition firing time' assumption, making the stochastic process associated to the SPN instead of CTMC. For example, Extended Stochastic Petri Nets (ESPN) [267] are SPN where transitions can be characterised by general firing time distributions. Deterministic and Stochastic Petri Nets (DSPN) [268] have immediate, exponential and deterministic transitions. Markov Regenerative Stochastic Petri Nets (MRSPN) [269] and Concurrent Generalised Stochastic Petri Nets (C-GSPN) [270] are generalisations of DSPNs. Numerical techniques based on the solution of the underlying stochastic process have been also proposed for the aforementioned SPN variants.

C.5.2.2. Queueing networks

Some classes of Big Data technologies such as streaming systems, in-memory databases and Hadoop clusters (see Chapter D), may be effectively modelled by means of a queueing network models. A **queueing network** model is a network of resources throughout which requests are routed after spending some time at each resource. Each queue can be used to model either a physical resource (e.g., CPU, network bandwidth etc.) or a software server (e.g. admission control or connection pools). Nowadays, applications are often tiered and queueing networks can easily capture the interactions between tiers.

[1] Observe that for non repairable systems the two measures are equivalent.

Several variants of queueing network models exist, such as product-form models [271] which can be solved analytically, and extensions to model blocking, fork/join behaviour and priorities [252].

A popular class of queueing networks used to model complex software systems are **layered queueing networks** (LQNs). An LQN adds the capability of describing request workflows and organising resource hierarchically, in layers. LQN models of an application can be built automatically from software architecture models expressed using formalisms such as UML or Palladio Component Models (PCM) [154]. Several evaluation techniques exist for LQNs such as [272]-[275].

The authors in [276] provide a discussion about the advantages and disadvantages of LQNs identifying a number of key limitations for their practical use. These include, among others, difficulties in modelling caching, lack of methods to compute percentiles of response times, trade-off between accuracy and speed. Evaluation techniques for LQNs that allow the computation of response time percentiles have been presented [275]. Recently these models have been further extended to represent the operational environment in which a system operates through an abstraction called **random environment**. Finally, we mention the existence of a class of models, referred to as Q**ueueing Petri Nets (QPNs)**, which add to queueing networks the possibility to include some elements of stochastic Petri nets in the specification of the service workflow of requests. QPNs models combine benefits of both approaches, but they require evaluation via simulation due to the current lack of analytical results for this class of models.

C.5.3 Monte Carlo simulation techniques

A common use of Monte Carlo simulation is to express the system behaviour using simulation languages, e.g. [277] or general-purpose programming languages. Also, simulation can be used as solution technique for systems modelled with the high level formalisms discussed in the previous section (i.e. Stochastic Petri nets [261], Queueing Networks [239] and their extensions).

Simulation allows to estimate performance and reliability measures by generating and analysing randomly chosen paths through the state-space. For each measure of interest, the simulation provides the estimated value and the confidence interval, i.e. the real (unknown) value falls into this interval with a certain probability (i.e. confidence level). The width of the confidence interval is a measure of the accuracy of the estimated value.

Different simulation approaches exist. When the measures of interest have a transient nature, then a common approach is the replication where N statistically independent simulation runs are executed and the measure is estimated considering the N independent measurements collected in the different runs. More efficient simulation approaches are used when the aim of the analysis is the estimation of steady state measures. Indeed, in such cases just a single simulation run can be executed where the simulation time is ‘long enough’ to bring the system into the steady state, and then use measurements taken across time (rather than measurements taken across replications). For example, the method of batch [278] partitions the simulation interval in successive epochs of fixed duration and for each epoch a measurement is collected.

The computational cost of the simulation depends on various factors: 1) the length of transient period, i.e. that is the simulation time until the steady state is reached, 2) the length of the simulation time needed to collect a ‘good sample’, and 3) the length of the epochs needed to ensure statistical independence of successive measurements. It is worth to observe that such factors depend on the system under analysis, so the choice of simulation input parameters is a critical point since both the statistical quality of the simulation results and computer time that is required to simulate a desired amount of system time rely on them. Concerning the length of the epochs, a possible rigorous solution is the regenerative points method [279], where a regeneration point is a time instant at which the system enters a given state (e.g. in an M/G/1 queue model this is the state where there are no jobs in the queue).

Moreover, statistical issues may arise in case of rare events, for example when the measure to be estimated is a small probability, e.g. the system failure probability. So-called variance reduction techniques have been developed to overcome them [280].

With respect to the required computer time approximate accelerated stochastic simulation approaches have been proposed to reduce it [281].

C.6. Performance and reliability prediction tools

In this section we revise the performance and reliability tools supporting the modelling and analysis with the formalisms of interest for DICE, i.e. Stochastic Petri Net (SPN) tools and Queueing Network tools.

C.6.1 Stochastic Petri Net tools

There is a variety of modelling and evaluation tools for SPN, most of them can be found in [282]. The survey in [283] provides a ranking of those tools considering a set of criteria including: multi-platform support, open source, embedded graphical animation, structural analysis and stochastic and coloured Petri Net support.

Table 12 compares the tools that provide support to Generalised Stochastic Petri Nets (GSPN) [264] and their extensions, such as:

- Stochastic Well Formed Nets (SWN) [266] and Coloured Generalised Stochastic Petri Nets (CGSPN), that are a stochastic variants of Coloured Petri Net [284], and
- extended Deterministic and Stochastic Petri Nets (eDSPN), that are DSPN [268] characterised by transitions with deterministic, exponentially and generally distributed delays.

The first column of the Table 12 provides a list of the features we consider relevant for the DICE framework. Particular attention has been paid to the quantitative analysis support.

From the quantitative point of view GreatSPN [285] and TimeNET [286] are better than PIPE [287], considering that for reliability analysis having transient solver is a need. Observe that TimeNET supports the simulation of rare events, which are often studied in reliability/performability models. Moreover, it includes both sequential and distributed simulation modules. GreatSPN and TimeNET provide support to extended classes of GSPN (transitions with general distributions). On the other hand, the advantage of using PIPE - together with the integrated modules of PeabraiN [288] – is that it is multi-platform (Java-based). It is worth to note that PeabraiN provides two types of steady state simulator, in particular the approximated one is used to accelerate the analysis.

Table 12: Comparison of SPN tools.

Features / Tools	GreatSPN [285]	TimeNET [286]	PIPE [287]	PeabraiN [288]
Open source (license)	proprietary, free for academic institutions	proprietary, free for non commercial use	GPL	GPL
PNML Format Interchange		X	X	X
Multi-Platform		X	X	X
Net composition	X	X		
PN classes	GSPN, SWN, eDSPN	GSPN, CSPN, eDSPN	GSPN,	GSPN, CGSPN

			CGSPN	
Qualitative Analysis	X	X	X	
Quantitative Analysis				
Exact numerical analysis	steady state, transient	steady state, transient	steady state	
Approximated numerical analysis		steady state		
Simulation	steady state	steady state, transient	steady state	steady state (exact,approx)
Rare event simulation		steady state		
Bounding techniques	X			X
Probabilistic model checking	CSL-TA			

C.6.2 Queueing Network tools

A similar comparison is given below in the Table 13 for queueing network modelling tools.

Table 13: Comparison of Queueing Network tools.

Features / Tools	JMT [289]	LINE [290]	LQNS [291]
Open source (license)	GPL	BSD-3	Proprietary, Free for academic use
File Format	XML, proprietary	same as LQNS	XML, proprietary
Multi-Platform	Any Java supported	Windows, Linux	Windows
Product-form models	X	X	X
Open models	X	Approximate	Approximate
Blocking	X		X
Fork-join	X	X	X
Priorities	X		X
Random environments		X	
Workflows	Partial	Partial	Comprehensive
Service phases	Partial	X	X
Nested layers?		Transformed to single layer	X
Exact numerical analysis	steady state, transient		
Approximated numerical analysis	product-form models	steady state, transient	steady state

Percentiles analysis	X	X	
Simulation	X		X
Rare event simulation			
Bounding techniques	product-form models		

C.7. Formal analysis of safety and privacy properties

The main goal of DICE is to define a Model-Driven Engineering (MDE) [59] approach and a Quality Assurance (QA) toolchain. One of the functionalities of the DICE IDE is to offer the ability to specify the DIAs through UML models. From these models, the toolchain will guide the developer through the different phases of quality analysis, *formal verification* being one of them.

The purpose of formal verification in DICE is to assess *safety and privacy risks* of Data-Intensive Applications (DIAs). More precisely, the DIAs will be transformed into DICE models. The verification and validation of these models will be achieved by: (1) annotating the models with safety and privacy formulas, and (2) automatically translating the models into a suitable formal representation.

For many decades it was thought that formal verification is hopeless due to cryptic notations and non-scalable techniques, as well as hard to use because of the dedicated tools. Moreover, the existing non-trivial case studies were not convincing enough for the software or hardware engineers.

Around 1990s perspectives for formal verification started to look more optimistic once techniques like model checking [292] and theorem proving were adopted. Researchers and practitioners began performing more and more industrialised case studies and thereby gaining the benefits of using formal methods. Nowadays, the formal verification techniques and tools face another challenge, namely they must undergo a deep technological transition to exploit the new available architectures. These new complex architectures were imposed by the Cloud Computing (the ease of accessibility of the cloud-based computing resources) and Big Data (large amount of data which have become easily accessible) emerging technologies.

There exist in the literature only few attempts of applying formal verification in the context of DIAs. For example, [293] focuses on techniques for checking specifications expressed in a metric temporal logic with aggregating modalities (using the MapReduce programming model). These specifications correspond to quantitative metrics [294], e.g. response-time, throughput or availability. Similarly, [295] proposes an approach for the offline monitoring of DIAs. Unlike [293], it uses metric first-order temporal logic for specifying properties of the system actions to be checked, as well as a formal framework for slicing logs. Their work also relies on MapReduce model. From a different perspective, [296] uses Big Data technology (Hadoop MapReduce [14]) for parallelising and distributing model checking techniques in order to make the verification of DIAs feasible or more efficient. None of these approaches focuses on verifying qualitative properties, like safety and privacy of DIAs. An attempt similar to ours is presented in [297], aiming to model the MapReduce model with Communicating Sequential Processes (CSP) formalism [298]. The authors present a very simplistic model together with an example and, as future work, it is planned to use a dedicated model checker [299] to check safety properties of the model. No explicit safety property was verified; however, the authors mention that the model is suitable to verify properties like 'If MASTER[2] node failed, there is no component handling this exception, the system cannot recover from the error and the MapReduce programme cannot be finished successfully.' Computation Tree Logic (CTL) [300] is used

[2] A MASTER node performs scheduling and failure handling.

in [301] as underlying formalism for representing the MapReduce model and expressing its safety properties. The general purpose model checker Uppaal [302] is used to implement the model and verify properties like load balancing and fault tolerance.

An architecture and technology-oriented formalisation of DIAs is presented in [303]. The authors present a multi-formalism approach which enables the modelling of the architecture (Lambda Architecture [17]) and application logic of DIAs as well as the environment. Although their approach uses Petri Nets for modelling the application logic, the work is important for DICE because the authors present a two-layer abstraction model enabling a very precise modelling of the application, but at the same time allowing to hide the complexity when necessary. Their work could be inspirational since the DICE profile comes also at different levels of abstraction Hence we have an example of what and how different elements, e.g. architecture components and queries, are modelled at different layers.

From the works cited above we can learn the following: (1) there are suitable logic-based formalisms modelling the MapReduce framework; (2) there does not exist a general level of abstraction for application architecture, application logic and environment; (3) there is no consensus on safety properties of interest to be verified; (4) there are no dedicated methods and tools for dealing with DIAs models, so researchers use general-purpose techniques and tools; (5) the research in this direction is in the early stage.

Hence, the challenge of the DICE research will be to exploit and adapt existing techniques for checking safety and privacy of DIAs. Therefore, in the remainder of this section we will focus on general-purpose verification methods and tools that we aim to use and adapt in the framework of the DICE project.

Two of the underlying formalisms used to enable formal verification which will be exploited by us are *temporal logic* and *first-order logic.* While temporal logic proved to be useful for formalising the application and expressing the properties to be checked, first-order logic is the working language of powerful state-of-the-art solvers that can be used to check the satisfiability of these properties. This is also the strategy we aim to pursue in DICE. Hence, below we'll present relevant research in these two directions and how we plan to exploit them.

Using temporal logic notation (see e.g. [304]) as the language of annotating the UML models, or software, is not appealing for any kind of actor/end user, unless they have previous experience with such notations. To overcome this shortcoming, temporal logic description of the models should be completely hidden from the modellers. One way to achieve this is to use a *pattern-based approach* to the presentation, codification and reuse of property specifications. For finite-state verification, research has been initiated in this direction in [305], which proposes *specification abstractions* having the property to be parameterisable, high-level, and formalism-independent. It turned out that their approach was successful since it is mainly the case that practitioners prefer guidance on how to optimise the usage of the language for solving a large class of problems, rather than using its full expressiveness. In a similar manner, the TRIO approach [306] a first-order linear temporal logic that supports a metric on time - allows template-based specification. TRIO specifications of systems consist of basic predicates and arithmetic temporal terms representing elementary phenomena of the system. The system behaviour over time is described by a set of TRIO formulae, which state how the predicates are constrained and how they vary over time in a purely declarative fashion. This formalism has been successfully applied to the case of embedded systems [307]. More recently, specification patterns have been used in Service-Based Applications (SBAs), in particular industrial SBAs in the banking domain [308]. *In the framework of the DICE project we plan to use and extend the TRIO formalism, as well as for specifying Big Data applications models.*

Similar to the state of the art, in DICE we will also consider using classical model checkers like SPIN [309], NuSMV [310] and Uppaal [302], which target continuous-time, real-time systems, and PRISM

[311], which handles probabilistic models, in order to test safety properties specific for DIAs. In this context we will also exploit the techniques introduced in [312] to include the possibility of modelling and verifying data transmission under real-time constraints. Additionaly to this, for efficiency purposes, Satisfiability Modulo Theories (SMT) solvers such as Z3 [313] and satisfiability solvers for temporal logics such as Zot [314] will be used as verification engines to build formal verification techniques for UML models with real-time constraints similar to b project [307]. However, in the DICE framework these techniques have to be extended in order to cope with the particularities of DIAs.

For what concerns the formal verification at different levels of abstraction and granularity, the work presented in [307], which is implemented in the CorrettoUML [315] formal verification tool for UML models, is relevant for DICE: we plan to decouple the semantics from the predicates that represent model elements. In this way one can change the semantics while translation from UML to the predicates remains unaffected. Since verification at all abstraction levels will be impossible due to the complexity of the model and the properties to be verified, we plan to adopt the so-called lightweight formal approaches [316], where verification formalisms are applied only where necessary. Moreover, [307] introduced stereotypes to allow actors to identify those parts of the system on which they want to focus verification activities. These stereotypes are then taken into account during the transformation phase, which only produces formal model for the tagged parts instead of the whole model. *A lightweight formal approach combined with stereotypes will be also used within DICE.* These, together with verification at different abstraction levels, will allow us to model the application as a set of components (e.g. nodes) performing the same kind of task (e.g. Map, Reduce) and check the same safety property for each component type. In order to avoid this repetition, a generalisation of the safety properties for an arbitrary number of components of the same kind is useful. Such generalisation was applied for many case studies, for example mutual exclusion protocols [317]. We believe that this parametricity is practically important, especially in the context of DIAs modelling.

Regarding privacy properties, research similar to DICE Model-Driven Engineering context is carried out in [318]. The authors develop a Model-Driven Development approach for the secure medical system Selkis. They built both the functional and security models (Platform Independent Models) of the application in SecureUML [319], translated them into a specification in the B method [320] and then used the animation tools of the B method in order to validate the functional model and to perform systematic tests. The modelling of privacy properties like data confidentiality and integrity made possible the detection of insiders' attacks. In DICE we aim at capturing privacy properties at the UML level using SecureUML and then formalising them in temporal logic.

We presented a short overview of existing methods and tools relevant in the context of DICE for the verification of safety, respectively, privacy of systems. We noticed that the verification tasks are approached from different perspectives. *Due to this consideration, in DICE safety and privacy properties will be handled by using different techniques and tools suitable to the given situation.*

C.8. Tools for Formal Verification

As was explained at the beginning of the chapter in formal verification approaches we aim to check - given a model of the system S and a requirement R, both formally expressed - whether R holds for S or not (i.e. whether S |= R). Depending on the formalisms used to describe S and R we can express and verify different kinds of properties (e.g. ordering, real-time, probabilistic etc.).

Many tools have been built that implement the various formal verification approaches. In this section we briefly list some of the most commonly used, and the candidates to be the starting point for the DICE formal verification tools.

Table 14 lists the tools of interest. For each tool we indicate its name, the formalism used to describe the system S (the 'System language') and the formalism used to describe the requirement R (the 'Property language'). The tools' websites can be found in the list of references.

Table 14: Formal verification tools.

Tool	System language	Property Language
SPIN [309]	Transition systems	LTL
	The input language of SPIN is called Promela, and it allows users to describe transition systems, i.e., finite state automata. The tool allows users to check for properties such as state reachability, assertion violations, and properties expressed in LTL; essentially, it allows for the verification of ordering properties (real-time properties can be checked provided we assume '1 state transition = 1 time instant'. The verification is based on the exhaustive exploration of the state space.	
NuSMV [310]	Transition systems	LTL/CTL
	NuSMV has a custom language for describing system models to be analysed, which corresponds to defining transition systems. It implements several decision procedures based on a symbolic representation of the state. Depending on the decision procedure, it uses a different formalism to express the property to be checked: CTL or LTL. As a consequence, it is capable of analysing ordering properties. An evolution of the tool has been recently released, called nuXmv [321], which is capable of analysing infinite-state systems and also real-time properties.	
Uppaal [302]	Timed Automata	Restricted version of CTL
	Uppaal is the most popular tool for analysing real-time systems. It uses Timed Automata as input language. To define the property to be analysed it employs a restricted variant of CTL which only allows certain combinations of temporal operators. Essentially, it allows users to check whether certain desired states are reachable or not. The decision procedure is based on the exhaustive exploration of the state space of the so-called 'region automaton' that is derived from the timed automaton.	
PRISM [311]	Probabilistic automata	PCTL/CSL
	PRISM is the reference tool for the formal verification of probabilistic systems. It allows users to use several languages to describe the system to be analysed: Discrete-Time Markov Chains (DTMC), Continuous-Time Markov Chains (CTMC), Markov Decision Processes (MDP), Probabilistic Timed Automata (PTA). For systems described through DTMC, MDP and PTA, the tools allow users to formalise the property to be checked through PCTL, whereas the properties to be analysed for CTMCs are expressed in CSL.	
Zot [314]	Metric Temporal Logic	Metric Temporal Logic
	Zot is a satisfiability checker for various kinds of temporal logics, most of which have a metric on time. More precisely, it can decide the satisfiability of formulae of LTL, MTL/TRIO, CLTL (Constraint LTL) and CLTLoc (Constraint LTL over clocks). To check whether property R holds for system S it reduces the problem to one of satisfiability of a logic formula. More precisely, when both S and R are described through temporal logic formulae, it checks whether S |= R or not by determining if S ∧ ¬R is satisfiable (S |= R holds if S ∧ ¬R is *not* satisfiable). Zot employs a so-called *bounded* decision procedure, where the temporal logic formula P whose satisfiability needs to be checked is translated into a formula of a decidable logic that captures the unfolding of P over k time instants (where k is the bound). Then, the satisfiability of the latter formula is checked through an off-the-shelf SAT of SMT (Satisfiability Modulo Theories) solver such as, for example, Z3[322].	

C.9. Software Anti-patterns

Flaws in software architecture design resulting in performance problems are known as software Performance Anti-patterns (AP).

The first study employing automated AP detection [323] involves construction of application design model based on monitoring data and specification of a set of rules. Each rule is a description of application behaviour if a specific anti-pattern is present. These rules are then applied to the design model in order to detect APs. This approach became the practice used in all subsequent work in the area. However, the work considered only Enterprise Java Bean applications and could not be used for other technologies. Additionally, the approach utilises run-time monitoring data, and thus is not suitable for anti-patterns detection on the early development stages.

Cortellessa [324] in his state of the art analysis has summarised the research activities undertaken by his research group in the area of performance anti-patterns specification, detection and solution. The paper stressed the importance of the automation of these activities and gave a summary of model-driven techniques employing UML or the Palladio Component Model (PCM) [154] and specifically developed Anti-patterns Modelling Language (Aemilia). [324] also proposed a ranking system to rate anti-patterns based on their contribution to violating requirements. This approach aims to facilitate more efficient and automated AP solving. Finally, open issues, such as accuracy of anti-pattern specification, conflict between anti-pattern solutions, ambiguity in formalisation, interdependencies among performance requirements and others were outlined.

Trubiani et. al. [325] built on the work cited and discussed in [324] by adding the ranking system to the detected anti-patterns. This system allows one to find anti-patterns causing the performance degradation and, based on the results, run the automated search for the optimal architecture configuration that would eliminate this 'guilty' anti-pattern. The authors though argue against the full automation of the anti-patterns solution process, as human experience can be useful to cut some alternatives or give priority to other alternatives than an optimisation algorithm would. The proposed methodology is implemented as a tool, built on PCM and Aemilia.

Wert et. al [326], [327] have also developed an automated AP detection tool, but did not extend it to the automated AP solution. While the tool in [325] requires the prior knowledge of the application software architecture and PCM is used to simulate it in order to obtain performance analysis results, Wert et. al.'s tool acquires performance analysis metrics through specifically designed set of experiments. Design of experiments technique used in this approach is based on the assumptions about application usage profiles.

Trubiani et. al [328] investigated potential benefits of synergy between two types of software performance analysis: Bottleneck Analysis (BA), aimed at identifying overloaded software components/hardware resources, and Performance Anti-patterns (PA) in order to enhance the efficiency of identification and solution of performance problems, and, as a result, present software developer with a broader set of promising alternative solutions. For the validation of this approach authors designed a case study, where they first executed two techniques separately, and then in sequence: PA after BA and BA after PA. The sequence 'bottleneck analysis after performance anti-patterns' yielded the highest improvement on the parameter of interest – response time of a certain operation (reduction by 90.8%). This approach also allows for the choice of anti-patterns to be resolved or discarded, but instead of ranking system introduced in Trubiani et. al. [325], the decision is made based on the results of BA. In conclusion, the authors discussed limitations of BA and PA if executed in isolation, trade-offs between algorithm complexity and its effectiveness and benefits of BA and PA synergy. However, there is still a limitation to the proposed

approach as its execution time depends on a number of application-specific parameters, e.g. number of software components/hardware resources, which may cause scaling problems if implemented in large-scale applications. Table 15 provides a summary of the methods discussed above. Automated AP detection and automated AP solution are chosen as dimensions for comparison to indicate if manual labour is required in the method application. Dependence on the specific technology/platform or reliance on the specifically-developed modelling language might become a challenge in the uptake of a method. Finally, implementation of a formal model in the method may be an advantage, as they are well understood and widely used. However, this may play a negative role as well, as formal models require large parameter space which 'explodes' with increase of an application's scale.

Table 15: State of the art in the area of software performance anti-patterns detection.

	Automated AP detection	Automated AP solution	Platform/technology independent	Utilises formal model	Specific modelling language
Parsons et. al [323]	Y	N	N	Y	N
Wert et. al. [326], [327]	Y	N	Y	N	N
Trubiani et. al. [325]	Y	Y	Y	Y	Y (Aemilia)
Trubiani et. al. 2 [328]	Y	Y	Y	Y	Y

C.10. Optimising Deployment Plans

The DICE project aims at providing an optimisation tool whose main goal is to identify a minimum cost deployment for DIAs providing performance (e.g. job execution time is within a deadline) and reliability (e.g. probability of no failure before the end of task is at least 99%) guarantees in a (public or private) Infrastructe as a Service (IaaS) system. Once a model-to-model transformation is created between UML models and the class of reliability and performance models discussed earlier, one can think of a search procedure that alters the definition of the resources needed by the application up to finding an assignment that satisfies the SLAs. This implies that each candidate deployment needs to be re-assessed using quality analysis tools to determine the fitness of that assignment and possibly suggest the optimal search directions for a new guess. The final output of this activity is the generation of a cost-optimal deployment plan from the UML model. Evidently, the development of efficient search methods is of paramount importance in this activity, since model-to-model transformations and solutions of formal models are both computationally expensive. Thus, we overview advanced methods that have been proposed to efficiently search the space of deployment allocations.

Identifying a Big Data cluster of minimum cost to operate a DIA under job completion time constraints is far from trivial. As a matter of fact, given the variety of resource types, their performance and pricing models available in the online catalogue of any of the Cloud providers in the market, the number of possible Cloud configurations grows combinatorially. This makes the exploration of the solution space a difficult and time consuming task even for determining a feasible configuration of small size systems. What is more, since the number of VMs to be allocated to a single job depends on their type, even in the simplistic scenario where only one type of VMs is selected the resulting problem belongs to the class of bi-level optimisation problems (i.e. a one problem is nested within another) that has been demonstrated to be NP-Hard also for the simpler linear-linear case [329]. More details on bi-level optimisation problems can

be found in [330]. Finally, the problem could result in being highly nonlinear in the constraints set. In particular, given a certain deadline for an application, a number of VMs and the associated type, the process of checking whether the application meets the requirements on the completion time (i.e. whether it terminates before the deadline or not) can be nonlinear in the size of the problem. The described problem, being intractable with exact algorithms even for very small cases, calls for a meta-heuristic solution approach [331].

One meta-heuristic of interest to DICE is local search. The rationale behind a basic local search is the idea to reach a good solution through a sequence of moves, each one leading from one solution to another. A move can be described informally as an atomic operation that changes the structure of a solution it is applied upon, generating a new (hopefully better) one. Indeed, not every move implies the generation of a better solution. A local-search-based algorithm starts from an initial solution and drives the optimisation process by selecting the moves to perform according to a certain strategy, which might involve moves that can lead towards better or worse result. Local search techniques within the larger class of meta-heuristics have proven to be highly adaptable and well suited to deal with optimisation problems where the evaluation of the objective function is a process that may require a long time. There are several reasons for this. One is that they are generally more reactive than other techniques. Furthermore, they tends to be more suitable to incorporate and exploit specific knowledge of the problem, derived from performance models, to drive the search toward more promising zones of the solution space.

C.11. Testing and Monitoring of Non-Functional Properties

C.11.1 Testing methods

In this section, we focus on state-of-the-art research in quality testing through model-driven approaches. Since the model-driven aspect is central to DICE, we do not review here standard testing approaches that are agnostic of the application model-driven architecture. Furthermore, we only consider testing methods that primarily focus on quality aspects (e.g. efficiency), ignoring functional testing, which is a mature area [332]. Model-driven functional testing methods are easy to integrate with UML 2.0, for example via the UML Testing Profile (UTP) distributed by OMG [333].

A systematic review performed in 2007 [334] reported that among 200 surveyed works in Model-Driven Testing, just 10 papers covered non-functional aspects, in particular performance testing and related dimensions (i.e. stability, resource utilisation and scalability).

Model-driven performance testing aims at using model-based reasoning to: 1) automatically generate a performance test suite, possibly using model-to-model transformations; 2) automate the execution of the test activities such as:

- **Load tests**: where the test workloads are similar to the ones faced in production. Therefore the aim is to expose the expected behaviour of the application under such workloads in order to verify compliance with service-level objectives.
- **Stress tests**: where the goal is to expose hardware and software bottlenecks by generating a workload that leads to resource exhaustion. Stress testing can expose worst-case scenarios, poor system configuration or verify the ability of self-adapting system to react to extreme conditions.
- **Stability tests**: where the application stability is assessed by long-running tests.
- **Spike tests**: where the system response is evaluated under unit workload spikes.

In the context of web-based enterprise systems, Model-Based Testing typically relies on the following classes of models:

- **User behaviour models**, commonly specified via formal models such as probabilistic automata or Markov chains, which can describe the navigation behaviour of users on a set of web pages. It is easy to define algorithms that use these models to randomly generate user sessions for testing purposes. Often the challenge in specifying these models is in the characterisation of the preconditions that should hold in order to generate semantically-valid user sessions [335]. UML sequence diagrams or PCM models [154] can be automatically translated into user behaviour models, as done for example for PCM by the *MDLoad* tool developed in the MODAClouds project [336]. Furthermore, methods to automatically extract user behaviour models have also been recently proposed [337].
- **Traffic models**: formal models are also commonly exploited to specify the frequency at which requests should be sent to the application. For example, Markov-modulated models [338], [339] have been used in recent years to automatically inject workload burstiness in a test suite. In the literature, traffic models have not been normally related to UML or PCM modelling artefacts and they are more often provided as input parameters to the workload generator.

Other classes of models for stress testing have also been considered in the literature, such as **network usage models** that can be extracted from UML diagrams to support the generation of workloads that are likely to expose faults related to network traffic [340].

One question left open by the methods discussed above concerns their ability to cope with the generation of large volumes of data or high-velocity streams. We have surveyed the relevant literature. For NoSQL datastores (see Section D.4), the main benchmark is the Yahoo! YCSB benchmark [341] which allows the generation of user-specific datasets based on the number of records/keys with a specific distribution. Clients can be configured to access the datastore in a random fashion or skewed with distributions that mimic internet applications/services and with different ratios of reads/writes. Traditional databases have a number of benchmarks mostly based on the TPC [342] suite of benchmarks, including the new TPCx-HS Big Data benchmark specification. The size of the data is defined within the specifications and is usually scaled from the minimum specification, e.g. in TPC-C depends on the number of warehouses. Data generation variability is controlled by defined functions designed to be semi-random. However, it has been shown that the data tends towards uniformity [343]. Data access is semi-random based on the same functions for data generation, but the access footprint tends towards randomness. It is important to note that the TPC benchmarks are specifications and their implementations are left to third parties.

Concerning stream-based systems, the unique feature of load/stress testing for stream-based application is that the parameters that we can change to put load on the system consist of two orthogonal dimensions: (1) data stream to the system, (2) query to retrieve information from the system. For example, for the former we can change:

- **Event injection rate**. Represents the number of events injected per second into the system. For example: {1K, 5K, 10K, 50K, 100K}
- **Number of users**. Represent the concurrent number of users submitting queries (e.g., /recommend and /similarity in Oryx [31]) to the system.

And regarding the second dimension, we can change:

- **Query inter-arrival time**. Represent the inter-arrival time between the query submission by a user.
- **Query parameters**. Represent the type of parameter that is passed to the query. For example in Oryx we can pass the userID and depending on the type of user the cost of running the query could be different.

In the research community for comparing approaches benchmarks like LRB benchmark [344] are used. However, to the best of our knowledge there is no available tool that generates loads automatically based on a user-defined usage pattern (like model-driven approaches for web-based applications, e.g. MODACLouds MDLoad testing tool [336]) or probabilistic characteristics. The Starfish system [345] is the most relevant approach. It represents a cost-based optimiser for MapReduce programmes which focuses on the optimisation of configuration parameters for execution of these programmes on the Hadoop platform. It relies on the profiler component that collects statistical information from executing the programmes and a what-if engine for fine-grained cost estimation processing.

Message-Oriented Middleware (MOM) is the previous generation of technology for modern event-driven applications like stock trading, event-based supply chain management, air traffic control and online auctions. However, with the popularity of stream-based applications, the popularity of MOMs diminished and they are now mostly used only in web-based non-Big Data systems. There exist a number of proprietary and open source benchmarks for messaging platforms — for example, the SonicMQ Test Harness [346], IBM's Performance Harness for Java Message Service [347], Apache's ActiveMQ JMeter Performance Test [348] and JBoss' Messaging Performance Framework [349].

Reliability testing tools are used to exercise an application and/or platform so that failures can be discovered within the design and functionality. To do this, these tools are used to generate load and/or faults within the application or platform. Test results can be used to identify bottlenecks and faults within the structure of the deployment. Traditionally these tests are run before live deployment where faults found will cause no issues to the production system. However, with the increase in usage of the Cloud, reliability tests are now being run on live production as fast redeployment and inexpensive elasticity have made this possible.

C.11.2 Testing tools

C.11.2.1. Grinder

Grinder [350] is a framework for distributed testing of Java applications. In addition to testing HTTP ports, it can test web services (SOAP/REST), and services exposed via protocols such as JMS, JDBC, SMTP, FTP and RMI. The tool executes test scripts written in Java, Jython [351] or Clojure [352]. Tests can be *dynamic*, in the sense that the script can decide the next action to perform based on the outcome of the previous one. The tool is mature, having been first released in 2003. It is quite popular with weekly download count of around 400 downloads. A limitation of Grinder is the lack of an organised community of volunteers to support this tool.

Current Version: 3

License type: Custom, BSD-style license.

C.11.2.2. Apache JMeter

JMeter [353] is a distributed load testing tool that offers similar features to Grinder in terms of load injection capabilities. If combined with Markov4JMeter [354], the tool can also feature probabilistic behaviour. Interestingly, the tool also features native support for MongoDB (a NoSQL database, see Section D.4). The tool excels in extensibility, which is achieved through plug-ins. A limitation of JMeter is that the web navigation may not be representative on websites where Javascript/AJAX play a major role for performance, since JMeter does not feature the same range of Javascript capabilities as a complex browser. JMeter has support of the large Apache Foundation community.

Current Version: 2.13
License type: Apache Public License 2.0 [187]

C.11.2.3. Selenium

Selenium [355] is a popular library that automates the control of a web browser, such as Internet Explorer or Firefox. It allows to execute tests using a native Java API. Furthermore, test scripts can be recorded by adding Selenium plug-in to the browser and manually executing the operations. An extension, formerly called SeleniumGrid and now integrated in the stable branch of the main Selenium tool, allows to perform distributed load testing.

Current Version: 2.13

License type: Apache Public License 2.0 [187]

C.11.2.4. MODAClouds MDload

The MDload tool [356] is a Java stress testing application built upon the Selenium library. The distinguished feature of the tool is the generation of requests driven by an engine capable of a layered model describing user behaviour and resource topology, which can be used online to decide the next request to issue to the system. MDload offers the capability to conduct a multi-threaded test using local Mozilla Firefox browser. The tool acts as a closed-loop workload generator, where clients need to complete requests before issuing new requests. Requests can also be injected via bursts, for increased realism. A limitation of MDload is the lack of integration with the distributed stress testing capabilities of Selenium.

Current Version: 1.0
License type: BSD-3 [223]

C.11.2.5. Chaos Monkey

Chaos Monkey [357] is an open source service that runs on Amazon Web Services [192]. This tool is used to create failures within Auto Scaling Groups [358] and to help identify failures within cloud applications. The design is flexible and can be expanded to work with additional cloud providers and provide additional functionality. The service has a fully configurable schedule for when actions will be taken and by default runs on non-holiday weekdays between 9am and 3pm. A similar tool has been developed within .NET [71] for Microsoft Azure [359] called WazMonkey [360] which offers similar functionality as Chaos Monkey such as rebooting or reimage role instances within a given Azure deployment at random.

Current Version: 2.4.0
License type: Apache Public License 2.0 [187]

Table 16 provides comparative summary of testing tools described in detail in the sections C.11.2.1-C.11.2.5.

Table 16: Comparative summary of testing tools.

	Type	Distributed Testing	User Behaviour Model	Traffic Model
Grinder [350]	Performance	Y	Y	Y
JMeter [353]	Performance	Y	Y*	Y
Selenium [355]	Performance	Y	N	N
MDload [356]	Performance	N	Y	Y
ChaosMonkey [357]	Reliability	Y	N	N

*: via the Markov4JMeter plug-in.

C.11.3 Monitoring tools

During recent years a large number of companies/organisations tried to move part or even entire infrastructure to the cloud. This trend can be observed due to the new easy ways of delivering applications, process large workloads and even empower the end-user to work with some model of the cloud.

Although the trend is so that lots of organisations tend to move their applications to the cloud, there are still problems that have to be addressed in order to ease usage of cloud technologies. One of the problems that have to be addressed is related to the way one can monitor the cloud in order to provide the user with aggregated information about cloud behaviour in different situations. The major problem that arises in the context of monitoring is which metrics have to be monitored. Besides this problem one has to figure out what sort of monitoring solution fits the cloud model deployed.

Below we present a short overview of different monitoring solutions that can be applied in different cloud deployments. We will start by presenting solutions that are based on Hadoop toolkit [361], then we'll provide a short overview of technologies used for monitoring NoSQL databases (see Section D.4 for the technology description and analysis) and some other solutions used in different applications.

C.11.3.1. Hadoop toolkit

Hadoop Performance Monitoring User Interface (UI) [361] provides a Hadoop built-in solution for quickly finding performance bottlenecks. It also can visually represent the configuration parameters which might be tuned for better performance. The monitoring toolkit provides a lightweight monitoring UI for the Hadoop servers. Although it is a lightweight solution, it allows its users to find out where and why bottlenecks occur in different instances of Hadoop. The toolkit is already present in any Hadoop distribution, hence in order to use it one only has to start the UI and do data queries.

License type: Apache Public License 2.0 [187]

C.11.3.2. SequenceIQ

SequenceIQ [362] provides a solution for monitoring Hadoop clusters. The architecture proposed in [363] and used in order to do monitoring is based on Elasticsearch [364], Kibana [365] and Logstash [366].

The general picture of their proposed architecture as presented in [363], is shown in the Figure 14 below:

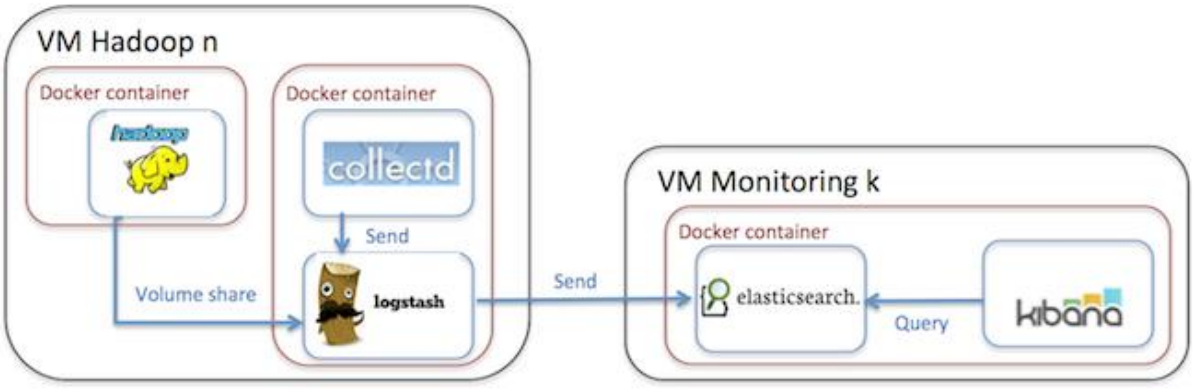

Figure 14. SequenceIQ high-level architecture [363].

The architecture proposed by [363] has the main objective of obtaining a clear separation between monitoring tools and some existing Hadoop deployment. For achieving this they use three Docker [199] containers.

In a nutshell, the monitoring solution consists of *client* and *server* containers. The server container takes care of the actual monitoring tools. In this particular deployment it contains Kibana for visualisation and Elasticsearch for consolidation of the monitoring metrics. Through the capabilities of Elasticsearch one can horizontally scale and cluster multiple monitoring components. The client container contains the actual deployment of the tools that have to be monitored. In this particular instance it contains Logstash, Hadoop [14] and the Collectd [367] modules. The Logstash connects to Elasticsearch cluster as client and stores the processed and transformed metrics data there.

License type: Apache Public License 2.0 [187]

C.11.3.3. Hadoop Vaidya

Hadoop Vaidya [368] (Vaidya in Sanskrit language means 'one who knows', or 'a physician') is a rule-based performance diagnostic tool for MapReduce jobs. It performs a post execution analysis of map/reduce job by parsing and collecting execution statistics through job history and job configuration files. It runs a set of predefined tests/rules against job execution statistics to diagnose various performance problems. Each test rule detects a specific performance problem with the MapReduce job and provides a targeted advice to the user. This tool generates an XML report based on the evaluation results of individual test rules.

License type: Apache Public License 2.0 [187]

C.11.3.4. Ganglia

Ganglia [369] is a scalable distributed monitoring system for high-performance computing systems such as clusters and grids. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport and RRDtool [370] for data storage and visualisation. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on thousands of clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes.

License type: BSD-License [223]

C.11.3.5. Apache Ambari

The Apache Ambari [371] project is aimed at making Hadoop management simpler by developing software for provisioning, managing, and monitoring Apache Hadoop clusters. Ambari provides an intuitive, easy-to-use Hadoop management web UI backed by its RESTful APIs.

License type: Apache Public License 2.0 [187]

C.11.3.6. Apache Chukwa

Chukwa [372] is an open source data collection system for monitoring large distributed systems. Chukwa is built on top of the Hadoop Distributed File System (HDFS) [33] and MapReduce framework and inherits Hadoop's scalability and robustness. Chukwa also includes a flexible and powerful toolkit for displaying, monitoring and analysing results to make the best use of the collected data.

License type: Apache Public License 2.0 [187]

C.11.3.7. Datastax-OpsCentre for Apache Cassandra

Datastax [373] provides a solution called OpsCentre [374] which can be integrated in order to monitor Cassandra [375] installation. Using OpsCentre one can monitor different parameters of the Cassandra instance and also different parameters provided by the actual machines on which it runs. Also, they expose an interactive web UI that can allow administrators to add or remove nodes from the deployment. One interesting feature provided by the OpsCentre is the automatic load balancing. A developer API is also provided in order to integrate with other services.

License: Free for Datastax-OpsCentre (not bundled)

C.11.3.8. MongoDB (MMS)

One way of monitoring and keeping under control a MongoDB deployment is using MMS [376]. MMS is a cloud service developed by the MongoDB team. It provides an integrated and easy way to provision, monitor, backup and scale MongoDB on the infrastructure of choice. It also includes out of the box support for Amazon Web Services (AWS) [192]..

License: Commercial.

C.11.3.9. Server Density

The tool provides an interesting way of monitoring different systems. Besides the monitoring itself, it provides a way to clearly visualise the data. One of the main advantages of Server Density [377] is its versatility. Its implementation ranges from the monitoring of the basic user profile to the monitoring of the high-throughput (over 30TB/month) processing of time-series data.

License: Commercial.

C.11.3.10. Manage Engine

Applications Manager enables the users to monitor the performance of Cassandra [375] and also perform administration tasks of all the nodes in a cluster in a centralised manner. One can collect different statistical data from the JVMs that run in a cluster. Besides Cassandra the tools offered by Manage Engine [378] cover a wide variety of other applications that can be monitored. When it comes to NoSQL databasess it can also monitor MongoDB instances and others.

License: Commercial.

Table 17 provides comparative summary of monitoring tools described in detail in the sections C.11.3.1-C.11.3.10.

Table 17: Comparative summary of monitoring tools.

Tool	License	Monitored technology	Tools used
Hadoop-toolkit [361]	Apache [187]	Hadoop [14]	-
SequenceIQ [362]	Multiple	Hadoop	Docker [199], Collectd [367], Logstash [366], elasticsearch and Kigtbana [365]
Hadoop Vaidya [368]	Apache	Hadoop	-
Ganglia [369]	BSD [223]	Clusters	-

Ambari [371]	Apache	Hadoop	-
Chukwa [372]	Apache	Hadoop	-
MMS (MongoDB) [376]	First 8 servers free	MongoDB	-
Datastax-OpsCentre [374]	Free*	Cassandra [375]	-
Server Density [377]	Pay	MongoDB, others	-
Manage Engine [378]	Pay	Cassandra, MongoDB, others	-

C.12. Monitoring Feedback Analysis

C.12.1 Tools for Detecting Anomalies with Machine Learning

The goal of this section is to review Machine Learning (ML) tools that may be applied to perform anomaly detection on the monitoring data acquired during testing. One of the most well known tools for machine learning is **Weka** [379]. It is an extensive collection of ML algorithms written in Java covering classification, regression, clustering, outlier detection etc. It also features a package system that allows its functionality to be extended with both official and non-official packages. Weka is not designed out of the box to run on any distributed systems such as Hadoop [14] but there is a set of wrappers [380] available that support MapReduce operations on top of Hadoop. Another interesting package built on top of Weka is called **MOA** (Massive Online Analysis) [381]. It is the most popular open source framework for data stream mining.

Scikit-learn [382] is a general ML framework developed in Python [128] with some code written in Cython [383] for increased performance. It is built on top of existing math and science Python packages: NumPy [384], SciPy [385] and matplotlib [386]. It has a wide variety of machine learning algorithms similar in scope to that of Weka. It is arguably the most mature Python-based framework. It has been used in a wide range of scientific and commercial applications.

KNIME (Konstanz Information Miner) [387] is an open source data analytics, reporting and integration platform. It is written in Java and based on the Eclipse project. It has two important features. First, KNIME integrates various other open source projects ML libraries and data manipulation utilities (such as Weka, LIBSVM [388] and R [389]). The second feature is that its user interface is based on a visual paradigm (workflows), not API.

ELKI [390] is another specialised ML framework. It aims to provide a suitable environment for the development and evaluation of advanced data mining algorithms and their interactions with database index structures. One of its most interesting features is the extensive collection of outlier detection algorithms.

The **R** language [389] is one of the most used environments among ML practitioners as the main goal of this system is to provide an environment for statistical computing and graphics. R is easily extensible by using packages. These packages can be written in Java [99], C/C++ [100] and Fortran [391]. The CRAN package page [392] contains a wide variety of specialised ML algorithms. Several packages were developed in order to enable Big Data processing within R, most of them being available under CRAN packages page.

Mathematical libraries such as **Mathematica** [393] and **Matlab** [394] have both an extensive ML libraries and are used both in academia and commercial products. They even possess special plug-ins/libraries that enable the implementation of MapReduce. For Mathematica it is HadoopLink [395] while for Matlab this is Distributed MapReduce [396].

There are tools that are designed for specific problems. One such tool is **Encog** [397] which focuses on neural networks. It features a wide array of neural network types and training algorithms. Although the most widely used version of this framework is written in Java, there are .NET [71] and C++ [100] variants as well. Another important characteristic of Encog is that it is one of the fastest ML tools for training neural networks [398].

In the last 10 years a significant trend in ML is the use of General-Purpose Computing on Graphics Processing Units (**GPGPU**) [399], which can in some instances outperform high-end CPUs (and bring costs of equipment required for this training from $1M down to $20,000) [400], [401]. A significant portion of these tools is geared towards deep learning scenarios. Major deep learning software frameworks have incorporated GPU acceleration, including **Caffe** [402], **Torch7** [403], **Theano** [404], **CUDA-Convnet2** [405], **DeepLearning4J** [406], **cuDNN** [407] and NVIDIA's **DIGITS** [408].

Most of the tools listed above are not Big Data ready in the sense that they are not designed to handle large datasets. Also, most frameworks are not horizontally scalable. In the following section we will detail machine learning tools which are in fact Big Data ready and almost all of them are deployed on a distributed environment making them horizontally scalable. Table 18 provides comparative summary of ML anomaly detection tools described in detail above.

Table 18: Machine Learning tools.

Name	License	User Community	Language
Weka [379]	GPL 3	Large	Java [99]
Scikit-learn [382]	BSD [223]	Large	Python [128]
KNIME [387]	GPL 3	Large	Java
ELKI [390]	AGPL 3 [186]	Moderate	Java
R [389]	GPL 3	Large	C, Fortran [391]
Encog [397]	Apache 2.0 [187]	Low	Java, .NET [71]
Caffe [402]	BSD 2.0	Moderate	C/C++ [100]
Theano [404]	Apache 2.0	Low	Python

C.12.2 Distributed Machine Learning Platforms

H2O [409] and **MLLib** [410] are two of the most extensively developed projects of fully distributed and scalable Machine Learning (ML) tools. Both feature distributed in-memory computations and are certified both for Spark (MLLib being the part of Apache Spark [411]) and Hadoop [14] distributions. This in-memory capability means that in some instances this framework can outperform strictly Hadoop based frameworks [412]. One important distinction is that each H2O node (which is a single JVM process) runs as a mapper in Hadoop. There are no combiners or reducers. Also, H2O has more built-in analytical features and a more mature REST-based interface for R [389], Python [128] and JavaScript [99] than MLLib.

Mahout [413] is an ML framework built on top of Hadoop and features an extensive collection of ML algorithms. The latest version (0.9) supports a variety of different execution engines such as Hadoop [14], Spark [411], H20 [409] and Fink [414]. It is widely regarded that Mahout is well-suited for high-latency analytics and unsupervised learning tasks.

Jubatus [415] is a distributed computing framework specifically designed for online ML of Big Data. A loose model-sharing architecture allows it to efficiently train and share ML models by defining three fundamental operations: Update, Mix and Analyse [416]. It supports the scaling of ML processing of up to 100,000 or more data point per second. All data is processed in memory. The main difference between Jubatus and Hadoop/Mahout is that the former can handle real-time deep analytics tasks while the latter is not designed for these tasks.

Another distributed stream processing framework which focuses on real-time online machine learning is **Trident-ML** [417] built on top of Apache Storm [19]. It processes batches of tuples in a distributed way which means that it can scale horizontally. However, Storm doesn't allow state updates to append simultaneously which hinders distributed model learning. Learning step can't be distributed and no distributed learning algorithms are currently implemented for Trident-ML.

The Apache **Oryx 2** [31] framework is a realisation of the Lambda Architecture [17] built on top of Spark [411] and Kafka [21]. It is a specialised framework that provides real-time large scale machine learning. It consists of three tiers: lambda, ML and application. The Lambda tier is further split up into batch, speed and serving tiers respectively. Currently it has only 3 end-to-end implementations that implement the batch, speed and serving layer (collaborative filtering, k-Means clustering, classification and regression based on random forest). Although it has only these three complete implementations, its main design goal is not that of a traditional ML libraries but more a Lambda Architecture-based platform for MLLib [410] and Mahout [413].

Vowpal Wabbit [418] is an open source fast out-of core learning system currently sponsored by Microsoft Research [419]. It has an efficient implementation of online ML. Learning is accomplished through progressive validation. The so-called hash trick presented in [420] is implemented as the core data representation which results in significant storage compression for parameter vectors. Vowpal Wabbit also reduces regression, multi-class, multi-label, structured prediction etc. to a weighted binary classification problem. This allows for significant computational optimisations.

One limitation of this framework is that some problems can't be solved with the resources of a single machine. However, a Hadoop-compatible computational model called AllReduce [421] was developed to tackle this problem. For example, using this model, a 1000-node cluster was able to learn a terafeature dataset in one hour [420]. Table 19 contains a list of fully distributed and scalable ML tools.

Table 19: Distributed ML frameworks[3].

Name	Licensing	ML Problem	Distributed Environment	Users Community	Language
Petuum [422]	Open source (Sailing Lab)	DL, CL, Reg, Metrics Learning, Topic modelling	Clusters or on Amazon EC2 [429], Google GCE [430]	Moderate / Large	C++ [100]
Jubatus [415]	LGPL v2.1	CL, Reg, AD, CU, Rec, Graph analysis	Zookeeper [20]	Moderate	C++
MLlib [410] (MLBase)	Apache 2.0 [187]	Reg, CL,Rec, CU	Spark [411]	Large	Scala, Java
Mahout [413]	Apache 2.0	Collaborative Filtering, CL, CU, Dimensionality Reduction, Topic Models	Hadoop [14], Spark, H2O [409]	Moderate	Java
Oryx 2 [31]	Apache 2.0	Rec, CL, Reg, CU	Hadoop, Spark	Low	Java
Trident-ML [417]	Apache 2.0	CL,Reg, CU, Feature Extraction	Storm	Low	Java
H2O [409]	Apache 2.0	DL, Reg, CL,CU, PCA	Hadoop	Moderate	Java
GraphLab Create [423]	Apache 2.0	CU, CL, Reg, DL, Rec	Hadoop, Spark, MPI	High	C++
Vowpal Wabbit [418]	Ms-PL	CL, Reg, CU	Hadoop	Moderate	C++
Deep Learning 4J [424]	Apache 2.0	DL	Hadoop, Spark,	Moderate	Java, Scala
MLBase [425]	MIT License [222]	CL	Julia [431]		Julia [432]
FlinkML [426]	Apache 2.0	CL, Reg, CU, Rec	Flink Hadoop	Low	Scala [433]
NIMBLE [427]	N/A	CU, Frequent Pattern Mining, AD	Hadoop	None	Java
SystemML [428]	N/A	Reg, PageRank	Hadoop	None	DML

The tools mentioned in this section should not be considered as an exhaustive list of all the available ML tools. Some tools which are specialised on a specific use case were not covered. One such tool is **Ellogon** [434], specifically designed for natural language processing. Instead we focused on general ML frameworks which can be adapted with relative ease to any DICE use case.

Another important consideration when choosing the ML tools is to consider the specific challenges that arise in the application domain. For example in anomaly detection applications, one will most likely

[3] CL=Classification, CU=Clustering, Reg = Regression, Rec = Recommendation, AD = Anomaly Detection, DL = Deep Learning

consider both point anomalies as well as collective and context dependent anomalies. Because of this, easily extensible libraries such as Weka [379], Encog [397] or ELKI [390] are of key interest to the project. At this point it is not clear if a distributed ML engine is required in the DICE context but during the project we will look into and possibly adapt distributed ML tools to fit the project's needs.

C.12.3 Model Parameter Estimation

Part of the feedback analysis concept focuses on reducing amount of manual labour for the software developer when studying software performance analysis results and making decisions about improvements/changes in the application code or design.

Van Hoorn et. al [435] presented Kieker – a tool that conducts performance analysis, reconstructs software architecture model and automatically annotates it with analysis results. Additionally it presents performance analysis results in graphical and textual form via web-based user interface. The tool can be applied both to concurrent and distributed systems and currently implemented for a number of specific platforms/technologies (Java [99], .NET [71] and COM). The tool does not provide automated anomaly detection, leaving this task to the developer.

Filling-theGap (FG) tool, proposed and developed by Perez et. al [436] is similar to van Hoorn's [435] in that it also can be deployed both in concurrent and distributed systems, automatically updates performance model and presents performance analysis results both in graphical and textual form. However, FG tool employs UML diagrams and is platform and technology-independent. This is an ongoing work and currently the tool does not support automatic anomaly detection, neither support to improvements in the application code or design (leaving interpretation of analysis results to the developer).

Calinescu et. al [437] developed a tool for continual verification of service-based systems. The proposed method solves a formal model (discrete time Markov chains obtained from the software model via model-to-model transformation), automatically annotates performance model with results and verifies if all model parameters are within required values. If some thresholds are violated, algorithm chooses another service from the pool of available services so that all performance requirements could be satisfied. In this way developer's involvement is minimal and is required only if algorithm cannot find any service within a pool to satisfy performance requirements. Then the warning is issued and developer should make a decision, e.g. change requirements or re-factor model/application code. Table 20 provides a summary of the tools discussed above. Automated anomaly detection, automated anomaly solution, automated model update and the option of reporting to the developer/architect are chosen as dimensions for comparison to indicate if manual labour is required in the tool application. Dependence on the specific technology/platform might become a challenge in the uptake of a tool. Finally, the attempt is made to assess if these tools can be used in Big Data applications.

Table 20: State of the art in the area of feedback analysis tools.

	Automated anomaly detection	Automated anomaly solution	Automated model update	Reports results to the developer?	Platform/system independent	Can be used in Big Data applications?
Van Hoorn et. al [435]	N	N	Y	Y (text, visual)	N	Likely
Perez et. al [436]	N	N	Y	Y (text, visual)	Y	Likely
Calinescu et. al [437]	N	N	Y	Y (warning)	N	Unclear, due to large parameter space

C.12.4 Trace Checking

Trace checking is a procedure for evaluating a formal specification against a log of recorded events produced by a system. Traces are produced at runtime by a monitoring engine, which collects events of the system and saves them to a storage while running. According to Bauer et al. [438], run-time verification consists of all the verification techniques which allow for checking whether a certain execution of a system satisfies a given property, and trace checking is a run-time verification technique as long as only one (or a finite number of) execution of the system is analysed at a time. Therefore, runtime verification mainly focuses on the detection of violations of properties occurring in the observed system executions.

Trace checking can be performed online or offline, depending on the moment when the trace checking algorithm is executed and the moment when the result of the processing is available for the analysis. The online approach includes procedures such that the trace to verify is the current execution of the system and the analysis carries out while the system is running. The result of the verification is immediately considered upon the procedure terminates and allows for checking the integrity of the system and the adherence to the requirements. The offline approach prescribes that the verification is done on previously recorded traces which correspond to past executions of the system.

Logic-based formalisms were inherited from model-checking and became very popular also in run-time verification to specify properties. Linear Temporal Logic (LTL) [439] is employed in [440], while extensions of LTL over continuous-time are considered in [441], which studies monitoring procedures of LTL properties enriched with special operators dealing with time, and in [442] and [443] which devise monitoring algorithms for Metric Temporal Logic (MTL) specifications [444].

The standard approach to deal with trace checking is to build a monitor which verifies the correctness of the trace with respect to a certain property (e.g., ‘the average/maximum number of events per hour in the last ten hours is less than K’). In general, the time and memory complexity of the procedure building the monitor is small, as the monitor is only generated once for each specification to verify. On the other hand, logs in Big Data applications are rather extensive and procedures which formally verify properties on them can be rapidly limited by time and memory requirements.

Dealing with huge amounts of data increased the research effort towards the design of procedures for run-time verification leveraging parallel computational mechanisms. [445]shows the relationship between trace checking and Boolean circuits for which there are efficient algorithms to evaluate the output if their graph satisfies some specific restrictions. Although for general LTL formulae trace checking cannot be expressed with such a class of circuits, parallelisation with respect to sub-formulae can be efficiently achieved. The decidability of the trace checking problem over timed trace is studied in [446].

The consolidation of Big Data approaches and frameworks, started from the advent of MapReduce in industrial and mainstream applications, have led the community to a mature stage where parallelisation is adopted in trace checking [295] and run-time verification [447]-[449]. All adopt first-order relations over finite domains to represent the events in the traces which, based on the terms occurring therein, can be split into several unrelated partitions that are then verified in parallel by independent threads. These approaches rely on splitting traces on data but not on the structure of the formula. MapReduce for trace checking is adopted to evaluate LTL formulae over traces in [450] and in [451]. The latter employs MapReduce to check traces with MTL specifications by parallelising the evaluation of the formulae on their syntactic structure. The memory scalability depends on the size of the temporal intervals occurring in the formulae. [452] addresses this limitation by exploiting a special semantics and a parametric decomposition of formulae which allow the evaluation of the truth value of sub-formulae to consider only bounded portions of the trace for minimising the memory usage.

D. Data-Intensive Technologies

D.1. Overview

This section introduces the technologies relevant to the DICE framework. For each technology we emphasise three dimensions:

- **General characteristics**: In this subsection the innovative aspects and typical application domains of the technology are explained. The description is accompanied by a diagramme showing a typical architecture of this technology. Where possible, a list of public cloud offerings available for this technology is also reported together with advertised costs and quality. As the DICE project targets open source framework, open source solutions for adoption of this technology in private clouds are also described. If known, pointers to existing Chef [49] recipes to automate deployment of the solutions are provided.

- **Quality assurance**: For quality assurance aspects a list of key monitoring metrics used to assess quality in this technology is given. If possible, the description focuses on unique metrics specific to this technology. The main quality assurance challenges are described for this technology, referring to the research literature. In case there exist a native support and configuration options for reliability and high-availability, it is also detailed, as well as native support and configuration options for scalability and performance, for hard deadlines and real-time processing, and for privacy and data protection.

- **Models** (if it applies): Finally, if there exist meta-models that explicitly address this technology, relevant citations are provided. Also, if there exist Quality of Service (QoS) prediction models that explicitly address this technology, relevant references are provided.

The technologies and concepts covered in this chapter are listed below:

- Hadoop/MapReduce/Spark (Section D.2).
- Streaming (Section D.3).
- NoSQL (Section D.4).
- Software Defined Networking (Section D.5).
- Cloud-based Blob Storage (Ceph and Amazon S3) (Section D.6).
- In-memory Databases (e.g. HANA) (Section D.7).

D.2. Hadoop and Spark

D.2.1 Overview

Below we discuss the main technological solutions adopted in the industry to develop and execute Data-Intensive applications to cope with variety (unstructured data) and volume dimensions: Apache Hadoop [14] and Spark [411]. Both frameworks can exploit a common infrastructure for data distribution (relying on the distributed file system HDFS [33]) and for resource management (that can be handled by the resource negotiator YARN [18]). For introduction purposes we provide an overview of HDFS and YARN first.

The **Hadoop Distributed File System (HDFS)** [33] transparently distributes data across a computational cluster. Every machine runs a *Data Node* agent, which is responsible for handling *blocks*, i.e. data chunks belonging to files in the file system (usually of the size around 64/128 MB), whilst a central *Name Node* holds metadata that map blocks to files. For fault-tolerance reasons, blocks are replicated three times by

default and possibly one replica is placed on a different rack with respect to the other two. When files are written to HDFS, the Name Node provides a list of Data Nodes that will process the corresponding blocks. The client then sends data to the first Data Node, which further forwards replicas in a pipelined fashion to the subsequent Data Nodes. Name Node presents a single point of failure in the HDFS. Therefore, for reliability reasons it is common practice to run a second Name Node in passive state. It periodically saves snapshots of file system metadata in order to provide a failover alternative in case the active Name Node fails.

Hadoop YARN [18], acronym for Yet Another Resource Negotiator, was developed to decouple resource management and application semantics, as well as to address scalability issues brought by the resource management model implemented in Hadoop 1.x. Initially, Hadoop relied on a central Job Tracker to perform tasks ranging from capacity allocation to progress monitoring, including checking possible failures or performance degradation, and addressing them. All these duties resulted to be overwhelming for a single node in very large clusters. The issue is now addressed in YARN by distributing responsibilities among several entities. A first class of components - the *Node Managers* - handles resources locally activating *containers*, an abstraction for a slice of the resources available. The central *Resource Manager* has the role of allocating resources for the execution of jobs, based on a configurable scheduling policy. However, all the duties related to application semantics, such as restarting failed tasks, tracking progress, performing speculative execution, among the others, are delegated to *Application Masters*, which are application-specific components. When a job is submitted for execution, the Resource Manager bootstraps a container for the corresponding Application Master. Then, the Application Master requests a number of containers to perform the computation. According to the current state of the cluster and the scheduling policy, the Resource Manager provides a token-authenticated lease that the Application Master will use to activate the obtained containers at Node Managers.

After this introductory description of the common infrastructure, we present in detail the two considered frameworks: Hadoop and Spark.

D.2.2 Apache Hadoop

D.2.2.1. Overview

Hadoop [14] project started as an open source implementation of MapReduce framework. Hadoop framework allows for parallel and distributed computing on large scale clusters of commodity hardware, focusing on batch processing of huge datasets. Furthermore, it guarantees beneficial properties of fault-tolerance, scalability, and automatic parallelisation and distribution of computation across the cluster at the expense of a simple and rigid programming paradigm. MapReduce jobs are composed of two main phases, namely, *Map* and *Reduc*e. The former takes as input unstructured data from HDFS [33], filtering them and performing a preliminary elaboration according to the instructions in a user-defined function. The intermediate results are returned as key-value pairs, grouped by key in the *Shuffle* phase and distributed across the network, so as to provide each node taking part in the Reduce phase with all the values associated with a set of keys. In the end, every Reduce node applies a second user-defined function to complete data elaboration and outputs to HDFS.

Upon the same infrastructure, the **Apache Tez** [453] project allows for a less rigid programming paradigm. Applications can be expressed as directed acyclic graphs, where vertices represent data elaboration tasks and edges represent data dependencies among tasks. Depending on the application graph, Tez automatically launches tasks as soon as their input data are available.

There are four main comprehensive Hadoop distributions available for adoption in private clouds: Hortonworks Data Platform (HDP) [454], Cloudera CDH [455], MapR [456], and IBM BigInsights [457].

The first is composed of only open source software and is the only distribution that runs both on Windows and Linux. CDH offers proprietary software alongside the main open source projects to perform cluster management tasks, whilst MapR also adds closed source features to Hadoop. IBM BigInsights, on the other hand, integrates software from the open source Hadoop stack with enterprise-grade IBM solutions, such as Big R [458] and Big Sheets [459]. In literature there are not any up-to-date comparative studies on these solutions. However, they differ in terms of installed software stack: all of them include Hive [460], Pig [461], and HBase [462], SQL-on-Hadoop [463] projects, but CDH and BigInsights do not provide Storm (see Section D.3.4.1) and Tez, and HDP does not provide Spark [411]. Only HDP and BigInsights provide the Ambari [371] management platform. HDP, CDH, and MapR can be deployed with outdated Chef cookbooks available on GitHub [464]-[466].

D.2.2.2. Hadoop public cloud offerings

This section provides an overview of the offerings available from the three public cloud providers having the largest market shares (i.e. Amazon, Microsoft and Google) with an additional overview of Flexiant path to release Hadoop services.

Amazon provides Elastic MapReduce (EMR) [29], which automatically deploys a Hadoop cluster on EC2 [429] virtual machines. Users are billed on a pay-per-use basis, with a price per instance and hour ranging from $0.058 to $0.756 for general purpose instances at the time of writing [467], depending on their computational capabilities. Data can be stored on HDFS [33] hosted on ephemeral storage, at the cost of a lower scalability. EMR distinguishes two main classes of nodes: one performing computation and data storage, whilst the other only contributing for computational purposes. This forces the number of instances in the former class to never decrease to avoid data loss. Further, if the total number of data nodes is increased, HDFS is not automatically rebalanced, hence existing datasets do not benefit from the added capacity. Alternatively, it is possible to use EMRFS [468], a custom version of HDFS hosted on top of Amazon S3 (see Section D.6.2), working around such limitations and allowing for data persistency. With this approach, nodes in the EMR cluster are only in charge of computation and access of S3 space, thus allowing for dynamically scaling. Both storage solutions are billed hourly.

Microsoft offers HDInsight [469] – a Hadoop cloud service backed by Azure [359]. The pricing mechanism is similar to Amazon EMR, with prices per instance-hour ranging from $0.08 to $1.41. Differently from EMR, HDInsight clusters use all the attached instances as data nodes, hence they can be easily scaled up, but can only be scaled down by restarting the currently running jobs. This poses a threat of data loss due to the HDFS Data Nodes that are shut down in the process. Alternatively, as for the Amazon, it is possible to use Azure Blob [470] Hadoop-compatible containers for storage, allowing for a dynamically variable cluster size without data loss concerns.

Finally, also Google Compute Engine [430] offers Hadoop cluster, but this service is geared towards development or test environments and is not as straightforward as the previous alternatives.

Within DICE, Flexiant goal is to build the necessary feature set to replicate the AWS EMR service in conjunction with Flexiant Bento Boxes [471] that can be leveraged to spin up/down the Hadoop cluster. In particular, Flexiant Cloud Orchestrator [180] will provide the following:

- The means to access an object storage service to host the input files.
- The means to template a Hadoop cluster with all the required nodes roles (JobTracker, TaskTracker, NameNode, DataNode) to provide both HDFS [33] and MapReduce functionality. This could be leveraged by enhancing the Bento Box feature, including the ability to specify the number of workers to create from a single OS image containing the Hadoop framework
- The means to submit a job to the Hadoop cluster, ideally picking it from the object storage service.

- The ability to tell the Hadoop cluster where to place the output files (again into the object storage service).
- The ability to destroy the cluster once the job is complete.

The object storage service will be provided externally by software like Ceph (see Section) or Swift [472] but Flexiant Cloud Orchestrator will retain the Control Panel functionalities that are required to configure the service and possibly to browse its content. The OS image used by the cluster will be prepared by Flexiant in conjunction with a Hadoop expert partner and using popular frameworks such as the one delivered by Hortonworks [454].

D.2.2.3. Hadoop Quality metrics

The key metrics to assess the quality of MapReduce jobs are throughput and compliance with deadlines [473], [474].

The Hadoop framework provides reliability guarantees by monitoring the execution of Map and Reduce tasks. In case of failure of a node, all the hosted Map tasks are automatically restarted on other nodes processing other replicas of the failed blocks, whilst Reduce tasks are transparently restarted and fetch again the associated values across the network. In this way, the framework efficiently handles failures re-computing only the needed portions of intermediate values or final results.

D.2.2.4. Hadoop meta- and QoS models

Meta-models and high level language for specifying Data-Intensive applications have been provided in [475], [303], which can be further translated into Stochastic Petri Nets for performance analysis. The authors' proposed framework supports what-if analysis for Hadoop based systems.

Several QoS models for Hadoop and HDFS can be found in literature, which can be listed in three main categories: empirical models mapping data size into execution times, approximate formulae evaluating jobs executions time starting from individual tasks execution time and formal models like queueing network and colored Petri Nets.

In the first category, Herodotou [476] provides formulae that take into account every main contribution to total execution times. Anyway, the author did not perform any validation of the proposal. Zhang et al. [477] use regression methods to estimate job completion times based on dataset size and available capacity.

Verma et al. [478] propose the ARIA framework for MapReduce job completion time prediction, adopting job profiles extracted from previous executions to predict performance under different resource allocations. Zhang et al. [479] build upon these findings to derive a new model that takes into account both user-defined functions via job profiles, and generic platform performance linked to framework code. Malekimajd et al. [480] extend the formulae proposed in [478] to the multiple users case and apply them to optimise Hadoop cluster configuration in public clouds. Yang and Sun [481] propose a model that separately considers job and system impacts on performance. Their results are not accurate, but under varying configurations show a realistic trend, which might help in optimising cluster resources utilisation. Lin et al. [482] propose a detailed model taking into account low level aspects of the framework mechanism, but validate it only in the simplistic case of single job execution.

At last, within formal models, Lin et al. [483] propose a tandem queues network with overlapping phases to predict MapReduce jobs performance, which correspond to the overlapping of the Map and Shuffle phases. Tan et al. [484] consider Reduce-intensive jobs and model the Map phase with an M/G/1 queue, whilst the Reduce phase with a multi-server queue. Bardhan and Menascé [485] adopt mean value analysis to solve a queueing network model and predict the Map phase duration. Vianna et al. [486] develop a hierarchical

model coupling a precedence tree and a queueing network and then apply approximate mean value analysis to estimate service time.

An attempt to model HDFS with Timed Coloured Petri Nets has been carried on in [487]. Through simulation they can also obtain an optimal number of replicas to provide a sufficient reliability of the system in terms of read and write successful operation. Castiglione et al. [488] used a Stochastic Petri Nets to model MapReduce applications performance and proposed a novel mean field analysis evaluation method.

D.2.3 Apache Spark

D.2.3.1. Overview

Apache Spark [411] started as a research project at UC Berkeley to address the performance issues related to the use of the MapReduce paradigm in applications that imply iterative algorithms or interactive processing [489]. Spark is an in-memory framework based on the novel concept of Resilient Distributed Datasets (RDD), i.e. read-only, partitioned collections of records. Both for consistency and performance, RDDs can only be created through transformations, i.e. bulk operations applied to data sources or parent RDDs. It is also possible to request actions on RDDs, such as counting elements or extracting other aggregate values from the datasets. Among the available transformations there are Map and Reduce tasks, but also more complex operations such as union, groupByKey, etc. Every RDD is immutable, may be required to persist, and stores information about its lineage. RDDs are lazily created when an action is requested on them: the framework obtains a graph from the chain of dependencies and launches computation accordingly, possibly pipelining subsequent elaboration on RDDs with narrow dependencies. Given its very good performance the use of Spark is advocated for batch but also for interactive job analysis and streaming. Further, Spark supports interfaces for Python [128], R [389], and Scala [433] (R is partially supported for streaming analysis).

D.2.3.2. Spark public cloud offerings

Contrasting to Hadoop, the main public cloud providers do not feature readily available Spark services. On the other hand, both Amazon and Microsoft provide instructions about the customisation of their cloud Hadoop clusters to run Spark. In the case of Google instead there is a publicly available third-party script allowing for easy deployment of Spark clusters on Compute Engine [490]. Databricks [491] - the software company founded by Spark developers - offers automated deployment and management of Spark clusters on Amazon AWS [192]. Unfortunately, pricing is not clearly stated on the company website.

Among the above mentioned Hadoop distributions, all but Hortonworks [454] also comprise Apache Spark as part of their software stack.

Apache Spark guarantees reliability exploiting information about the lineage of RDDs. In the event of node failures the framework identifies missing partitions and schedules re-computation only of those. Dependencies in the lineage are used to obtain a graph of required transformations and this is followed up to the closest available ancestor.

D.2.3.3. Spark Quality metrics and models

The key metrics to assess quality of Spark jobs are, as for Hadoop, throughput and compliance with deadlines [492].

Being a rather young and poorly documented project, no performance models are available yet. However, in literature it has been observed that the shuffle phase is a bottleneck and some works investigate the optimisation of specific parts of the framework, as in [493], [494] even if some recent works have shown

how business intelligence workloads are mainly CPU-bounded and little improvement can be achieved if network and I/O in general are improved [495].

D.3. Streaming

D.3.1 Introduction

Stream processing is a technology designed to collect, integrate, analyse and visualise data streams or sensor data with a high ratio of event throughput versus number of queries, all in real-time as the data is being produced.

Stream processing solutions are designed to handle high volume in real time without disrupting the activity of existing sources and enterprise systems and with a scalable, highly available and fault-tolerant architecture [496], [497].

Use cases and application domains:

- Telecom: Security breaches, network outages, bandwidth allocation...
- Financial services: anti-fraud, operational risks, order routing, pricing...
- Retail: shrinkage, stock outs, offers, pricing...
- Manufacturing: preventive maintenance, quality assurance, supply chain optimisation...
- Transportation: driver monitoring, predictive maintenance, routes, pricing...
- Web: Application failures, operational issues, personalised content
- Research: Digital signal processing, databases, operating systems and networks, complex event processing, machine learning...

D.3.2 Stream processing architecture

Streaming architectures are typically organised in three layers [498]:

1. Data collection: A large scale data collection layer (Sensors, social media monitoring, financial market data, business transactions, machine-to-machine data etc.).
2. Real-time stream processing.
3. Data output: Redirect the processed data to other sub-systems for further processing.

Data collection and data output usually is performed through a highly efficient message broker such as Kafka [21] or RabbitMQ [499]. Figure 15 shows a typical streaming processing solution [500].

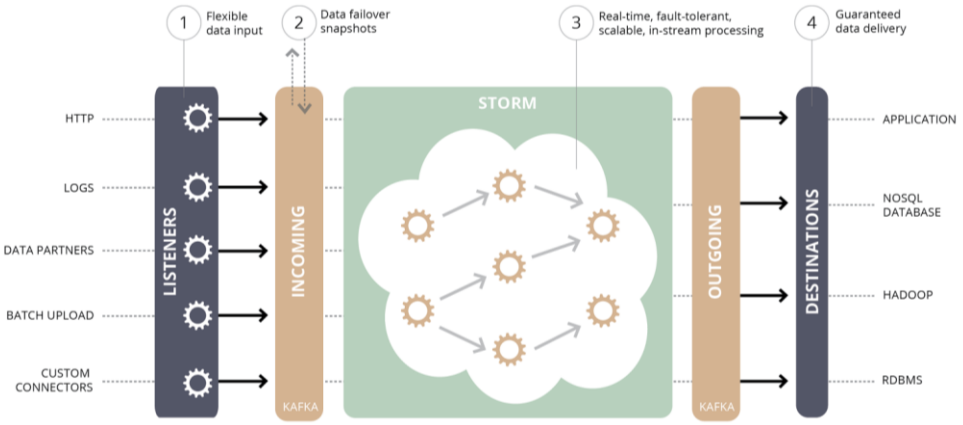

Figure 15. Architecture of a typical streaming processing solution [500].

D.3.3 Public cloud offerings

Table 21 lists some of the most important public and commercial cloud offerings with some streaming components integrated.

Table 21: Public and commercial cloud offerings for stream processing.

Product	Components	Pricing	Free trial	Observations
Microsoft HD Insight [501]	Hadoop + Yarn + Spark + Storm [502]	Difficult to calculate [503]	One month	Based on Apache components
AWS Kinesis [504]	Kinesis API + Integration with AWS [192] and Storm	Depends on the amount of records streamed [505]	No	Black boxed components available through APIs
Databricks [491]	Spark	Not publicly available	One month	-
Cloudera [506]	Kafka + Spark + HBase [455]	-	-	Based on Apache components. More private cloud oriented
Google Cloud Dataflow [27]	Dataflow SDK + Google Cloud Platform (Kafka, RabbitMQ [499], HBase [462], …)	Difficult to calculate	Two months	Beta

As a conclusion, most public clouds offerings are based on open source software components, mostly provided by the Apache foundation, while others provide custom APIs and SDKs for the streaming process.

D.3.4 Open source solutions.

A detailed view of some of the most important streaming solutions is described below:

D.3.4.1. Apache Storm

Apache Storm [19], [507] is a distributed real-time computation system for processing large volumes of high-velocity data.

Features:

- Fast: one million 100 byte messages per second per node.
- Scalable.
- Fault tolerant: workers automatically restarted.
- Reliable: guarantees every message is processed.
- Easy to operate: ready for production.

Figure 16 shows a typical Storm architecture [508].

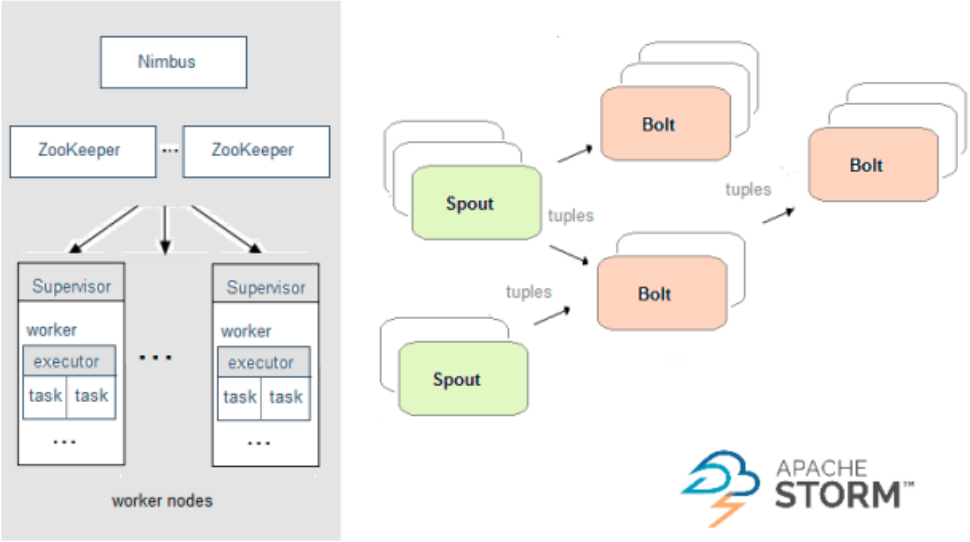

Figure 16. Typical Storm architecture [508].

D.3.4.2. Apache Spark Streaming

Apache Spark Streaming [509] is a batch Big Data processor that provides a module for streaming processing.

Features:

- API: supports Java [99], Scala [433] and Python [128].
- Fault tolerance: recovers both lost work and operator state.
- Integration: Allows reuse the same code for batch processing.

Figure 17 shows a typical Spark architecture [508].

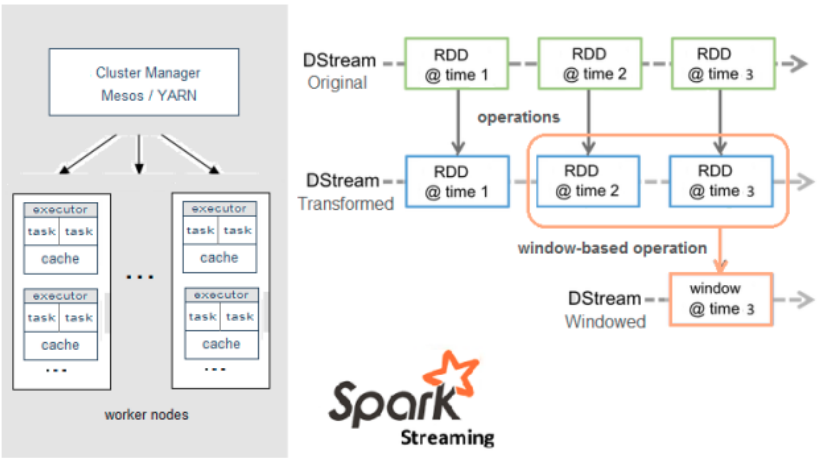

Figure 17. Typical Spark architecture [508].

D.3.4.3. Comparison between Storm and Spark

For what concerns streaming analysis, the main difference between Storm and Spark resides in the different processing models implemented. Storm processes the incoming pieces of information (tuples) one at a time and this leads to a lower latency but also to a lower throughput compared to Spark which is based on the mini-batch analysis paradigm. In a nutshell, a mini-batch is a collection of tuples arrived at the systems during a (short) time window. This approach raises the latency to the order of seconds.

Moreover, in Storm each tuple is acknowledged back to the source implementing the at-least-once fault tolerance mechanism. This means that Storm cannot avoid considering twice the same tuple while recovering from a fault. Spark, instead, tracks the data at batch level guaranteeing to analyse each batch exactly once. From the programming languages point of view, Storm is developed in Clojure [352], while Spark in Java and Scala [433]. Storm natively supports a larger set of languages but many third-party projects aiming at extending Spark-supported languages are available. Storm is undoubtedly more widely adopted in real-life deployments, whereas only one company officially uses Spark in production. Finally, although both solutions can run on their own and on Mesos clusters [15], Spark also supports YARN [18] natively.

This short comparison, summarised in the Table 22, is inevitably partial and circumstantial. The performance, for example, greatly depends on the version and configuration of the tools. Moreover, recently Storm introduced Trident [417], which is a solution for the mini-batch processing that implements many features that are typical of Spark.

Table 22: Storm vs Spark comparison.

	Storm [19], [507]	**Spark** [411]
Stream type	Stream processing (one record at time)	Mini-batching
Latency	Less than a second	Seconds
Fault tolerance mechanism	At least once (may have duplicated)	Exactly once
Implementation language	Clojure	Scala and Java
Directly supported languages	Java, Scala, Clojure, Python, Ruby [103]	Java, Scala, Python
Production deployments	Many	Only one
Resource Management	Mesos	YARN, Mesos
Throughput	10k records/s/node	400k records/s/node

D.3.4.4. Apache Samza

Apache Samza [510] is a distributed stream processing framework. It makes use of Apache Kafka [21] and Apache Hadoop YARN [18].

Features:

- Simple API: provides a very simple callback-based ‘process message’ API comparable to MapReduce.
- Managed state: Snapshots and restoration of streaming processors’ state.
- Fault tolerance: When a cluster fails, tasks are transparently migrated.
- Durability: No messages are ever lost.

- Scalability: Partitioned and distributed.
- Pluggable.
- Processor isolation: Hadoop's security model and resource isolation through Linux CGroups.

Figure 18 shows a typical Samza architecture [508].

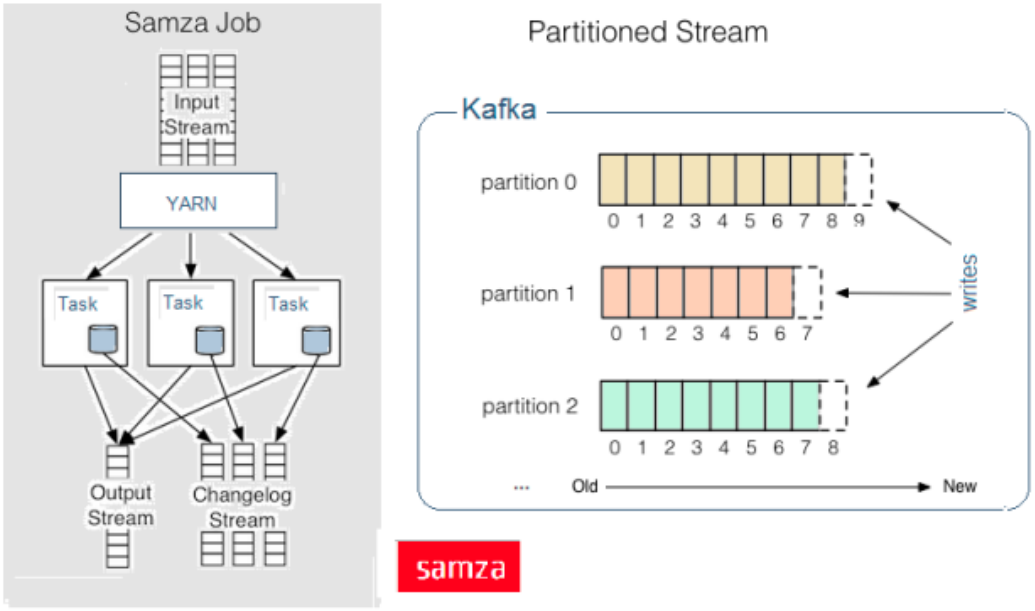

Figure 18. Typical Samza architecture [508].

D.3.4.5. Other solutions

Other open source solutions not as mature as the above ones:

- Hadoop streaming [511].
- Apache S4 [512].
- Streamparse [513].
- Stratio [514].
- Summingbird [515].
- Facebook Scribe [516].
- LinkedIn Pinot [517].

Comparison matrix of Streaming solutions discussed in detail in the sections D.3.4.1-D.3.4.4 is presented in the Table 23.

Table 23: Comparison of streaming processing frameworks [508], [518]-[520].

Technology	Apache Storm	Apache Spark	Apache Samza
Delivery semantics	at-most-once at-least-once exactly-once (with Trident)	exactly-once	at-least-once
State management	stateless, stateful (with Trident)	stateful	stateful
Latency	sub-second	seconds	sub-second
Language support	Java, Clojure and any non JVM language	Scala, Java, Python	Java, Scala, only JVM languages
Maturity	***	**	**
Ease of use	**	***	*
Originates from	Twitter	AMPLab	LinkedIn
Companies using it	Groupon, Twitter, Yahoo!, Spotify, Yelp	Alibaba Taobao, Amazon, Autodesk, Ebay, Yandex	LinkedIn, Intuit, MobileAware, Fortscale
Stream source (S)	spouts	receivers	consumers
Stream primitive (P)	tuple	Dstream	message
Stream computation (C)	bolts	transformations windows operations	tasks

D.3.5 Message queues

- Apache Kafka [21]
- Apache ActiveMQ/Apollo [521]
- RabbitMQ [499]

D.3.6 General characteristics

D.3.6.1. Key monitoring metrics

- Latency, as the difference between the time message enter the system until it contribute to the final results
- 24x7 streaming
- I/O troughput
- Maximum delivery time

D.3.6.2. Main quality assurance challenges

Real-time processing is the next large step in Big Data analysis. Main difficulties have to be with the ease of configuring and deploying testing environments and make them ready to be deployed in production without breaking the streaming flow.

D.3.6.3. Reliability

Some streaming processors guarantee that each message will be fully processed. This is the case of Storm [522].

Streaming processors are fault-tolerant: when workers die, they will be automatically restarted. If a node dies, the worker will be restarted on another node [523].

D.3.6.4. Scalability

Streaming processing solutions are inherently parallel and run across a cluster of machines. Different parts of the processors can be scaled individually by tweaking their parallelism even on the fly [524], [525].

D.3.6.5. Efficiency

Streaming processing solutions can process more than 100 000 messages per second per node.

D.3.6.6. Privacy

Usually privacy and data protection is provided by other collateral components. For example, Apache Ranger delivers a comprehensive approach to security for a Hadoop cluster. It provides central security policy administration across the core enterprise security requirements of authorisation, audit and data protection [526], [527].

D.4. NoSQL

Traditionally, the Relational DataBase Management Systems (RDBMS) were the main type of databases in use in the application. With the growth of the level of distribution of both storage and computation, and with the ever growing amount of data, the NoSQL (Not only SQL) database model became more suitable for certain data-intensive use cases. For example, Facebook, Google or LinkedIn support millions of connection in each seconds. The data does not exhibit the strong structural and relational requirements, but instead requires a quick and simple scaling up to handle increase of the data sources and the growth of the users accessing the data.

NoSQL databases are data storage solution suitable for building very large and scalable web application that have to deal with analysing and moving huge data sets. NoSQL refers to those distributed databases that avoid the rigidity of the relational model in favor of other kind of data models. The reason to move to a NoSQL database arises from a mismatch between the needs of modern DIAs compared to the following characteristics of relational databases:

1. Rigidity of the data model. In a relational data model, data have a precise structure and all those belonging to a certain relation have to share exactly the same structure. This means that, for instance, it is difficult to represent, as part of an e-commerce application, situations in which we have a catalog of heterogeneous products all having a size, but where the way to represent such size can vary from by height, width and depth for one item and radius for others.
2. Difficulty of handling large quantities of data. Relational databases have been designed and built to handle data that stay within a few Gigabytes. When considering Terabytes and Petabytes, which are the targets for DIAs, they fail and should be replaced by simpler systems that make less strict assumptions about the ACID properties of data.

The main kinds of not relational data models offered by NoSQL are the following:

- *Key value*: each data item is identified by a unique key that is used to retrieve it. The actual data item is stored as a sequence of bytes and is opaque to the NoSQL database.
- *Document-based*: this model is an evolution of the key-value one. Each data item is still univocally identified by a key, but the internal structure of the data is known by the database and it is not fixed and equal for all data items.
- *Column-based*: these databases owe their name to the data model proposed by Google BigTable paper [528]. Data are stored inside structures named Columns, which, in turn, are contained inside Column Families, and are indexed by Keys.

- *Graph-based*: in this case the data model follows the graph theory and allows adjacent nodes to be directly connected one with the other. Such kind of data model is often adopted to represent highly interconnected structures such as social networks.

The data models described above are quite different from each other, but generally aim at simplifying the organisation of data to make them more suitable for distribution. Distribution and replication are in fact the two mechanisms to enable scalability of the database with respect to the growth of the quantity of data to be handled. More specifically, replication of data on different nodes increases tolerance to faults and network partitions. Distribution of data allows the processing work to be split among different nodes so that it becomes more manageable while the size of the whole data set grows. Distribution of data occurs by partitioning them horizontally or vertically. Horizontal partitioning, also called sharding, means that different data items can reside in different nodes of a database cluster. Vice versa, vertical partitioning applies to columns-based databases and means that data distribution occurs by column.

In the field of distributed databases the CAP theorem [529] state that it is impossible for a distributed database system to achieve at the same time the three properties of *Consistency*, *Availability* and *Partition tolerance*, but just two of them can be guaranteed simultaneously. While classical centralised relational databases aim at guarantee the consistency and availability properties, NoSQL databases tend to privilege availability and partition tolerance. More specifically, they typically fulfill the BASE requirements (basic availability, soft state, eventually consistent).

D.4.1 High level architecture of a NoSQL database

A typical architecture for a distributed NoSQL database is the one shown in Figure 19. The Figure highlights the presence of different distributed nodes each one managing its local storage. Such nodes are typically coordinated by a master node that receives the requests from clients and propagates them to the others. Other nodes are used to manage the life-cycle of the others.

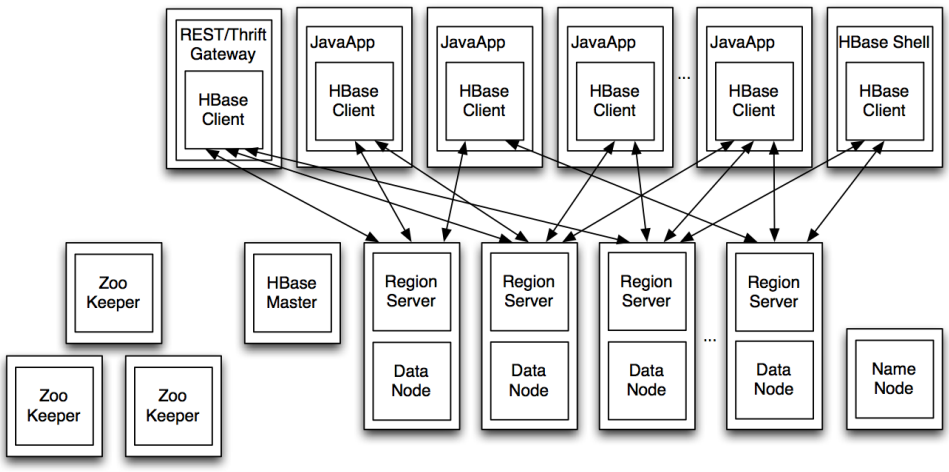

Figure 19. Architecture of a distributed NoSQL. This Figure refers to the HBASE architecture.

D.4.2 List of public cloud offerings available for this technology

Table 24 lists some of the DaaS (Database as a Service) available on the market. While their quality e.g., in terms of number of entities written or retrieved in the time unit, or of possible parallel requests handled is of crucial importance for an application developer [529], still such information is not available to users of such services.

Table 24: Public cloud offerings for NoSQL (Database as a Service).

Name	Data model	Cost
Amazon DynamoDB [531]	Key-value and Document	*Free usage tier.* The prices below depend on the region. *Provisioned Throughput Capacity* • Write Throughput: less than $0.01/hour for every 10 units of Write Capacity (approx. 36,000 writes/hour); • Read Throughput: less than $0.01/hour for every 50 units of Read Capacity (approx. 180,000 strongly consistent reads/hour, or 360,000 eventually consistent reads/hour). *Indexed Data Storage* • First 25 GB stored/month is free; • less than $0.4/GB/month thereafter *Reserved Capacity* • depends on the time; less than $0.0150 hour for 100 Write/Read Capacity Units *Data Transfer IN*: free *Data Transfer OUT*: • first 1GB/month free of charge • Up to 10 TB/month: $0.09/GB • lower price for more transfer
Amazon SimpleDB [532]	Column-based	*Free usage tier* *Machine Utilisation* • First 25 Amazon SimpleDB Machine Hours consumed per month are free; • $0.2/Amazon SimpleDB Machine Hour consumed thereafter *Data Transfer IN*: free *Data Transfer OUT*: • first 1GB/month free of charge • Up to 10 TB/month: $0.25/GB • lower price for more transfer *Structured Data Storage* • first 1GB/month free of charge • less than $0.4/GB/month thereafter
Redis Labs [533]	Key-value	There are available 10 types of virtual machines from different providers, for each of them certain plan costs are available depending on the features provided.
Google Cloud Datastore [534]	Column-based	There are two types of costs: free cota/day, respectively prices if one exceeds the quotas *Stored Data*: • free 1GB total limit; • after $0.18/GB/month *Read Operations*: • free 50; • after $0.06/100k operations *Write Operations*: • free 50k; • after $0.06/100k operations *Small Operations*: • free 50k; • after free
Azure DocumentDB [535]	Document	It is billed based on the number of collections contained in a database account. There are three types of collections S1, S2 and S3: • *S1*: SSD Storage 10GB; Requests Units 250/sec.; Scale out limits up to 100; SLA 99.95%; $0.034/hr (~$25/mo) • *S2*: SSD Storage 10GB; Requests Units 1000/sec.; Scale out limits up to 100; SLA 99.95%;$0.067/hr (~$50/mo)

		• *S3*: SSD Storage 10GB; Requests Units 2500/sec.; Scale out limits up to 100; SLA 99.95%;$0.134/hr (~$100/mo).
Azure Redis Cache [536]	Key-value	It gives the ability to use a secure open source Redis cache, managed by Microsoft, to build highly scalable and responsive applications by providing you super-fast access to your data. It is offered in two tiers: • *basic* - a single cache node (ideal for development/test and non-critical workloads); • *standard* - replicated cache in a two-node Primary/Secondary configuration; automatic replication between the two node; high-availability SLA. There are six configuration types,two of them being, for example: • *C0*: cache size 250MB; basic $0.022/hr (~$16/mo); standard $0.055/hr (~$41/mo) • *C6*: cache size 53 GB; basic $0.84/hr (~$625/mo); standard $2.10/hr (~$1,562/mo)
Azure Storage [537]	Multiple	Azure has four types of storage, each coming with two options *standard*, respectively *premium*. We detail here the prices for the standard storage. • *block blobs*, suitable for streaming and storing documents, videos, pictures, backups, and other unstructured text or binary data. For example, for the first 1 TB / Month, prices are starting from $0.024/GB for LRS (Locally redundant storage - maintaining three copies of your data) and gets to $0.061/GB for RA-GRS (Read access geo-redundant storage - replicating the data to a secondary geographic location, and also providing read access to the data in the secondary location). In this category one can buy storage capacity up to Over 5,000 TB / Month • *page blobs* and *disks*, suitable for random read and write operations (VHD images). For example, for the first 1 TB / Month, prices are starting from $0.05/GB for LRS and gets to $0.12/GB for RA-GRS. In this category one can buy storage capacity up to over 5,000 TB / Month. • tables and queues; tables offer NoSQL storage for unstructured and semi-structured data (suitable for web applications, address books, and other user data). Queues provide a reliable messaging solution for your apps. For example, for the first 1 TB / Month, prices are starting from $0.07/GB for LRS and gets to $0.12/GB for RA-GRS. In this category one can buy storage capacity up to over 5,000 TB / Month. • *files (preview)*: suitable for sharing the files between applications running in the virtual machines using familiar Windows APIs or file REST API. For LRS option, the price is $0.04/GB. Their cost depends on how much one stores, the volume of storage transactions out data transfer, which data redundancy option is chosen.

MongoLab [538]	Document	There exist three types of predefined databases for various needs:
		1. Sandbox - for development and prototyping, free
		2. Shared - for small datasets and light workloads, $15 and more
		3. Dedicated - large datasets and demanding workloads, $180 and more
		There exists also other plans, the machines can be chosen to be hosted by different providers (Amazon, Google, Microsoft Azure):
		• dedicated cluster plans
		  ○ standard, e.g. RAM 68GB SSD 700GB $3520/month
		  ○ high storage, e.g. RAM 68GB SSD 1TB $3790/month
		  ○ high performance, e.g. RAM 61GB SSD 640GB $5890/month
		• dedicated single-node plans, e.g. RAM 68GB SSD 700GB $2045/month

Rackspace [539]	Redis – Key-value	*Redis*. There are three options for using Redis at Rackspace: Managed Cloud, Redis as a Service and Private Cloud. For example, using Redis as a Service, one should pay from $59 for 500MB up to $7499 for 100GB.

	MongoDB - Document	*MongoDB*. The same three usage options as above. For example, using MongoDB as a Service, one should pay from $19 for 1GB/month and customized price for storage in the interval 100GB - 10PB.

D.4.3 Open source solutions

List of open source solutions for adoption of NoSQL technology in private clouds is presented in the Table 25:

Table 25: Open source solutions for NoSQL.

Name	Website	Data model	Chef Cookbook
Apache HBase	http://hbase.apache.org/	Column-based	Version 0.1.0
Cassandra	https://cassandra.apache.org/	Column-based	Version 0.2.4
Hypertable	http://hypertable.org/	Column-based	Version 0.3.2
Accumulo	http://accumulo.apache.org/	Key-value	Not found
Cloudata	https://github.com/gruter/cloudata	Column-based	Not found
MongoDB	http://www.mongodb.org/	Document	Version 0.16.2
CouchBase Server	http://www.couchbase.com/	Document	Version 1.3.1
CouchDB	http://couchdb.apache.org/	Document	Version 2.5.2
RethinkDB	http://www.rethinkdb.com	Document	Version 0.1.0
SequoiaDB	http://www.sequoiadb.com/en/index.php?p=index&j=2	Document	Not found
RavenDB	https://github.com/ravendb/ravendb	Document	Not found
Riak	http://riak.basho.com/	Key value	Version 3.1.0
Redis	http://redis.io/	Key value	Version 3.0.4
Aerospike	http://www.aerospike.com/	Key-value In-memory	Version 0.0.12
LevelDB	https://github.com/google/leveldb	Key value	Version 1.18.0
Berkeley DB	http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/	Key value	Not found
Oracle NoSQL Database	http://www.oracle.com/technetwork/database/database-technologies/nosqldb/overview/index.html	Key value	Not found
Dynomite	https://github.com/moonpolysoft/dynomite/wiki/dynomite-framework	Key value	Version 0.2.2
HamsterDB	http://hamsterdb.com/	Key value	Not found

Table 25 lists in column ‘Chef Cookbook’ the version supported by Chef [49] recipes.

D.4.4 Quality Assurance

D.4.4.1. Key monitoring metrics

There is not much work in the literature aiming at assessing the performance of NoSQL. Yahoo has defined in 2010 a benchmarking framework by identifying a set of workloads useful to analyse the performance of NoSQL [540]. A very recent application of such benchmarks is reported in [541]. Another work going in this direction is described in [529]. In general, the metrics being considered are the following:

- Throughput by workload: number of CRUD (Create, Read, Update and Delete) operations performed on the database in the time unit, considering different types of workloads.
- Latency by workload.
- Support to parallel connections: that is, ability of the database to support various client connections without cutting them or showing unforeseen failures.

D.4.4.2. Main quality assurance challenges.

As already mentioned, the Brewer's CAP Theorem [529] states that in a distributed system it is impossible to have at the same time availability, consistency and partition tolerance satisfied at the same level. NoSQL databases tend to privilege availability and partition tolerance.

While most NoSQL databases and, in particular, the column-based ones have been developed to support high scalability with respect to the growth of data, they delegate to the developer most of the activities related to the optimisation of CRUD operations. For instance, if a developer wants to extract from a NoSQL the data items fulfilling a certain complex condition, most likely, this developer will have to extract and filter all data in the database in order to find the ones that are interesting to him/her. If such extraction and filtering operations are programmed in a non-optimised way, all the advantages offered by NoSQL in terms of scalability may be lost at the application level because of the execution of many operations on such data.

D.4.4.3. Reliability

NoSQL stores use data sharding and replication by design. Several configuration options allow database administrators to tune between performance and reliability, such as: number of replica (shards), number of writes to consider a write operation successful, number of reads to consider a read operation successful etc.

D.4.4.4. Efficiency

Unlike relational database management systems that were designed initially as monolithic systems, NoSQL stores are designed upfront as distributed systems. They support scale-out approach; they allow ease addition or removal of nodes in/from the storage nodes cluster.

D.4.4.5. Safety

Database NoSQL have been built for offering high performance and for being available also in the presence of network partitioning. However, they have not been developed for meeting hard-deadlines.

D.4.4.6. Privacy

NoSQL has not been designed with security as a priority. Many NoSQL products allow and even recommend the use of a ‘trusted environment’ assuming that only trusted machines can access the database's TCP ports. SSL connections are another measure to be enabled in order to offer better protection at network layer.

At the application layer, products such as Riak [542] and MongoDB [543] already include support for users’ authentication and authorisation, similarly to relational database management systems. To conclude,

security is a shared responsibility between level of protection offered by the product and the deployment of the product within an organisation and it safe to say that the latest versions of NoSQL software are no less-secure than their relational systems.

D.4.5 Models

We are not aware of general meta-models focusing on NoSQL. In [544] a meta-model specifically developed to support migration between column-based NoSQL of different vendors is defined.

D.4.5.1. QoS prediction models.

QoS research for NoSQL databases focuses on performance in terms of response times and throughput, availability and scalability and consistency guarantees. Most studies that address performance, availability and scalability are benchmarking studies with limited work in modelling, e.g. [545] and utilisation of models in feedback loops, e.g. [546]. Research into predicting consistency guarantees, i.e., the probability of retrieving stale data from quorum-like NoSQL databases, has been studied using Monte Carlo simulation in [547].

D.5. Software-Defined Networking

Big Data systems are usually massively parallel and require at some stage of the computation to transfer partial results from one node to another (e.g. the copy shuffle phase of Hadoop and Spark). Data transfers could become the bottleneck of the computation [548] and Software-Defined Networking (SDN) solutions are widely adopted within data centres nowadays to configure the underlying networking infrastructure to cope with such issues.

SDN is a new network paradigm based on the decoupling of the Control Plane from the Data Plane to make computer networks more programmable [549]. SDN provides a global vision of all network nodes state inside a logically centralised controller. Now programmers have the opportunity to easily create applications that before had to be taught and developed over multiple nodes in a distributed way. All the nodes had to be configured one by one in order to obtain the desired global network behaviour. It is much easier reasoning over a complete network graph rather than on single nodes.

In the past, network devices were closed, proprietary, vertically-integrated systems with the drawback that third-party programmes could not easily manage them. In addition the configuration interfaces varied from vendor to vendor and even from device to device of the same vendor.

Through the creation of a software-based application, operators can control in an easier way the infrastructure, having the ability to customise, optimise and deploy new services without having to upgrade the underlying hardware.

D.5.1 Architecture

SDN architecture consists of three main layers: 1) Data plane, 2) Control plane and 3) Application layer. Data plane includes the elements of the network infrastructure (physical and virtual switches). These devices implement the forwarding behaviour dictated by the controller, which installs forwarding rules through an abstract interface.

The Network Operating System (NOS) lies in the control plane. NOS is the intermediate layer between the underlying network and the application layer. It gives a consistent abstract view of the network global state and offers an interface for controlling the network to the application layer. The NOS acts like a standard computer OS and abstracts the resources of the whole network to the applications executed on top of it.

The application layer is placed at the top of the SDN stack. It contains applications that provide network services. The above-defined abstractions allow third-party developers to easily deploy new services in different heterogeneous networks (data centres, WAN, mobile networks, etc.).

To allow the communication between the layers, well-defined interfaces are used:

- Southbound API: for the communication between the controller and the network infrastructure (e.g. OpenFlow API).
- Northbound API: for the communication between the network applications and the controller.
- EastWest API: In the case of a multi-controller-based architecture, this interface manages interactions between the various controllers.

Table 26 reports the most widely adopted open source SDN solutions that support OpenFlow [550], the most used protocol for managing network devices. Two examples of SDN for WANs inter-data center are B4 [551] and SWAN [552], deployed by Google and Microsoft respectively.

Table 26: Open source SDN solutions supporting OpenFlow.

Name	Description
OpenDaylight [553]	A collaborative open source project hosted by The Linux Foundation. Its goal is to accelerate the adoption of SDNs and create a solid foundation for Network Functions Virtualisation (NFV).
OpenContrail [554]	An Apache 2.0-licensed project that is built using standards-based protocols and provides all the necessary components for network virtualisation–SDN controller, virtual router, analytics engine, and published northbound APIs. It is the open source version of Juniper Networks Contrail controller
Floodlight [555]	The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller. It is supported by a community of developers including a number of engineers from Big Switch Networks.
Ryu [556]	A component-based SDN framework. Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications.
FlowVisor **Error! Reference source not found.**	FlowVisor is an experimental SDN controller that enables network virtualisation by dividing a physical network into multiple logical networks.
ONOS **Error! Reference source not found.**	ONOS is the Open Network Operating System developed by the Open Networking Lab (ON.Lab) from Stanford and UC Berkeley. Although ONOS is now aimed at carrier networks, ON.Lab intends it as an alternative to the vendor-driven OpenDaylight initiative and hopes to extend it to enterprises in the future.

D.5.2 Quality Assurance in Software Defined Networking

Traffic engineering (TE) is an important mechanism to optimise the performance of a data network at both traffic and resource level. SDN allows for centralised visibility of global network topology and status, thus providing great incentive for new TE techniques. SDN TE mechanisms mainly focus on four areas: flow management, fault tolerance, topology update and traffic analysis **Error! Reference source not found.**. Flow management deals with the process of installation of new forwarding rules: when a flow does not

match any rule in the switch, its first packet is sent to the controller that will install new forwarding rules. This process can introduces delay and significant overhead if a significant number of n*ew flows arrive at switches. Some solutions exploit wildcards rules to reduce the control plane load* **Error! Reference source not found.**-**Error! Reference source not found.**, *others deploy multiple controllers architecture to obtain load-balancing* **Error! Reference source not found.**-*[568]*.

Fault tolerance is a mandatory requirement to ensure network reliability: networks should detect and react to a failure transparently and without affecting user experience. Fault tolerance is needed for both Data Plane and Control Plane. For the data plane some solutions such as [569] and **Error! Reference source not found.** are based on a restoration approach (backup path rules are installed reactively by the controller), while others such as **Error! Reference source not found.** and [571] have a proactive approach (protection paths are pre-computed and pre-installed, so no additional switch-controller signalling is required). Control plane reliability is handled by replicating the controller with a primary-backup approach **Error! Reference source not found.** or by means of a distributed cluster of controllers **Error! Reference source not found.**, **Error! Reference source not found.**.

Topology update mechanisms are related to planned network policies changes: updates must guarantee consistency so that each individual packet or flow is not handled by a conflicting mix of old and new policy.

Some approaches modifies packet's header fields with an ID of the policy and temporarily keeping both the new and old flow rules **Error! Reference source not found.**, **Error! Reference source not found.**, others are time-based **Error! Reference source not found.**, **Error! Reference source not found.**.

Finally traffic analysis tools aim to supply instruments to monitor traffic, collect statistics at different aggregation levels, check network invariants and debug programming errors.

Some approaches are active and based on polling from the controller **Error! Reference source not found.**, **Error! Reference source not found.**. Others are passive and based on the analysis of control messages **Error! Reference source not found.**. Some others exploit dedicated servers for monitoring traffic and triggering updates to the controller **Error! Reference source not found.**.

D.5.3 Current Issues

Networks consist of a wide variety of devices, and each time a new network function has to be developed, network administrators must express policies through a tedious box-by-box configuration, dealing with a multitude of protocols and vendor-specific interfaces. In the next paragraphs is explained why SDN simplifies the network management and how is possible to avoid the usage of low level APIs.

D.5.4 Controllers architecture and low levels APIs

In SDN, the presence of a unique logically centralised controller simplifies and redefines how to manage networks, letting the applications to be network aware.

The controller is in charge of both providing an abstract view of the network topology to the application layer and of managing and communicating the applications requirements down to the dataplane layer **Error! Reference source not found.**.

OpenFlow [550], API reference of SDN, is a standard open interface which allows the communication between the controller and the dataplane layer (southbound API). The controller uses the OpenFlow messages to install rules, query traffic statistics and learn the network topology.

While SDN makes it possible to programme the network, it does not make it easy. Since OpenFlow is a low level APIs, a programmer is forced to write application in a low level language by handling each single packet field in each single network device.

D.5.5 Languages and network policies

Openflow low level API makes difficult also the definition of policies, since a programme must both take into account the shared rule-table space and manipulate single packet fields.

Several high level languages (Pyretic **Error! Reference source not found.**, Merlin **Error! Reference source not found.**, etc.) were born to encourage programmers to focus on how to specify a network policy at a high level of abstraction, rather than how to implement it using low-level OpenFlow mechanism.

One of the disadvantages of programming using OpenFlow is the non-modularity. Today, controllers allow programming only monolithic controller applications bringing to an increasing complexity in debugging and testing. Thanks to the presence of the northbound programming interface, it is possible to define a module for each control application. Each policy can be developed independently and the interactivity between them is orchestrated by runtime software, a compiler that lies in the controller. Each policy is compiled and transformed in specific low-level forwarding rules to be installed in the switches, avoiding each kind of collision between them.

High level languages allows to define policies operating on an abstract view of the network: for example multiple underlying switches can be abstracted as a single derived virtual switch or, alternatively, one underlying switch can be represented as multiple derived virtual switches. The compiler will produce low-level rule according to the actual topology.

D.5.6 SDNs in DICE

The integration of SDN technologies into Big Data applications allows a more accurate control of network resources than traditional switching technologies. This can be incorporated at design time by specifying requirements for the Data-Intensive Application and at deployment time reserving the necessary network resources for the specific data flows in the different phases of the processing, ensuring predictable quality levels expected from the network.

D.6. Cloud-based blob storage

D.6.1 Ceph

Ceph is an open source distributed storage technology for efficient handling of Big Data requirements [587]. Ceph presents object, block, and file storage from a single distributed computer cluster and supports data replication for fault tolerance. Ceph’s object storage system allows users to mount Ceph as a thinly provisioned block device, automatically stripes and replicates the data across the cluster. Ceph’s distributed block storage distributes the storage requirements of VMs across a large number of storage devices as opposed to centralised Storage Area Network (SAN) and Network-Attached Storage (NAS) solutions. This way, each VM disk data is stored across many (at least 2) different smaller storage devices called 'replicas'. If one replica is lost, the VM’s disk will not be corrupted or the data lost. Ceph’s file system and Ceph metadata server cluster provide a service that maps the directories and file names of the file system to objects stored within expandable Reliable Autonomic Distributed Object Store (RADOS) [588] clusters ensuring high performance and avoidance of heavy loads. This makes Ceph a relevant and apt solution for the handling of large data sets. Ceph supports a RESTful API that is compatible with the data access model of the Amazon S3 API.

D.6.1.1. Typical architecture of CEPH

Ceph clusters are made up of machines running the following types of services:

- Monitors (MON): these maintain a map of the current state of the Ceph cluster.
- Object Storage Devices (OSD): these store the data.
- Metadata server (MDS): these maintain metadata on the placement of files in the cephfs filing system.
- Auxiliary services, such as RADOS gateways to S3 and SWIFT.

An example of a Ceph configuration can be found in Figure 20 [588].

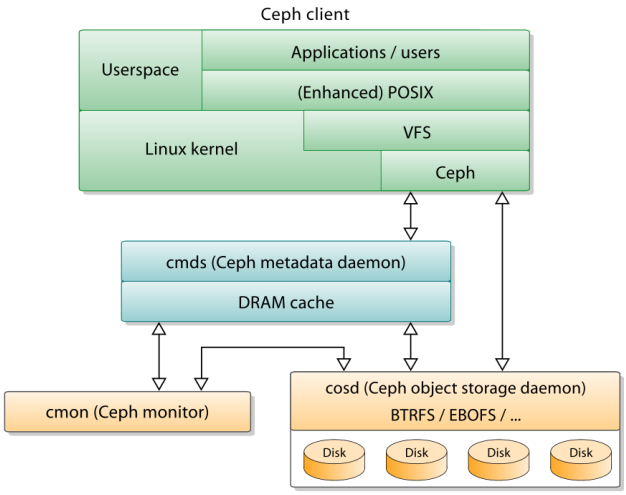

Figure 20. Ceph integration example [589].

D.6.1.2. Cloud offerings using CEPH

Flexiant Cloud Orchestration (FCO) [180] is a cloud orchestration software solution and allows service providers to manage their own public, private or hybrid cloud solution. FCO supports the use of CEPH as a storage solution for the node data storage, however pricing and quality will depend on the service provider.

OpenStack [188] is a cloud operating system that allows control of compute, storage, and networking resources throughout a datacenter. OpenStack supports the use of CEPH as a storage solution, however pricing and quality will depend on the datacenter operator [590]. An example of a public cloud offering using OpenStack and Ceph is Dreamhost.

Dreamhost is a public cloud company that uses OpenStack and Ceph for their cloud and storage solution. Currently Dreamhost offers multiple options for its storage solution. These can be found in the Table 27 [591].

Table 27: Dreamhost pricing storage [591].

Storage Included	Monthly Price	Effective Price/GB
40 GB	$0.95	2.38¢
200 GB	$4.50	2.25¢
1,024 GB	$19.95	1.95¢
2,048 GB	$34.95	1.71¢
20,480 GB	$299.95	1.46¢

D.6.1.3. Open Source solutions

Ceph is an open source project [592] and can be easily adapted to any cloud environment.

D.6.1.4. Quality assurance

D.6.1.4.1. Key monitoring metrics

- Available Disk Space.
- IOWait.
- Read Operations per second.
- Write Operations per second.
- Network Throughput.
- Cluster health.
- OSD status.

D.6.1.4.2. Quality assurance challenges for CEPH literature.

Distributed block storage is a newer technology than centralised storage. Arguably, it is more difficult to administer, and carries the risks of being less well tested in mainstream production environments. In particular, tuning it for performance can be more difficult. However, if deployed properly it offers enhanced scalability, can be very economical and can perform as well as a centralised storage solution. CEPH does this by using CRUSH, a data distribution function. CRUSH enforces data replica separation for improved data safety [593].

D.6.1.4.3. Reliability and high-availability support for CEPH

Ceph's foundation is the Reliable Autonomic Distributed Object Store (RADOS), which provides applications with object, block, and file system storage in a single unified storage cluster. Ceph's RADOS provides scalability that can handle thousands of client hosts or KVMs accessing Petabytes to Exabytes of data.

D.6.1.4.4. Scalability and performance support for CEPH

To provide scalability Ceph uses the CRUSH algorithm [594]. This algorithm helps liberate storage clusters from the scalability and performance limitations imposed by centralised data table mapping as it replicates and re-balances data within the cluster dynamically and delivers high-performance and infinite scalability.

D.6.1.4.5. Privacy and data protection with CEPH

Data security is typically Cloud provider-dependent. However, Ceph does offer some data security by default. Ceph provides a cephx authentication system, which authenticates users operating Ceph clients. With Ceph a user contacts a monitor that is used to authenticate them and distribute keys. The monitor returns an authentication data structure that contains a session key for use in obtaining Ceph services. This key is self encrypted and only the user can request services from the Ceph monitors.

The client then uses the session key to request its desired services from the monitor, and the monitor provides the client with a ticket that will authenticate the client to the OSDs that actually handle data. Ceph monitors and OSDs share a secret, so the client can use the ticket provided by the monitor with any OSD or metadata server in the cluster. This form of authentication will prevent attackers with access to the communications medium from either creating bogus messages under another user's identity or altering another user's legitimate messages, as long as the user's secret key is not divulged before it expires [595]. The users can grant or revoke access to objects or buckets for other users in an S3 compatible way.

D.6.2 Amazon Simple Service Storage (Amazon S3)

D.6.2.1. Overview

Amazon S3 (Simple Storage Service) [596] is an online file storage web service offered by Amazon Web Services (AWS) [192]. Amazon provides storage through web services interfaces as well as through the use of their APIs (REST, SOAP) [596].

The main benefits of S3 and why it has grown in popularity is due to its low storage cost, redundancy and unlimited storage capacity. S3 is typically used to back data off site, storage for large amounts of data generated or even hosting static websites.

S3 stores data as objects within 'buckets.' These buckets are unique to an AWS account and are identified by a unique, user-assigned key. Objects are stored with buckets and can store as many objects as required. They can either be accessed using the web interface or API to write, read and delete objects. These Objects can be up to 5 Terabytes in size, however within a single PUT the object can only be a maximum of 5GB.

Data stored within S3 is stored on redundant servers in multiple data centres and can be publicly accessible if required. One example in which this is used is for the hosting of static websites.

D.6.2.2. Public cloud offerings of S3

AWS is the only public cloud that offers this technology. A full breakdown of pricing can be found in Table 28 and Table 29 [597] (note these prices are the US Standard pricing and may change depending on AWS region used).

Table 28: AWS Storage Pricing US Standard [597].

	Standard Storage	Reduced Redundancy Storage
First 1 TB / month	3.00¢ per GB	2.40¢ per GB
Next 49 TB / month	2.95¢ per GB	2.36¢ per GB
Next 450 TB / month	2.90¢ per GB	2.32¢ per GB
Next 500 TB / month	2.85¢ per GB	2.28¢ per GB
Next 4000 TB / month	2.80¢ per GB	2.24¢ per GB
Over 5000 TB / month	2.75¢ per GB	2.20¢ per GB

Table 29: Data Transfer Pricing US Standard [597].

	Pricing
Data Transfer IN To Amazon S3	
All data transfer in	0.00¢ per GB
Data Transfer OUT From Amazon S3 To	
Amazon EC2 in the Northern Virginia Region	0.00¢ per GB
Another AWS Region	2.00¢ per GB
Amazon CloudFront	0.00¢ per GB
Data Transfer OUT From Amazon S3 To Internet	
First 1 GB / month	0.00¢ per GB
Up to 10 TB / month	9.00¢ per GB
Next 40 TB / month	8.50¢ per GB
Next 100 TB / month	7.00¢ per GB

D.6.2.3. Open source solutions with S3

With AWS services there are no open source options as they must all be used from within an AWS account. However AWS allows S3 to be used with an on-site solution but only with the use of an AWS

storage gateway VM [598]. An example of this can be found in within section detailing on site usage with S3.

Other solutions such as Ceph provide S3 compatibility by offering a S3 compatible API, while native installation of S3 is currently not available.

D.6.2.4. Quality assurance.

D.6.2.4.1. Key S3 monitoring metrics

- Size of all objects present in bucket(s).
- Number of objects present in bucket(s).
- Get transfer speed.
- Push transfer speed.

D.6.2.4.2. Main quality assurance challenges.

AWS conduct continuous research and development of S3, but this research is not made public. External research is conducted. It looks at the S3 solution as a black box approach and focuses on pricing, transfer speeds and multi cloud reliability.

D.6.2.4.3. Configuration options for reliability and high-availability with S3

Amazon S3 Standard Storage is designed to achieve 99.999999999% durability [599]. To achieve this level of durability S3 redundantly stores objects on multiple devices across multiple facilities in an Amazon S3 Region. The service is designed to sustain concurrent device failures by quickly detecting and repairing any lost redundancy. S3 also verifies the integrity of data using checksums.

Reduced Redundancy Storage is a storage option within S3 that enables customers to reduce their costs by storing data at lower levels of redundancy than S3's standard storage offering. With RRS, data is replicated fewer times so the durability offered is only 99.99%. However, both are backed by Amazon S3's Service Level Agreement.

D.6.2.4.4. Scalability and performance for S3

The total volume of data and number of objects that can be stored is unlimited. Individual Amazon S3 objects can range in size from 1 byte to 5 Terabytes however the largest object that can be uploaded in a single PUT is 5 Gigabytes.

D.6.2.4.5. Privacy and data protection within S3

Within S3 a user can choose to make their data private or publicly accessible. Amazon state they will only track usage for billing purposes, however if required by law this data can be made available to relevant authorities. It is up-to the end user to encrypt their data before storing within S3.

D.7. In-Memory Analytics

D.7.1 Introduction

In-memory analytics is a computing paradigm aimed at processing large volumes of data in main memory, while minimising usage of disk I/O. Often, in-memory databases rely on column based representation and compression techniques to optimise processing and memory usage. Thanks to the constant decrease of DRAM costs, this approach is now mainstream in databases for analytical workloads, which can then enjoy vastly shortened response times compared to the recent past. As a result, business analytics is now in

strong demand and several companies are developing in-memory database-as-a-service offerings in the cloud.

In-memory analytics offerings rely on in-memory database technologies. Since the offerings in this area are mostly commercial, we focus on describing a specific commercial implementation, i.e., the SAP HANA platform.

D.7.2 Diagram showing a typical architecture of this technology.

N/A, this is a single box, there is no architecture. In case of high-availability architecture, standard layouts exist, e.g., [600].

D.7.3 Public cloud offerings available for this technology.

In-memory DBMS are mostly commercial and include, among others, SAP HANA, ScaleOut and Actian Matrix that are all available on Amazon AWS. In-memory extensions are also features in Oracle and Microsoft SQL databases.

D.7.4 Open source solutions for adoption of this technology in private clouds.

Open source in-memory DBMS include Apache Derby, HyperSQL (HSQLDB) and SQLite.

While Apache Spark is not an in-memory DBMS, it is an open source solution to perform some in-memory analytics.

D.7.5 Quality assurance

D.7.5.1. Key monitoring metrics.

Typical metrics include CPU Utilisation, query response times, threading level, thread affinity, mean memory consumption, peak memory consumption.

D.7.5.2. Main quality assurance challenges.

In-memory databases are a new type of DBMS that is not well understood in terms of quality assurance. While benchmarking studies are available that prove that in-memory databases can be orders of magnitude faster than traditional databases, the definition of techniques for optimal deployment, configuration and resource management of these DBMS is still an open research problem.

As noted in [601], a significant problem for quality assurance of in-memory databases is the ability to capture *time-varying threading levels* that are used internally to these databases to process queries efficiently. At present, there are limited studies available on this aspect of performance.

Another important problem is the characterisation of *peak memory consumption*, since this affect memory sizing. The goal is to minimise the impact of disk spill-off when memory consumption exceeds the available physical memory. Understanding peak memory requirements is simple testing a reference workload, but it is generally hard to predict on unobserved configurations.

Characterising and assigning *threads affinities* is also important, due to cache locality. As the threading level of a query is dynamically increased, there can be in NUMA architectures substantial latencies for jobs that reference data stored in a cache residing on a different socket.

D.7.5.3. Reliability

Some in-memory DMBS come with high-availability (HA) capabilities, such as backup & restore, disaster recovery and reliable architectures with stand-by servers and standard layouts to replicate data in order to minimise the changes of data loss.

D.7.5.4. Efficiency

Admission control allows to limit the maximum number of queries that can simultaneously execute on an in-memory database system. This is the primary control known to limit excessive resource utilisation.

D.7.5.5. Privacy

In-memory DMBS and related database-as-a-service products typically offer multi-tenancy. Multi-tenancy implies privacy and data protections, which can be enforced at various granularity levels, from separate index servers to separate tables, up to separating individual table rows.

D.7.6 Models

D.7.6.1. Meta-models.

To the best of our knowledge, no meta-models explicitly captures the properties of in-memory databases. Such an extension, in order to feed design-time reasoning, would require at least an explicit characterisation in the meta-model of memory consumption and parallelism levels for each query type, which does not appear in principle too complex. It would be instead more complex to derive an optimisation tool to decide the deployment and configuration of such systems.

D.7.6.2. Quality of Service prediction models.

A number of works have explored the problem of performance modelling of in-memory databases, such as [601] that defines a queueing network model to predict query response times and CPU utilisations. Similar works, in the arena of general DBMS, have adopted machine learning approach to predict response times that appear partly applicable to in-memory DBMS [602], [603].

Conclusion

In this deliverable, we have provided a comprehensive overview of the state-of-the-art in several areas related to the DICE project including, but not limited to: DevOps, Model Driven Engineering (MDE), Quality Engineering Methods, architecture styles and technologies for Big Data. One of the main takeaways that emerge from this deliverable is the breadth, heterogeneity and complexity of the challenges that exist in building high-quality data-intensive applications:

- First, we noticed a generalised lack of extensions to annotate, model or test the behaviour of complex software systems in terms of data usage, especially in the domain of MDE. On the one hand this is challenging, since it implies that DICE has to build a foundation in MDE to describe Data-Intensive Applications. On the other hand, having a "first mover" advantage like this increases the opportunities for impact and exploitation.

- Next, we observed that the technology stack of Big Data is very heterogeneous, and this poses the question on how a 3-years research project like DICE could produce a useful contribution. In fact, Big Data is so new and diverse that it does not seem always possible to come up with abstractions (e.g., UML annotations or forma models) that can be reused for more than 1-2 implementations of a given technology. It seems therefore more interested for DICE to focus on modelling a few technologies of clear impact and diffusion (e.g., MapReduce, Spark and Storm) rather than supporting a plethora of diverse technologies for the sake of comprehensiveness. Extensibility of the approach to other technologies should still be assessed and, where possible, supported via appropriate guidelines.

- Another major lesson that was learned by the state of the art review is the disruptive growth of DevOps in recent months, which is attractive large parts of the industry towards changing their product offering and the way they address service delivery problems. This realisation has led us to elaborate the positioning of DICE at the crossroad between Model-Driven Engineering and DevOps, as we discuss in Deliverable D1.2 "Requirements specification".

- The investigation in Chapter D has led us to assess and compare the commercial offerings and open source implementations of some popular Big data technologies. After performing this review, the consortium has defined an initial software stack for DICE to support in the different areas of interest of the project, such as Apache Hadoop, Apache Spark, Apache Storm, Amazon S3, Apache Cassandra, and Cloudera Oryx 2. The analysis in this deliverable has led us to believe that such technology stack is sufficiently broad to cover commons classes of data-intensive applications.

- Lastly, it should be remarked that the analysis performed in this deliverable has driven the definition of deliverable D1.2, where we report an extensive set of requirements for the DICE project and an overview of the general technical approach.

In conclusion, Big Data is a diverse, heterogeneous area where model-driven software engineering has a lot of potential, but where language, tool and formal model support is still lacking. This opens exciting possibilities for the DICE project to make an impact in this area, by defining a novel UML profile and tools for building Data-Intensive Applications with quality guarantees

References

[1] NIST 'Big Data Interoperability Framework: Volume 1, Definitions'. Draft Release. National Institute of Standards and Technology. April 2015.

[2] NIST 'Big Data Interoperability Framework: Volume 2, Big Data Taxonomies'. Draft Release. National Institute of Standards and Technology. April 2015.

[3] 'The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things' Vernon Turner, John F. Gantz, David Reinsel, and Stephen Minton, Report from IDC for EMC April 2014.

[4] Big Data Value Association http://www.bdva.eu/

[5] 'Exploring Data-Driven Innovation as a New Source of Growth – mapping the policy issues raised by 'Big Data'', Report from OECD 18 June 2013.

[6] Applying assumptions from the McKinsey Global Institute report 'Big Data: The next frontier for innovation, competition, and productivity', June 2011, to the European healthcare sector.

[7] 'Big Data Vendor Revenue and Market Forecast 2013-2017', article, Wikibon, February 2014.

[8] Big Data Market by Types (Hardware; Software; Services; BDaaS - HaaS; Analytics; Visualisation as Service); By Software (Hadoop, Big Data Analytics and Databases, System Software (IMDB, IMC): Worldwide Forecasts & Analysis (2013 – 2018), available online at: www.marketsandmarkets.com, August 2013.

[9] 'Worldwide Big Data Technology and Services 2013–2017 Forecast', report, IDC, December 2013.

[10] '*Big and open data in Europe - A growth engine or a missed opportunity?'* Sonia Buchholtz, Maciej Bukowski, Aleksander Śniegocki (Warsaw Institute for Economic Studies), report commissioned by demosEUROPA, 2014.

[11] Big Data Value calculation based on http://www.eskillslandscape.eu/ict-workforce-in-europe/

[12] '*The European Data Market*', Gabriella Catteneo, IDC, presentation given at the NESSI summit in Brussels on 27 May 2014, available online at: http://www.nessi-europe.eu/?Page=nessi_summit_2014

[13] IDC European Vertical Markets Survey, October 2013. http://www.idc.com/getdoc.jsp?containerId=M07W

[14] Apache Hadoop https://hadoop.apache.org/

[15] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, Shenker, S., Stoica, I. 'Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center'. In NSDI. March 2011.

[16] Marz, N. (2013). Big Data: Principles and best practices of scalable real-time data systems. O'Reilly Media.

[17] A repository dedicated to the Lambda Architecture (LA), http://lambda-architecture.net/

[18] Hadoop YARN, http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html, accessed on 08/05/2015.

[19] Apache Storm http://storm.apache.org/

[20] Apache Zookeeper https://zookeeper.apache.org/

[21] Apache Kafka http://kafka.apache.org/

[22] Fan, W. and Bifet, A. (2013). Mining Big Data: current status, and forecast to the future. ACM SIGKDD Explorations Newsletter, 14(2):1–5.

[23] Robak, S., Franczyk, B., and Robak, M. (2013). Applying Big Data and linked data concepts in supply chains management. In Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on, pages 1215–1221. IEEE.

[24] Nicolas Bär, 'Investigating Lambda Architecture', Master Thesis, August 2014, University of Zurich.

[25] SRBench benchmark http://www.w3.org/wiki/SRBench

[26] DEBS Grand Challenge 2014 http://lsds.doc.ic.ac.uk/projects/SEEP/DEBS-GC14

[27] Google Cloud Dataflow, https://cloud.google.com/dataflow/

[28] Amazon Kinesis, http://docs.aws.amazon.com/kinesis/latest/dev/key-concepts.html

[29] Amazon Elastic MapReduce, http://aws.amazon.com/elasticmapreduce/, accessed on 08/05/2015.

[30] http://lambdoop.com/

[31] Oryx 2 http://oryxproject.github.io/oryx/

[32] Oryx 1 https://github.com/cloudera/oryx

[33] Hadoop Distributed File System (HDFS) http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[34] http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

[35] http://arxiv.org/abs/1203.6402

[36] http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html

[37] R. C. Fernandez, P. Pietzuch, J. Koshy, J. Kreps, D. Lin, N. Narkhede, J. Rao, C. Riccomini, and G. Wang. Liquid: Unifying Nearline and Offline Big Data Integration. In CIDR, 2015.

[38] http://blog.acolyer.org/2015/02/04/liquid-unifying-nearline-and-offline-big-data-integration/

[39] Loukides, Mike (2012-06-07). 'What is DevOps?'

[40] Loukides, Mike (2011-05-03) 'Software Velocity 2011 - retrospective'

[41] FP7 Research and Innovation http://ec.europa.eu/research/fp7/index_en.cfm

[42] BTO Research Technical Briefing 'DevOps@Unicredit', May 2015

[43] Ant http://wiki.mxunit.org/display/default/Continuous+Integration+--+ANT+-+The+Glue+between+Jenkins+and+your+Tests
[44] Capistrano http://capistranorb.com/
[45] RPM Package Manager http://www.rpm.org/
[46] Cobbler http://cobbler.github.io/
[47] Crowbar https://crowbar.github.io/
[48] Puppet https://puppetlabs.com/puppet/what-is-puppet
[49] Chef https://www.chef.io/chef/
[50] Nagios https://www.nagios.org/
[51] Sensu https://sensuapp.org/
[52] Software Engineering Institute - Blog 'Continuous Integration and DevOps'
[53] Matt Weinberger, DevOps explained: A philosophy of speed, not momentum
[54] Gartner. 'Research Note'
[55] Hüttermann, Michael (2012). 'DevOps for Developers'. Apress.
[56] PuppetLabs, 2014. 'State of DevOps Report '
[57] Model-Driven Architecture Specification and Standardisation, The Object Management Group (OMG) - http://www.omg.org/mda/
[58] OMG Object Management Group http://www.omg.org/
[59] Model-Driven Engineering Reference Guide, An Enterprise Architect's Perspective - http://www.theenterprisearchitect.eu/blog/2009/01/15/mde-model-driven-engineering-reference-guide/
[60] MODAClouds EU http://www.modaclouds.eu/
[61] MODAClouds Documentation Portal, D4.2.1, MODACloudML Version 1 - http://www.modaclouds.eu/wp-content/uploads/2012/09/MODAClouds_D4.2.1_MODACloudMLDevelopmentInitialVersion.pdf
[62] Reuse and Migration of legacy applications to Interoperable Cloud Services - EU REMICS, Methodology - http://www.remics.eu/publicdeliverables
[63] ARTIST EU Project Homepage, Resource Gallery - http://www.artist-project.eu/vision
[64] JUNIPER EU Project Homepage, Resource Gallery - http://www.juniper-project.org/
[65] CORBA http://www.corba.org/
[66] Architecture-Driven Modernisation Taskforce, Will Ulrich, Tactical Strategy Group - http://www.omg.org/news/meetings/workshops/ADM_2005_Proceedings_FINAL/T-1_Ulrich.pdf
[67] Unified Modelling Language: Infrastructure, 2011. Version 2.4.1, OMG document: formal/2011-08-05.
[68] OMG: UML Profile for MARTE: Modelling and Analysis of Real-Time and Embedded Systems, June 2011. Version 1.1, OMG document: formal/2011-06-02.
[69] Bernardi, S., Merseguer, J., and Petriu, D.C.: A dependability profile within MARTE. Software and Systems Modelling, 10(3):313–336, 2011.
[70] J2EE Java Enterprise Edition SDK http://www.oracle.com/technetwork/java/javaee/download-141771.html
[71] .NET Framework http://www.microsoft.com/net
[72] Object Constraint Language, 2014. OMG document: formal/2014-02-03, v2.4.
[73] Avizienis, A., Laprie, J.C., Randell, B. and Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. on Dependable and Secure Computing, 01(1):11–33, 2004.
[74] Leveson, N.G.: Safeware. Addison-Wesley, USA, 1995.
[75] Bernardi, S., Merseguer, J. and Petriu, D.C.: Model-driven Dependability Assessment of Software Systems. Springer, 2013.
[76] Basin, D., Doser, J., and Lodderstedt, T.: Model Driven Security: From UML Models to Access Control Infrastructures. ACM Transactions on Software Engineering and Methodology, 15(1): 39-91, 2006.
[77] Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., and Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. on Information Systems Security (TISSEC) 4(3):224–274, 2001.
[78] Rodríguez, R., Merseguer, J. and Bernardi, S.: Modelling Security of Critical Infrastructures: A Survivability Assessment. The Computer Journal 2014. DOI :10.1093/comjnl/bxu096.
[79] Jürjens, J: Secure systems development with UML. Springer 2005, ISBN 978-3-540-00701-2.
[80] CARiSMA, https://www-secse.cs.tu-dortmund.de/carisma/index.shtml.
[81] XMI XML Meta-data Interchange http://www.omg.org/spec/XMI/
[82] Enterprise Architect https://www.sparxsystems.es/
[83] Papyrus https://www.eclipse.org/papyrus/
[84] Eclipse Public License 1.0 https://www.eclipse.org/legal/epl-v10.html
[85] Eclipse Papyrus WIKI http://wiki.eclipse.org/Papyrus_User_Guide
[86] UML Activity, Sequence, Profile Diagrams https://www.eclipse.org/papyrus/usersTutorials/usersTutorialsIndex.php
[87] UML Diagrams (and Models) with Papyrus http://lowcoupling.com/post/47802411601/uml-diagrams-and-models-with-papyrus

[88] UML Sequence Diagrams http://lowcoupling.com/post/47844944042/uml-sequence-diagrams
[89] UML State Machines http://lowcoupling.com/post/47845261252/uml-state-machines
[90] UML Use case http://www.cs.fsu.edu/~baker/swe1/restricted/notes/tutorial_papyrus/tutorial_papyrus.html
[91] UML Timing diagram http://www.eclipse.org/papyrus/project-info/new_and_noteworthy_0.10.0.php
[92] Papyrus for Real-Time Embedded Systems, Charles Rivet https://www.youtube.com/watch?v=kfD7Ejxo6dQ
[93] Modelio https://www.modelio.org/
[94] Modeliosoft https://www.modeliosoft.com/
[95] BPMN standard http://www.bpmn.org/
[96] http://rd.softeam.com/prototypes/martedesigneralfaformodelio2x
[97] MOSKitt https://moskitt.gva.es/redmine
[98] ArgoUML http://sourceforge.net/projects/argouml/
[99] Java programming language http://www.oracle.com/technetwork/java/index.html
[100] C++ programming language https://isocpp.org/
[101] C# programming language https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx
[102] PHP https://www.php.net/
[103] Ruby programming language https://www.ruby-lang.org/ru/
[104] Delphi programming language http://www.delphibasics.co.uk/
[105] StarUML version 1. FreeSoftware http://staruml.sourceforge.net/v1/index.php
[106] StarUML version 2. Commercial http://staruml.io/
[107] GNU GPL http://www.gnu.org/licenses/gpl-3.0.en.html
[108] UML Designer http://www.umldesigner.org/
[109] UML2 https://wiki.eclipse.org/MDT-UML2
[110] https://www.eclipse.org/sirius/
[111] MARTE Designer http://marketplace.obeonetwork.com/module/marte
[112] http://www.nomagic.com/products/magicdraw.html
[113] SysML http://sysml.org/
[114] UPDM http://www.omg.org/spec/UPDM/
[115] XML Schema http://www.w3.org/standards/xml/schema
[116] WSDL – Web Service Description Language http://www.w3.org/TR/wsdl
[117] BluAge http://www.bluage.com/
[118] http://www-03.ibm.com/software/products/en/ratisoftarch
[119] Sparx Systems http://www.sparxsystems.com/
[120] CEA http://www.cea.fr/english-portal
[121] Atos http://atos.net/en-us/home.html
[122] http://www.citma.gva.es/ (the website is in Spanish)
[123] Tigris.org http://www.tigris.org/
[124] Obeo http://www.obeo.fr/en/
[125] No Magic, Inc. http://www.nomagic.com/
[126] IBM http://www.ibm.com/uk/en/
[127] ActionScript http://help.adobe.com/livedocs/specs/actionscript/3/wwhelp/wwhimpl/js/html/wwhelp.htm
[128] Python https://www.python.org/
[129] Visual Basic https://msdn.microsoft.com/en-us/vstudio/ms788229.aspx
[130] VB .NET https://msdn.microsoft.com/en-us/vstudio/hh388573
[131] DDL https://msdn.microsoft.com/en-us/library/ff848799.aspx
[132] Ada http://www.adaic.org/
[133] VHDL http://www.eda.org/twiki/bin/view.cgi/P1076/WebHome
[134] Verilog http://www.verilog.com/
[135] BPEL http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
[136] Visual Studio https://www.visualstudio.com/
[137] AndroMDA http://www.andromda.org/
[138] NetBeans https://netbeans.org/
[139] EA User Guide http://www.sparxsystems.com.au/bin/EAUserGuide.pdf
[140] Modelio User Manual https://www.modelio.org/documentation/user-manuals.html
[141] Papyrus User Manual http://wiki.eclipse.org/Papyrus_User_Guide
[142] MOSKitt User Manual www.moskitt.org/eng/med/print/manuales/
[143] ArgoUML Documentation http://argouml-users.net/index.php?title=Documentation
[144] UML Designer Documentation http://www.umldesigner.org/ref-doc/usage.html
[145] http://www.nomagic.com/files/manuals/MagicDraw%20UserManual.pdf
[146] Enterprise Architect Forum http://www.sparxsystems.com.au/cgi-bin/yabb/YaBB.cgi

[147] Modelio Forum https://www.modelio.org/forum/index/6-general.html
[148] MOSKitt forum https://joinup.ec.europa.eu/software/gvcase-db/forum/all
[149] ArgoUML Forum http://argouml-users.net/forum/
[150] UML Designer Forum http://www.obeonetwork.com/group/uml-designer
[151] https://community.nomagic.com/magicdraw-f22.html
[152] Eclipse Mars project https://projects.eclipse.org/releases/mars
[153] J. Oldevik, A. Solberg , Ø. Haugen, and B. Møller-Pedersen Evaluation Framework for Model-Driven Product Line Engineering Tools,, Software Product Line, 2006, pp 589-618
[154] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for model-driven performance prediction. Journal of Systems and Software,82(1), 3-22. 2009
[155] D. Varró, and A. Balogh. The model transformation language of the VIATRA2 framework. Science of Computer Programming, 68(3), 214-234. 2007
[156] Model Editing VIATRA2 http://wiki.eclipse.org/VIATRA2/GettingStarted/Model_Editing
[157] VIATRA2 http://www.eclipse.org/viatra/
[158] UML Model transformation tool http://sourceforge.net/projects/umt-qvt/
[159] UPUPA http://sealabtools.di.univaq.it/tools.php
[160] fUML http://www.omg.org/spec/FUML/
[161] Meta-Object Facility Query-Views-Transformations Specification - http://www.omg.org/spec/QVT/
[162] AMMA - ATLAS Model Management Architecture https://wiki.eclipse.org/AMMA
[163] Atlas-Transformation Language Specification - https://eclipse.org/atl/
[164] Derek Palma and Thomas Spatzier, Topology and Orchestration Specification for Cloud Applications Version 1.0, OASIS standard, November 2013 http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html
[165] Open Source CONvention OSCON 2015 http://www.oscon.com/open-source-2015
[166] TOSCA-TC SubCommittee (SC) on interoperability https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca-interop
[167] YAML http://yaml.org/
[168] SaltStack ALT http://saltstack.com/
[169] Ansible automation tool for application deployment http://www.ansible.com/home
[170] Vagrant – tool for configuring development environments https://www.vagrantup.com/
[171] http://en.wikipedia.org/wiki/Orchestration_%28computing%29
[172] http://www.ubuntu.com/cloud/tools/juju
[173] https://jujucharms.com/docs/stable/authors-charm-components
[174] https://jujucharms.com/u/asanjar/spark/trusty
[175] http://getcloudify.org/
[176] http://getcloudify.org/2012/03/20/big-data-in-the-cloud-using-cloudify.html
[177] http://alien4cloud.github.io/index.html
[178] https://brooklyn.incubator.apache.org/
[179] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=camp#technical
[180] Flexiant Cloud Orchestrator: https://www.flexiant.com/flexiant-cloud-orchestrator/
[181] Rundeck: http://rundeck.org/
[182] CAMF: https://projects.eclipse.org/proposals/cloud-application-management-framework
[183] CELAR: https://github.com/CELAR/c-Eclipse
[184] Open-TOSCA Winery: https://projects.eclipse.org/projects/soa.winery
[185] Open-TOSCA VinoThek: http://install.opentosca.org/
[186] AGPL – GNU Affero General Public License https://gnu.org/licenses/agpl-3.0.en.html
[187] Apache license 2.0 https://www.apache.org/licenses/
[188] OpenStack – Open source software for creating public and private clouds https://www.openstack.org/
[189] Apache CloudStack – open source cloud computing https://cloudstack.apache.org/
[190] FCO Blueprints https://www.flexiant.com/2013/07/11/what-are-cloud-blueprints/
[191] FCO Triggers https://www.flexiant.com/2014/11/13/flexiant-trigger-plugin-technology/
[192] Amazon Web Services (AWS) http://aws.amazon.com/
[193] SoftLayer http://www.softlayer.com/
[194] VMware vSphere https://www.vmware.com/uk/products/vsphere
[195] VMware vCloudAir http://vcloud.vmware.com/uk/explore-vcloud-air/what-is-vcloud-air
[196] Apache Java Multi-Cloud Toolkit https://jclouds.apache.org/
[197] Flexiant Hypervisors comparison https://www.flexiant.com/2014/02/12/hypervisor-comparison-kvm-xen-vmware-hyper-v/
[198] MIKELANDGELO http://www.mikelangelo-project.eu/
[199] Docker https://www.docker.com/

[200] LXC https://linuxcontainers.org/
[201] IPMI http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html
[202] OSV http://osv.io/
[203] Martin Fowler, *Contionuous Integration*, May 2006, http://www.martinfowler.com/articles/continuousIntegration.html
[204] http://en.wikipedia.org/wiki/Continuous_integration
[205] http://www.yegor256.com/2014/10/08/continuous-integration-is-dead.html
[206] http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix
[207] TeamCity https://www.jetbrains.com/teamcity/
[208] XCode https://developer.apple.com/xcode/
[209] Jenkins https://jenkins-ci.org/
[210] Hudson http://hudson-ci.org/
[211] http://en.wikipedia.org/wiki/Jenkins_%28software%29
[212] https://www.atlassian.com/software/bamboo
[213] JIRA issue tracking service https://www.atlassian.com/software/jira
[214] Stash https://www.atlassian.com/software/stash
[215] http://www.thoughtworks.com/products/go-continuous-delivery
[216] Mingle http://www.thoughtworks.com/mingle/
[217] https://github.com/Strider-CD/strider
[218] Node.js https://nodejs.org/
[219] http://buildbot.net/
[220] https://circleci.com/
[221] http://en.wikipedia.org/wiki/Xvfb
[222] MIT license http://opensource.org/licenses/MIT
[223] BSD license https://en.wikipedia.org/wiki/BSD_licenses
[224] Software versioning https://en.wikipedia.org/wiki/Software_versioning
[225] Revision control https://en.wikipedia.org/wiki/Revision_control
[226] CVS - Concurrent Versions System https://en.wikipedia.org/wiki/Concurrent_Versions_System
[227] http://subversion.apache.org/
[228] http://git-scm.com/
[229] SHA-1 https://en.wikipedia.org/wiki/SHA-1
[230] Loeliger, Jon and McCullough Matthew, Version Control with Git, Second Edition, O'Reilly Media, Inc, August 2012
[231] https://mercurial.selenic.com/
[232] http://www.openoffice.org/
[233] Versions Maven Plugin: http://www.mojohaus.org/versions-maven-plugin/
[234] Python-versioneer: https://github.com/warner/python-versioneer
[235] Dist::Zilla: http://dzil.org/index.html
[236] Perl programming language https://www.perl.org/
[237] Casale, G., Ardagna, D., Artac, M., Barbier, F., Di Nitto, E., Henry, H, Iuhasz, G., Joubert, C., Merseguer, J., Ion Monteanu, V., Perez, J. F., Petcu, D., Rossi, M., Sheridan, C., Spais I., Vladusic, D., DICE: Quality-Driven Development of Data-Intensive Cloud Applications, In the proceedings of the MISE 2015 workshop
[238] List of Web Services standards https://en.wikipedia.org/wiki/List_of_web_service_specifications#Web_Service_Standards_Listings
[239] Lazowska, E.D., Zahorjan, J., Graham, S.G., and Sevcik, K.C.: Quantitative system performance: computer system analysis using queueing network models. Prentice-Hall, Inc. 1984, ISBN: 0-13-746975-6.
[240] JOHNSON, B. W. 1989. Design and Analysis of Fault-Tolerant Digital Systems. Addison-Wesley
[241] C. Baier, J-P. Katoen, 2008. Principles of Model Checking, MIT Press
[242] C.A. Furia, D. Mandrioli, A. Morzenti, M. Rossi, 2012. Modelling Time in Computing, Springer.
[243] Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. 1999. Patterns in property specifications for finite-state verification. Proceedings of the 1999 International Conference on Software Engineering (ICSE)
[244] S. Konrad and B. H. C. Cheng. Real-time specification patterns. Proceedings of the 2005 International Conference on Software Engineering (ICSE), pages 372-381
[245] L. Grunske. Specification patterns for probabilistic quality properties. In Proceedings of the 2008 International Conference on Software Engineering (ICSE), pages 31-40
[246] D. Bianculli, C. Ghezzi, C. Pautasso, and P. Senti, 2012. Specification patterns from research to industry: A case study in service-based applications. Proceedings of the 2012 International Conference on Software Engineering (ICSE), pages 968-976
[247] Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modelling. Wiley, Wiley professional computing series, 1991, ISBN: 978-0-471-50336-1.

[248] Nicol, D.M., Sanders, W.H., Trivedi, K.S.: Model-Based Evaluation: From Dependability to Security. IEEE Trans. Dependable Sec. Comput. 1(1): 48-65 (2004).
[249] Trivedi, K.S.: Probability and Statistics with Reliability, Queueing, and Computer Science Applications. 2nd Edition, Wiley, 2001, ISBN: 978-0-471-33341-8,
[250] U.S. Nuclear Regulatory Commission: Fault Tree Handbook. NUREG-0492, January 1981.
[251] Dugan, B.J., Bavuso, S.J., and Boyd M.A.: Dynamic fault-tree models for fault tolerant computer systems. IEEE Transactions on Reliability, 41(3):363-373, 1992.
[252] Bolch, G., Greiner, S., de Meer, H. and Trivedi, K.S.: Queueing Networks and Markov Chains: Modelling and Performance Evaluation with Computer Science Applications. Wiley-Interscience, New York, NY, USA (1998).
[253] Haverkort, B.R., Marie, R., Rubino, G. and Trivedi, K.S.: Performability Modelling: Techniques and Tools. Wiley & Sons, 2001.
[254] Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press, 1994, ISBN-13: 978-0691036991.
[255] Muppala, J., Sathaye, A., Howe, R., and Trivedi, K.S.: Dependability Modelling of a Heterogeneous VAXcluster System Using Stochastic Reward Nets. Hardware and Software Fault Tolerance in Parallel Computing Systems, D. Avresky (ed.), pp. 33-59, Ellis Horwood Ltd., 1992.
[256] Sahner, R.A., Trivedi, K.S., and Puliafito, A.: Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package. Kluwer Academic Publishers, 1996.
[257] Buchholz, P.: Exact and Ordinary Lumpability in Finite Markov Chains. Journal of Applied Probability, 31(1),pp. 59-75, 1994.
[258] Bobbio, A. and Trivedi, K.S.: An Aggregation Technique for the Transient Analysis of Stiff Markov Chains. IEEE Trans. Computers 35(9): 803-814 (1986).
[259] Miner, A.S., and Ciardo, G.: Efficient Reachability Set Generation and Storage Using Decision Diagrams. ICATPN 1999: 6-25.
[260] Malhotra, M., Muppala, J.K., and Trivedi, K.T.: Stiffness-tolerant methods for transient analysis of stiff Markov chains. Microelectronics and Reliability, vol.34, pp. 1825-1841, 1994.
[261] Molloy, M.K. On the Integration of Delay and Throughput Measures in Distributed Processing Models. PhD thesis, UCLA, Los Angeles (CA), 1981.
[262] Kelling, C.: Conventional and Fast Simulation Techniques for Stochastic Petri Nets. Forschungsbericht des Fachbereichs Informatik, Heft 25, Technische Universität Berlin, Berlin, 1996.
[263] Rodríguez, R.J., Júlvez, J., and Merseguer, J.: On the Performance Estimation and Resource Optimisation in Process Petri Nets. IEEE T. Systems, Man, and Cybernetics: Systems 43(6): 1385-1398 (2013).
[264] Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., and Franceschinis, G.: Modelling with Generalised Stochastic Petri Nets, Wiley & Sons, 1994, ISBN:0471930598.
[265] Sanders, W.H., and Meyer, J.F.: Stochastic Activity Networks: Formal Definitions and Concepts. European Educational Forum: School on Formal Methods and Performance Analysis 2000: 315-343.
[266] Chiola, G., Dutheillet, C., Franceschinis, G., and Haddad, S.: Stochastic Well-Formed Colored Nets and Symmetric Modelling Applications. IEEE Trans. Computers 42(11): 1343-1360 (1993).
[267] Dugan, B.J., Trivedi, K.S., Geist, R., and Nicola, V.F.: Extended Stochastic Petri Nets: Applications and Analysis. Performance 1984: 507-519.
[268] Lindemann, C.: Performance Modelling with Deterministic and Stochastic Petri Nets. SIGMETRICS Performance Evaluation Review 26(2): 3 (1998).
[269] Choi, H., Kulkarni, V.K., and Trivedi, K.S.: Markov Regenerative Stochastic Petri Nets. Perform. Eval. 20(1-3): 337-357 (1994).
[270] Puliafito, A., Scarpa, M., and Trivedi, K.S.: Petri Nets with k Simultaneously Enabled Generally Distributed Timed Transitions. Perform. Eval. 32(1): 1-34 (1998).
[271] Baskett, F., Chandy, K.M., Muntz, R.R., and Palacios, F.G.: Open, closed, and mixed networks of queues with different classes of customers. J. ACM 22(2), 248–260 (1975).
[272] Franks, G., Al-Omari T., Woodside, C.M., Das, O., and Derisavi, S.: Enhanced modelling and solution of layered queueing networks. IEEE Trans Softw. Eng 35(2):148–161, 2009.
[273] Omari, T., Franks, G., Woodside C.M., and Pan A.: Efficient performance models for layered server systems with replicated servers and parallel behaviour. J Syst Softw 80(4):510–527, 2007.
[274] Tribastone, M.: A fluid model for layered queueing networks. IEEE Trans Softw Eng 39(6):744–756, 2013.
[275] Pérez J.F., and Casale G. Assessing sla compliance from Palladio component models. In: Proceedings of the 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC '13, Timisoara, Romania, pp 409–416, 2013.
[276] Bacigalupo, D., van Hemert, J., Chen, X., Usmani, A., Chester, A., He, L. Dillenberger, D., Wills, G., Gilbert, L., and Jarvis, S.: Managing dynamic enterprise and urgent workloads on clouds using layered queueing and historical performance models. Simul. Model Prac. Theory 19:1479–1495, 2011.

[277] Nance, R.R.: A History of Discrete Event Simulation Programming Languages. SIGPLAN Not., 28(3), March 1993, pp. 149-175.
[278] Banks J, Carson J.S., Nelson, B.L., and Nicol D.M.: Discrete Event System Simulation, Upper Saddle River, N.J., Prentice-Hall, 2000.
[279] Shelder, G.S.: Regenerative Stochastic Simulation. Boston, Prentice-Hall, 1993.
[280] Gillespie, D.T.: Approximate accelerated stochastic simulation of chemically reacting systems. Journal of Chemical Physics, 115(4), 2001.
[281] Heidelberger, P.: Fast Simulation of Rare Events in Queueing and Reliability Models. ACM Trans. Model. Comput. Simul., Jan. 1995, 5(1), pp.43-85.
[282] University of Hamburg: Petri Net tool database. Accessed: 2015-06-02. URL: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html.
[283] Charalambous, A.: Extension of PIPE2 to Support Coloured Generalised Stochastic Petri Nets. Imperial College, Tech.Rep. 2014.
[284] Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Basic Concepts. EATCS Monographs on Theoretical Computer Science, vol. 1. Springer, Germany (1992).
[285] GreatSPN2.0 http://www.di.unito.it/~greatspn/index.html
[286] TimeNET https://www.tu-ilmenau.de/sse/timenet/general-information/petri-nets/
[287] PIPE http://pipe2.sourceforge.net/
[288] PeabraiN http://webdiis.unizar.es/GISED/?q=tool/peabrain
[289] JMT http://jmt.sf.net
[290] LINE https://line-solver.sourceforge.net
[291] LQNS http://www.sce.carleton.ca/rads/lqns/
[292] Clarke, E. M. (2008). The birth of model checking. 25 Years of Model Checking, 1-26.
[293] Krstić, S. (2014). Quantitative properties of software systems: specification, verification, and synthesis. Companion Proceedings of the 36th International Conference on Software Engineering. ACM.
[294] Kwiatkowska, M. (2013). Advances in quantitative verification for ubiquitous computing. Theoretical Aspects of Computing–ICTAC 2013, 42-58.
[295] Basin, D., Caronni, G., Ereth, S., Harvan, M., Klaedtke, F., & Mantel, H. (2014). Scalable Offline Monitoring. Runtime Verification. Springer International Publishing.
[296] Camilli, M. (2014). Formal verification problems in a Big Data world: towards a mighty synergy. Companion Proceedings of the 36th International Conference on Software Engineering. ACM.
[297] Yang, F., Su, W., Zhu, H., & Li, Q. (2010). Formalising MapReduce with csp. Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on. IEEE.
[298] Hoare, C. A. R. (1985). Communicating sequential processes (Vol. 178). Englewood Cliffs: Prentice-hall.
[299] Refinement, F. (2003). FDR2 User Manual. Formal Systems (Europe) Ltd.
[300] Clarke, Edmund M, Orna Grumberg, and Doron Peled. Model checking. MIT press, 1999.
[301] Di Noia, T., Mongiello, M., & Di Sciascio, E. A computational model for MapReduce job flow.
[302] Uppaal. http://uppaal.org/
[303] Barbierato, E., Gribaudo, M., & Iacono, M. (2013). Modelling apache hive based applications in Big Data architectures. Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
[304] Fisher, M. (2011). An Introduction to Practical Formal Methods Using Temporal Logic. John Wiley & Sons.
[305] Dwyer, M. B., Avrunin, G. S., & Corbett, J. C. (1999). Patterns in property specifications for finite-state verification. Software Engineering, 1999. Proceedings of the 1999 International Conference on. IEEE.
[306] Ciapessoni, E., Mirandola, P., Coen-Porisini, A., Mandrioli, D., & Morzenti, A. (1999). From formal models to formally based methods: an industrial experience. ACM Transactions on Software Engineering and Methodology (TOSEM), 8(1), 79-113.
[307] Baresi, L., Blohm, G., Kolovos, D. S., Matragkas, N., Motta, A., Paige, R. F., Radjenovik A. & Rossi, M. (2015). Formal verification and validation of embedded systems: the UML-based MADES approach. Software & Systems Modelling, 1-21.
[308] Bianculli, D., Ghezzi, C., Pautasso, C., & Senti, P. (2012). Specification patterns from research to industry: a case study in service-based applications. Proceedings of the 34th International Conference on Software Engineering. IEEE Press.
[309] The SPIN Model checker. http://spinroot.com/
[310] NuSMV: A New Symbolic Model Checker. http://nusmv.fbk.eu/
[311] The PRISM probabilistic model checker. http://www.prismmodelchecker.org/
[312] Bersani, M. M., Rossi, M., & Pietro, P. S. (2013). A tool for deciding the satisfiability of continuous-time metric temporal logic. In Temporal Representation and Reasoning (TIME), 2013 20th International Symposium on(pp. 99-106). IEEE

[313] De Moura, L., & Bjørner, N. (2008). Z3: An efficient SMT solver. Tools and Algorithms for the Construction and Analysis of Systems, 337-340.
[314] The Zot bounded model/satisfiability checker. http://zot.googlecode.com
[315] CorrettoUML https://github.com/deib-polimi/Corretto
[316] Jackson, D. (2001). Lightweight formal methods. FME 2001: Formal Methods for Increasing Software Productivity, 1-1.
[317] Ghilardi, Silvio et al. 'Towards SMT model checking of array-based systems.' Automated Reasoning (2008): 67-82.
[318] Ledru Y., Idani A. and Richier J.L.. Validation of a security policy by the test of its formal B specification - a case study. FormaliSE: FME Workshop on Formal Methods in Software Engineering. ICSE 2015.
[319] Lodderstedt T., Basin D., and Doser J. SecureUML: A UML-based modelling language for model-driven security. The Unified Modelling Language (2002): 426-441.
[320] Abrial, J. (2005).The B-Book: Assigning programmes to meanings. Cambridge University Press.
[321] The nuXmv model checker. https://nuxmv.fbk.eu/
[322] The Z3 Theorem Prover. https://github.com/Z3Prover
[323] Parsons T., Murphy, J. 'Detecting Performance Antipatterns in Component Based Enterprise Systems', Journal of Object Technology, vol. 7, no. 3, March - April 2008, pp. 55-90.
[324] Cortellessa V., 'Performance antipatterns: state of the art and future perspectives', 10th European Workshop, EPEW 2013, Venice, Italy, September 16-17, 2013. Proceedings, pp. 1-6. DOI: 10.1007/978-3-642-40725-3_1
[325] Catia Trubiani, Anne Koziolek, Vittorio Cortellessa, Ralf Reussner, Guilt-based handling of software performance antipatterns in palladio architectural models, Journal of Systems and Software, Volume 95, September 2014, Pages 141-165, ISSN 0164-1212.
[326] Wert et. al, 'Supporting swift reaction: automatically uncovering performance problems by systematic experiments'. Proceedings of the 2013 International Conference on Software Engineering, pp. 552-561 .
[327] Wert et. al., 'Automatic detection of performance anti-patterns in inter-component communications'. Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures, 2014, pp. 3-12 .
[328] Trubiani et. al, 'Exploring Synergies between Bottleneck Analysis and Performance Antipatterns'. Proceedings of the 5th ACM/SPEC international conference on Performance engineering, 2014. Pages 75-86 .
[329] Jeroslow, R. G. The polynomial hierarchy and a simple model for competitive analysis. Mathematical Programming, 32, 146–164, 1985.
[330] Colson, B., Marcotte, P., Savard G. An overview of bilevel optimisation. Annals of Operations Research, 153, 235–256, 2007.
[331] El-Ghazali, T. Metaheuristics: From Design to Implementation. Wiley Publishing, 2009.
[332] Paul Baker et al. Model-Driven Testing, 183 pages, Springer, 2008.
[333] OMG http://utp.omg.org
[334] Arilo Dias Neto et al. A Survey on Model-based Testing Approaches: A Systematic Review, WEASELTech'07, ACM, 2007.
[335] Diwakar Krishnamurthy, Jerome A. Rolia, Shikharesh Majumdar. A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems. IEEE Trans. Software Eng. 32(11): 868-882 (2006).
[336] Gabriel Iuhasz et al. MODAClouds Deliverable D6.5.3: Runtime Environment Final Release, March 2015. (Available at http://www.modaclouds.eu)
[337] André van Hoorn, Christian Vögele, Eike Schulz, Wilhelm Hasselbring, Helmut Krcmar. Automatic Extraction of Probabilistic Workload Specifications for Load Testing Session-Based Application Systems, in Proc. of Valuetools 2014.
[338] Ningfang Mi, Giuliano Casale, Lucy Cherkasova, Evgenia Smirni. Injecting realistic burstiness to a traditional client-server benchmark. Proceedings of the 6th international conference on Autonomic computing, 149-158, 2009.
[339] Giuliano Casale, Amir Kalbasi, Diwakar Krishnamurthy, Jerry Rolia. BURN: Enabling Workload Burstiness in Customised Service Benchmarks. IEEE Trans. Software Eng. 38(4): 778-793 (2012).
[340] Vahid Garousi, Lionel C. Briand and Yvan Labiche. Traffic-aware Stress Testing of Distributed Systems Based on UML Models, Proc. of ICSE 2006.
[341] YCSB. https://github.com/brianfrankcooper/YCSB
[342] Transaction Processing Council Benchmarks. http://www.tpc.org
[343] Tilmann Rabl, Meikel Poess, Hans-Arno Jacobsen, Patrick O'Neil, and Elizabeth O'Neil. 2013. Variations of the star schema benchmark to test the effects of data skew on query performance. In Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13), ACM, New York, NY, USA, 361-372.
[344] http://www.cs.brandeis.edu/~linearroad/
[345] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, 'Starfish: A Self-Tuning System for Big Data Analytics,' Proc. 5th Biennial Conf. on Innovative Data Syst. Res. (CIDR '11) (Asilomar, CA, 2011), pp. 261–272.
[346] https://community.progress.com/community_groups/technology_partner/w/technologypartner/401.sonic-test-harness.aspx
[347] IBM Hursley, Performance Harness for Java Message Service, 2005. http://www.alphaworks.ibm.com/tech/perfharness
[348] ActiveMQ, JMeter performance test, 2006 http://activemq.apache.org/jmeter-performance-tests.html

[349] JBoss, JBoss JMS Messaging Performance Framework, 2006 http://docs.jboss.org/jbossmessaging/docs/guide-1.0.1.SP5/html_single/
[350] Grinder: http://grinder.sourceforge.net/
[351] Jython http://www.jython.org/
[352] Clojure http://clojure.org/
[353] Jmeter: http://jmeter.apache.org/
[354] Markov4Jmeter: https://www.se.informatik.uni-kiel.de/en/research/projects/markov4jmeter
[355] Selenium: http://www.seleniumhq.org/
[356] MDload: https://github.com/imperial-modaclouds/modaclouds-mdload
[357] Chaos Monkey https://github.com/Netflix/SimianArmy
[358] AWS Autoscaling http://aws.amazon.com/autoscaling/
[359] Microsoft Azure http://azure.microsoft.com/en-gb/
[360] WazMonkey https://github.com/smarx/WazMonkey
[361] Hadoop toolkit, https://code.google.com/p/hadoop-toolkit/wiki/HadoopPerformanceMonitoring
[362] SequenceIQ, http://sequenceiq.com/
[363] http://blog.sequenceiq.com/blog/2014/10/07/hadoop-monitoring/
[364] Elasticsearch, https://www.elastic.co
[365] Kibana, https://www.elastic.co/products/kibana
[366] Logstash, http://logstash.net/
[367] Collectd https://collectd.org/
[368] Hadoop Vaidya, http://hadoop.apache.org/docs/r1.2.1/vaidya.html
[369] Ganglia, http://ganglia.info
[370] RRDtool http://oss.oetiker.ch/rrdtool/
[371] Apache Ambari, https://ambari.apache.org/
[372] Apache Chukwa, https://chukwa.apache.org/
[373] DATASTAX, http://www.datastax.com
[374] OpsCentre, http://www.datastax.com/what-we-offer/products-services/datastax-opscenter
[375] Apache Cassandra http://cassandra.apache.org/
[376] MMS MongoDB, https://www.mongodb.com/cloud
[377] Server Density, https://www.serverdensity.com/plugins/mongodb
[378] Manage Engine, https://www.manageengine.com/
[379] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I. H. The WEKA Data Mining Software: An Update SIGKDD Explor. Newsl., ACM, 2009, Vol. 11(1), pp. 10-18
[380] http://weka.sourceforge.net/packageMetaData/distributedWekaBase/index.html
[381] Bifet, A., Holmes, G., Kirkby, R. and Pfahringer, B. MOA: Massive Online Analysis J. Mach. Learn. Res., JMLR.org, 2010, Vol. 11, pp. 1601-1604
[382] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 2011, Vol. 12, pp. 2825-2830
[383] Cython C-Extension for Python http://cython.org/
[384] http://www.numpy.org/
[385] http://docs.scipy.org/doc/numpy/index.html
[386] http://matplotlib.org/
[387] Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K. and Wiswedel, B. KNIME: The Konstanz Information Miner Studies in Classification, Data Analysis, and Knowledge Organisation (GfKL 2007) Springer, 2007
[388] LIBSVM https://www.csie.ntu.edu.tw/~cjlin/libsvm/
[389] R programming language http://www.r-project.org/
[390] Achtert, E., Kriegel, H.-P., Schubert, E. and Zimek, A. Interactive Data Mining with 3D-parallel-coordinate-trees Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data ACM, 2013, pp. 1009-1012
[391] Fortran http://www.fortran.com/
[392] http://cran.r-project.org/web/packages/available_packages_by_name.html
[393] Wolfram Research, I. Mathematica 2015
[394] Gdeisat, M. and Lilley, F. Matlab by Example: Programming Basics Elsevier Science Publishers B. V., 2013
[395] https://github.com/shadanan/HadoopLink
[396] http://www.mathworks.com/help/matlab/large-files-and-big-data.html
[397] Heaton, J. Programming Neural Networks with Encog 2 in Java Heaton Research, Inc., 2010
[398] http://www.heatonresearch.com/wiki/Encog_Performance
[399] GPGPU https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units

[400] A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, B. Catanzaro, Deep learning with COTS HPC Systems, Proceedings of the 30th International Conference on Machine Learning, Atlanta, USA, 2013
[401] H. Schwenk, A. Rousseau, M Attik, Large, Pruned or Continuous Space Language Models on a GPU for Statistical Machine Translation, Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modelling for HLT, June 2012, Montreal, Canada
[402] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S. and Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding Proceedings of the ACM International Conference on Multimedia ACM, 2014, pp. 675-678
[403] Collobert, R., Kavukcuoglu, K. and Farabet, C. Torch7: A Matlab-like Environment for Machine Learning BigLearn, NIPS Workshop 2011
[404] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N. and Bengio, Y. Theano: new features and speed improvements Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012
[405] https://code.google.com/p/cuda-convnet2/
[406] http://deeplearning4j.org/gettingstarted.html
[407] Ashari, A., Tatikonda, S., Boehm, M., Reinwald, B., Campbell, K., Keenleyside, J. and Sadayappan, P. On Optimising Machine Learning Workloads via Kernel Fusion Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming ACM, 2015, pp. 173-182
[408] http://devblogs.nvidia.com/parallelforall/digits-deep-learning-gpu-training-system/
[409] H2O http://0xdata.com/product/
[410] Franklin, M., Gonzalez, J., Jordan, M. I., Pan, X., Smith, V., Sparks, E., Talwalkar, A., Venkataraman, S. and Zaharia, M. MLlib 2013
[411] Apache Spark https://spark.apache.org/
[412] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. and Stoica, I. Spark: Cluster Computing with Working Sets Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing USENIX Association, 2010, pp. 10-10
[413] Owen, S., Anil, R., Dunning, T. and Friedman, E. Mahout in Action Manning Publications Co., 2011
[414] https://flink.apache.org/
[415] jubat.us
[416] Hido, S., Tokui, S. and Oda, S. Jubatus: An Open Source Platform for Distributed Online Machine Learning NIPS 2013 Workshop on Big Learning, Lake Tahoe
[417] Jain, A. and Nalya, A. Learning Storm Packt Publishing, 2014 inproceedings (Jia14)
[418] Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A. and Vishwanathan, S. Hash Kernels for Structured Data J. Mach. Learn. Res., JMLR.org, 2009, Vol. 10, pp. 2615-2637
[419] Microsoft Research http://research.microsoft.com/en-us/
[420] Agarwal, A., Chapelle, O., Dudik, M. and Langford, J. A Reliable Effective Terascale Linear Learning System CoRR, 2011, Vol. abs/1110.4198
[421] Rosen, J., Polyzotis, N., Borkar, V. R., Bu, Y., Carey, M. J., Weimer, M., Condie, T. and Ramakrishnan, R. Iterative MapReduce for Large Scale Machine Learning CoRR, 2013, Vol. abs/1303.3517
[422] Petuum http://petuum.github.io/
[423] GraphLab Create https://dato.com/products/create/
[424] Deeplearning4j http://deeplearning4j.org/
[425] MLBase http://www.mlbase.org/
[426] FinkML https://ci.apache.org/projects/flink/flink-docs-master/libs/ml/
[427] NIMBLE http://users.cis.fiu.edu/~lzhen001/activities/KDD2011Program/docs/p334.pdf
[428] SystemML http://researcher.watson.ibm.com/researcher/files/us-ytian/systemML.pdf
[429] Amazon EC2 http://aws.amazon.com/ec2/
[430] Google GCE https://cloud.google.com/compute/
[431] Julia parallel computing http://julia.readthedocs.org/en/latest/manual/parallel-computing/
[432] Julia programming language http://julialang.org/
[433] Scala programming language http://www.scala-lang.org/
[434] http://ellogon.org/
[435] van Hoorn et. al., 'Kieker: a framework for application performance monitoring and dynamic software analysis'. Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, 2014. Pages 247-248.
[436] Perez et. al., 'Towards a DevOps Approach for Software Quality Engineering'. Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development, 2015. Pages 5-10
[437] Calinescu et.al., 'Adaptive Model Learning for Continual Verification of Non-Functional Properties'. Proceedings of the 5th ACM/SPEC international conference on Performance engineering, 2014. Pages 87-98.
[438] Bauer, A., Leucker, M., Schallhart, C.: Runtime Verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol, 20.4. September 2011.

[439] Pnueli 1977. The temporal logic of programmes. In Symposium on the Foundations of Computer Science (FOCS). IEEE Computer Society Press, Providence, Rhode Island, 46–57.
[440] Havelund and Rosu 2002. Synthesising Monitors for Safety Properties. In Tools and Algorithms for Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 2280. 342–356.
[441] Hakansson, J., Jonsson, B., and Lundqvist, O. 2003. Generating online test oracles from temporal logic specifications. Journal on Software Tools for Technology Transfer (STTT) 4, 4, 456–471.
[442] Thati, P. and Rosu, G. 2005. Monitoring algorithms for metric temporal logic specifications. Electronic Notes in Theoretical Computer Science (ENTCS) 113, 145–162.
[443] Oded Maler, Dejan Nickovic 2004: Monitoring Temporal Properties of Continuous Signals. Proceedings of FORMATS/FTRTFT: 152-166
[444] Alur, R., Henzinger, T. A.: A Really Temporal Logic. J. ACM 41.4, pp. 181-203. ACM (1994)
[445] Kuhtz, L., Finkbeiner: LTL path checking is efficiently parallelisable. In: Proc of ICALP 2009. LNCS, vol. 5556, pp. 235–246. Springer (2009)
[446] Markey, N., Raskin, J.: Model Checking Restricted Sets of Timed Paths. CONCUR 2004. LNCS, vol. 3170, pp. 432-447. Springer (2004).
[447] Rosu, G., Chen, F.: Semantics and algorithms for parametric monitoring. Logical Methods in Computer Science 8(1) (2012)
[448] Medhat, R., Joshi, Y., Bonakdarpour, B., Fischmeister, S.: Parallelised runtime verification of first-order LTL specifications (2014), technical report
[449] Bauer, A., Falcone, Y. : Decentralised LTL monitoring. In: Proc of FM 2012. LNCS, vol. 7436, pp. 85–100. Springer (2012)
[450] Barre, B., Klein, M., Soucy-Boivin, M., Ollivier, P.A., Hallé, S.: MapReduce for parallel trace validation of LTL properties. In: Proc. of RV 2012. LNCS, vol. 7687, pp. 184–198. Springer (2012)
[451] Bianculli, D., Ghezzi, C., Krstic , S.: Trace checking of metric temporal logic with aggregating modalities using MapReduce. In: Proc. of SEFM 2014. LNCS, vol. 8702, pp. 144–158. Springer (2014)
[452] Bersani, M.M, Bianculli D., Ghezzi, C., Krstic, S., San Pietro, P.]: Lazy Semantics for MTL: An Application to Trace Checking with MapReduce . Submitted to ATVA 2015
[453] Apache Tez, http://tez.apache.org, accessed on 08/05/2015.
[454] Hortonworks Data Platform, http://hortonworks.com/hdp/, accessed on 08/05/2015.
[455] Cloudera CDH, http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html, accessed on 08/05/2015.
[456] MapR, https://www.mapr.com/products/mapr-distribution-including-apache-hadoop, accessed on 08/05/2015.
[457] IBM BigInsights, http://www-01.ibm.com/software/data/infosphere/hadoop/enterprise.html, accessed on 08/05/2015.
[458] Big R https://www-01.ibm.com/support/knowledgecenter/SSPT3X_3.0.0/com.ibm.swg.im.infosphere.biginsights.analyze.doc/doc/t_analyze_data_bigr.html
[459] Big Sheets http://www-01.ibm.com/software/ebusiness/jstart/bigsheets/
[460] Hive https://hive.apache.org/
[461] Pig https://pig.apache.org/
[462] HBase http://hbase.apache.org/
[463] SQL-on-Hadoop https://www.mapr.com/why-hadoop/sql-hadoop/sql-hadoop-details
[464] HDP cookbooks, rackerlabs/hdp-cookbooks, accessed on 08/05/2015.
[465] Cloudera cookbook, RiotGamesCookbooks/cloudera-cookbook, accessed on 08/05/2015.
[466] MapR cookbooks, boorad/mapr_cookbooks, accessed on 08/05/2015.
[467] Amazon EC2 pricing http://aws.amazon.com/elasticmapreduce/pricing/
[468] EMRFS http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-fs.html
[469] Microsoft HDInsight, http://azure.microsoft.com/en-gb/services/hdinsight/, accessed on 08/05/2015.
[470] Azure Blob http://justazure.com/azure-blob-storage-part-one-introduction/
[471] https://www.flexiant.com/2012/12/03/application-provisioning/
[472] Swift http://docs.openstack.org/developer/swift/
[473] Villalpando, Bautista, April, and Abran. 'Performance analysis model for Big Data applications in cloud computing.' *Journal of Cloud Computing* 3.1 (2014): 1–20.
[474] Verma, Cherkasova, and Campbell. 'Profiling and evaluating hardware choices for MapReduce environments: An application-aware approach.' *Performance Evaluation* 79 (2014): 328–344.
[475] Barbierato, Gribaudo, and Iacono. A Performance Modelling Language For Big Data Architectures. *ECMS* 2013: 511–517.
[476] Herodotou. 'Hadoop performance models.' *arXiv preprint arXiv:1106.0940* (2011).
[477] Zhang, Rajasekaran, Duan, Wood, and Zhu. 'Minimising Interference and Maximising Progress for Hadoop Virtual Machines.'

[478] Verma, Cherkasova, and Campbell. 'ARIA: automatic resource inference and allocation for MapReduce environments.' *Proceedings of the 8th ACM International Conference on Autonomic Computing* 14 Jun. 2011: 235-244.
[479] Zhang, Cherkasova, and Thau Loo. 'Parameterisable benchmarking framework for designing a MapReduce performance model.' *Concurrency and Computation: Practice and Experience* 26.12 (2014): 2005–2026.
[480] Malekimajd, Rizzi, Ardagna, Ciavotta, Passacantando, and Moghavar. 'Optimal Capacity Allocation for executing MapReduce Jobs in Cloud Systems.' *SYNASC 2014*: 385–392.
[481] Yang and Sun. 'An analytical performance model of MapReduce.' Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on 15 Sep. 2011: 306–310.
[482] Lin, Meng, Xu, and Wang. 'A practical performance model for Hadoop MapReduce.' *Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on* 24 Sep. 2012: 231–239.
[483] Lin, Zhang, Wierman, and Tan. 'Joint optimisation of overlapping phases in MapReduce.' *Performance Evaluation* 70.10 (2013): 720–735.
[484] Tan, Wang, Yu, and Zhang. 'Non-work-conserving effects in MapReduce: diffusion limit and criticality.' *The 2014 ACM international conference on Measurement and modelling of computer systems* 16 Jun. 2014: 181–192.
[485] Bardhan and Menascé. 'Queueing network models to predict the completion time of the map phase of MapReduce jobs.' *Proceedings of the Computer Measurement Group International Conference* 3 Dec. 2012.
[486] Vianna, Comarela, Pontes, J. Almeida, V. Almeida, Wilkinson, Kuno, and Dayal. 'Analytical performance models for MapReduce workloads.' *International Journal of Parallel Programming* 41.4 (2013): 495–525.
[487] Aguilera-Mendoza and Llorente-Quesada. Modelling and simulation of Hadoop Distributed File System in a cluster of workstations. *Model and Data Engineering* 2013: 1–12.
[488] Castiglione, Gribaudo, Iacono, and Palmieri. 'Exploiting mean field analysis to model performances of Big Data architectures.' *Future Generation Comp. Syst.* 37 (2014): 203–211.
[489] Zaharia, Chowdhury, Das, Dave, Ma, McCauley, Franklin, Shenker, Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *Berkeley University Technical Report* 2012. https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf
[490] Spark GCE, sigmoidanalytics/spark_gce, accessed on 07/05/2015.
[491] Databricks Inc., https://databricks.com, accessed on 07/05/2015.
[492] Ousterhout, Panda, Rosen, Venkataraman, Xin, Ratnasamy, Shenker, Stoica. The Case for Tiny Tasks in Compute Clusters. https://www.eecs.berkeley.edu/~apanda/papers/tiny-tasks.pdf
[493] Chowdhury. Performance and Scalability of Broadcast in Spark. *Berkeley University Technical Report* 2010. http://www.cs.berkeley.edu/~agearh/cs267.sp10/files/mosharaf-spark-bc-report-spring10.pdf
[494] Davidson, and Or. 'Optimising Shuffle Performance in Spark.' University of California, Berkeley — Department of Electrical Engineering and Computer Sciences, Tech. Rep (2013).
[495] Ousterhout, Rasti, Ratnasamy, Shenker, Chun. Making Sense of Performance in Data Analytics Frameworks. https://www.eecs.berkeley.edu/~keo/publications/nsdi15-final147.pdf
[496] Whänner K. Sep 10 2014 Real-Time Stream Processing as Game Changer in a Big Data World with Hadoop and data Warehouse. May 7 2015 http://www.infoq.com/articles/stream-processing-hadoop
[497] http://venturebeat.com/2014/03/19/without-stream-processing-theres-no-big-data-and-no-internet-of-things/
[498] http://www.happiestminds.com/sites/all/themes/happiestminds/pdf/ANALYTICS_ON_BIG_FAST_DATA_USING_A_REALTIME_STREAM_DATA_PROCESSING_ARCHITECTURE.pdf
[499] RabbitMQ https://www.rabbitmq.com/
[500] http://blog.infochimps.com/wp-content/uploads/2012/10/StormKafka.png
[501] http://azure.microsoft.com/es-es/pricing/details/hdinsight/
[502] http://azure.microsoft.com/es-es/documentation/articles/hdinsight-component-versioning/
[503] http://azure.microsoft.com/es-es/pricing/calculator/
[504] http://aws.amazon.com/es/kinesis/
[505] http://aws.amazon.com/kinesis/pricing/
[506] http://www.cloudera.com/content/cloudera/en/products-and-services.html
[507] http://hortonworks.com/hadoop/storm/
[508] http://dzone.com/articles/streaming-big-data-storm-spark
[509] http://spark.apache.org/streaming/
[510] http://samza.apache.org/
[511] http://hadoop.apache.org/docs/r1.2.1/streaming.html
[512] http://incubator.apache.org/s4/
[513] https://github.com/Parsely/streamparse
[514] http://docs.stratio.com/modules/streaming-cep-engine/development/
[515] https://github.com/twitter/summingbird
[516] https://github.com/facebookarchive/scribe
[517] https://github.com/linkedin/pinot

[518] http://lambda-architecture.net/components/2014-06-30-speed-components/
[519] https://samza.apache.org/learn/documentation/0.9/comparisons/storm.html
[520] http://samza.apache.org/learn/documentation/0.7.0/comparisons/spark-streaming.html
[521] ActiveMQ http://activemq.apache.org/
[522] https://storm.apache.org/documentation/Guaranteeing-message-processing.html
[523] https://storm.apache.org/documentation/Fault-tolerance.html
[524] https://storm.apache.org/about/scalable.html
[525] https://labs.spotify.com/2015/01/05/how-spotify-scales-apache-storm/
[526] http://hortonworks.com/hadoop/ranger/
[527] https://github.com/apache/storm/blob/master/SECURITY.md
[528] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.
[529] Brewer, Eric. Towards Robust Distributed Systems. http://www.cs.berkley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
[530] M. Scavuzzo, E. Di Nitto, D. Ardagna, Building Data-Intensive Applications Exploiting Data as a Service Systems: Experiences and Challenges. Politecnico di Milano. Technical Report n. 2014.10.
[531] Amazon DynamoDB https://aws.amazon.com/dynamodb/
[532] Amazon SimpleDB https://aws.amazon.com/simpledb/
[533] Amazon Redis Labs https://redislabs.com/
[534] Google Cloud Datastore https://cloud.google.com/datastore/
[535] Azure DocumentDB http://azure.microsoft.com/en-us/services/documentdb/
[536] Azure Redis Cache http://azure.microsoft.com/en-us/services/cache/
[537] Azure Storage http://azure.microsoft.com/en-us/services/storage/
[538] MongoLab https://mongolab.com/company/
[539] Rackspace http://www.rackspace.com/data/managed-nosql
[540] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing* (SoCC '10). ACM, New York, NY, USA, 143-154.
[541] End Point. Benchmarking Top NoSQL Databases Apache Cassandra, Couchbase, HBase, and MongoDB. Revised: May 27, 2015. http://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf
[542] http://basho.com/fundamentals-of-nosql-security/
[543] http://docs.mongodb.org/manual/security/
[544] M. Scavuzzo, E. Di Nitto, S. Ceri, Interoperable Data Migration between NoSQL Columnar Databases, Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International.
[545] R. Osman and P. Piazzolla (Sept 2014). Modelling Replication in NoSQL Datastores. 11th International Conference on Quantitative Evaluation of SysTems (QEST 2014), Sept 8–10, 2014, Florence, Italy.
[546] C. Stewart, A. Chakrabarti, and R. Griffith. Zoolander: Efficiently meeting very strict, low-latency slos. In 10th ICAC, 2013.
[547] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically bounded staleness for practical partial quorums. PVLDB, 5(8):776{787, 2012.
[548] Aaron Davidson, Andrew Or. Optimising Shuffle Performance in Spark. UC Berkeley technical report. http://www.cs.berkeley.edu/~kubitron/courses/cs262a-F13/projects/reports/project16_report.pdf. Last accessed 19th May 2015.
[549] N. Feamster, J. Rexford, E. Zegura. The Road to SDN: An Intellectual History of Programmable Networks
[550] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks
[551] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart and Amin Vahdat. B4: Experience with a Globally-Deployed Software Defined WAN
[552] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, Roger Wattenhofer. Achieving High Utilisation with Software-Driven WAN
[553] https://www.openstack.org/assets/presentation-media/OpenDaylightOpenStackIcehouseSummit.pdf
[554] http://www.opencontrail.org/
[555] http://www.projectfloodlight.org/floodlight/
[556] http://osrg.github.io/ryu/
[557] Rob Sherwood , Glen Gibb , Kok-Kiong Yap , Guido Appenzeller, Martin Casado, Nick McKeown , Guru Parulkar. FlowVisor: A Network Virtualisation Layer, Available online at: http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf

[558] Pankaj Berde , Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, Guru Parulkar. ONOS: Towards an Open, Distributed SDN OS, Proc. of HotSDN'14. Available online from: http://www-cs-students.stanford.edu/~rlantz/papers/onos-hotsdn.pdf

[559] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo and Wu Chou. A roadmap for traffic engineering in SDN-OpenFlow networks, Computer Networks, Volume 71, 4 October 2014, Pages 1–30

[560] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: scaling flow management for high performance networks, Proc. of SIGCOMM'11. Available online from: http://conferences.sigcomm.org/sigcomm/2011/papers/sigcomm/p254.pdf

[561] J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A.R. Curtis, S. Banerjee, Devoflow: cost-effective flow management for high performance enterprise networks, Proc. of Hotnets'10. Available online from: http://conferences.sigcomm.org/hotnets/2010/papers/a1-mogul.pdf

[562] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with DIFANE, Proc. of SIGCOMM'10. Available online from: http://www-bcf.usc.edu/~minlanyu/writeup/sigcomm10.pdf

[563] A. Tootoonchian, Y. Ganjali, Hyperflow: a distributed control plane for openflow. Available online from: http://www.cse.iitd.ac.in/~siy107537/csl374/a5/files/Tootoonchian.pdf

[564] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al., Onix: a distributed control platform for large-scale production networks. Available online from: https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Koponen.pdf

[565] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, Balanceflow: controller load balancing for openflow networks, Proc. of 2nd International Conference on Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE

[566] S. Hassas Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications. Proc. of HotSDN'12. Available online from: http://conferences.sigcomm.org/sigcomm/2012/paper/hotsdn/p19.pdf

[567] E. Ng, Maestro: A System For Scalable Openflow Control. Available online from: http://www.cs.rice.edu/~eugeneng/papers/TR10-11.pdf

[568] D. Erickson, The Beacon Openflow Controller, Proc. of HotSDN'13. Available online from: http://yuba.stanford.edu/~derickso/docs/hotsdn15-erickson.pdf

[569] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks. Available online from: https://www.usenix.org/system/files/conference/hot-ice12/hotice12-final33_0.pdf

[570] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, Enabling fast failure recovery in openflow networks, Proc of DRCN, 2011.

[571] D. Staessens, S. Sharma, D. Colle, M. Pickavet, P. Demeester, Software defined networking: Meeting carrier grade requirements, Proc. LANMAN, 2011.

[572] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, Openflow: meeting carrier-grade recovery requirements, Computer Communications 36(6): 656-665 (2013)

[573] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, P. Castoldi, Openflow-based segment protection in ethernet networks, Optical Communications and Networking, IEEE/OSA Journal of (Volume:5 , Issue: 9), 2013.

[574] P. Fonseca, R. Bennesby, E. Mota, A. Passito, A replication component for resilient openflow-based networking, NOMS 2012: 933-939.

[575] M. Luo, Y. Tian, Q. Li, J. Wang, W. Chou, Sox – a generalised and extensible smart network openflow controller, First SDN World Summit, 2012.

[576] M. Luo, Y. Zeng, J. Li, An adaptive multi-path computation framework for centrally controlled networks, Journal of Computer Networks, Elsevier, February 2015.

[577] M. Reitblatt, N. Foster, J. Rexford, D. Walker, Consistent updates for software-defined networks: Change you can believe in!, Proc. Hotnets, 2011.

[578] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, D. Walker, Abstractions for network update. Proc. of ACM SIGCOMM 2012.

[579] T. Mizrahi, Y. Moses, Time-based updates in software defined networks, SDNRG, IETF Meeting, Berlin, July 2013.

[580] T. Mizrahi, Y. Moses, Time-based Updates in Openflow: A Proposed Extension to the Openflow Protocol, CCIT Report #835, July 2013, EE Pub No. 1792, Technion, Israel.

[581] A. Tootoonchian, M. Ghobadi, Y. Ganjali, Opentm: traffic matrix estimator for openflow networks, PAM'10 Proceedings of the 11th international conference on Passive and active measurement.

[582] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with opensketch, Proc of NSDI 2013.

[583] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H.V. Madhyastha, Flowsense: monitoring network utilisation with zero measurement cost, PAM 2013.

[584] T. Benson, A. Anand, A. Akella, M. Zhang, Microte: fine grained traffic engineering for data centers, Proc of Co-Next 2011.

[585] Joshua Reich, Christopher Monsanto, et al. Modular SDN Programming with Pyretic. ;login: issue: October 2013, Volume 38, Number 5.

[586] Robert Soulé, Shrutarshi Basu, Parisa Jalili Marandi, Fernando Pedone, Robert Kleinberg, Emin Gün Sirer, and Nate Foster. Merlin: A Language for Provisioning Network Resources. Proc. of CONEXT 2014.
[587] http://ceph.com/
[588] RADOS https://ceph.com/dev-notes/the-rados-distributed-object-store/
[589] http://docs.flexiant.com/display/DOCS/Ceph+Support
[590] https://www.openstack.org/summit/openstack-summit-hong-kong-2013/session-videos/presentation/ceph-the-de-facto-storage-backend-for-openstackd
[591] https://www.dreamhost.com/cloud/computing/
[592] https://github.com/ceph/ceph
[593] http://www.ssrc.ucsc.edu/Papers/weil-osdi06.pdf
[594] Ceph: A Scalable, High-Performance Distributed File System Sage A. Weil Scott A. Brandt Ethan L. Miller Darrell D. E. Long (http://ceph.com/docs/master/rados/operations/crush-map/)
[595] https://ceph.com/docs/v0.79/rados/operations/auth-intro/
[596] http://aws.amazon.com/s3/
[597] http://aws.amazon.com/s3/pricing/
[598] http://aws.amazon.com/storagegateway/
[599] http://aws.amazon.com/s3/faqs/
[600] Analysis of SAP HANA High Availability Capabilities. http://www.oracle.com/technetwork/database/availability/sap-hana-ha-analysis-cwp-1959003.pdf
[601] K.Molka *et al*. Memory-Aware Sizing for In-Memory Databases. In Proc. NOMS 2014, 10 pages, IEEE Press.
[602] Jennie Duggan, Ugur Çetintemel, Olga Papaemmanouil, Eli Upfal. Performance prediction for concurrent database workloads. Proceedings of SIGMOD 2011, 337-348.
[603] A. J. Elmore, S. Das, A. Pucher, D. Agrawal, A. El Abbadi, and X. Yan, 'Characterizing tenant behaviour for placement and crisis mitigation in multitenant DBMSs,' in Proceedings of SIGMOD 2013, pp. 517– 528, ACM Press.

