Article Title

Infrastructures for High-Performance Computing: Cloud Computing Development Environments

Author and Co-author Contact Information

Fabrizio Marozzo (corresponding author)
c/o DIMES – University of Calabria
Via P. Bucci 42c
87036 Rende (CS)
Italy
Email: fmarozzo@dimes.unical.it
Telephone: +39 0984 494782

Paolo Trunfio
c/o DIMES – University of Calabria
Via P. Bucci 41c
87036 Rende (CS)
Italy
Email: trunfio@dimes.unical.it
Telephone: +39 0984 494788

Abstract

This article describes some of the most representative cloud computing development environments classified into four types. *Integrated development environments* are used to code, debug, deploy and monitor cloud applications that are executed on a cloud infrastructure. *Parallel-processing development environments* are used to define parallel applications for processing large amount of data that are run on a cluster of virtual machines provided by a cloud infrastructure. *Workflow development environments* are used to define workflow-based applications that are executed on a cloud infrastructure. *Data-analytics development environments* are used to define data analysis applications through machine learning and data mining tools provided by a cloud infrastructure.

Keywords

Azure ML; BigML; Data-analytics development environments; DMCF; Eclipse; Hadoop; Integrated development environments; IntelliJ; Parallel-processing development environments; Spark; Swift; Visual Studio; Workflow development environments.

1. Introduction

Developing cloud applications may be a complex task, with specific issues that go beyond those of stand-alone application programming. For instance, cloud programming must deal with deployment, scalability and monitoring aspects that are not easy to handle without the use of ad-hoc environments *(Talia, Trunfio and Marozzo 2015)*. In fact, to simplify the development of cloud applications, cloud computing development environments are often used. This article describes some of the most representative cloud computing development environments currently in use. The environment presented in this paper are classified into four types:

- *Integrated development environments*, which are used to code, debug, deploy and monitor cloud applications that are executed on a cloud infrastructure. The environments discussed in this article are *Eclipse*, *Visual Studio* and *IntelliJ*.

- *Parallel-processing development environments*, which are used to define parallel applications for processing large amount of data that are run on a cluster of virtual machines provided by a cloud infrastructure. The environments presented here are *Hadoop* and *Spark*.

- *Workflow development environments*, which are used to define workflow-based applications that are executed on a cloud infrastructure. The examples discussed here are *Swift* and *DMCF*.

- *Data-analytics development environments*, which are used to define data analysis applications through machine learning and data mining tools provided by a cloud infrastructure. The examples presented in this article are *Azure ML* and *BigML*.

2. Integrated Development Environments

Eclipse

Eclipse[1] is one of the most popular integrated development environments (IDEs) for software programmers that can be used to define applications in C++, Java, JavaScript, PHP, Python, R and so on, which can be run and deployed on multiple operating systems and computing platforms, including the most popular cloud computing infrastructures.

The Eclipse platform can be extended by installing plug-ins, such as development toolkits for novel programming languages and/or systems. Plug-ins can be programmed using Eclipse APIs and can be run on any of the supported operating systems. At the core of Eclipse is an architecture for discovering, loading, and running plug-ins. In addition to providing a development environment for programming languages, Eclipse supports development for most popular application servers (e.g., Tomcat, GlassFish) and is often capable of installing the required server directly from the IDE. It supports remote debugging that allows programmers to debug the code of applications running on servers.

Eclipse provides three types of products:

- *Desktop IDE*, for defining and running Java applications, C/C++ software, PHP web pages and so on in a desktop PC;
- *Cloud IDEs*, a Cloud IDE to develop software using a browser;
- *IDE Platforms*, a set of frameworks and common services to support the use of Eclipse as a component model.

Eclipse allows to program cloud applications for the main public and private cloud infrastructures. For example, *AWS Toolkit for Eclipse* is an open source plug-in that allows developers to define, debug, and deploy Java applications on Amazon Web Services. *IBM Eclipse Tools for Bluemix* enables the deployment and integration of many services from Bluemix into applications. Finally, *Google Plugin for Eclipse* simplifies the development of web applications that utilize Google cloud technology.

[1] Eclipse, https://eclipse.org/

Visual Studio

Microsoft Visual Studio[2] is a Microsoft IDE that allows users to develop, test, and deploy applications for the web, desktop, cloud, mobile, and game consoles. It is fully integrated with Microsoft technologies such as Windows API, Microsoft Office, Microsoft Azure and Windows Store.

Visual Studio includes a code editor for supporting code completion and refactoring. An integrated debugger helps to observe the run-time behaviour of programs and find problems. The functionality of the IDE can be enhanced through plug-ins, such as visual tools aiding in the development of GUI of web, desktop and mobile applications.

Visual Studio supports the most popular programming languages. For example, C++ for performance across a wide range of devices; Python for cross-platform scripting; R, for data processing; Node.js for scalable applications in JavaScript; C# as a multi-paradigm programming language for a variety of platforms, including the cloud.

Visual Studio allows to program cloud applications for Microsoft Azure and other cloud infrastructures. For example, *Visual Studio Tools for Azure* allows building, managing, and deploying cloud applications on Azure, whilst the *AWS Toolkit for Visual Studio* is a plugin that permits to develop, debug, and deploy .NET applications that use Amazon Web Services.

IntelliJ

IntelliJ[3] is an IDE for web, mobile, cloud and enterprise development that supports languages like Java, JavaScript, Groovy and Scala. It is developed by the JetBrains company and is available in an Apache Licensed community edition, and in a commercial edition. The community edition allows Java and Android development. The commercial edition extends the community edition with support for web and enterprise development. Both community and commercial editions support cloud development with sponsorship of the main cloud providers.

One important feature of IntelliJ is code completion made by analyzing the source code of a user. Specifically, IntelliJ indexes user source code, for providing relevant code suggestions and on-the-fly code analysis. As the previous two systems discussed above, IntelliJ supports plugins for adding additional functionality to the IDE, such as version control systems (e.g., GIT), databases (e.g., Microsoft SQL Server) and automatic task runners (e.g., Grunt).

IntelliJ IDEA supports the most popular Java application servers, such as Tomcat, JBoss and Glassfish. A developer can deploy, debug and monitor an application onto an application server. Moreover, IntelliJ IDEA provides dedicated plug-ins that allows programmers to manage Docker virtual machines.

IntelliJ allows developers to create and interact with cloud applications using the API of the most popular cloud infrastructures. For example, through *AWS Manager Plugin* it provides integration with AWS services like EC2, RDS and S3, whilst *Cloud Tools for IntelliJ* is a Google-sponsored plugin that allows IntelliJ developers to interact with Google Cloud Platform services.

---

[2] Microsoft Visual Studio, https://www.visualstudio.com/

[3] IntelliJ, https://www.jetbrains.com/idea/

3. Parallel-processing development environments

Hadoop

Apache Hadoop[4] is commonly used to develop parallel applications that analyse big amounts of data. It can be adopted for developing parallel applications using many programming languages (e.g., Java, Ruby, Python, C++) based on the MapReduce programming model *(Dean and Ghemawat, 2004)* on a cluster or on a cloud platform. Hadoop relieves developers from having to deal with classical distributed computing issues, such as load balancing, fault tolerance, data locality, and network bandwidth saving. The Hadoop project is not only about the MapReduce programming model (Hadoop MapReduce module), as it includes other modules such as:

- *Hadoop Distributed File System (HDFS)*: a distributed file system providing fault tolerance with automatic recovery, portability across heterogeneous commodity hardware and operating systems, high-throughput access and data reliability.
- *Hadoop YARN*: a framework for cluster resource management and job scheduling.
- *Hadoop Common*: common utilities that support the other Hadoop modules.

With the introduction of YARN in 2013, Hadoop turns from a batch processing solution into a platform for running a large variety of data applications, such as streaming, in-memory, and graphs analysis. As a result, Hadoop became a reference for several other frameworks, such as: Giraph[5] for graph analysis; Storm[6] for streaming data analysis; Hive[7], which is a data warehouse software for querying and managing large datasets; Pig[8], which is as a dataflow language for exploring large datasets; Tez[9] for executing complex directed-acyclic graph of data processing tasks; Oozie[10], which is a workflow scheduler system for managing Hadoop jobs.

Hadoop is available in most cloud infrastructures. For example, the *HDInsight* service by Microsoft Azure, the *Amazon Elastic MapReduce (EMR)* service by AWS, and *Google Cloud Dataproc* service by Google Cloud.

Spark

Apache Spark[11] is an open-source framework for in-memory data analysis and machine learning developed at UC Berkeley in 2009. It can process distributed data from several sources, such as HDFS, HBase, Cassandra, and Hive. It has been designed to efficiently perform both batch processing applications (similar to MapReduce) and dynamic applications like streaming, interactive queries, and graph analysis. Spark is compatible with Hadoop data and it can run in Hadoop clusters through the YARN module. However, in contrast to Hadoop's two-stage MapReduce paradigm in which intermediate data are always stored in distributed file systems, Spark stores data in a cluster's memory and queries it repeatedly so as to obtain better performance for several classes of applications (e.g., interactive jobs, real-time queries, and stream data) (Xin et al., 2013). The Spark project has different components:

---

[4] http://hadoop.apache.org/

[5] http://giraph.apache.org/

[6] http://storm.apache.org

[7] http://hive.apache.org

[8] http://pig.apache.org

[9] http://tez.apache.org/

[10] http://oozie.apache.org/

[11] http://spark.apache.org

- Spark Core contains the basic functionalities of the library such as for manipulating collections of data, memory management, interaction with distributed file systems, task scheduling, and fault recovery.
- Spark SQL provides API to query and manipulate structured data using standard SQL or Apache Hive variant of SQL.
- Spark Streaming provides an API for manipulating streams of data.
- GraphX is a library for manipulating and analyzing big graphs.
- MLlib is a scalable machine learning library on top of Spark that implements many common machine learning and statistical algorithms.

Several big companies and organizations use Spark for big data analysis purpose: for example, Ebay uses Spark for log transaction aggregation and analytics, Kelkoo for product recommendations, SK Telecom analyses mobile usage patterns of customers.

Similarly to Hadoop, most cloud infrastructures provide Spark as a service, like *IBM Analytics for Apache Spark*, *Azure HDInsight* and *Google Cloud Dataproc*.

4. Workflow development environments

Swift

Swift *(Wilde et al., 2011)* is a implicitly parallel scripting language that runs workflows across several distributed systems, like supercomputers, clusters, grids and clouds. The Swift language has been designed at the University of Chicago and at the Argonne National Lab to provide users with a workflow-based language for cloud computing.

Swift separates the application workflow logic from runtime configuration. This approach allows a flexible development model. The Swift language allows invocation and running of external application code and allows binding with application execution environments without extra coding from the user. Swift/K is the previous version of the Swift language that runs on the Karajan grid workflow engine across wide area resources. Swift/T is a new implementation of the Swift language for high-performance computing. In this implementation, a Swift program is translated into an MPI program that uses the Turbine and ADLB runtime libraries for scalable dataflow processing over MPI. The Swift-Turbine Compiler (STC) is an optimizing compiler for Swift/T and the Swift Turbine runtime is a distributed engine that maps the load of Swift workflow tasks across multiple computing nodes. Users can also use Galaxy *(Giardine et al., 2005)* to provide a visual interface for Swift.

The Swift language provides a functional programming paradigm where workflows are designed as a set of code invocations with their associated command-line arguments and input and output files. Swift is based on a C-like syntax and uses an implicit data-driven task parallelism *(Wozniak, Wilde, and Foster, 2014)*. In fact, it looks like a sequential language, but being a dataflow language, all variables are futures, thus execution is based on data availability. When input data is ready, functions are executed in parallel. Moreover, parallelism can be exploited through the use of the *for each* statement. The Turbine runtime comprises a set of services that implement the parallel execution of Swift scripts exploiting the maximal concurrency permitted by data dependencies within a script and by external resource availability. Swift has been used for developing several scientific data analysis applications, such as prediction of protein structures, modeling the molecular structure of new materials, and decision making in climate and energy policy.

Data Mining Cloud Framework

The Data Mining Cloud Framework (DMCF) is a software system developed at the University of Calabria for designing and executing data analysis workflows on clouds *(Belcastro, Marozzo, Talia, and Trunfio, 2015).* A

Web-based user interface allows users to compose their applications and submit them for execution over cloud resources, according to a Software-as-a-Service (SaaS) approach.

The DMCF architecture has been designed to be deployed on different cloud settings. Currently, there are two different deployments of DMCF: i) on top of a Platform-as-a-Service (PaaS) cloud, i.e., using storage, compute, and network APIs that hide the underlying infrastructure layer; ii) on top of an Infrastructure-as-a-Service (IaaS) cloud, i.e., using virtual machine images (VMs) that are deployed on the infrastructure layer.

The DMCF software modules can be grouped into *web components* and *compute components* DMCF allows users to compose, check, and run data analysis workflows through a HTML5 web editor. The workflows can be defined using two languages: *VL4Cloud* (Visual Language for Cloud) *(Marozzo, Talia and Trunfio 2016)* and *JS4Cloud* (JavaScript for Cloud) *(Marozzo, Talia and Trunfio 2015)*. Both languages use three key abstractions:

- *Data* elements, representing input files (e.g., a dataset to be analyzed) or output files (e.g., a data mining model).
- *Tool* elements, representing software tools used to perform operations on data elements (partitioning, filtering, mining, etc.).
- *Tasks*, which represent the execution of Tool elements on given input Data elements to produce some output Data elements.

The DMCF editor generates a JSON descriptor of the workflow, specifying what are the tasks to be executed and the dependency relationships among them. The JSON workflow descriptor is managed by the DMCF workflow engine that is in charge of executing workflow tasks on a set of workers (virtual processing nodes) provided by the cloud infrastructure. The workflow engine implements a data-drive task parallelism that assigns workflow tasks to idle workers as soon as they are ready to execute.

5. Data-analytics development environments

Microsoft Azure Machine Learning

Microsoft Azure Machine Learning[12] (Azure ML) is a SaaS that provides a Web-based machine learning environment for the creation and automation of machine learning workflows. Through its user-friendly interface, data scientists and developers can perform several common data analysis/mining tasks on their data and automate their workflows.

Using its drag-and-drop interface, users can import their data in the environment or use special readers to retrieve data form several sources, such as Web URL (HTTP), OData Web service, Azure Blob Storage, Azure SQL Database, Azure Table. After that, users can compose their data analysis workflows where each data processing task is represented as a block that can be connected with each other through direct edges, establishing specific dependency relationships among them. Azure ML includes a rich catalog of processing tool that can be easily included in a workflow to prepare/transform data or to mine data through supervised learning (regression e classification) or unsupervised learning (clustering) algorithms. Optionally, users can include their own custom scripts (e.g., in R or Python) to extend the tools catalog. When workflows are correctly defined, users can evaluate them using some testing dataset. Users can easily visualize the results of the tests and find very useful information about models accuracy, precision and recall. Finally, in order to use their models to predict new data or perform real time predictions, users can expose them as Web services. Always through a Web-based interface, users can monitor the Web services load and use by time.

Azure Machine Learning is a fully managed service provided by Microsoft on it Azure platform; users do not need to buy any hardware/software nor manage virtual machine manually. One of the main advantage of

[12] https://azure.microsoft.com/en-us/services/machine-learning/

working with Azure Machine Learning is its auto-scaling feature: models are deployed as elastic Web services so as users do not have to worry about scaling them if the models usage increased.

BigML

BigML[13] is provided as a Software-as-a-Service (SaaS) for discovering predictive models from data sources and using data classification and regression algorithms. The distinctive feature of BigML is that predictive models are presented to users as interactive decision trees. The decision trees can be dynamically visualized and explored within the BigML interface, downloaded for local usage and/or integration with applications, services, and other data analysis tools. Extracting and using predictive models in BigML consists in multiple steps, as detailed in the following:

- *Data source setting and dataset creation*. A data source is the raw data from which a user wants to extract a predictive model. Each data source instance is described by a set of columns, each one representing an instance feature, or field. One of the fields is considered as the feature to be predicted. A dataset is created as a structured version of a data source in which each field has been processed and serialized according to its type (numeric, categorical, etc.).
- *Model extraction and visualization*. Given a dataset, the system generates the number of predictive models specified by the user, who can also choose the level of parallelism level for the task. The interface provides a visual tree representation of each predictive model, allowing users to adjust the support and confidence values and to observe in real time how these values influence the model.
- *Prediction making*. A model can be used individually, or in a group (the so-called ensemble, composed of multiple models extracted from different parts of a dataset), to make predictions on new data. The system provides interactive forms to submit a predictive query for a new data using the input fields from a model or ensemble. The system provides APIs to automate the generation of predictions, which is particularly useful when the number of input fields is high.
- *Models evaluation*. BigML provides functionalities to evaluate the goodness of the predictive models extracted. This is done by generating performance measures that can be applied to the kind of extracted model (classification or regression).

6. Closing Remarks

This article described four categories of cloud computing development environments. *Integrated development environments* are used to code, debug, deploy and monitor cloud applications that are executed on a cloud infrastructure. *Parallel-processing development environments* are used to define parallel applications for processing large amount of data that are run on a cluster of virtual machines provided by a cloud infrastructure. *Workflow development environments* are used to define workflow-based applications that are executed on a cloud infrastructure. *Data-analytics development environments* are used to define data analysis applications through machine learning and data mining tools provided by a cloud infrastructure. For each category, the article described some of the most representative cloud computing development environments currently in use, including their main features, provided services and available implementations.

Cross References

Infrastructures for High-Performance Computing: Cloud Computing

Infrastructures for High-Performance Computing: Cloud Infrastructures

References

[13] https://bigml.com

Belcastro, L., Marozzo, F., Talia, D. and Trunfio, P. (2015) Programming visual and script-based big data analytics workflows on clouds. Advances in Parallel Computing, 26, pp. 18-31.

Dean, J., Ghemawat, S. (2004) MapReduce: Simplified data processing on large clusters, in: 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04), San Francisco, USA.

Giardine, B. et al., "Galaxy: A platform for interactive large-scale genome analysis," *Genome Res*, 15:1451–1455, 2005.

Justin, M., Wozniak, J. M., Foster, I. (2014), "Language features for scalable distributed-memory dataflow computing," *Proc. Data-flow Execution Models for Extreme-scale Computing at PACT*.

Marozzo, F., Talia, D., and Trunfio, P. (2016) "A Workflow Management System for Scalable Data Mining on Clouds". IEEE Transactions On Services Computing (IEEE TSC).

Marozzo, F., Talia, D., and Trunfio, P. (2015) "JS4Cloud: Script-based Workflow Programming for Scalable Data Analysis on Cloud Platforms". Concurrency and Computation: Practice and Experience, vol. 27, n. 17, pp. 5214--5237, Wiley InterScience.

Talia, D., Trunfio P. and Marozzo, F. (2015). Data analysis in the Cloud. Elsevier, The Netherlands.

Wilde, M., Hategan, M., Wozniak, J. M., Clifford, B., Katz, D. S., and Foster I. (2011) "Swift: A language for distributed parallel scripting", *Parallel Computing*, 37(9):633–652.

Xin, R. S., Rosen, J., Zaharia, M., Franklin, M. J., Shenker, S. and Stoica, I. (2013). Shark: SQL and rich analytics at scale. In Proceedings of the *2013 ACM SIGMOD International Conference on Management of Data* (SIGMOD '13). New York, USA.