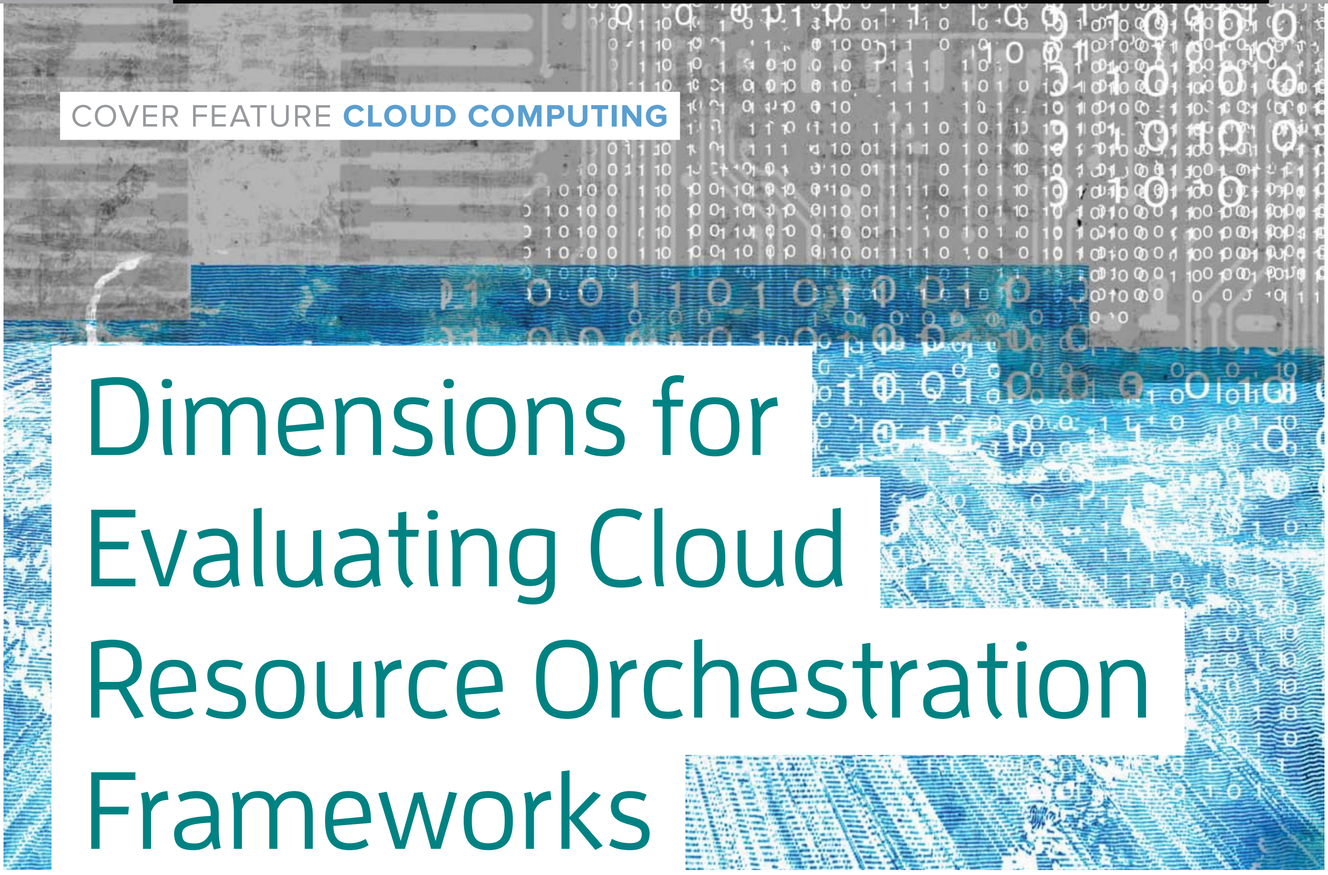

Dimensions for Evaluating Cloud Resource Orchestration Frameworks

Alireza Khoshkbarforoushha, Australian National University and Commonwealth Scientific and Industrial Research Organisation

Meisong Wang, Australian National University

Rajiv Ranjan, Newcastle University

Lizhe Wang, Chinese Academy of Sciences

Leila Alem, Commonwealth Scientific and Industrial Research Organisation

Samee U. Khan, North Dakota State University

Boualem Benatallah, University of New South Wales

Despite the proliferation of cloud resource orchestration frameworks (CROFs), DevOps managers and application developers still have no systematic tool for evaluating their features against desired criteria. The authors present generic technical dimensions for analyzing CROF capabilities and understanding prominent research to refine them.

Cloud computing assembles a large network of virtualized services—from storage and networking to databases and workload-balancing software—and enables instant access to virtually unlimited software and hardware resources. It offers a number of considerable advantages, including no upfront investment, lower operating costs, and infinite scalability. However, orchestrating the right set of cloud resources to fit application architecture optimized for the desired quality of service (QoS) remains a challenging technical problem. In simple terms, cloud resource orchestration is the selection, deployment, monitoring, and runtime management of software and hardware resources to ensure that applications meet QoS targets, such as availability, throughput, latency, security, cost, and reliability under uncertainties.[1]

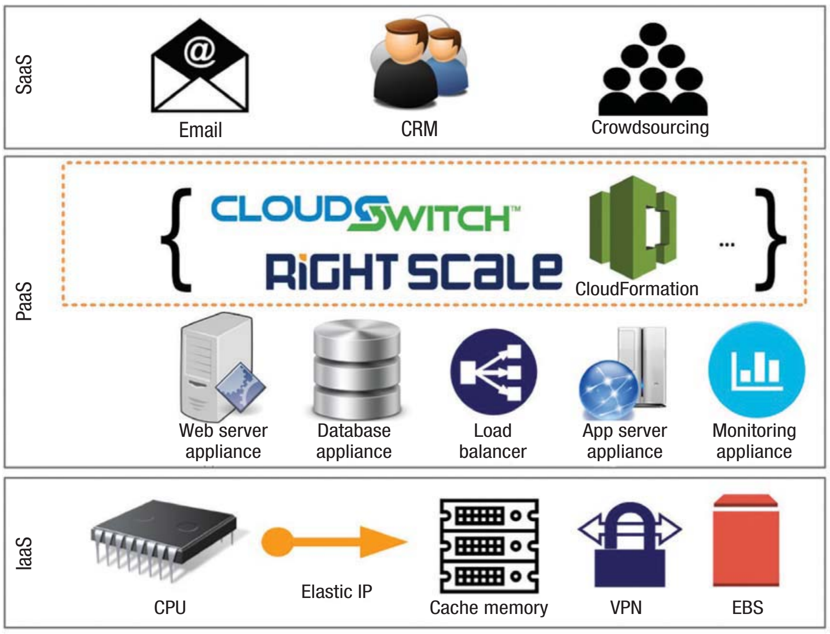

FIGURE 1. Cloud resource orchestration framework's (CROF's) position in the cloud ecosystem. Despite the many cloud-standardization projects, the community has not yet defined a comprehensive standard to cover all cloud layers including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). CRM: customer relationship management; EBS: elastic block store; VPN: virtual private network.

Since the inception of cloud computing in mid-2000, academic groups and industry vendors have developed various cloud resource orchestration frameworks (CROFs) to simplify application management. A CROF aids software engineers, DevOps managers, and infrastructure administrators in migrating to and managing in-house applications in cloud environments. Figure 1 shows how a CROF fits into the cloud computing ecosystem.

Amazon Web Services (AWS) Elastic Beanstalk is an example of an easy-to-use CROF for deploying and scaling multitier Web applications developed with popular programming languages, including Java, .NET, PHP, Node.js, Python, and Ruby. Another example is Ooyala's CROF for delivering multimedia content online. At the infrastructure layer, Ooyala's CROF leverages AWS Elastic Compute Cloud (EC2) and Simple Storage Solution (S3) resources for content distribution and storage.

PROLIFERATION WITHOUT STANDARDIZATION

With the proliferation of CROFs from competing vendors and academic groups has come an often bewildering array of frameworks with similar (if not identical) functionalities. For example, any of the CROFs from RightScale, Bitnami, EngineYard, and CloudSwitch are suitable for managing Web applications over AWS and other public cloud infrastructures. Similarly, CloudFront, Ooyala, MetaCDN, or RackSpace Cloud Files could be used to manage content delivery–network (CDN) applications. This diversity makes selection a risky task, as wrong decisions could lead to vendor lock-in, poor application QoS, excessive costs, and unwanted administration overhead. Moreover, migration from one cloud provider to another is nontrivial, if not impossible. Despite the many cloud standardization projects (http://cloud-standards.org), the community has no comprehensive standard.[2]

From the service provider's perspective, the lack of standardization among resource-orchestration techniques is no accident. Most approaches aim to improve productivity by automating the structured processes that IT departments or professional programmers handle—those more equipped for the hardware and software resource coordination that orchestration requires.

In addition to traditional control and dataflow programming constructs, there is a need for rich abstractions to support consistency across physical and logical layers; exception handling; and flexible, efficient resource coordination and management. All these supporting mechanisms must balance the tensions between the rich semantics and simplicity of comprehension, which are essential to successful practical use. Resource orchestration in cloud environments must also contend with the scale and variety of resource types and the uncertainties inherent in a cloud environment. Finally, integration and interoperational dependencies intensify the challenge of building application architecture over the cloud.[1]

EVALUATING CROFs

The variety of CROFs offered can make selection challenging for software engineers, solution architects, or DevOps managers who want to migrate their applications to the cloud. To aid that process, we developed technical dimensions for CROF analysis that provide insights into existing frameworks by answering why, what, and how frameworks complete resource orchestration. We then analyzed the strengths and weaknesses of popular frameworks in light of these dimensions and surveyed recent research work pertinent to each dimension.

Application domain

This dimension refers to the type of applications that the frameworks have been targeted and customized to, including *multitier Web applications*, *CDN*, and *large-scale data processing* (also known as big data). Multitier Web applications refers to migrating in-house Web, mobile, or gaming applications to public or private clouds for improved scalability, availability, and so on, as well as for conducting development and test activities in the cloud environment.

CROFs that target CDN applications give businesses and Web application developers an easy and cost-effective way to distribute content (including images, videos, and webpages) to sporadic geographical locations with low latency and high data-transfer speeds. CDN offerings normally use edge computing, which pushes the frontier of computing applications, data, and services away from centralized nodes to servers on the Internet's edge.

CROFs with large-scale data-intensive computing platforms rely predominantly on MapReduce, a simple yet effective programming model.[3] MapReduce adopts an inherently divide-and-conquer strategy in which a single problem is broken into multiple individual subtasks that include instances of the map and reduce tasks. This strategy is further reinforced by parallelizing subtasks in a cluster.

CROFs WITH CLOUD INTEROPERABILITY—TO PORT APPLICATIONS TO OR USE RESOURCES FROM MULTIPLE CLOUDS—ARE MORE LIKELY TO BE ADOPTED.

Resource type

This dimension refers to the resource type the framework can orchestrate. *Infrastructure* resources, which include network, CPU, and blob storage, require looking at aspects such as IP type or ports, cores or addressing bit, and storage size and format. *Platform* resources include application servers (such as the Java application server), monitoring services, database servers, and the like. *Software* components and subprocesses include customer relationship management (CRM) business processes that are formed and executed through software-component orchestration to deliver a business service to end users. An example is the Salesforce CRM platform.

All three resource categories (infrastructure, platform, and software) are mapped to the so-called cloud service model that includes the infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS) layers.

Resource access component

This dimension captures what interfaces the CROF uses to interact with cloud resources: command line, Web portal, or Web services API. All three are abstractions through which an administrator or DevOps manager manipulates cloud resources.

Command line. The command-line interface wraps cloud-specific API actions as commands or scripts that are executable through Linux (`bash` or `sh`) or Windows-based (`command.com` or `cmd.exe`) shell environments. Although command-line orchestration frameworks can be easier to implement, their use requires sophisticated technical expertise and a deep understanding of cloud resources and related orchestration operations.

Web portal. The portal presents cloud resources as user-friendly artifacts, such as buttons and checkboxes, and resource catalogs. Visual artifacts and catalogs aim to simplify resource selection, assembly, and deployment. A catalog manages resource-entity sets that can be instantiated to create CPU, storage, and network objects.

The portal also features an editor that lets users assemble and deploy applications by dragging appliance entities from the catalog, and integrating and configuring them through customized configuration-management interfaces.

These features make the Web portal simpler and more flexible for the administrator to use than the command-line tool.

Web services API. The Web services API enables other tools and systems to integrate cloud-resource-management operations into their functionalities. As such, it provides the highest abstraction and greatest simplicity of the three interface types, particularly when some cloud-platform functionalities must be integrated into other tools and systems.

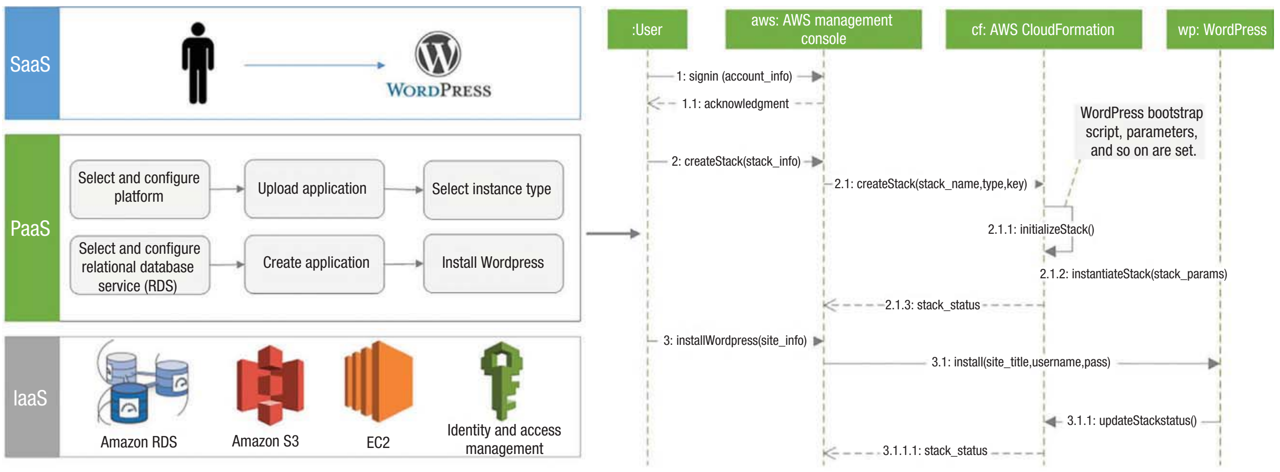
SaaS
WordPress
PaaS
Select and configure platform
Upload application
Select instance type
Select and configure relational database service (RDS)
Create application
Install Wordpress
IaaS
Amazon RDS
Amazon S3
EC2
Identity and access management
:User
aws: AWS management console
cf: AWS CloudFormation
wp: WordPress
1: signin (account_info)
1.1: acknowledgment
2: createStack(stack_info)
2.1: createStack(stack_name,type,key)
WordPress bootstrap script, parameters, and so on are set.
2.1.1: initializeStack()
2.1.2: instantiateStack(stack_params)
2.1.3: stack_status
3: installWordpress(site_info)
3.1: install(site_title,username,pass)
3.1.1: updateStackstatus()
3.1.1.1: stack_status

FIGURE 2. An example of cloud orchestration operations in Amazon Web Service (AWS) involving the deployment of WordPress through AWS Elastic Beanstalk or AWS CloudFormation with homogeneous interoperability that lets administrators manage the application across different cloud services and coordinate and streamline communication among resources within the same cloud. EC2: Elastic Compute Cloud; S3: Simple Storage Solution.

Interoperability

Cloud interoperability is a key factor in the decision to switch to cloud computing, so any platform with high interoperability has a greater chance of being adopted. Recognizing this competitive edge, platform developers are attempting to expand their multicloud capability. This dimension refers to a CROF's ability to port applications across multiple clouds or to use resources from multiple clouds to compose and host applications.

Although interoperability helps prevent cloud provider lock-in, it is nontrivial to design and implement generic resource orchestrators that can work with various clouds because APIs must be specific to each cloud provider.

Interoperability can be *hybrid or homogeneous*. Hybrid resource orchestrators operate across multiple clouds, transparently integrating their resources (IaaS, PaaS, and SaaS) as part of a single-resource leasing abstraction. With a hybrid orchestrator, administrators can manage and automate both application movement across clouds and the communication among resources hosted in different clouds. In contrast, a homogeneous orchestrator can orchestrate only the resources in a single cloud or from cloud providers who apply a similar technology stack in managing their resources.

Resource selection

CROFs differ in the degree to which they automate the selection of software and hardware resources. Selection involves identifying and analyzing alternatives (cloud resources) on the basis of the decision maker's (typically the administrator's) preference. Decision makers must not only identify as many feasible alternatives as possible but also choose the one that best fits their selection criteria.

In resource selection, approaches can be either manual or automated. In the *manual* approach, a resource orchestrator assumes that administrators have sophisticated technical knowledge that can help them select the most appropriate cloud resources. In an *automated* approach, the orchestrator implements a recommendation system (RS) that helps with resource selection to best fit the desired QoS, features, and cost.[4–6] The RS ranks different resources according to predefined selection criteria and presents them to the administrator.

Application deployment

The scale and complexity of applications and cloud resources make them increasingly difficult and expensive to administer and deploy. A 2010 study of enterprise applications in Fortune 100 companies contained some interesting statistics.[7] The total number of distinct appliances required for each application varied from 11 to over 100, depending on application type. Some applications required up to 19 distinct front-end webservers, 67 application servers, and 21 back-end databases.

Clearly, to ensure that configurations are error free, any application-deployment technique must account for dependencies across appliances. Existing deployment tools, which provide varying automation levels, fall into manual, script-based, language-based, and model-based approaches. Higher automation levels are preferable to enhance correctness, speed, and ease of documentation.

In the *manual* approach, DevOps managers configure and integrate appliances manually by inserting the XML and text snippets into configuration files. The *script-based* approach consists of shell scripts that execute on CPU resources hosting the appliances. The scripts directly modify the appliance-specific configuration file. Both manual and script-based approaches have limited ability to express dependencies, react to changes, and verify configurations, which results in erroneous application configuration in large-scale deployments.

Language-based approaches are more powerful and expressive, declaratively defining appliance configurations. *Model-based* approaches define a desired state for each appliance. Once instantiated on CPU resources, appliances automatically fetch and execute their state definition from a centralized repository. Both language- and model-based approaches can handle dependencies across appliances and automatically react to changes, such as resource failure, by activating adaptation actions.

Figure 2 illustrates an instantiation of cloud orchestration operations in AWS, which features manual resource selection and a script-based application deployment in a single-cloud environment.

Runtime QoS adaptation

CROFs must be able to adapt to dynamic exceptions either manually or automatically, such as acting appropriately when some threshold is reached. *Manual* adaptation does not provide any autoscaling facility; in the best case, the CROF alerts the administrator through an email of the need to manually configure the instances to adapt to new conditions. A CROF with *automatic* adaptation will adapt to exceptions through the use of reactive and predictive techniques.

Reactive techniques respond to events only after reaching a predefined threshold that is determined through monitoring the state of hardware and software resources. Although these techniques are simple to define and implement (nothing more than if-then-else statements), they are not sufficient to ensure guaranteed QoS in some cases, such as during a peak demand for resources.

Predictive techniques can dynamically anticipate and capture the relationship between an application's QoS targets, current hardware resource allocation, and changes in application-workload patterns to adjust hardware allocation. Overall, predictive techniques build on the integration of theoretical workload prediction and resource performance models.[6,8,9] Workload prediction models forecast workload behavior across applications in terms of CPU, storage, I/O, and network bandwidth requirements.

Predictive models will be the foundation of next-generation resource-provisioning frameworks, which will completely understand workload and resource demands and will therefore have a higher resilience to uncertainties.

FRAMEWORK ANALYSIS

To give administrators a glimpse of how our technical dimensions aid CROF evaluation, we chose 14 CROFs that formed a mix of proprietary and open source frameworks. All frameworks have active users, reasonably stable versions, and good documentation. Table 1 shows the results of evaluating the frameworks in terms of our seven dimensions.

We also investigated the latest research studies, particularly in resource selection, application deployment, and runtime QoS adaptations.

Application domain

Most off-the-shelf or open source CROFs support managing and migrating in-house multitier Web applications over to a cloud environment. CloudSwitch, CA AppLogic, EngineYard, Bitnami, AWS Elastic Beanstalk, CloudBees, and RightScale are some frameworks in this category.

RightScale, GoGrid, and RackSpace have large-scale data processing as well; however, Amazon and Google offer independent platforms: Amazon Elastic MapReduce and Google BigQuery. The solutions for large-scale data processing are built on Hadoop, an open source framework in which data and processes are distributed across a resizable cluster of computing server instances. Consequently, once the application and data are ready for processing, the administrator must specify and configure the number and types of computing resources to crunch the deluge of data. This task is challenging because large-scale data processing platforms like Hadoop have as many as 200 configuration parameters, which precludes ad hoc settings or at least makes them risky for job performance.

In light of these constraints, CROFs should come with what-if analysis capabilities so that users can configure and tune cloud resource parameters and data-intensive computing platform settings at the IaaS and PaaS layers.

GoGrid and Rackspace also offer CDN services, with Rackspace using the Akamai network to efficiently deliver content to edge nodes. For CDN applications, Amazon's CloudFront is integrated with other AWS platform to distribute content with low latency and at high data-transfer speeds.

Resource type

Most cloud frameworks orchestrate virtual appliances (application and database servers, for example) in the PaaS layer according to major infrastructure assets, such as Amazon EC2 and S3 for computing and storage. Other frameworks, such as Google App Engine (GAE), operate on Google datacenters and Windows Azure, building on Microsoft's infrastructure services.

TABLE 1. Mapping cloud resource orchestration frameworks (CROFs) to evaluation dimensions.

CROF	Application domain	Resource type	Resource access component	Interoperability	Resource selection mode	Application deployment mode	Runtime QoS adaptation
CloudSwitch	Web (migration)	PaaS	Command line, Web portal, Web services API	Multicloud (Amazon EC2, Terremark)	Manual	Script and model based	Reactive
RightScale	Web, gaming, mobile apps, large-scale data processing, development and test	PaaS	Command line, Web portal	Multicloud (AWS, Datapipe, Google Compute Engine, HP Cloud, IDC Frontier, Rackspace, Softlayer, Windows Azure, CloudStack, OpenStack)	Manual	Script and language based	Reactive
CA AppLogic	Web application	PaaS	Command line, Web portal, Web services API	Single cloud	Manual	Script, language, and model based	Reactive
Engine Yard	Web and mobile	PaaS	Command line, Web portal, Web services API	Multicloud (AWS, Windows Azure, CloudStack, Terremark)	Manual	Script and language based	Reactive
Bitnami	Web	PaaS	Command line, Web services API	Multicloud (AWS, Windows Azure, Amazon EC2, RightScale)	Manual	Script based	Reactive
AWS Elastic Beanstalk	Web, development and test, large-scale data processing with Amazon Elastic MapReduce, CDN with CloudFront	PaaS	Command line, Web portal, Web services API	Single cloud (Amazon EC2, S3 services)	Manual	Manual and script based (for bootstrapping applications via CloudFormation)	Reactive
CloudBees	Web, development and test	PaaS	Command line, Web portal, Web services API	Multicloud (AWS, OpenStack, HP Cloud Services)	Manual	Manual	Reactive
OpenNebula	None	IaaS	Command line, Web portal, Web services API	Multicloud (vCloud, OpenStack, Eucalyptus, Amazon EC2)	Manual	Manual and model based	Reactive
Eucalyptus	Development and test	IaaS and PaaS	Command line, Web portal, REST-based API	Multicloud (AWS)	Manual	Manual	Reactive
CohesiveFT	Web application (migration)	PaaS	Web portal	Multicloud (IBM SmartCloud Enterprise, Amazon EC2, Amazon VPC, ElasticHosts, Cloud Sigma, Flexiant, Eucalyptus, OpenStack, vCloud)	Manual	Model based	Not known
Google App Engine	Web and mobile, development and test, large-scale data processing with BigQuery	PaaS (IaaS services through Google Compute Engine)	Web portal, REST-based API	Single cloud	Manual	Manual	Reactive
Microsoft Azure	Web and mobile, development and test	PaaS	Command line, Web portal, Web services API	Multicloud (Engine Yard)	Manual	Manual	Reactive
GoGrid	Web, large-scale data processing, CDN, development and test	IaaS, PaaS, and PaaS	Command line, Web portal, REST-based API	Multicloud (Windows Azure)	Manual	Manual	Reactive
RackSpace	Web, large-scale data processing, CDN (Akamai)	IaaS and PaaS	Command line, Web portal, Web services API	Multicloud (OpenStack)	Manual	Manual	Reactive

*AWS: Amazon Web Services; CDN: content-delivery network; EC2: Elastic Compute Cloud; IaaS: infrastructure as a service; PaaS: platform as a service; REST: Representational State Transfer; S3: Simple Storage Service; SaaS: software as a service.

Apart from proprietary IaaS platforms are open source solutions, including OpenStack, which serves the IaaS layer of frameworks such as RackSpace, NeCTAR, OpenNebula, CloudStack, and Eucalyptus. Among these, OpenStack and CloudStack, both Apache-licensed cloud computing programs, feature well-defined and documented APIs as well AWS.

Resource access component

All 14 CROFs we evaluated support Web-based interfaces. The provided UIs from cloud frameworks generally do not differentiate between public users and administrators in their simplicity and expressiveness. However, OpenNebula offers various kinds of user interfaces for accessing the virtual computing environment through two Web services—OCCI (http://opennebula.org/documentation) and EC2—and two Web interfaces, OpenNebula Sunstone for administrators and OpenNebula Self-Service for public users.

RECENT DEVELOPMENTS AIM TO SIMPLIFY INTEROPERABILITY BY IMPLEMENTING A SINGLE API THAT ABSTRACTS APIs IN MULTIPLE CLOUDS.

Having a Web services API increases the chance that administrators and developers will be able to integrate orchestrator functionalities into their tools as needed. Most of the 14 frameworks evaluated have APIs, although the functionalities provided are sometimes more limited than their Web user interfaces. EngineYard, for example, is still working on having the API as a first-class citizen on their platform.

Interoperability

RightScale is one of the pioneers in promoting interoperability, supporting more than 10 public or private clouds through its multicloud platform. Cloud-specific differences are sufficiently abstract that users can focus on running applications and access resources on their own terms through either the RightScale Dashboard or API.

CohesiveFT is another highly interoperable platform that provides a kind of software factory for assembling and deploying servers to many public or private cloud platforms. In the same manner, CloudSwitch (www.verizonenterprise.com/solutions/cloud) provides a topology manager that abstracts the cloud provider's details from the provisioning and management infrastructure that the application requires. Consequently, CloudSwitch can accommodate new providers by allowing them to model and install their interfaces without affecting any existing cloud providers.

The dominance of AWS and its user diversity makes it a desirable interoperability target for almost every cloud provider. CloudStack, OpenStack, and Windows Azure are other potential targets. As Table 1 shows, they are supported by Bitnami, OpenNebula, EngineYard, and CloudBees.

Recent developments aim to simplify interoperability by implementing a single API that abstracts APIs in multiple clouds. Examples include Delta Cloud (http://deltacloud.apache.org) and JCloud (http://jclouds.apache.org). The most recent release of Delta Cloud abstracts dozens of cloud providers such as Amazon EC2, GoGrid, OpenNebula, OpenStack, Rackspace, Eucalyptus, and Windows Azure APIs into a single API.

Although these APIs can simplify implementation across multiple clouds, developers must still accommodate the heterogeneities in appliance packaging, virtualization technology, resource naming, and so on.

Resource selection

Most CROFs have ad hoc resource selection, which means that application composition is largely up to the administrator, who needs specific and deep knowledge about workload demands and balancing. The growing use of cloud computing technologies is forcing industry to work on smarter approaches such as an RS. In the near future, CROFs could guide administrators in more precise resource selection. For example, CloudSwitch features CloudFit (www.techrepublic.com/resource-library/whitepapers/cloudswicth-architecture-overview), which evaluates the fit of the application composer's requested configurations against available cloud resources such as Amazon EC2. By automatically selecting the appropriate combination of resources (processor, memory, and storage, for example), CloudFit ensures that cloud deployments operate with sufficient performance and reliability.

A cloud framework could also employ log files along with best practices to adjust a fitness function that balances a resource pool against user requirements, in much the same way expert systems have done in the medical domain.

Many research efforts have focused on automatic resource selection. One proposed framework, CloudGenius, automates decision making on the basis of a model tailored for web-server migration to the cloud.[4] The framework leverages the analytic hierarchy process (AHP) to translate cloud service selection steps into a multicriteria decision-making problem. CloudRecommender[5] adopts a novel declarative approach for selecting cloud-based infrastructure services, automatically mapping users' specified application requirements to cloud-service configurations.

Another proposed framework incorporates an online decision-support system for resource management that addresses both task scheduling and resource-management optimization.[6] The framework uses fuzzy and neural network–based methods to predict virtual machine workload patterns and migration time.

Application deployment

In this dimension, CA AppLogic is a step ahead, providing a sound model-based deployment that lets administrators define appliance workflow by dragging and dropping resource icons and thus specifying dependencies among appliances, which avoids using all resources at the same time. Another interesting feature is that stopping the application stops appliances in the reverse of their start order.

CloudSwitch also features model-based deployment, although at a different abstraction level relative to CA AppLogic. CloudSwitch uses cloud isolation technology, a sandboxing technology that hides the hosting environment complexity of the public cloud from an application, enabling the administrator to automatically assign cloud resources to application components. Every data item is encrypted end to end to ensure security and privacy.

WITH MODEL-BASED APPROACHES, AN APPLICATION WILL WORK IN THE CLOUD JUST AS IT WOULD IN THE DATACENTER WITH THE SAME CONFIGURATIONS AND IDENTITIES.

Model-based approaches, such as those in CA AppLogic and CloudSwitch, guarantee that the application will work in the cloud just as if it were in the datacenter, because all configurations (IP and MAC addresses) and identities will be the same.

At the same abstraction level as CloudSwitch, CohesiveFT enables images to be created and configured dynamically on the basis of the users' preferences. The images can then be uploaded to the cloud infrastructure.

The AWS CloudFormation service complements the deployment of AWS's Elastic Beanstalk by automating the creation and management of related AWS infrastructure resources. The service instantiates a template—a text file in JavaScript Object Notation that describes all the required AWS resources for running the application—and a stack, which is a set of AWS resources that are created and managed as a single unit when CloudFormation instantiates a template. An interesting feature of CloudFormation is automatic rollback on error, which guarantees that stacks are fully created or not created at all.

A recently proposed peer-to-peer architecture uses a component repository to manage software-component deployment, enabling elasticity by using the underlying cloud infrastructure provider.[10] The provided peer-to-peer architecture has three logical layers:

- a design tier holds the description of services,
- a management tier manages service provisioning, and
- a cloud infrastructure tier enables the creation of on-demand infrastructure.

A proof-of-concept implementation is proposed in which the design tier contains a template designer for both the Eclipse and Netbeans integrated development environments and a cloud-infrastructure tier that supports EC2 and an internal HP cloud platform. Because the implementation is open source, it can be extended to include new components and features.

Runtime QoS adaptation

As Table 1 shows, all 14 evaluated frameworks have reactive QoS adap-

ABOUT THE AUTHORS

ALIREZA KHOSHKBARFOROUSHHA is a doctoral student in the College of Engineering and Computer Science at the Australian National University (ANU) and a graduate researcher in Data61 of the Commonwealth Scientific and Industrial Research Organisation (CSIRO) in Canberra, Australia. His research interests include data-intensive systems and streaming workload management in the cloud. Khoshkbarforoushha received an MS in computer science from Tarbiat Modares University, Tehran. He is a member of ACM. Contact him at a.khoshkbarforoushha@anu.edu.au.

MEISONG WANG is a master of research student in the College of Engineering and Computer Science at ANU and an assistant researcher at CSIRO. His research interests include cloud computing and data-intensive systems. Wang received an MS in software engineering from the Harbin Institute of Technology. Contact him at deanmeisong@gmail.com.

RAJIV RANJAN is an associate professor in the School of Computing Science at Newcastle University. His research interests include cloud computing, the Internet of Things, and big data. Ranjan received a PhD in computer science from the University of Melbourne. He is a member of IEEE. Contact him at raj.ranjan@ncl.ac.uk.

LIZHE WANG is a professor at the Institute of Remote Sensing & Digital Earth of the Chinese Academy of Sciences and a ChuTian Chair Professor in the School of Computer Science at the China University of Geosciences. His research interests include high-performance computing, e-science, and spatial data processing. Wang received a D.Eng. from the University of Karlsruhe. He is a Fellow of the Institution of Engineering and Technology (IET) and the British Computer Society (BCS). Contact him at lizhe.wang@gmail.com.

LEILA ALEM is an adjunct professor in the User Centered Technology Design Research Centre Faculty of Engineering and Information Technology at the University of Technology, Sydney. While conducting the research reported in this article, Alem was a principal research scientist at CSIRO. Her research interests include human–computer interaction, computer-supported cooperative work, and wearable computing. Alem received a PhD in artificial intelligence from University of Nice Sophia Antipolis. Contact her at leila.alem@uts.edu.au.

SAMEE U. KHAN is an associate professor in the Department of Electrical and Computer Engineering at North Dakota State University. His research interests include the optimization, robustness, and security of the cloud; grid, cluster, and big data computing; social networks; wired and wireless networks; power systems; smart grids; and optical networks. Khan received a PhD in computer science from the University of Texas at Arlington. He is a Fellow of the IET and the BCS. He is an ACM Distinguished Lecturer, a member of ACM, and a Senior Member of IEEE. Contact him at samee.khan@ndsu.edu.

BOUALEM BENATALLAH is a Scientia Professor in the School of Computer Science and Engineering at the University of New South Wales, Australia. His research interests include API engineering, Web services composition, federated cloud services orchestration, crowdsourcing services, and business process management. Benatallah received a PhD in computer science from Grenoble University, France. He is a member of IEEE and ACM. Contact him at boualem@cse.unsw.edu.

tation. Some CROFs provide different APIs, leaving the implementation of autoscaling and runtime adaptation to the developer. GoGrid, for example, does not automate platform-infrastructure scaling, allowing cloud users to manually scale server and storage resource configurations up or down through the customer portal. When the computational server is virtualized, only physical memory allocation changes; CPU and local storage allocation remain the same. GoGrid gives developers the option of implementing a custom autoscaling service through its APIs (https://wiki.gogrid.com/index.php/API).

Work continues in predictive QoS adaptation. One proposed strategy uses prediction-based resource measurement and provisioning based on a neural network and linear regression.[8] The prediction method uses historical data generated from running a standard client–server benchmark for training forecasting models on Amazon EC2.

Taking a different tack from a general predictive framework, other researchers proposed predictive models for resource provisioning for read-intensive multtier applications in the cloud.[9] Yet another effort produced a resource-prediction and provision scheme that uses time-series analysis based on an autoregressive integrated moving average (ARIMA) as the foundation for a prediction model.[11]

Another interesting study resulted in an online temporal data-mining system to model and predict virtual machine demands.[12] The system extracts high-level characteristics from the virtual machine–request stream and notifies the provisioning system to prepare virtual machines.

With the diversity of CROFs—many with the same functions—software engineers, solution architects, and DevOps managers can find it challenging to select the most suitable one for their needs. Our seven dimensions facilitate CROF evaluation by providing a common set of characteristics for analysis, breaking down framework capabilities, deepening the awareness of their strengths and weaknesses, and enabling more informed selection decisions. C

REFERENCES

1. R. Ranjan et al., "Cloud Resource Orchestration Programming: Overview, Issues, and Directions." *IEEE Internet Computing*, vol. 19, no. 5, 2015, pp. 46–56.
2. G.A. Lewis, "Role of Standards in Cloud-Computing Interoperability," *Proc. 46th IEEE Hawaii Int'l Conf. System Sciences* (HICSS 12), 2012, pp. 1652–1661.
3. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Comm. ACM*, vol. 51, no. 1, 2008, pp. 107–113
4. M. Menzel et al., "CloudGenius: A Hybrid Decision Support Method for Automating the Migration of Web Application Clusters to Public Clouds," *IEEE Trans. Computers*, vol. 64, no. 5, 2014, pp. 1336–1348.
5. M. Zhang et al., "A Declarative Recommender System for Cloud Infrastructure Services Selection," *Proc. 9th Int'l Conf. Economics of Grids, Clouds, Systems, and Services* (GECON 12), 2012, pp. 102–113.
6. F. Ramezani, J. Lu, and F. Hussain, "An Online Fuzzy Decision Support System for Resource Management in Cloud Environments," *Proc. Joint IEEE/IFSA World Congress and NAFIPS Ann. Meeting* (IFSA/NAFIPS 13), 2013, pp. 754–759.
7. M. Hajjat et al., "Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud," *ACM SIGCOMM Computer Comm. Rev.*, vol. 40, no. 4, 2010, pp. 243–254.
8. S. Islam et al., "Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud," *Future Generation Computer Systems*, vol. 28, no. 1, 2012, pp. 155–162.
9. W. Iqbal et al., "Adaptive Resource Provisioning for Read Intensive Multi-Tier Applications in the Cloud," *Future Generation Computer Systems*, vol. 27, no. 6, 2011, pp. 871–879.
10. J. Kirschnick et al., "Towards an Architecture for Deploying Elastic Services in the Cloud," *Software: Practice and Experience*, vol. 42, no. 4, 2012, pp. 395–408.
11. W. Fang et al., "RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center," *Proc. 9th IEEE Int'l Conf. Services Computing* (SCC 12), 2012, pp. 609–616.
12. Y. Jiang et al., "ASAP: A Self-Adaptive Prediction System for Instant Cloud Resource Demand Provisioning," *Proc. 11th IEEE Int'l Conf. Data Mining* (ICDM 11), 2011, pp. 1104–1109.