Cloud Resource Orchestration Programming

Overview, Issues, and Directions

The pervasiveness and power of cloud computing alleviates some of the problems application administrators face in their existing hardware and locally managed software environments. However, the rapid increase in scale, dynamicity, heterogeneity, and diversity of cloud resources necessitates having expert knowledge about programming complex orchestration operations (for example, selection, deployment, monitoring, and runtime control) on those resources to achieve the desired quality of service. This article provides an overview of the key cloud resource types and resource orchestration operations, with special focus on research issues involved in programming those operations.

Rajiv Ranjan
CSIRO and University of New South Wales

Boualem Benatallah
University of New South Wales

Schahram Dustdar
Vienna University of Technology, Austria

Michael P. Papazoglou
Tilburg University, the Netherlands

Over the last few years, cloud computing has emerged as the new model of distributed computing by offering hardware and software resources as virtualization-enabled services. Cloud computing[1] providers such as Amazon Web Services (AWS) and Microsoft Azure give application owners the option to deploy their application over a network with a virtually infinite resource pool with practically no upfront capital investment and with modest operating costs. Today, cloud computing systems (see http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf) follow a service-driven, layered software architecture model (see Figure 1), with Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

Key to exploiting the potential of cloud computing is the issue of *resource orchestration* (RO).[2,3] Based on the analysis of several research papers, commercial products, and analysts reports, we define RO as the set of operations that cloud providers (such as AWS) and application owners (such as Netflix) undertake (either manually or automatically via computer programs) for selecting, deploying, monitoring, and dynamically controlling the configuration of hardware and software resources as a system of quality of service (QoS)-assured components that can be seamlessly delivered to end users. As Figure 1a shows, RO operations

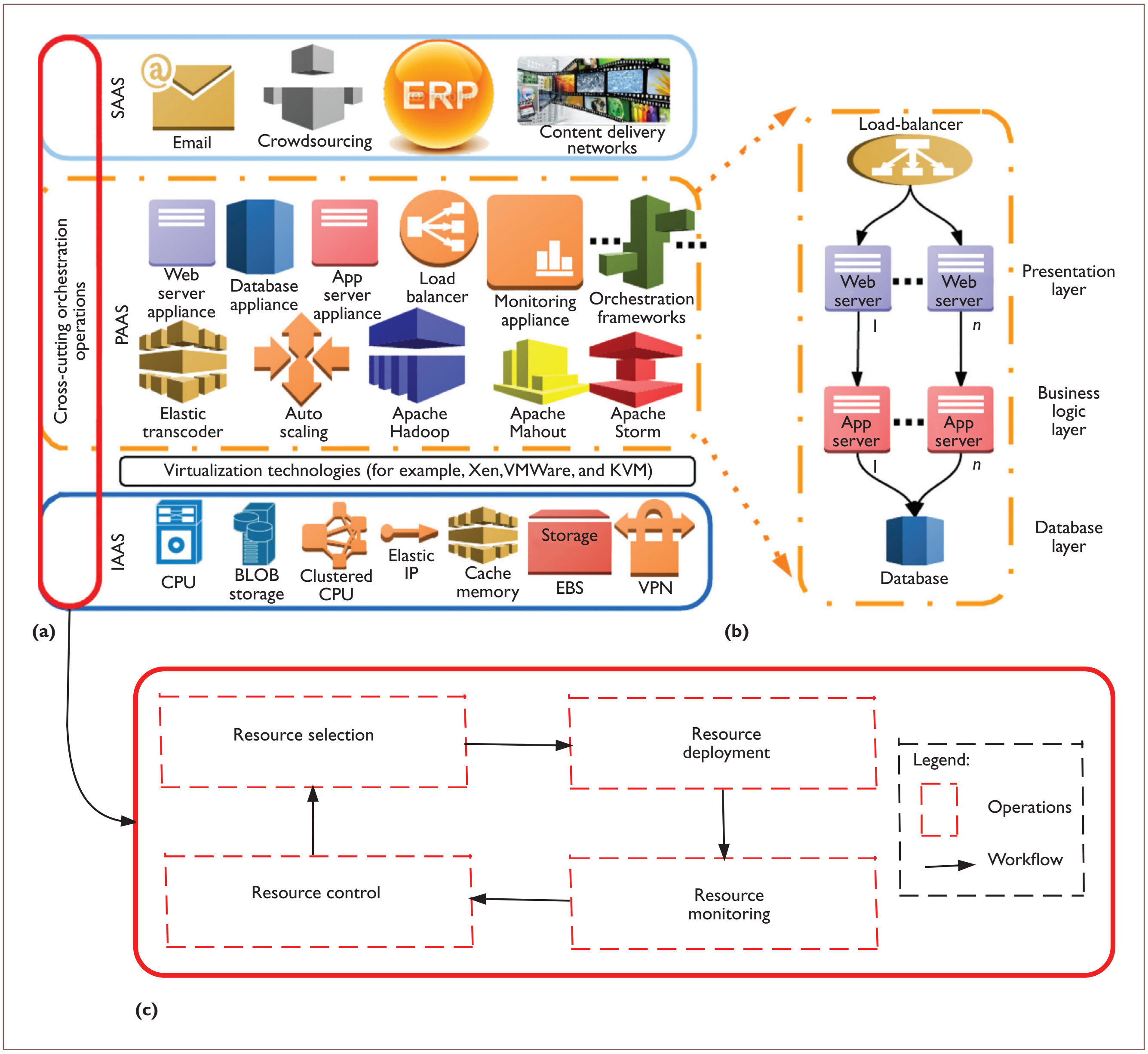

Figure 1. Overview of a cloud computing system. (a) Reference cloud resource stack. The architecture provides a layered approach to characterizing resources based on their attributes and granularity. (b) High-level architecture of a multilayered enterprise application consisting of clients, a load balancer, Web servers, application servers, and database management servers. The flow of requests between these layers is often complex. Each layer might instantiate multiple software resources, and each software resource might need to be replicated on multiple hardware resources (for example, CPUs), while load balancers distribute requests across instances of software resources. (c) Abstract resource orchestration (RO) operations in the lifecycle of an enterprise application. (BLOB = Binary Large Object; EBS = Elastic Block Store; ERP = enterprise resource planning; IaaS = Infrastructure as a Service; PaaS = Platform as a Service; SaaS = Software as a Service; and VPN = virtual private network.)

span across all the layers of a cloud computing stack. The overall goal of RO is to ensure successful hosting and delivery of applications (SaaS) by meeting QoS objectives of cloud application owners (for example, maximizing availability and throughput, while minimizing latency and avoiding an overload) and resource providers (maximizing usage, energy efficiency, profit, and so on), respectively. A recent report from the Open Data Center Alliance defines 19 usage scenarios of RO, spanning across all three layers of the cloud stack (see www.opendatacenteralliance.org/docs/ODCA_Service_Orch_MasterUM_v1.0_Nov2012.pdf).

Programming RO is challenging, because cloud applications are composed of heterogeneous software and hardware resources that are deployed across the cloud stack and might have complex integration and interoperation dependencies. Currently, orchestrating cloud resources requires human familiarity with the various providers and extensive manual programming. This is inadequate, given the dynamic variation of application resource requirements, and the proliferation of autonomous and heterogeneous cloud service providers offering resources at different layers (IaaS, PaaS, and SaaS). Dynamic variation of application resource requirements[4,5] arise from a number of factors, including resource capacity demand (such as bandwidth, memory, and processing power), failures (of a network link or resource), end-user access patterns (number of users, request arrival pattern burstiness, request service time distribution, and user location), and variations in resource prices. Modern configuration management solutions such as Amazon OpsWorks and Puppet provide support for describing resource configuration over cloud services. However, even sophisticated professional programmers and system administrators regularly resort to understanding different low-level cloud service APIs, command-line languages, Web interfaces, and procedural programming, to create and maintain complex cloud resource configurations. Given the importance of resources orchestration to cloud service consumers, major cloud service providers are rapidly improving their cloud resource-management capabilities. Recent offerings such as CloudSwitch (see https://home.cloudswitch.com), Azure Fabric Controller (see http://fabriccontroller.net), and AWS CloudFormation (see http://aws.amazon.com/cloudformation) exemplify such trends.

To help navigate this terrain, here we characterize cloud resources orchestration in a multilayered stack and highlight the main research challenges involved with programming orchestration operations for different cloud resource types.

RO Operations for Hosting Enterprise Applications on the Cloud

The application architecture (such as content delivery networks, streaming Big Data analytics applications, and high-performance computing applications) determines how, when, and which orchestration operations should be affected on cloud resources. Though lack of space doesn't permit discussion about all application architectures, here we discuss some orchestration operations for managing typical enterprise applications (see https://media.amazonwebservices.com/AWS_Web_Hosting_Best_Practices.pdf). Figure 1b depicts the high-level architecture of an enterprise application, which consists of multiple software resource layers, including the presentation, business logic, and data layers. Across each layer, we must program a number of orchestration operations to control the resources at design time, as well as at runtime, to fulfill the QoS objectives. We detail the operations in the following paragraphs (see also Figure 1c).

Selecting resources (at design and runtime). An application owner analyzes candidate software resources to determine whether we can select them for realizing the required functionality satisfying certain resource requirements and constraints (for example, interoperability with other software resources, compatibility with target hardware resources, cost, availability, and so on). Next, we select the compatible hardware resources that we can allocate to software resources.

Deploying resources (both design time and runtime). This operation involves instantiating software resources on cloud services and configuring them for communication and interoperation with other software resources. Integrating an application server with the database server (see Figure 1b) is a salient example of this orchestration operation.

Monitoring resources (runtime). Monitoring QoS attributes of cloud applications involves detecting event patterns (such as a load spike) from information produced by deployed resources (for example, application usage statistics).

Controlling resources (runtime). Based on event patterns detection, a resource orchestrator can react to deviations in application behaviors and initiate (policy-based) corrective actions, ideally without disrupting the runtime system. An example resource control operation could be to horizontally scale a database server by migrating it from a small CPU resource configuration to an extra-large CPU resource in AWS Elastic Compute Cloud (EC2) for improving throughput.

Cloud Resource Types and Orchestration Challenges

Now, let's look at each resource type through examples (see Figure 1a) and analyze the core research challenges involved with programming orchestration operations.

IaaS

The CPU, storage, and network resources in cloud environments are supplied by a collection of data centers installed with hundreds to thousands of physical resources such as cloud servers, storage repositories, and network backbone. These resources expose configuration attributes (see Table 1) that define consumable features and functions that are available from hardware resources. Providers manage these physical resources through hardware virtualization technologies, such as Xen,[6] Citrix, and VMware (see www.vmware.com/au/virtualization).

A CPU resource is essentially a piece of virtualization software running on the physical cloud server. It's the most common method of exposing the computational power to software resources; where we get finer-granularity accessibility and flexibility at the super-user level that can help customize the placement of software resources for QoS. The CPU resource emulates the properties of a physical CPU resource by providing a virtual CPU: a network card, physical memory, and hard disk. Table 1 shows orchestration operations relevant to IaaS resources.

The second IaaS-level hardware resources are the Binary Large Object (BLOB) data storage resources, which let users store raw application data on virtualized disks and access them anytime from any point on the Internet. BLOB storage (such as AWS Simple Storage Service) can hold video, audio, photos, and archived email messages, and let applications store and access data from any point on the Internet. This storage type aims to enforce fault-tolerant behavior through redundancy. For example, Azure provides different levels of redundancy[7,8] options for its BLOB and other types of storage resources (queues and tables), including local redundant storage, geo redundant storage, and read access–geo redundant storage.

A CPU resource has access to its local hard disk. However, by default, the local disk is nonpersistent; once the instance of a CPU resource is terminated, its local storage contents are purged. To overcome this issue, cloud providers offer off-instance storage resources that persist independently from the life of a CPU resource. These off-instance storage resources are referred to as the Elastic Block Store (EBS) and XDrive in AWS EC2 and Microsoft Azure, respectively. Principal advantages of designing applications using off-instance storage include the following: automatic data replication — this prevents data loss due to a single point of failure; and point-in-time data snapshot creation and backup to cloud-specific BLOB storage resources.

As the need for high-volume data transfer and communication across network boundaries grows for applications, networking resources (for example, routers, switches and communication bandwidth, AWS elastic IP, OpenFlow, and the AWS security group) become a vital component at the IaaS level. Network resources provide a variety of functionality, including bandwidth, virtual overlays for isolating traffic, guaranteeing message delivery delay, encrypting communication channels, and network monitoring.

Programming IaaS-Layer Orchestration Operations

Here we discuss research issues in programming orchestration operations at the IaaS layer.

Selecting optimal IaaS resources. The diversity of offerings at this layer leads to complex decision-making problems of optimal comparison and selection of IaaS resources from multiple cloud providers. For example, how does an application engineer compare the cost and performance features of hardware resources offered by different providers such as AWS and Azure? Similarly, an engineer can choose one provider for storage-intensive applications and another for computation-intensive applications. During the selection process, an engineer must consider many attributes (see Table 1), including goals, comparison benchmarks, and resource type alternatives. The main research challenges include how to identify and formulate selection criteria and solve qualitative (that is, the virtualization format and cloud location) and quantitative (for example, minimizing response time and cost) QoS constraints while considering a large number of IaaS resource alternatives and application use cases. Existing approaches have focused on applying combinatorial optimization,[9] evolutionary optimization,[10] and multicriteria[11] decision-making techniques for solving the selection problem.

Controlling concurrency. Orchestration operation on a particular class of hardware resources (such as a CPU resource) is enforced by invoking their

Table 1. IaaS, PaaS, and SaaS resource types, their attributes, and list of supported orchestration operations.

Service	Hardware resources	Attributes	Supported orchestration operations
IaaS	CPU	Cores, speed, family, physical memory capacity, storage capacity, addressing bits, I/O performance, renting cost, type (single or cluster of templates), resource sharing (multitenant or dedicated), physical location of cloud, availability zone, availability, performance statistics, service-level agreement (SLA), security, privacy, and integrity	Start, stop, restart, select, mount off-instance storage, monitor, reconfigure, assign IP, select cloud location, select availability zone, scale-in, scale-out, authorize, and authenticate
	BLOB storage	Type (persistent or nonpersistent), storage size, storage format, renting cost, location of host cloud, availability zone, availability, performance statistics, SLA, security, privacy, and integrity	Create new buckets, upload file, download file, scale-in, scale-out, monitor, encrypt, decrypt, authorize, and authenticate
	Network	IP Type (static or dynamic), version (IPV4 or IPV6), renting cost, message encryption cost, URL, data transfer-in cost, data transfer-out cost, connection hour, availability, performance statistics, SLA, security, privacy, and integrity	Allocation of IP addresses, URL, ports, availability zone, VPN to CPU resources, and monitor
PaaS		Feature (Web server, database server, load balancer, authorization server, and so on), virtualization format (such as Xen and VMware), environment (host operating system, implementation language such as Java, .Net, PHP, or Ruby on Rails), legal and regulatory issues, security, reliability, integrity, licensing terms and costs, initialization scripts, availability, performance statistics, and SLA	Start, stop, restart, select, allocate hardware resources, integrate with other appliances, install script, monitor, create, migrate, scale-in, scale-out, login, log-out, install software, replicate, synchronize, backup, delete, encrypt data, decrypt data, authorize, and authenticate
SaaS		Feature (email, customer relationship management, ERP, social networking, document management, and crowdsourcing), legal and regulatory issues, security, privacy, integrity, reliability, licensing terms and costs, availability, performance statistics, SLA, and data portability	Customize, accounting, billing, select, data porting, authentication, and authorization

respective (provider-specific) Web service API. Programming applications that can be hosted across distributed IaaS resources require a developer to orchestrate concurrent computation and communication across heterogeneous cloud services, in a manner that's robust to delays and failures. For example, in a multistep orchestration operation of allocating a CPU resource to a software resource, followed by assigning an elastic network IP and mounting an EBS resource — if one of the immediate operations fails or throws an unexpected error, a trivial implementation would fail stop, leaving the system in inconsistent state. Ensuring deadlock-free orchestration to deal with a high level of concurrency and network traffic arising from potentially large numbers of overlapping requests, recent efforts[2,12] have advocated programming resource orchestration based on declarative programming languages.

Configuring dynamic resources. The impetus behind cloud computing is the ever-increasing demand to manage growth and increase computing flexibility by dynamically scaling up or down resources based on demand.[4,5] However, existing cloud resource-provisioning techniques don't effectively support dynamic resource configuration. For instance, applications or workloads can't be dynamically and automatically partitioned or migrated arbitrarily from one cloud service to another if demand cycles increase. Moreover, dynamic configuration of resources is a complex issue because of lack of visibility and control across heterogeneous services at different layers. Advanced cloud resource orchestration techniques[13] have focused on developing an analytical application workload-prediction model for forecasting application resource requirements, and developing adaptive resource management techniques that can dynamically configure resources to meet requirements and constraints. While initial research results are promising, more than that, in many cases there's research from the fields of autonomic computing that we can leverage to a certain extent — however, designing effective dynamic cloud resource orchestration techniques that cope with large-scale heterogeneous cloud environments remains a deeply challenging problem.

Allocating cloud resources energy-efficiently. In recent years, energy-efficient allocation[12] of hardware resources to applications has emerged as a critical requirement, due to the worldwide focus on minimizing the carbon footprint. Efforts have focused on fabricating energy-efficient hardware, such as low-power, energy-efficient CPUs and solid-state drives to minimize energy consumption. The research community has also focused on software-based approaches to minimize energy consumption, such as resource allocation and task consolidation. That said, what remains a difficult and open research problem is the development of energy-efficient IaaS resource orchestration techniques that take into account application-specific service-level agreements (SLAs) while making resource allocation decisions for software resources.

Data security and privacy. The most significant difference between cloud security and traditional security controls stems from the fact that users spanning different corporations and trust levels often interact with the same set of computing resources. The security and availability of general cloud resources is dependent upon the security of basic APIs. From authentication and access control to encryption and activity monitoring, we must design these interfaces to protect against both accidental and malicious attempts to circumvent policy. For example, consider BLOB storage resources that have limited data security and privacy features, such as simple access control based on trusted credentials. BLOBs only support fine-grained security and privacy features to protect its end users from the following risks: data exposure (confidentiality), data tampering (integrity), and denial of access to data (availability). Recent research efforts have focused on developing additional third-party security infrastructures[14] to ensure the security, privacy, and integrity of data — not only while being transmitted over network links but also while at rest on BLOB resources.

Interoperability. To improve resilience, an intuitive solution is to deploy applications across multiple IaaS providers. Unfortunately, most of the existing providers aren't compatible with each other. They tend to have proprietary APIs, which aren't explicitly designed for cross-cloud interoperability. To tackle such heterogeneities, there's a requirement to enforce standardization across layers of the cloud resource stack. Recent developments — including Delta Cloud, jclouds, and Dasein Cloud (see http://dasein-cloud.sourceforge.net) — simplify this task by implementing a single API that abstracts APIs related to multiple clouds such as AWS EC2 and GoGrid. We can orchestrate fundamental cloud resources such as CPU, appliances, and storage via SOAP/RESTful APIs. However, orchestrating monitoring, load balancing, and auto-scaling RO operations to handle uncertainties in application and resource behaviors across clouds via a unified API still isn't viable, and hence remains an open research problem. The Topology and Orchestration Specification for Cloud Applications (TOSCA; see www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca#overview) is an interoperability specification that provides building blocks to support cross-stack orchestration of cloud resources.

PaaS

The PaaS layer features a rich pool of software appliances that facilitate the end-to-end lifecycle of developing, testing, deploying, and hosting applications. The following software resource categories are relevant at this layer.

Appliances. Appliances[15] are pre-configured, self-contained, virtualization-enabled, and pre-built software resource units (database, Web server, application server, Apache Hadoop, Apache Storm, load balancers, and so on) that we can integrate with other compatible appliances for designing complex applications. Primarily, it's the goal of the resource orchestrator to select, assemble, deploy, and manage a set of appliances (refer to https://solutionexchange.vmware.com/store/category_groups/virtual-appliances) delivering a particular application functionality.

For instance, several reusable appliances (see http://cloud.dzone.com/news/sql-vs-nosql-cloud-which) emerged in the area of Big Data processing (refer to http://cloud.dzone.com/articles/small-cross-section-big-data), including *SQL* and *NoSQL appliances*.[16] SQL appliances (see http://aws.amazon.com/rds) provide traditional relational database systems (such as MySQL, SQL Server, PostGres, and Oracle). NoSQL appliances (for example, Neo4j, CouchDB, MongoDB, Cassandra, and Amazon Dynamo) offer efficient support for unstructured data management and limited-to-no support for atomicity, consistency, isolation, and durability (ACID) transaction principles of SQL-like database systems.

In addition, to process Big Data produced by social media, mobile devices, the Internet of Things, business transactions, and content distribution, there has been a paradigm change from the traditional "one shot" machine-learning approach to elastic and virtualized cloud-based machine learning (ML) and data-processing appliances that are able to mine continuous, high-volume, open-ended data streams.

Distributed ML appliances[17] (such as Apache Mahout, MLBase, GraphLab, R, FlexGP, Vowpal Wabbit, MOA, and Pegasus) implement a wide range of ML algorithms (for example, clustering, decision trees, latent Dirichlet allocation, regression, and Bayesian) that are capable of mining datasets in parallel by leveraging a distributed set of machines.

Special data processing appliances — such as Apache S4 (see http://incubator.apache.org/s4), Twitter Storm, Amazon Kinesis, StreamBase, and Apache Hadoop — enable programming of applications that rapidly process massive amounts of data in parallel on large sets of machines. To speed up the ML algorithms, these data processing appliances simplify the process of distributing the training and learning tasks across a parallel set of resources.

We distinguish between *basic* and *composite appliances*. A basic appliance (such as AWS Relational Database Service [RDS], Apache Mahout, Apache Storm, and SQL Azure) delivers single-abstract functionality that might not be sufficient to design a fully functional application stack. Examples include Web server, database, and monitoring appliances. However, multiple basic appliances might need to be integrated to create a functional application. On the other hand, a composite appliance encapsulates a number of software resource units to support a standalone, fully functional application. For instance, Bitnami's Redmine composite appliance encapsulates multiple software resource units, including MySQL and Ruby on Rails.

We can classify appliances as *customizable* and *non-customizable*. Bitnami and rPath offer appliances that we can customize in terms of their mapping to hardware resources. For instance, we can map a Bitnami appliance (see http://wiki.bitnami.com/Applications) to one of the AWS CPU resource types, depending on anticipated QoS targets. Similarly, with customizable appliances, users have the flexibility to mount EBS volumes, if persistence of application data is a requirement. On the other hand, providers such as AWS EC2 offer noncustomizable appliances that we can integrate directly (without any further modification to its hardware resource configuration) into an application. For instance, AWS offers a load balancer and a monitoring appliance such as AWS CloudWatch (see http://aws.amazon.com/cloudwatch) for integration with other appliances (such as an app server appliance) to be hosted on AWS EC2.

Programming PaaS-Layer Orchestration Operations

Now, let's discuss research issues in programming orchestration operations at the PaaS layer.

Selecting optimal appliances. This operation requires an understanding of the technical details, features, and interoperation ability of competing appliances. In particular, the orchestrator needs to evaluate whether an appliance can deliver the requested functionality (such as stream data processing, database server, and source code management server). If a group of appliances is going to be selected, then they must meet integration constraints. Finally, an appliance's compatibility with the virtualization technology of the target cloud must be considered during the selection process. Other important selection criteria include the appliance's host operating system and its programming environment. To solve the appliance selection problem, research efforts have focused on applying multicriteria decision making[11] and semantic-based services' discovery techniques.[18]

Integrating appliances. An application can be composed of several appliances. The deployment procedures and order of their executions are unique to each application and cloud environment. Dependencies between various appliances in an application must be taken into account to ensure correct deployments. In today's interoperability solutions, templates are used as an interoperation mechanism to combine appliances. These templates capture appliances' unstructured information that's notoriously difficult to use as a means to support information with other compatible templates to render composite offerings at the PaaS level. Today, no description language exists to describe and combine metadata descriptions of PaaS appliances in a uniform manner. Instead, a plethora of such formalisms exists, with varying types of concerns and different capabilities. Nor is there a consistent way to model the dependencies between operational and deployment dimensions to create end-to-end combinations of modular cloud stack offerings to meet application or consumer demands.

Comprehensive monitoring.[19] Although the cloud provider offers proprietary monitoring appliances, such as CloudWatch by AWS and Fabric Controller by Azure, they have the following limitations: an inability to monitor application components deployed across multiple cloud providers; and an inability to support QoS (such as latency and availability) monitoring for individual software resources (for a Web, application, or database server, for example). To improve this situation, recent research efforts have focused on developing monitoring techniques that can monitor both hardware and software resources and support a comprehensive list of QoS parameters for each resource type.

Managing Big Data. While research groups have focused on large-scale data management in traditional enterprise settings, cloud computing and its available NoSQL and SQL appliances have their own research challenges with regards to programming orchestration operations (such as selection, scale-in, scale-out, synchronize, replicate, and backup). Supporting ad hoc querying on top of NoSQL appliances and providing hard data-consistency guarantees remains an open research problem. Further, it's not clear how NoSQL appliances will perform for different classes of applications (for enterprises or streaming Big Data, for example) and workload (decision support, I/O intensive, and so on). Developing techniques that can impart the intelligence[16] of characterizing the data density (density and distribution of data; or composition of queries) to a cloud-based load-balancing appliance (such as AWS Elastic Load Balancer) for improving the QoS (including query latency and database service throughput) remains a popular research topic.

Cloud data security. Similar to BLOB storage resources, data stored on NoSQL and SQL appliances aren't secured at a finer granularity level. The application data managed by these appliances are vulnerable to theft, because adversaries can gain access to private data and malicious database administrators might capture or leak data. Hence, research efforts[20] have focused on developing techniques that efficiently support the following: encryption and decryption of data without causing additional query and data-processing overhead (time and space); and execution of a variety of traditional SQL (equality checks, order comparisons, aggregates, and joins) or NoSQL queries over encrypted data.

SaaS

SaaS integrates multiple, interoperable PaaS- and IaaS-level cloud resources to deliver applications to end users — for example, social network services such as Facebook and Twitter. The orchestration operations for managing lifecycles of SaaS-level resources are handled behind the scenes by SaaS providers. A SaaS provider can own or rent the underlying PaaS- and IaaS-level resources. SaaS applications are accessed over the Internet and typically are charged on a subscription basis. Table 1 also shows the orchestration operation relevant to SaaS.

Programming SaaS-Layer Orchestration Operations

Next, we discuss research issues in programming orchestration operations at the SaaS layer.

Managing Big Data. Social network applications generate massive amounts of data. Timely acquisition and processing of data from social networks play an important role in many application scenarios, such as emergency situation awareness, customer sentiment analysis, and syndromic biosurveillance. However, the rate at which social networks produce data (Twitter generates 40,000 messages per second) leads to many complex orchestration challenges, whether it involves aggregating data in real time or processing efficient, QoS-aware data in real time for detecting an event (such as a disease outbreak, flood, or earthquake) within a specified time constraint.

Optimizing a QoS location-aware network. With the growing popularity of online multimedia services such as YouTube and Netflix, there's a need to orchestrate QoS-aware content delivery to end users. Users of online multimedia services tend to consume content based on their own interest, popularity, and content type. However, key challenges exist in detecting users' hotspots such that a suitable cloud data center can be chosen to initiate dynamic content migration.

Data portability with security and privacy. SaaS providers such as SalesForce.com need to provide service features that let users — such as small- and medium-sized enterprises (SMEs) and governments — migrate their existing application data to cloud environments with minimal effort. All existing in-house application data must be exported transparently, then formatted and parsed to suit the data format supported by cloud-hosted SaaS installation. Porting in-house applications (including CRM and email servers) remains a challenging issue for SMEs and government organizations. This is mainly due to a lack of technologies that can seamlessly migrate existing security (authorization, authentication, and accounting) mechanisms to public SaaS installations.

Distributed denial of service (DDoS). With the pay-as-you-go cloud model, the conventional DDoS attacks targeting SaaS can be transformed into a new attack that can create significant economic loss for the application owners. For example, a high rate and volume of malicious service requests lead to excessive cloud usage bills, due to unnecessary scaling of the CPU resources, as well as network data transfer to and from the SaaS. To thwart such attacks, Fahd Al-Haidari and his colleagues[21] propose an early approach, where their Shield system detects and mitigates distributed DoS attacks against CPU resources. However, further research is required to determine whether researchers can apply similar techniques for thwarting attacks against a variety of application models — in particular, those that must perform real-time analytics over a data stream flowing from external sources (such as mobile devices and sensors).

Issues Overlapping across IaaS, PaaS, and SaaS Layers

Having looked at each service individually, now we turn our attention to overlapping issues across the service layers.

Holistic cloud interoperability. Cloud interoperation requires, in addition to common APIs, clear separation of concerns (and control), flexible mappings of cloud resources to services, and higher-level forms of abstraction to automate the orchestration of resources. It also requires creating a cloud solution with one of its portions running on internal systems, and another portion delivered from the external cloud environment in which there's ongoing data exchanges and process coordination between the internal and external cloud environments. We believe that *holistic cloud interoperability* to address operational and deployment concerns mentioned at different cloud stack layers is needed. Holistic cloud interoperability fuses two interoperability issues.

The first, *horizontal cloud interoperability*, is cross-cloud stack interoperability between the same tiers in different cloud stacks — such as cross-SaaS, cross-PaaS, or cross-IaaS interoperation. For instance, a logistics application can combine order management, inventory, and payroll services from diverse providers at the SaaS level. These are services that can be composed unambiguously and safely with other similar services at the same horizontal levels of the cloud stack.

The first, *vertical cloud interoperability*, is between downstream-compatible cloud tiers in different stacks — such as SaaS to PaaS and PaaS to Iaas. For instance, an order-management service at the SaaS level might appropriately interface with a specialized Oracle sales forecasting data repository from an external provider at the PaaS level. These are services that can be composed unambiguously and safely with other conformant services from adjoining downstream levels in the cloud stack. For instance, the payroll solution Oracle's PeopleSoft Enterprise ePay platform runs on an Android smartphone operating platform so that we can use it on mobile devices.

Dependencies of RO operations across IaaS and PaaS layers. Table 1 shows that some RO operations are applicable to both IaaS and PaaS resources — for instance, start, stop, restart, scale-in, and scale-out operations. Though semantically these operations are compatible, each operation needs to cater to the type of resource (such as IaaS or PaaS) under consideration while programming the orchestration. For instance, the "start" operation for a MySQL database appliance (at the PaaS level) means first instantiating a virtual CPU resource (by invoking an IaaS-level API), followed by starting the database instance via a specific command (such as `start mysqld`). On the other hand, the "start" operation for an Apache Tomcat Web server appliance will be similar to a database appliance at the IaaS level but different at the PaaS level (for example, `catalina start`). Developing APIs to resolve such dependencies across the layers automatically will continue to be an active area of research.

Cloud computing is a vast, complex, and evolving technology landscape, embracing a multilayered resource stack that must be orchestrated in an intricate manner to ensure that the application delivers an acceptable QoS level to the end users. We characterized cloud resources in a multilayered stack based on their attributes, granularity, and supported orchestration operations. This work will help readers clearly understand the core cloud computing concepts, interrelationship between different resource types, and relevant research challenges.

A taxonomy of cloud resources is discussed elsewhere.[22,23] In Lamia Youseff and her colleagues' work, they list a unified ontology for describing the cloud resource.[24] They use composability as the methodology for developing the resource ontology. Although these works help in understanding the issues of cloud computing, they fail to capture the essence of programming resource orchestration frameworks and associated research challenges. Further, they don't consider all of the resource types that we captured here. Arguably, our work is the first attempt at capturing orchestration operations and related research issues involved with programming orchestration frameworks across all layers of a cloud resource stack.

Acknowledgments

This article is a concise version of our technical report available at http://arxiv.org/abs/1204.2204. Additional references appear in a "Web Extra" document available at doi:10.1109/MIC.2015.20.

References

1. L. Wang et al., eds., *Cloud Computing: Methodology, Systems, and Applications*, CRC Press, 2011.
2. C. Liu et al., "Cloud Resource Orchestration: A Data-Centric Approach," *Proc. 5th Biennial Conf. Innovative Data Systems Research*, 2011, pp. 241–248.
3. A. Wieder et al., "Conductor: Orchestrating the Clouds," *Proc. 4th Int'l Workshop on Large-Scale Distributed Systems and Middleware*, 2010, pp. 44–48.
4. J. Schad et al., "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, 2010, pp. 460–471.
5. A. Iosup et al., "On the Performance Variability of Production Cloud Services," *Proc. IEEE/ACM Int'l Symp. Cluster, Cloud, and Grid Computing*, 2011, pp. 104–113.
6. P. Barham et al., "Xen and the Art of Virtualization," *Proc. 19th ACM Symp. Operating Systems Principles*, 2003, pp. 164–177.
7. Microsoft, *Windows Azure Storage Redundancy Options and Read Access Geo Redundant Storage*, blog, 11 Dec. 2013; http://blogs.msdn.com/b/windowsazurestorage/archive/2013/12/04/introducing-read-access-geo-replicated-storage-ra-grs-for-windows-azure-storage.aspx.
8. B Calder et al., "Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency," *Proc. 23rd ACM Symp. Operating Systems Principles*, 2011, pp. 143–157; http://doi.acm.org/10.1145/2043556.2043571.
9. M. Hajjat et al., "Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud," *Proc. ACM Sigcomm*, 2010, pp. 243–254.
10. H. Wada et al., "Evolutionary Deployment Optimization for Service Oriented Clouds," *J. Software: Practice and Experience, Special Issue on Search-Base Software Eng.*, vol. 41, no. 5, 2011, pp. 469–493.
11. M. Menzel and R. Ranjan, "CloudGenius: Decision Support for Web Server Cloud Migration," *Proc. 21st Int'l Conf. World Wide Web*, 2012, pp. 979–988.
12. A. Beloglazov et al., "A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems," *Advances in Computers*, vol. 82, 2011, pp. 47–111.
13. L. Vaquero et al., "Dynamically Scaling Applications in the Cloud," *Sigcomm Computer Comm. Rev.*, vol. 41, no. 1, 2011, pp. 45-52.
14. J. Yao et al., "TrustStore: Making Amazon S3 Trustworthy with Services Composition," *Proc. 2010 10th IEEE/ACM Int'l Conf. Cluster, Cloud, and Grid Computing*, 2010, pp. 600–605.
15. M. Menzel et al., "A Configuration Crawler for Virtual Appliances in Compute Clouds," *Proc. 2013 IEEE Int'l Conf. Cloud Eng.*, 2013, pp. 201–209.
16. C. Curino et al., "Relational Cloud: A Database Service for the Cloud," *Proc. 5th Biennial Conf. Innovative Data Systems Research*, 2011; http://web.mit.edu/ralucap/www/cidr11.pdf.
17. Y. Low et al., "Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud," *Proc. VLDB Endowment*, vol. 5, no. 8, 2012, pp. 716–727.
18. J. Kang and K.-M. Sim, "Ontology and Search Engine for Cloud Computing System," *Proc. Int'l Conf. System Science and Eng.*, 2011, pp. 276–281; doi:10.1109/ICSSE.2011.5961913.
19. K. Alhamazani et al., "An Overview of the Commercial Cloud Monitoring Tools: Research Dimensions, Design Issues, and State-of-the-Art," *Springer J. Computing*, vol. 97, no. 4, 2015, pp. 357–377.
20. R. Popa et al., "CryptDB: Protecting Confidentiality with Encrypted Query Processing," *Proc. 23rd ACM Symp. Operating Systems Principles*, 2011, pp. 85–100.

21. F. Al-Haidari, M.H. Sqalli, and K. Salah, "Enhanced EDoS-Shield for Mitigating EDoS Attacks Originating from Spoofed IP Addresses," *Proc. 11th IEEE Int'l Conf. Trust, Security, and Privacy in Computing and Comm.*, 2012, pp. 1167–1174.
22. C.N. Hoefer and G. Karagiannis, "Cloud Computing Services: Taxonomy and Comparison," *J. Internet Services and Applications*, vol. 2, no. 2, 2011, pp. 81–94.
23. B.P. Rimal, C. Eunmi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," *Proc. 5th Int'l Joint Conf. INC, IMS, and IDC*, 2009, pp.44–51; doi:10.1109/NCM.2009.218.
24. L. Youseff, M. Butrico, and D. Da Silva, "Toward a Unified Ontology of Cloud Computing," *Proc. Grid Computing Environment Workshop*, 2008; doi:10.1109/GCE.2008.4738443.

Rajiv Ranjan is a Julius Fellow, senior research scientist, and project leader in the Information Engineering Laboratory at CSIRO. His research interests include cloud and service computing. Ranjan has a PhD in computer science and software engineering from the University of Melbourne. Contact him at raj.ranjan@csiro.au.

Boualem Benatallah is a professor at the University of New South Wales Sydney. His research interests include Web service protocol analysis and management, enterprise services integration, large-scale and autonomous data sharing, process modeling, and service-oriented architectures for pervasive computing. Benatallah has a PhD in computer science from Grenoble University, France. Contact him at boualem@cse.unsw.edu.au.

Schahram Dustdar is a full professor of computer science (informatics) and he heads the Distributed Systems Group at the Vienna University of Technology. His research interests include Internet technologies. Dustdar is a member of the Academy Europeana, an ACM Distinguished Scientist, and recipient of the 2012 IBM Faculty Award. Contact him at dustdar@dsg.tuwien.ac.at; dsg.tuwien.ac.at/.

Michael P. Papazoglou is the chair of Computer Science Department and the executive director of the *European Research Institute in Service Science* (ERISS) at the University of Tilburg, the Netherlands. He's also the scientific director of the EU Network of Excellence in Software Services and Systems (S-Cube). His research interests include federated and distributed information systems, enterprise application integration, e-Business integration, and service-oriented computing. Papazoglou has a PhD in microcomputer systems engineering from the University of Dundee, Scotland. Contact him at mikep@uvt.nl.