CHAPTER

1 INTRODUCTION. SOFTWARE ARCHITECTURE FOR CLOUD AND BIG DATA: AN OPEN QUEST FOR THE ARCHITECTURALLY SIGNIFICANT REQUIREMENTS

Rami Bahsoon[*], Nour Ali[†], Maritta Heisel[‡], Bruce Maxim[§], Ivan Mistrik[¶]

[*] *University of Birmingham, UK* [†] *University of Brighton, UK* [‡] *Universität Duisburg-Essen, Germany* [§] *University of Michigan-Dearborn, USA* [¶] *Independent Researcher, Germany*

1.1 A PERSPECTIVE INTO SOFTWARE ARCHITECTURE FOR CLOUD AND BIG DATA

Cloud has revolutionized the way we look at software architectures. The emergence of cloud and its "as-a-service" layers (e.g., software, platform, data, infrastructure-as-a-service, etc.) have significantly induced the architecture of software systems. Cloud marketplaces, multitenancies, federation, elastic and on-demand access have enabled new modalities to the way we incept, compose, architect, deploy, maintain, and evolve architectures of software systems. Properties related to dynamic access of resources, resource pooling, rapid elasticity and utility service provision, economies of scale, dynamicity and multitenancy are arguably the emergent "architecturally significant requirements" [1] related to the cloud. These properties have influenced not only the behavior of the software system benefiting from the cloud, but also its structure, style, and topology. It has also moved architecting practices towards architecting for uncertainty, where architecture design decisions are mere complex to anticipate the extent to which they can operate in dynamic environments, and cope with operational uncertainties and continuous changes [2]. More interestingly, the cloud business model has also transformed architecting into economics-driven architecting, where utility monitoring, risk avoidance, utilization, technical debt monitoring, and optimizing for Service Level Agreements (SLA) are among the business objectives. In this context, architecting in/for the cloud has become an exercise that requires continuous alignments between enterprise and technical objectives.

The unbounded scalability of the cloud has "accidentally" led to "data farming", due to the volume and variety of data accumulated and/or assimilated across various service cloud layers and constituent architectural components. The accidental presence of the phenomena had steered a new hype for "big data" and a need for architecting for big data in the presence of the cloud. The hype has consequently provided new opportunities and modalities for data-driven services. In addition, data has introduced new constraints and requirements of architecturally significant nature.

Software Architecture for Big Data and the Cloud. DOI: 10.1016/B978-0-12-805467-3.00001-6

Architecting for the cloud and data is difficult to decouple. The coupling has led to new architecture styles and design paradigms that are designed with the evolution of data and its management as first class entities. Several architecture styles and architecture-centric development processes that leverage the benefits of the cloud and big data have emerged. The fundamentals of these styles cannot be understood in isolation in what we term as "cloud architecturally significant requirements."

We quest for cloud architecturally significant requirements and explain their implications on architecting for/in the cloud in the presence of big data. We posit that properties that relate to multitenancy, dynamism and elasticity, service Level Agreements (SLAs) constraints, value-seeking architecting through autonomic decisions, cloud marketplaces, big data management for volume, velocity, veracity, and scale are essentially architecturally significant requirements that are implied by the cloud as a distributed on-demand operating environment. Awareness of these requirements can help architects and practitioners to formulate architecture design decisions and choices, which are cloud-explicit. We hope to provide insights that can help in the systematic architecting for cloud-based systems and lessen generality and ad hoc practices.

1.2 CLOUD ARCHITECTURALLY SIGNIFICANT REQUIREMENTS AND THEIR DESIGN IMPLICATIONS

The software architecture community had come to the conclusion that not all nonfunctional requirements or quality attributes (QA) can have measurable or significant effects on software architectures, their design decisions and choice. Consequently, architecturally significant requirements were advocated as the determinants for this category of requirements that relate to architectures of software systems [1,10]. The observation is valid without any doubt: not all requirements can have direct implications on the architecture, constraint its process, and/or explicitly affect the choice of architecture design decisions, tactics, style, etc. Though it may be difficult to objectively determine the architectural significance nature of these requirements, researchers have suggested some indicators [1], which can relate to the extent to which requirements are important and unique to the operation of the system; its responsibility in affecting functionalities and nonfunctionalities that are visible within the architecture; the significant character of the requirement to the stakeholders involved; observable effects on Service Level Agreements performance; business objectives and constraints and/or technical debts implication of the choice. Nevertheless, difficulties arise from the fact that these indicators cannot be clearly assessed and can be observed across various trade-off points.

In the context of the cloud, many indicators are fundamentally linked to the characteristics of the paradigm itself and its distribution environment. Cloud-architecture significant requirements stem from cloud properties that relate to the fundamentals of elastic computing, scalability, multitenancy of the cloud, etc. This implies that the cloud environment and the environment economies of scale are key determinants to these requirements. They can consequently affect the way we resolve trade-offs and decide on cloud-based architecture design decisions, styles, etc.

Cloud architecturally significant requirements and measurable effects can be observed on Services Level Agreements (SLA) compliance and violations. The choice of architecture design decisions is essentially SLA-centric; where SLA promises mandate design choices. Conversely, architecture design decisions and choices tend to inform, refine, and elaborate SLAs. The process is continuous; it is intertwined, interleaved, and informed by the requirements of various multitenants. The challenge is that the

process has to reconcile various multitenant requirements and their constraints. This begs the question: is it possible to classify multitenants and relate their *commonality* requirements to architecture design decisions and choices of cloud-based systems? How do *variability* requirements within multitenancy inform of these choices? These questions are difficult for researchers to answer without understanding the anticipated behavior of the tenant, the dynamism of their behavior and the environment.

1.2.1 DYNAMISM AND ELASTICITY AS CLOUD ARCHITECTURALLY SIGNIFICANT REQUIREMENTS

Cloud is fundamentally dynamic. Dynamism induces the architecture of cloud-based systems in several ways. It can be implied from the fundamental properties of the cloud that have to do with *elastic computing* and multitenancy. Elasticity is defined as "the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible" [9]. Dynamism can be inherited from cloud service providers, where performance and service provision instability are the norms. As the cloud is a shared environment, it is imperative that architecting process for the cloud transits into resource and capacity planning in the realm of dynamism and benefiting from the economies of scale. The objective is to optimize for Quality of Services (QoS) provision and to improve cloud-based services utilization.

Dynamism is essentially a cloud architecturally significant requirement. It calls for new tactics and architecture strategies for predicting behavior in the cloud environment and proactively resolving complex trade-offs at runtime. It can also constrain the design of load balancing tactics and scheduling mechanisms that are essential for scalable, available, and dependable architectures. Dynamism makes uncertainty and unpredictability a pronounced concern in the architecting process; it begs for architecture-centric solutions that tame it. Solutions need to exploit various types of machine learning, planning, and intelligence to predict and consequently plan for realizing behavioral requirements. It also calls for architectural control and actuating mechanisms for managing trade-offs and mitigating their implications on various quality trade-offs. Analytic solutions that are grounded on market-based control, strategy dynamics, economics and finance are among the widely used techniques for managing and coping with dynamism.

One pending issue and open challenge that faces architects is to scale these decisions to cope with various levels of dynamism, ranging from relatively stable to dynamic and/or excessively hyper. Autonomic solutions are advocated in the heart of the architecture design process, where architecture design decisions are judged by their ability to deal with operational and evolution uncertainties. Among the autonomic decisions that architects consider is matching cloud users' requests with the resources capable of meeting users' requirements within the specified SLA constraints. Architectures are designed to dynamically allocate resources, effectively monitor resources' states and rapidly adapt to changes in workload. Architectures should seamlessly and efficiently resolve conflicting user goals and requirements, where scale, heterogeneity, dynamism, and complex trade-offs for nonfunctional requirements are the norm.

1.2.2 MULTITENANCY AS CLOUD ARCHITECTURALLY SIGNIFICANT REQUIREMENT

By multitenants we mean a software architecture which is designed to support instantiation, where multitenants can have a dedicated share of instances, including configuration, user management, individual tenant functionality, data and nonfunctional properties. Designing for multitenancy is a challenging exercise, as architects need to design instantiation taking into account multiple users in a given tenancy and across multiple ones. Architects need to consider the diverse needs, wishes, context, requirements, and quality-of-services constraints within and across tenants. They have to architect, realizing the commonality, variability, veracity, diversity, and scale of both functional and nonfunctional requirements supporting individuals, tenants, and the operating environment. Architects need to predict what may seem unpredictable from variation in workload patterns, likely sudden spikes or a one-off, etc. within and across tenants. Architects need to formulate design decisions and tactics that are flexible enough to cater for continuous changes in users' requirements in a single tenancy and across multitenancy. They also need to consider how changes associated with adapting to new environments relate to the cloud, how mobility, Fog, Edge, Internet of Things (IoT) and federation can affect design decisions and their evolution over time. Architects also need to predict changes in functional and nonfunctional requirements during the lifetime of the cloud-based system while it evolves. It is imperative that dynamic and "hyper" scale is the norm, where a new multitenant can join the environment and others may leave. Architectural design frameworks need to provide mechanisms to deal with similar uncertainties and to provide fairness and prevent "greed" in situations where providers stretch the economies of scales by accommodating more than what the environment can normally handle.

1.2.3 SERVICE LEVEL AGREEMENTS (SLAS) CONSTRAINTS AS CLOUD ARCHITECTURALLY SIGNIFICANT REQUIREMENT

Service Level Agreements (SLAs) constraints and specifications are essentially cloud architecturally significant requirements. Cloud-based system architects are challenged by the need for aligning Service Level Agreements (SLAs) with the architecture and its services provision, whether they are functional or nonfunctional in nature. Any violation of these agreements relating to performance, availability, reliability, security, compliance, etc., could lead to degradation of the provider's reputation or penalties in the form of monetary or service credit repayment [3]. Henceforth, architecting becomes a "strategy" game and challenge, where architects need to take cost-effective and efficient decisions, anticipate the risks and value of these decisions against likely or unpredictable changes in requirements and operating environments. This transits architecting process into service-level agreement-centric and value-driven design for operation and evolution, where alignment of the business objectives and technical drivers are at the heart of the process. Architecture design decisions hope to guarantee that violations for SLAs can be reduced by ensuring that critical requirements are provisioned through dependable resources. Therefore, the problem at hand is what architectural strategies, tactics and styles could help cloud providers meet the dynamic requirements of cloud-based systems, while adhering to service level terms. Aspects of the SLA life cycle, such as service discovery, SLA negotiation and reconciliation, monitoring and reporting can liken themselves with architecture tactics and components that meet these objectives.

1.2.4 CLOUD MARKETPLACES AS ARCHITECTURALLY SIGNIFICANT REQUIREMENT

Several cloud marketplaces have emerged, offering cloud application developers and architects ready means for online access and use of software applications and instances. Cloud-based architectures, an emerging class of architectures, can be composed of web-services, which can be leased or bought off the cloud marketplaces. It is believed that the value of the application and its underlying architecture is a function of the value of the individual web-services composing the application. These architectures can "trade" instances of abstract web services off the cloud, which can serve as a basis for dynamic synthesis and composition of the architecture. That is, for a given abstract service A, there exist multiple concrete services $Ai \ldots An$ in the market offering comparable functionalities, but differing in their price and Quality of Service (QoS) provision. Cloud is also emerging as a marketplace for service provision, where the selection of concrete web service instances from cloud-based market as a dynamic "search problem", which needs to be optimized to reduce probable risks [7,5], QoS and price [4], and technical debt [6].

Cloud-based architectures encompass a set of architecture strategies and tactics, which can take advantage of "goods" (e.g., service instances) traded in this market. Market mechanisms have lent themselves neatly to the notion of valuation of cost and benefits via utilities. "Buyers" are cloud-based application architectures, and sellers are vendors and providers of the services, which realize quality attributes, functional requirements, price and environmental constraints [7]. The formulation has enabled new modalities to the way we develop, maintain and deploy cloud-based architectures, where decentralization, scalability, modularity, substitutability are among the selling features. This has effectively steered new architecting paradigms, such as microservices [12], DevOps, etc.

The emergence of cloud marketplaces and instantiation has enabled dynamic composition of such architectures through substitution, where maximizing value, and minimizing debts and risks can be among objectives. Technical debt in cloud-based architectures can be attributed to suboptimal, poor and/or quick substitution and composition decisions [6]. This can be related to situations, where (i) the selected service features do not fully match the requirements of the application; fixes may be required to reengineer the solution to better fit the requirements, (ii) composing the service may have negative impact on the overall quality of the architecture; the composition may negatively affect the overall utility of the architecture and qualities like scalability, reliability, security, real time performance and the overall stability, (iii) in the absence of publically available benchmarks and historical performance data to verify the service provision and its compliance, uninformed quick selection and substitution decisions may carry likely risks, which can lead to Service Level Agreement (SLA) violations, (iv) the potentials of the architecture following composition are not fully utilized and the operational cost tends to exceed that of the generated benefits, (v) poor and quick decisions may add a value in the short-term, but can introduce long-term debt in situations where scaling up is unavoidable. Technical debt can also occur accidentally: cloud service providers may rush the release of their services introducing technical debt, which can be attributed to pressures to meet deadline and catch up the market. This can be, for example, through falsely accelerating the velocity by reducing testing and verification of the features delivered with the services. In situations where the service(s) are selected based on the sole trust of the service provider, the choice may accidentally introduce a debt, which is often left to the application developer to visualize and manage. As Agile Service Networks (ASN) become a reality, tools and techniques that allow organizations to autonomously evaluate the long-term value of the structure, the technical debt introduced by selection and substitution decisions will be needed. The impact of dynamic composition decisions of the architecture on value creation must be calculable, and actionable

by autonomic software, which can self-manage the architecture for value creation, debt reduction, and long-term sustainability.

1.2.5 SEEKING VALUE AS CLOUD ARCHITECTURALLY SIGNIFICANT REQUIREMENT

Cloud-based software architectures can dynamically evolve to seek value [4]. The evolution of such architectures shall add value to the enterprise, end users, cloud provider and environment in which they operate. We viewed evolution of cloud-based application through continuous adaptations as a value-seeking and value-maximizing activity. The added value can be attributed to the Operational flexibility primitives of the cloud. Value can be linked to Service Level Agreements (SLA) compliance, Quality of Service (QoS) improvements, revenue streams for cloud providers and the multitenant cloud consumers, better utilization of the cloud economies of scale, environmental sustainability and CO_2 footprint reduction. At runtime, cloud-based architectures can specify the type of web-services to meet their functional and nonfunctional requirements. They can specify their constraints for price, computing power, energy consumption, and demands to virtualized and physical resources. Indeed, the need for web service substitution could be attributed to a business objective or a technical one, such as changes in load, the need to scale up, improving the current system' QoS, reducing the operational cost, or by upgrading to a new web service, which has been released in the marketplace. If the architecture wishes to create an added value, it can dynamically substitute its services. Scenarios, where federated clouds with different specialized and standardized services collaborate can be envisaged. These collaborations can be leveraged by an enterprise to construct cloud-based architectures, which are dynamic and self-adaptive by changing the web services they utilize. The recognition of Agile Service Networks (ASN) that spring up in modern business practices is a testament to this.

A common way of representing value for architecture tactics (e.g., provision of resource to improve QoS) is to formulate this using utility functions. This could take different forms (e.g., additive utility), where each formulation impacts the market's ability to reach desirable emergent state(s). There are many existing market mechanisms in the microeconomics literature. Each theoretically studied mechanism can be shown to reach some known solution concept (e.g., Nash equilibria). A self-adaptive mechanism, which uses these utility models, can assist analysts, architects and enterprises in providing built-in support for an autonomic architecture management supporting continuous evolution for added value. The inputs to these modes can be continuous data and feeds available through published benchmarks (e.g., cloudharmony.com; spec.org), which can continuously steer the selection and composition of cloud-based architectures. IT analysts, planners, and software architects and engineers can use simulation toolkits, which facilitate the scientific and engineering queries, what-if, sensitivity and cost-benefit analyses for sustainability and value. The use of microeconomic trading mechanisms such as the Continuous Double Auction (CDA), for example, has inspired the design of decentralized, robust, and scalable self-adaptive architecture for the cloud [4]. The work demonstrates how value-based reasoning can dynamically and adaptively steer the evolution of cloud-based architectures for value added by optimizing for utilities expressing QoS concerns. The space of strategies that a buyer (i.e., architecture) may choose during its interaction is yet another dimension. A buyer may be risk averse when trading web services instances, which will sustain or improve the utility of the architecture. These instances shall provide the assurance for SLA compliance and reduce the risk in likely violations [5]. In a market context, value can be estimated through tracking an underlying asset (also referred to as

twin asset). The premise is that the twin asset can serve the evaluation through analogy in the absence of historical performance data of the asset under valuation, expert judgment, etc. [8].

1.3 BIG DATA MANAGEMENT AS CLOUD ARCHITECTURALLY SIGNIFICANT REQUIREMENT

The popularity, availability, unbounded scalability and service provision across various cloud layers, software, storage, infrastructure, platform, etc., had "accidentally" led to farming of data. The phenomena have hyped and strengthen what is known as big data, presence and opportunities. However, the hype has introduced new challenges to the architecting process, where data-centric architecting has become unavoidable and among the main drivers. The hype can be attributed to the natural incubation of data, represented by its volume, veracity and variety accumulated and/or assimilated across various cloud service layers and constituent architectural components.

Though big data is concerned with extracting valuable information from data spanning areas of society, business, science, medicine, engineering, entertainment, logistics among the many others, its presence within/from the cloud has constrained the architecting processes and product, and enabled new data-driven services. Such datasets are often from various sources (Variety), yet unstructured, such as social media, sensors, scientific applications, surveillance, video and image archives, Internet texts and documents, Internet search indexing, medical records, business transactions and web logs; and are of large size (Volume) with fast data in/out (Velocity). More importantly, big data has to be of high value (Value) and establish trust in it for business decision making (Veracity). Various technologies are being discussed to support the handling of big data such as massively parallel processing databases, scalable storage systems, cloud computing platforms and MapReduce.

Architecting for the cloud and big data is difficult to decouple. The coupling has led to new architecture styles and design paradigms that are designed with the evolution of data and its management as first class entities. The elasticity, multitenancy and the unbounded scalability of the cloud have steered new opportunities for data-driven architecting. The hype has provided new modalities for data-driven services. In addition, data has introduced new constraints and requirements of architecturally significant nature. The architecting process is challenged by having data assimilation, aggregation, classification and intelligent data analysis at the heart of the architecting process where data security, privacy, scalability, performance, availability and integrity and associated trade-offs are among the architecture significant requirements that influence architecture design decisions.

Database schema unification, portability, adaptation, search and mining efficiency, and their associated nonfunctional requirement trade-offs become pronounced cloud architecturally significant requirements that constraint and inform the architecture design process. The coupling of cloud and data has led to new opportunities for utilizing data through data as services and available via marketplaces, etc. Conversely, publically available service provision data and benchmarks have informed the composition decisions of cloud-based architectures, guided by historical information in relation to likely behavior and its dependability provision, cost, value, and risks. In dynamic and continuously evolving context, the challenge is to understand how value can be captured modeled, analyzed, and how the various trade-offs are involved.

Big data architectures, in the realm of the cloud, can essentially relate to architectures of data-driven services enabled by the cloud. The commonly used data-driven services relate to analytics

services; these services are enabled and steered by the cloud as high-performance, scalable and on-demand "high-refresh" computing paradigms. Discussions on big data architectures can also relate to fundamentals of inducing the architecture with specific big data technologies and their architecturally significant requirements implications such as parallelism and concurrency.

1.3.1 BIG DATA ANALYTICS ENABLED BY THE CLOUD AND ITS ARCHITECTURALLY SIGNIFICANT REQUIREMENTS

Buyya et al. [11] investigated the architecture elements of big data analytics enabled by the cloud. Among the architecture significant requirements that big data analytics architecture need to consider is context related. Contextual information compiles information, such as logs of infrastructure operations, product releases, business metrics, and data compiled from social networks analysis among the others. In other words, the contextual architecture elements tend to refer to a broad spectrum of information processed from various media, whether they are structured or unstructured, technical, business, organization, sentiment, etc. The variety of contextual related data calls for the inception of other architectural elements, which are essential for big data analytics. The essential architectural elements have to do with the selection of appropriate sources of data for prediction; data filtering, requiring a search for most relevant data to be included in the selection, prediction and modeling; data extraction and ensuring the timeliness of the extracted data in relation to decisions, architecture reaction which are based on this data.

Other important architecturally significant requirements that relate to analytics have to do with the integrity, persistence, reliability, scalability, and general dependability of the raw data and the processed information. These requirements crosscut the entire lifecycle of dealing with the context ranging from selection, filtering, modeling, prediction, refreshing, and timeliness, etc.

The nature of the data can impose requirements of architecture significance. For example, data and information that deal with identities can raise ethical concerns and can call for solutions that maintain privacy and security of the identities, where data is arguably the individual's asset, and compliance for confidentiality can be a pronounced constraint. Architecture solutions need to consider locality vs. distribution, decentralization vs. centralization of the computation and transmission. Solutions need also to consider the motion of data and devise architecture solutions that cater for constraints for full, limited or no motion. Data exploitation and sharing under the realm of cloud data-as-service needs to consider compliance related requirements among the other requirements. Data of public use and interest can also introduce its own architecture challenge for solutions that cater for trust monitoring, scalability, availability, dependability and decay of the data and processed information to the beneficial services.

1.3.2 ARCHITECTURALLY SIGNIFICANT REQUIREMENTS IN REALM OF COMPETING BIG DATA TECHNOLOGIES

Several big data technologies exist. It is an architecture challenge to select the "right" technology that induces the architecting process and solution. Examples of widely used technologies include but are not limited to Hadoop/YARM, Spark, Pig and Hive, Mahout, and Cassandra, CouchDB, BlinkDB, HDFS. These frameworks are engineered to serve specific functionalities and features for data analytic, data parallel processing, etc.

For example, Hadoop/YARM is a general-purpose, distributed, application management framework that supersedes MapReduce framework for processing data in Hadoop clusters. The Map phase of MapReduce is concerned with the parallel processing of data. In this phase, the data is split into discrete fragments for processing. In the Reduce phase, the output of the map phase is aggregated to generate the outcome. The framework promises efficient and scalable processing of data across various nodes. MapReduce can be affected at the nodes that process that data. The premise is that it significantly reduces the network I/O, keeping I/O on the local disk or rack. The architecture principle that underlies this model is that the computation is localized; processing takes place at the data side as opposed to moving the data itself to the computation node. The principle acknowledges that limiting the motion of data can have implication on scalability, performance, security of both data and computation.

MapReduce has limited capabilities when supporting real-time and/or near real-time processing such as stream processing. This is because MapReduce has been essentially designed for batch-processing. Storm, on the other hand, addresses this limitation and is suitable for processing unbounded streams of data, enabling real-time processing of large volumes of high-velocity data. Storm claims to be capable of processing over a million records per second on a cluster node. Spark provides an environment and interface for programming entire clusters; it can focus on implicit data parallelism and fault-tolerance. Domain-specific applications that benefit from STORM include real-time customer service management, cybersecurity and threat analytics, data monetization, operational dashboards, etc.

Other related environments and tooling such as Pig and Hive (high level query languages/tools that ease the complexity of writing MapReduce queries), Mahout (providing high-level analytics tasks that create scalable machine learning algorithms to deal with Recommendation, Classification, and Clustering), and Cassandra, CouchDB, BlinkDB, HDFS (file systems and NOSQL databases) have also provided new opportunities for inducing and realizing architectures that meet modern needs.

Architecture styles can be induced by these technologies and can benefit from the tooling support. Each technology and tooling support provides pragmatic architecture realizations and ready solutions influenced by their strengths and limitations. It can also enable effective and efficient analytics, with minimum resource consumption. The choice can address recurring problems that relate to batch processing, parallel processing, data motion and related trade-offs.

REFERENCES

[1] C. Lianping, A. Babar, B. Nuseibeh, Characterizing architecturally significant requirements, IEEE Softw. 30 (2) (2013) 38–45, http://dx.doi.org/10.1109/MS.2012.174.

[2] T. Chen, R. Bahsoon, Self-adaptive and online QoS modeling for cloud-based software services, IEEE Trans. Softw. Eng. (2016), http://dx.doi.org/10.1109/TSE.2016.2608826.

[3] F. Faniyi, R. Bahsoon, A systematic review of service level management in the cloud, ACM Comput. Surv. 48 (3) (2016) 43:1–43:27.

[4] V. Nallur, R. Bahsoon, A decentralized self-adaptation mechanism for service-based applications in the cloud, IEEE Trans. Softw. Eng. (2013).

[5] F. Alrebeish, R. Bahsoon, Implementing design diversity using portfolio thinking to dynamically and adaptively manage the allocation of web services in the cloud, IEEE Trans. Cloud Comput. 3 (3) (2015) 318–331, 2015.

[6] E. Alzaghoul, R. Bahsoon, CloudMTD: using real options to manage technical debt in cloud-based service selection, in: The Fourth International Workshop on Managing Technical Debt. The 35th ACM/IEEE International Conference on Software Engineering, San Francisco, CA, ACM Press, 2013.

[7] F. Faniyi, R. Bahsoon, Self-managing SLA compliance in cloud architectures: a market-based approach, in: Proceedings of the International ACM Sigsoft Symposium on Architecting Critical Systems, Bertinoro, Italy, ACM Press, 2012.
[8] R. Bahsoon, W. Emmerich, An economics-driven approach for valuing scalability in distributed architectures, in: Proc. of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), Vancouver, Canada, IEEE Computer Society Press, 2008.
[9] N. Herbst, S. Kounev, R. Reussner, Elasticity in cloud computing: what it is, and what it is not, in: Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013), San Jose, CA, June 24–28, 2012.
[10] P.C. Clements, L. Bass, Relating business goals to architecturally significant requirements for software systems (CMU/SEI-2010-TN-018). Retrieved from Pittsburgh, PA, USA, 2010.
[11] R. Buyya, K. Ramamohanarao, C. Leckie, R. Calheiros, A. Dastjerdi, S. Versteeg, Big data analytics-enhanced cloud computing: challenges, architectural elements, and future directions, in: IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), 2015.
[12] S. Hassan, R. Bahsoon, Microservices and their design trade-offs: a self-adaptive roadmap, in: The 13th IEEE International Conference on Services Computing (SCC), June 27–July 2, 2016, San Francisco, USA, 2016.