Resource-Aware Applications for the Cloud

Reiner Hähnle [1] and Einar Broch Johnsen [2]

[1] Technical University of Darmstadt, Germany haehnle@cs.tu-darmstadt.de

[2] University of Oslo, Norway einarj@ifi.uio.no

Making full usage of the potential of virtualized computation requires nothing less than to rethink the way in which we design and develop software.

Capitalizing on the Cloud

The planet's data storage and processing is about to move into the clouds. This has the potential to revolutionize how we will interact with computers in the future. A cloud consists of virtual computers that can only be accessed remotely. It is not a physical computer, you do not necessarily know where it is, but you can use it to store and process your data and you can access it at any time from your regular computer. If you still have an old-fashioned computer, that is. You might as well access your data or applications through your mobile device, for example while sitting on the bus.

Cloud-based data processing, or cloud computing, is more than just a convenient solution for individuals on the move. Although challenges remain concerning the privacy of data stored in the cloud, the cloud is already emerging as an economically interesting model for a start-up, an SME, or simply for a student who develops an app as a side project, due to an undeniable added value and compelling business drivers [1]. One such driver is *elasticity*: businesses pay for computing resources when they are needed, instead of provisioning in advance with huge upfront investments. New resources such as processing power or memory can be added to a virtual computer on the fly, or an additional virtual computer can be provided to the client application. Going beyond shared storage, the main potential in cloud computing lies in its scalable virtualized framework for data processing, which becomes a shared computing facility for your multiple devices. If a service uses cloud-based processing, its capacity can be automatically adjusted when new users arrive. Another driver is *agility*: new services can be deployed quickly and flexibly on the market at limited cost, without initial investments in hardware.

The EU believes[1] that cloud-based data processing will create 2.5 million new jobs and an annual value of 160 billion euros in Europe by 2020. Another study[2] predicts

[1] Digital Agenda for Europe, http://ec.europa.eu/digital-agenda/en/european-cloud-computing-strategy

14 million new jobs worldwide until 2015. However, reliability and control of resources are barriers to the industrial adoption of cloud computing today. To overcome these barriers and to regain control of the virtualized resources on the cloud, client services need to become resource-aware.

Challenges

Cloud computing is not merely a new technology for convenient and flexible data storage and implementation of services. Making full usage of the potential of virtualized computation requires nothing less than to rethink the way in which we design and develop software.

Empowering the Designer. The elasticity of software executed in the cloud means that designers have far reaching control over the resource parameters of the execution environment: the number and kind of processors, the amount of memory and storage capacity, and the bandwidth. These parameters can even be changed dynamically, at runtime. This means that the client of a cloud service not only can deploy and run software, but is also in full control of the trade-offs between the incurred cost and the delivered quality-of-service.

To realize these new possibilities, software in the cloud must be *designed for scalability*. Nowadays, software is often designed based on specific assumptions about deployment, including the size of data structures, the amount of RAM, and the number of processors. Rescaling requires extensive design changes, if scalability has not been taken into account from the start.

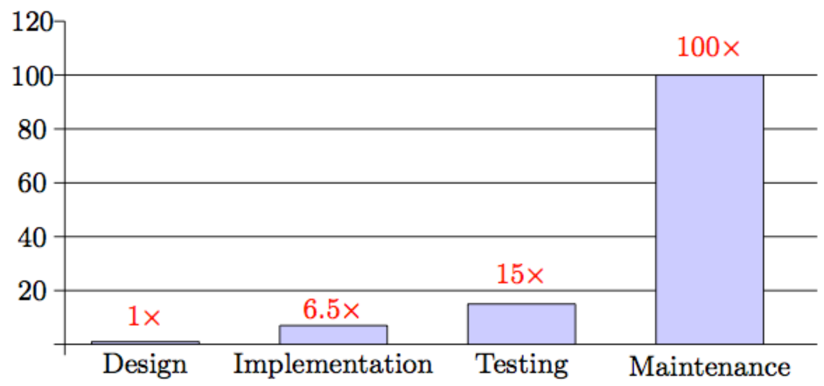

Fig. 1: Relative costs to fix software defects for static infrastructure (source: IBM Systems Sciences Institute). The columns indicate the phase of the software development at which the defect is found and fixed.

[2] IDC White Paper Cloud Computing's Role in Job Creation, http://www.microsoft.com/en-us/news/features/2012/mar12/03-05cloudcomputingjobs.aspx

Deployment Aspects at Design Time. The impact of cloud computing on software design, however, goes beyond scalability issues: traditionally, deployment is considered a late activity during software development. In the realm of cloud computing, this can be fatal. Consider the well-known cost increase for fixing defects during successive development phases [2]. IBM Systems Sciences Institute estimates that a defect which costs one unit to fix in design, costs 15 units to fix in testing (system/acceptance) and 100 units or more to fix in production (see Fig. 1). This cost estimation does not even consider the *impact cost* due to, for example, delayed time to market, lost revenue, lost customers, and bad public relations.

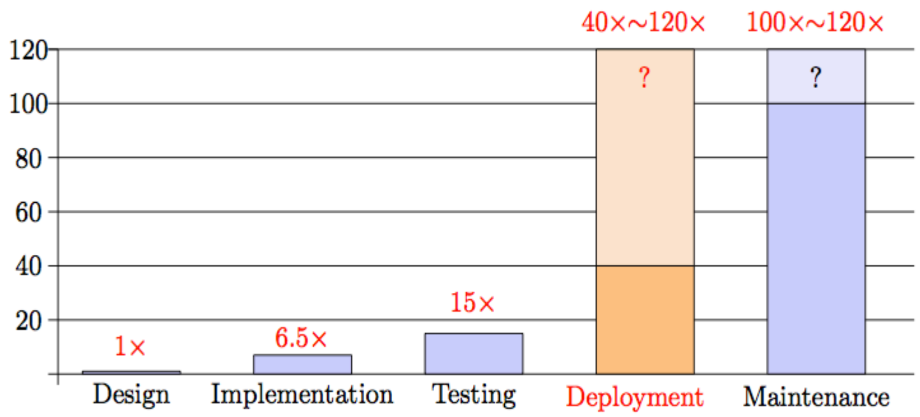

Fig. 2: Estimate of relative costs to fix software defects for virtualized systems with elasticity, from [3].

Now, these ratios are for *static* deployment. Considering the high additional complexity of resource management in virtualized environments, it is reasonable to expect even more significant differences; Fig. 2 conservatively suggests ratios for virtualized software in an *elastic* environment. This consideration makes it clear that it is essential to detect and fix deployment errors, for example, failure to meet a service level agreement (SLA), already *in the design phase*.

To make full usage of the opportunities of cloud computing, software development for the cloud demands a design methodology that (a) takes into account deployment modeling at *early* design stages and (b) permits the detection of *deployment errors* early and efficiently, helped by software tools, such as simulators, test generators, and static analyzers.

Controlling Deployment In The Design Phase

Our analysis exhibits a *software engineering challenge*: how can the validation of deployment decisions be pushed up to the modeling phase of the software development chain without convoluting the design with deployment details?

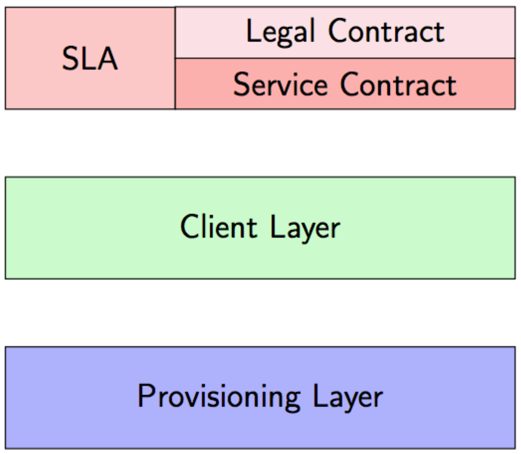

Fig. 3: Conceptual parts of a deployed cloud service.

When a service is developed today, the developers first design its functionality, then they determine which resources are needed for the service, and ultimately the provisioning of these resources is controlled through an SLA, see Fig. 3. The functionality is represented in the *client layer*. The *provisioning layer* makes resources available to the client layer and determines available memory, processing power, and bandwidth. The SLA is a legal document that clarifies what resources the provisioning layer should make available to the client service, what it cost, and states penalties for breach of agreement. A typical SLA covers two aspects: (i) a *legal contract* stating the mutual obligations and the consequences in case of a breach; (ii) the technical parameters and cost figures of the offered services, which we call the *service contract*.

How can the validation of deployment decisions be pushed up to the modeling phase of the software development chain without convoluting the design with deployment details?

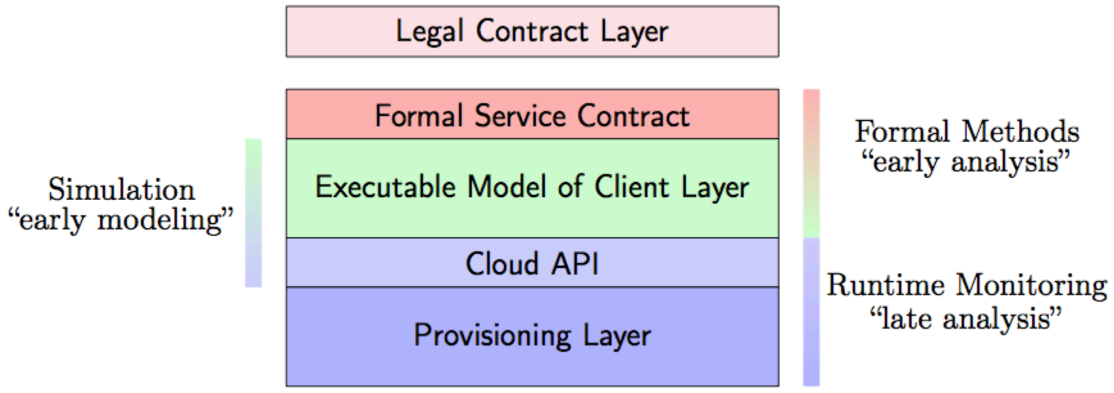

Fig. 4: Making services resource-aware.

So far, the different parts of a deployed cloud service live in separate worlds, but we need to connect them. In a first step the provisioning layer is made available to the client, so that the client can observe and modify resource parameters. We call this the *Cloud API*. This is *not* the same as the APIs that cloud environments provide to their clients now: our goal is to move deployment aspects into the *design phase*. We advocate that client behavior is represented by an *abstract behavioral model* of the client application, for example in an executable modeling language such as ABS [4]. Such a model can realistically be created during the design phase. The Cloud API is then an abstract interface to the provisioning layer, see Fig. 4. Such "early modeling" of client behavior makes it possible to simulate different client-side provisioning schemes and observe their impact on cost and performance.

To connect SLAs with the client layer, the key observation is that the service contract aspects of an SLA can be given a formal semantics. This enables formal analysis of client behavior with respect to the SLA at *design time*. Possible analyses include resource consumption, performance, test case generation, and even functional verification [3]. For modeling languages such as ABS this is highly automated [5]. "Early analysis" makes assumptions about the Cloud API explicit and enables the generation of monitors in the provisioning layer. Runtime monitors then provide "late analysis".

Opportunities

Making deployment decisions at design-time shifts control from the provisioning layer to the client layer. The client service becomes resource-aware. This provides a number of attractive opportunities.

Fine-grained provisioning. Business models for resource provisioning on the cloud are becoming similarly fine-grained as those we know from other industry sectors such as telephony or electricity. It is becoming increasingly complex to decide which model to select for your software. Design-time analysis and comparison of deployment decisions allow an application to be deployed according to the optimal payment model for the expected end-users. Cloud customers can take advantage of fine-grained provisioning schemes such as spot price.

Tighter provisioning. Better profiles of the resource needs of the client layer help cloud providers to avoid over-provisioning to meet their SLAs. Better usage of the resources means more clients can be served with the same amount of hardware in the data center, without violating SLAs and incurring penalties.

Application-specific resource control. Design-time analysis of scalability enables the client layer to make better use of the elasticity offered by the cloud, to know beforehand at what load thresholds it is necessary to scale up the deployment to avoid breaking SLAs and disappointing the expectations of the end-users.

Application-controlled elasticity. Going one step further, we foresee autonomous, resource-aware services that run their own deployment strategy. Such a service will monitor the load on its virtual machine instances as well as the end-user traffic, and make its own decisions about the trade-offs between the delivered quality of service and the incurred cost. The service interacts with the provisioning layer through an API to dynamically scale up or down. The service may even request or bid for virtual machine instances with given profiles on the virtual resource market place of the future!

Summary

We argued that the efficiency and performance of cloud-based services are boosted by moving deployment decisions up the development chain. Resource-aware services give the client better control of resource usage, to meet SLAs at lower cost. We identify formal methods, executable models, and deployment modeling as the ingredients that can make this vision happen. A concrete realization of our ideas is currently being implemented as part of the EU FP7 project Envisage: Engineering Virtualized Services (http: //www.envisage-project.eu).

Acknowledgments. This work has been partially supported by EU project FP7-610582 Envisage: Engineering Virtualized Services.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comp. Sys.*, 25(6):599–616, 2009.

[2] B. W. Boehm and P. N. Papaccio. Understanding and controlling software costs. *IEEE Trans. SW Eng.*, 14(10):1462–1477, 1988.

[3] E. Albert, F. de Boer, R. Hähnle, E. B. Johnsen, and C. Laneve. Engineering virtualized services. In M. A. Babar and M. Dumas, editors, *2nd Nordic Symp. Cloud Computing & Internet Technologies*, pages 59–63. ACM, 2013.

[4] E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen. ABS: A core language for abstract behavioral specification. In B. Aichernig, F. de Boer, and M. M. Bonsangue, editors, *Proc. 9th Intl. Symp. on Formal Methods for Components and Objects*, volume 6957 of LNCS, pages 142–164. Springer, 2011.

[5] E. Albert, F. de Boer, R. Hähnle, E. B. Johnsen, R. Schlatte, S. L. Tapia Tarifa, and P. Y. H. Wong. Formal modeling of resource management for cloud architectures: An industrial case study. *J. of Service-Oriented Computing and Applications*, 2013. Springer Online First, DOI 10.1007/s11761-013-0148-0.