Dynamic Provisioning of System Topologies in the Cloud

Thomas Ritter[1]
[1] Institute of Parallel and Distributed Systems (IPVS), University of Stuttgart, Universitaetsstr. 38, 70569 Stuttgart, Germany, thomas.ritter@ipvs.uni-stuttgart.de
Bernhard Mitschang[2]
[2] Institute of Parallel and Distributed Systems (IPVS), University of Stuttgart, Universitaetsstr. 38, 70569 Stuttgart, Germany, bernhard.mitschang@ipvs.uni-stuttgart.de
Cataldo Mega[3]
[3] Industry Solutions Development, IBM Deutschland Research & Development GmbH Boeblingen, Schoenaicher Str. 220, 71032 Boeblingen, Germany, cataldo_mega@de.ibm.com

Abstract. Today's IT infrastructures of companies are dimensioned to cover highly diverse workloads. In particular, it must be guaranteed that peak workloads can be processed according to concerted Service Level Agreements (SLAs). Consequently, companies have to cope with high acquisition costs of IT resources such as machines and software including costs for their maintainance and operation to meet these requirements. Thereby, achieving a high utilization of the available resources during most of the time is not possible. Based on these facts, companies endeavor to outsource their IT infrastructure to IT service providers, which in turn intend to offer respectively tailored and on-demand usable IT services using cloud computing paradigms. Obviously, the IT service providers are anxious to minimize the total cost of ownership (TCO) of their operating environments. Therefore, their goal is to minimize the amount of the provisioned IT resources by meeting tenant-specific SLAs and to maximize the utilization of the hosted IT resources by sharing them among multiple tenants (multi-tenancy). This paper presents a dynamic and cost-efficient provisioning approach of multi-tenant capable system topologies based on a Monitor-Analyze-Plan-Execute (MAPE) loop concept. For workload estimation and derivation of a capable resource topology, the MAPE loop is executed regularly regarding specified time intervals, which forms a proactive dynamic provisioning approach. Thereby, the proposed provisioning techniques apply heuristics which already encapsulate concrete performance information instead of using complex performance model solutions. Finally, a topology calculation model is developed which is the base for the proposed dynamic provisioning approach. This model enables provisioning capabilities supporting customer demands, cost-efficient utilization of resource instances, and sharing of resources by multiple tenants.

Keywords: cloud computing, dynamic provisioning, heuristics, MAPE loop, multi-tenancy

1.1 Introduction

For cutting their IT costs, including capital expenditure (CAPEX) as well as maintenance effort, companies are anxious to outsource respectively offload their operational IT infrastructure to dedicated (remote) IT service providers. In doing so companies, in this case usually called *tenants* of IT services, and IT service providers define amongst others a *Service Level Agreement* (SLA) [1][2] reflecting non-functional requirements a service has to provide – the so-called *Quality of Service* (QoS). Thereby, core SLA requirements reflecting the tenant point of view are *on-demand* usability and the *pay-per-use* business model. From the IT service provider perspective it is intended to minimize the costs of operation required to offer tenant-specific IT servcices. In this regard, challenges such as *virtualization* and *elasticity* of resources, denoted as *economy of scale*, and *multi-tenant-capable* services need to be established on provider side. That means, providers need to offer highly scalable IT infrastructures (dynamic system topologies) to cover flexibility requirements with respect to resource acquisition in a way that guarantees the specified SLAs at any time. Furthermore, IT providers aspire to minimize the amount of provisioned infrastructure resources. This in turn optimizes the overall utilization of the resources by sharing them among multiple tenants. Last but not least fewer provisioned resources reduce the total cost of ownership (TCO) of the environment on provider side.

All these issues are reflected in the fundamental ideas of the *cloud computing* paradigm [3]. By utilizing cloud computing, IT service providers get the ability to offer IT services according to specific tenant requirements. With respect to the provisioning of dynamic IT infrastructures, on which IT services in the cloud are based on, it raises the question how such an infrastructure can be built and maintained automatically considering prerequisites such as multi-tenancy, SLA compliance, economy of scale and minimal TCO? Or with other words, how can the provisioning of IT resources in a cloud environment be managed reliably and efficiently?

The primary purpose of this paper is to deal with the design and implementation of a framework for a multi-tenant-capable IT infrastructure in cloud environments using *dynamic provisioning* techniques based on *autonomic computing*. Thereby, the proposed provisioning techniques apply *heuristics* which already encapsulate concrete performance information instead of using performance models. These models are often system specific and complex, which makes it rather difficult to apply them on differing systems. The provisioning of system topologies follows the control loop concept for autonomic computing, the so-called *Monitor-Analyze-Plan-Execute* (*MAPE) loop* [4]. For workload estimation and derivation of a capable resource topology, this control loop is executed regularly regarding specified time intervals, which forms a *proactive* dynamic provisioning approach. Finally, a topology calculation model has to be developed which takes characteristics such as customer demands, cost-efficient utilization of resource instances, and sharing of resources by multiple tenants into account.

Beyond the addressed characteristics, cloud computing infrastructures have to provide additional important features, e.g., security, which will be covered in future work.

The remainder of the paper is organized as follows: Section 2 describes the state of the art of autonomic computing. Section 3 presents our dynamic provisioning framework based on the MAPE loop concept. Finally, section 4 outlines conclusions and some future work proposals.

1.2 Principles of Autonomic Computing

The *autonomic computing paradigm*, initially proposed by IBM in 2001 [4], was created in order to deal with the steadily increasing complexity of managing (installing, configuring, optimizing, maintaining) IT systems and environments, especially in the context of dynamic provisioning of system topologies in the cloud. Thereby, autonomic computing provides methods and techniques to handle fundamental cloud computing aspects, e.g., *on-demand services* and *elasticity* [6].

As discussed in [5], the self-management capability of an autonomic system consists of four basic categories *self-configuration*, *self-optimization*, *self-healing* and *self-protection*. Ideally, all these categories should be covered in the context of dynamic system topologies in a cloud to ensure the conformity with the agreed SLA rules, (also called *Service Level Objectives* - SLO) between customer (tenant) and IT service provider. In this paper, we focus on the two autonomic computing categories self-configuration and self-optimization.

1.2.1 The MAPE Loop

Looking under the hood, an autonomic system consists of various so-called *autonomic elements* containing resources and providing various services. Furthermore, an autonomic element itself can be composed by multiple *managed elements*, e.g., CPU, storage, or even software resources [5]. Control functions of managed elements are performed by the *autonomic manager* implementing an intelligent control loop. Basically, this loop consists of four phases, which will be traversed in the following order: *Monitor*, *Analyze*, *Plan* and *Execute (MAPE)*.

For a system component to be self-managing, it has to exhibit an automated method to collect all details it needs from the system (*Monitor*); to analyze those details to determine if something needs to be changed (*Analyze*); to create a plan or sequence of actions that specifies the necessary changes (*Plan*); and to perform those actions (*Execute*) [4].

1.2.2 Proactive vs. Reactive Provisioning

There exist two approaches how underlying algorithms calculate new system topologies: *proactive* (long-term) and *reactive* (short-term). Proactive approaches calculate system topologies regularly regarding specified time intervals, e.g., every hour, in order to flexibly provision resources depending on peak workloads that are predicted for the length of the respective time interval. On the other side, reactive

behaviour of a provisioning approach is required to evaluate whether the tenant-specific SLOs are fulfilled depending on actual workloads. The results are then used to calculate system topologies for new workloads. The advantage of proactive mechanisms is that system topologies can be provisioned before actual workload peaks occur. An example combining both techniques can be found in [12]. In this paper, we only consider approaches utilizing proactive dynamic provisioning.

1.3 Dynamic Provisioning Framework based on MAPE

The purpose of this paper is to present a framework for dynamic provisioning of system topologies for common and multi-tenant-capable application scenarios, whereas we combine a proactive provisioning approach with the MAPE loop concept.

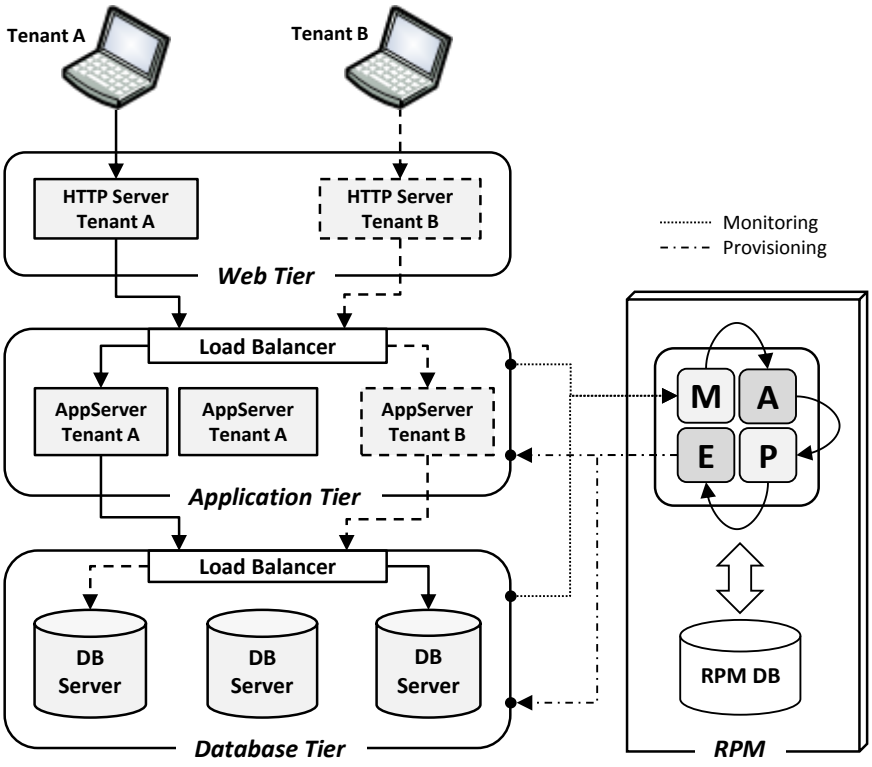

Fig. 1.1 System topology applying the MAPE loop concept

Thereby, the prerequisites are to allocate a minimal system resource topology with consequently high utilization in order to satisfy given customer SLAs for incoming workloads.

Based on these requirements, Fig. 1.1 shows a common three-tier system architecture. Critical resources (application servers, database servers), i.e. in view of workload, are monitored and managed (dynamically provisioned) by a framework built upon the MAPE loop concept. We call it *Resource Provisioning Manager* (RPM).

The RPM is triggered proactively, i.e. it is executed regarding specific time intervals in order to trigger resource provisioning depending on peak workloads predicted for the length of the respective time intervals. Once started, the RPM unit traverses the individual phases of the MAPE loop. As a result, the RPM calculates and generates so-called *provisioning flows*. These are a kind of a procedure containing instructions and rules, which map to particular resources to be provisioned or de-provisioned. Finally, the provisioning flows are executed in the execute phase, where the new system topology is built or respectively adapted.

In the following, we describe the core components of our dynamic provisioning framework and how they are interact in more detail.

1.3.1 The RPM database

Operational RPM data is preserved in a database. Table 1.1 illustrates a high-level overview of the various RPM data characteristics.

Table 1.1 RPM database – High-level overview of RPM data sets

Data Sets	Type	Comment
Heuristics of System Performance (HSP)	static	Encapsulates system performance information reflecting various workloads. They base on measured data from test runs.
Tenant-specific SLOs (TSSLO)	static	Rules for tenant-specific service functions. Example: For 95% of 10.000 incoming search requests per minute, the response time in respect to one search must be less than 5 seconds.
Sample Results (SR)	dynamic	Reflect results of sampled service function requests executed by tenants.
Workload History (WH)	dynamic	Aggregates requests and response times for every tenant-specific function.
Sample Analysis Results (SAR)	dynamic	Reflect comparison results from sample result data with the defined tenant-specific SLO target values.
Tenant-specific Workload Mixes (TSWM)	dynamic	Workload predictions for all important tenant-specific service functions.
Provisioning Flows (PF)	dynamic	Procedure containing instructions and rules for particular resources to be provisioned or de-provisioned.

In view of the data content the RPM database distinguishes between two types of data sources - *static data* and *dynamic data*. Dynamic data are the outcome of a successfully finished MAPE loop phase, e.g., the workload history data resulting from the monitor phase. One key differentiator of dynamic data with respect to static data is that the former are more volatile as dynamic data typically change for every started MAPE cycle, but static data do rather not. The semantics of dynamic data are explained in the following sections in more detail. Static data represents common or general data, i.e. heuristics and tenant-specific SLOs, which are prerequisites for dedicated MAPE loop operations. They are not calculated as part of the MAPE loop iteration, as it is the case with dynamic data.

1.3.2 Benefits of Heuristics of System Performance

Our intention was to develop a solution for dynamic provisioning which generally provides a constant good Quality of Service, for example in view of usability, serviceability, scalability, and performance.

Typically, utility-based optimization techniques [7] embed a performance model of the system into an optimization framework. In contrast to this the dynamic provisioning approach we propose in this paper applies *heuristics*, which comprise information about system performance. The main reason for using heuristics instead of a performance model is the creation of such a model is often a complex and expensive procedure. By utilizing heuristics no performance model is required, because they already encapsulate performance information of components and their respective functions for different amounts of workloads.

The initial heuristics we are using for calculating resource topologies base on measured data of generic tasks from test runs on the resources, e.g., server (de-)provisioning time, and number of specific requests (read, write) which can be processed in a certain time interval.

1.3.3 Monitor Phase

During the monitor phase, actual resource workloads for application and database serves on a per tenant basis are gathered, aggregated and persisted to the RPM database (Workload History - WH), as depicted in Fig. 1.2.

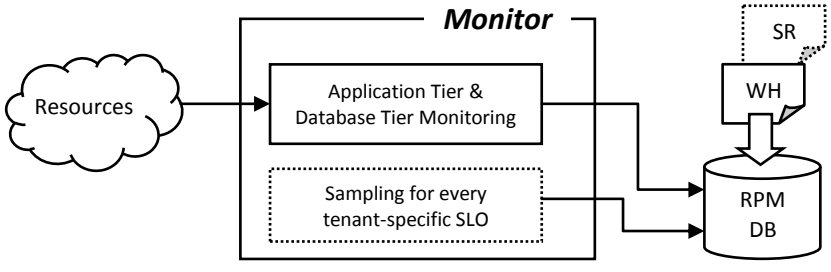

Fig. 1.2 Internal workflow of the Monitor Phase

Furthermore, in order to be prepared to support workload peaks higher than predicted in the previous MAPE loop cycle, SLO verification tests (composed of defined and known workloads) are executed. This is done using so-called *samples*. A sample is related to one specific function of the service. Thus, samples are analog to service function requests executed by tenants and enable the service provider during the analyze phase to check whether the promised SLOs can be guaranteed. Finally, the sampling results (SR), which are depicted by dotted items, are stored in the RPM database.

1.3.4 Analyze Phase

The data resulting from the monitor phase are required in the analyze phase as shown in Fig. 1.3. The objective of this phase is to predict tenant-specific workload peaks for critical system resources and to predict future workloads, e.g., by utilizing *time series forecasting techniques* [9], which use a statistical model to predict future values based on previously observed values. The result are tenant-specific workload mixes (TSWM) which consists of workload predictions for all tenant relevant functions, such as read and write requests.

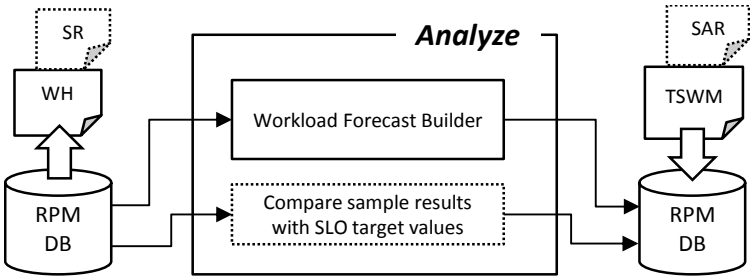

Fig. 1.3 Internal workflow of the Analyze Phase

Results of samples executed in the monitor phase are also used to derive whether more resources have to be provisioned in order to meet future workloads. This is done while comparing the results from the sample execution with the defined SLO target values. In case less resources are needed regarding future workloads, the system topology is not changed, since the frequency of the expensive provisioning of resources should be kept as minimal as possible. The results of the sample analysis (SAR) as well as tenant-specific workload mixes are saved in the RPM database.

1.3.5 Plan Phase

In the plan phase, the proposed MAPE loop algorithm calculates the system topology. Here, all the previously created, collected, and derived data that are persisted in the RPM database are used for the calculation of provisioning flows, that later create or adapt the real system topology. As illustrated in Fig. 1.4, the workflow of the plan phase is a two-stage process.

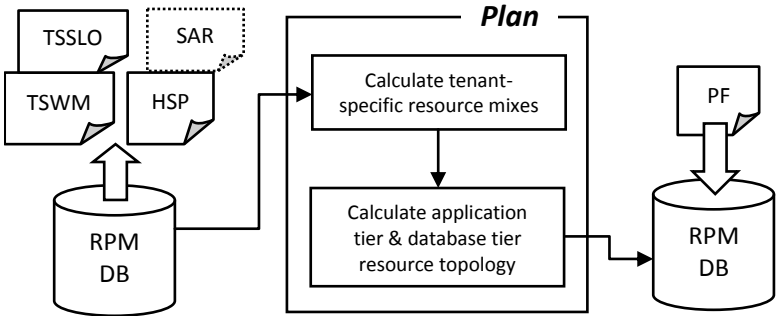

Fig. 1.4 Internal workflow of the Plan Phase

The first step calculates *tenant-specific resource mixes*. Thereby, resource mixes base on tenant-specific workload mixes (TSWM), sample result analysis data (SAR), tenant-specific SLOs (TSSLO) and on the heuristics comprising information about system performance (HSP). These resource mixes are similar to workload mixes, but contain the demand of a specific tenant on every demandable resource, e.g., application server.

Based on the tenant-specific resource mixes the second step of the plan phase - the *resource topology optimization* - is started. The objective of this step is to calculate an optimized resource topology for the application tier as well as the database tier.

To be able to do so, the resources or capacity a system provides need to be modeled at an adequate generic abstraction layer. The adoption of such an abstraction layer is based on the idea of Amazon's EC2 Compute Units [8]. Therefore, the term *Partial System Unit* (PSU), an equivalent of partial system capacity, is established as abstraction for a piece of capacity a system (S) provides. Briefly, the overall system capacity (S) is the sum of its partial system capacities (PSUs). This relationship can be abstracted using the following formula:

$$S = \sum_{i} PSU_i \tag{1}$$

It's introduction is based on the knowledge that there are typically a number of resources showing differences in specific characteristics (CPU, main memory). Thus, the major purpose of the abstraction layer is to be able to early abstract for individual resources, which finally enables to use a generic provisioning algorithm for different system topologies.

Fig. 1.5 illustrates an example of an internal representation of an application tier system topology modeled via the PSU approach.

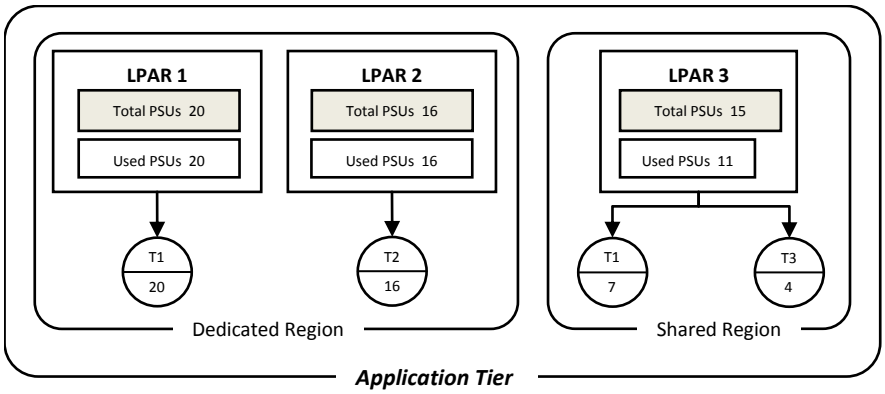

Fig. 1.5 Internal representation of an application tier system topology model

In this example, resources (LPARs - Logical Partitions - denote virtual hardware resources) are shown as rectangles and tenants as circles. The first entry of an resource element specifies its maximum or total available amount of PSUs, whereas the second entry describes the amount actually assigned respectively used by tenants. Tenants are distinguished by the first entry of an tenant element. PSUs used by a tenant of the respective resources are defined by the second entry of an tenant entity.

Additionally, we introduce the concept of *dedicated* and *shared regions* for the calculation of the optimized resource topology. Thereby, a dedicated region contains LPARs which may only be used by a single tenant. However, LPARs of the shared region may be utilized by multiple tenants which enables a multi-tenancy-awareness of the modell. The main reason for the classification of system topologies into a dedicated region and a shared region is that this reduces the complexity of the topology calculation. This calculation can be mapped to constraint programming techniques [10][11] in order to find optimized sets of resources for given workloads and a given system model. Results of the plan phase are a set of provisioning or de-provisioning flows (PF). These provisioning flows are again stored in the RPM database.

1.3.6 Execute Phase

Provisioning flows resulting from the plan phase are finally executed in the execute phase in order to transform the old system topology into a new one. They are abstract descriptions of single transforming steps. To make these abstract descriptions executable, they mapped to concrete deployment and configuration commands, e.g., scripts or workflows for virtualization frameworks.

1.4 Conclusions and Future Work

In this paper we introduced a dynamic cloud provisioning solution of a common two-tier system topology based on a MAPE loop concept. This solution enables sharing of resources between tenants and an automatic resource provisioning considering tenant-specific demands. A first prototype of that approach has been implemented and is currently tested in the area of scalable archiving services.

For future work we will concentrate on prototype extensions and detailed scenario testing. On the other side, for the plan and analyze phase, the concept of an reactive provisioning has to be worked out in addition to our proactive one.

Acknowledgement

The authors would like to thank Andreas Boerner, who contributed with his master thesis on this topic, and Peter Reimann, who carefully reviewed this article.

References

[1] A. Paschke, E. Schnappinger-Gerull. A Categorization Scheme for SLA Metrics. In Multi-Conference Information Systems. 2006.
[2] A.Boerner. Orchestration and Provisioning of Dynamic System Topologies. Master's thesis, University of Stuttgart, 2011.
[3] F. Leymann. Cloud Computing: The Next Revolution in IT. URL http://www.ifp.uni-stuttgart.de/publications/phowo09/010Leymann.pdf, Photogrammetric Week '09, 2009.
[4] IBM. An architectural blueprint for autonomic computing. Autonomic Computing White Paper, 2006.
[5] J. O. Kephart, D. M. Chess. The Vision of Autonomic Computing. Computer, 2003.
[6] P. Mell, T. Grance. The NIST Definition of Cloud Computing (Draft). National Institute of Standards and Technology, 2011. URL http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf.
[7] D. Ardagna, B. Panicucci, M. Trubian, L. Zhang. Energy-Aware Autonomic Resource Allocation in Multi-Tier Virtualized Environments. IEEE Transactions on Services Computing, 99, 2010.
[8] URL: http://aws.amazon.com/de/ec2, accessed on 14th Sep 2011
[9] A. K. Palit, D. Popovic. Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications. Advances in Industrial Control. Springer, 2005.
[10] Y. Chen, S. Iyer, X. Liu, D. Milojicic, A. Sahai. SLA Decomposition: TranslatingService Level Objectives to System Level Thresholds. Technical report, HP Laboratories Palo Alto – Enterprise Systems and Software Laboratory, 2007.
[11] H. N. Van, F. D. Tran, J.-M. Menaud. SLA-Aware Virtual Resource Management for Cloud Infrastructures. Computer and Information Technology, International Conference on, 1:357 – 362, 2009.
[12] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, A. Saxena. Intelligent Workload Factoring for a Hybrid Cloud Computing Model. In Proceedings of the 2009 Congress on Services - I, pp. 701 – 708. 2009.