A Framework for Optimized Distribution of Tenants in Cloud Applications

Christoph Fehling, Frank Leymann, Ralph Mietzner

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{fehling, leymann, mietzner}@iaas.uni-stuttgart.de

BIBTEX:

```
@inproceedings{FehlingLM10,
   author    = {Christoph Fehling and Frank Leymann and Ralph Mietzner},
   title     = {{A Framework for Optimized Distribution of Tenants in Cloud
                Applications}},
   booktitle = {Proceedings of the 3rd IEEE International Conference on Cloud
                Computing, CLOUD 2010},
   year      = {2010},
   pages     = {252--259},
   doi       = {10.1109/CLOUD.2010.33}
   publisher = {IEEE Computer Society}
}
```

2010 IEEE 3rd International Conference on Cloud Computing

A Framework for Optimized Distribution of Tenants in Cloud Applications

Christoph Fehling, Frank Leymann, Ralph Mietzner
Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstr. 38, 70563 Stuttgart, Germany
firstname.lastname@iaas.uni-stuttgart.de

***Abstract*—To be successful a cloud service provider has to target a preferably large customer group to leverage economies of scale. Therefore an application offered as a service in the cloud is often configurable regarding non-functional qualities, such as location or availability. Since many of these qualities depend on the resources on which the service is hosted, a large number of computing environments has to be managed by the service provider.**

This paper analyses the challenges arising from such a scenario and identifies several optimization opportunities originating from an intelligent distribution of users among the functionally equal resources with different quality of services. A framework enabling the development of distribution strategies exploiting these opportunities is defined. It allows modeling of resources, their deployment dependencies, and users with specific demands. An architecture and prototype of a management system is introduced to handle the required resource provisioning and user request routing. Several optimization strategies are defined and their performance is evaluated using statistical data of an existing cloud service provider.

***Keywords*-cloud, multi-tenancy, provisioning, SaaS**

I. INTRODUCTION

Providing computing resources over a network is a business model that currently gains widespread acceptance. A customer accesses a resource or service hosted by a provider instead of managing it on his own premise. Depending on the type of resource offered this is called Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS). These * *as a Service* offerings are typically subsumed under the term *cloud computing* [1]. One significant aspect which makes these * as a Service offerings successful, is the sharing of resources between customers, also called *tenants* in this context. This allows providers to exploit economies of scale by leveling the load of multiple customers on the same infrastructure. To increase this effect the number of targeted costumers has to be maximized. One attempt is to provide a service that fits all customers. However, as different customers have varying demands for a service, this approach is often not feasible. To be able to address a large customer base with different demands, a provider must ”catch the long tail” [2] by offering multiple variants of a service fitting different customers' needs. The potential of this approach is evident in Amazon's success on the book market which is largely based on selling small amounts of very specific books to a large group of customers with very specific individual interests. Analogous, a SaaS providers can exploit the same effect by providing a service with customizable non-functional qualities to address different costumers. These often have different demands regarding, for example, the location, security, availability, and ultimately the resulting price of a service. However, this introduces a complex challenge for the provider. Since service qualities are often highly dependent on the used computing infrastructure, a large number of computing environments has to be managed and tenants have to be assigned to the different resources of these environments in an efficient manner. Thus, a framework is needed that enables providers to model services with different quality of service (QoS) and non-functional properties. The provider can then use algorithms on top of this model to compute efficient customer distributions that take requirements of the customers into account.

After covering related work in Section II, such a framework is introduced that deals with the optimized distribution of customers in cloud applications built using a service-oriented architecture. Section III describes in detail in which scenarios optimization opportunities arise and derives a set of properties that a system targeted by optimization should display. The necessary models to describe users, their demands, system resources, as well as their dependencies are defined in Section V. These models form the basis for an optimization algorithm introduced in Section VI. A management architecture handling the provisioning of resources and routing of customer requests is introduced in Section VII. We evaluate different user distribution strategies in Section VIII, and finish with a conclusion in Section IX.

II. BACKGROUND AND RELATED WORK

SaaS applications allowing their resources to be shared between costumers are said to be *multi-tenant aware*. In [2] it is differentiated between four levels of tenant awareness. The first level corresponds to that of traditional application service providers (ASP) [3] meaning that every resource is instantiated exclusively for one tenant. Applications on the second level may be configured individually, but are still instantiated separately for every tenant. On the third level, the configuration data is extracted from the application. Only a single instance is present which utilizes this data to display a different behavior for each tenant. The fourth and last level dictates that multiple instances utilize the extracted configuration data and are load balanced to allow serving of a larger tenant number.

Designing and building of multi-tenant aware applications introduces several new challenges. A central one is *tenant isolation* meaning that even though resources are shared every tenant perceives the application as if he were the only tenant. In [4] a framework is introduced to achieve the required isolation regarding security, performance, availability, and administration. In [5] we introduced additional patterns which may be used to develop multi-tenant aware service-oriented applications. The patterns are used to describe which parts of an application are shared between tenants and which parts must be available individually for every tenant. The patterns also describe how tenants accessing the application are routed to the correct service for that tenant through tenant context-based routers. Each tenant request is associated with a tenant context. This context is used by a tenant context-based router to identify resources assigned to the tenant and route the request respectively.

None of these previous works, however, considers performance optimization which may be obtained by a smart distribution of application users among the required resources. Within the scope of this paper optimization of user distribution among resources forming forth level tenant-aware applications is investigated. One prerequisite is that resource and system properties may be described to form the basis for such an optimization. In [6] a method is introduced to describe the costs to operate Grid resources. This is done by assigning so-called *units of trade* to resources. Units of trade can be, for example, computing time, messages exchanged, or the number of transactions performed. Every unit of trade may then be associated with a price. Section V-A shows that this approach may also be used to describe non-cost oriented properties which may then be targeted by optimization. It is also necessary to determine the number of users served by one resource. Capacity planning techniques, such as those presented in [7], introduce models to describe the workload, performance, and availability of a system. Using these models the required capabilities of resources to serve an amount of users may be assessed.

III. Motivating Scenario

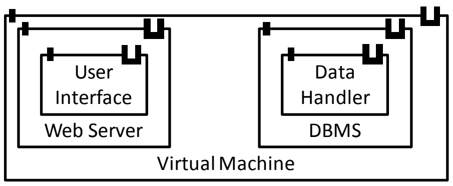

Figure 1. Deployment Dependencies of the Motivating Scenario

Consider a service provider offering a calendar service. Through a web interface users may store appointments in a database and share them with others. Application components and their dependencies on other software are shown in Figure 1. This is from now on called *deployment diagram* and is described further in Section V. To be successful a service provider has to achieve three important goals, (i) maximize the target costumer group, (ii) operate resources in an efficient manner, and (iii) adjust flexibly to changing demands.

In order to increase the target customer group, the service provider needs to offer a certain flexibility for customers, when subscribing to the application. That flexibility can be, for example, the subscription to only a subset of the components, or the selection of one of several quality levels. Tenants are thus allowed to subscribe to any subset of application components. In the scope of the example application they may want to subscribe only to the data handler component and use another application as the user interface. This use case can be employed by customers that want to take advantage of high-available, managed storage at a provider that they cannot supply on premise. Another customer may, for example, only subscribe to the user interface component, as this customer wants to integrate his own storage component.

Another desirable system property is the possibility to adjust the QoS individually to a tenant's demands. High QoS regarding performance, availability, or security increase the price of the offered service and tenants are unlikely to pay for higher QoS than they actually require. In this scenario the provider offers the service at three different quality levels, gold, silver, and bronze. Those differ in non-functional properties such as response time or availability assurances. Offering a service in different quality levels allows a provider to cater to different needs of different customers, while keeping the number of variants of the service low. As one goal of a cloud based approach is to exploit economies of scale by offering the same service to as many tenants as possible, providers will need to find the right balance between standardization (i.e., keeping the number of variants low) and attracting as many customers as possible by offering fine-granular customization options for the QoS of a service. For example, Amazon EC2 [8] offers their compute service in multiple quality levels, ranging from small to quadruple extra large instances that differ considerably in compute power and memory size.

In this scenario, the provider offers to host the service in his own private data center as well as in Amazon's public cloud as shown in Figure 2. Note that the private data center may also use cloud computing technologies to manage its resources. It would then be referred to as a *private cloud* [9]. Since offered quality levels are highly dependent on resource properties, the provider is forced to use multiple computing environments. In order to operate the resources residing in these environments efficiently, and thus

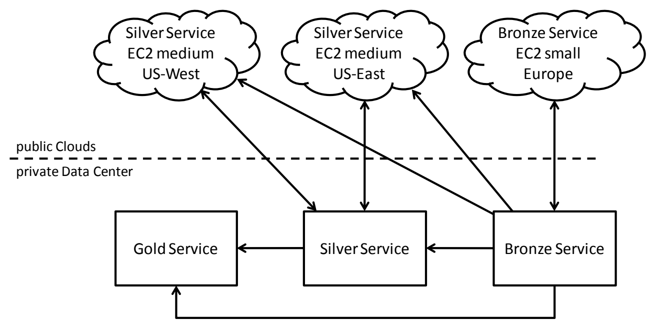
Silver Service EC2 medium US-West
Silver Service EC2 medium US-East
Bronze Service EC2 small Europe
public Clouds
private Data Center
Gold Service
Silver Service
Bronze Service

Figure 2. Computing Environments of the Motivating Scenario

optimize their utilization, resources may substitute others based on their QoS. In Figure 2 possible substitutions are depicted as arrows. They denote the possible movement of users from one computing environment to a substituting one. For example users requesting the bronze service may also be assigned to all other resource types. At first this does not seem profitable, however, there are circumstances under which such an "upgrade" can lead to a better utilization of the overall system.

To adjust flexibly to changing demands, the system must be able to handle changing user numbers efficiently. Through integration of computing environments, the dynamicity of the EC2 cloud can be combined with the probably smaller price of local resources. How well rapidly increasing user

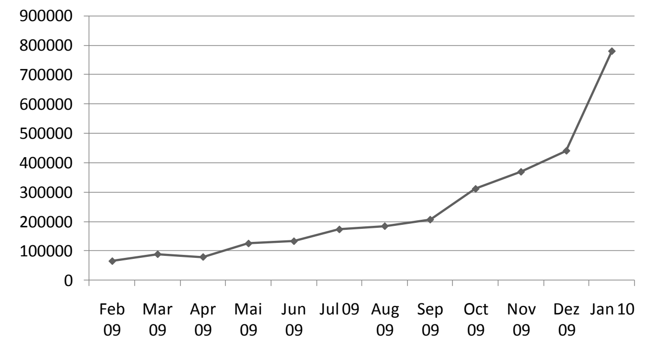

Figure 3. Unique Visitors of Ooyala.com per Month [10]

numbers may be handled by dynamic clouds, has been shown, for example, by the startup company Ooyala. It offers video hosting, transformation, and streaming based completely on EC2 virtual machines. In April 2009 their page had roughly 80.000 unique visitors, as shown in Figure 3. In less than one year, this number increased to more than 780.000 in January 2010. Therefore, the number of required resources is now almost ten-times as high. For service providers of Web 2.0 offerings such an increase is likely to happen due to rapidly increasing popularity. In case of Ooyala it originated from offering their service as a Facebook application. With the introduced approach, Ooyala would also be able to offer the service to costumers that do not allow their videos to reside in public clouds by integrating Ooyala's private data center or even resources residing on the costumers' premises.

Of course the required integration of multiple computing environments introduces several challenges. Instead of balancing users among equal resources, as has been researched in [11], resources differing in properties such as capacities, price, and dynamicity now have to be considered.

IV. Optimization Opportunities through Tenant Distribution

In the following, generic cases are derived from the motivating scenario which may profit from an optimized distribution of users among heterogeneous computing environments.

A. *Static Resources and increasing User Numbers*

In several cases resources in local data centers are cheaper than those of public clouds [12], [13]. However private servers are a very static resource. While a new EC2 instance is accessible within minutes, ordering and installation of a private physical server requires a significantly greater amount of time. To achieve the same dynamicity in a private data center, managed as a private cloud, massive overprovisioning would be necessary. Therefore, without integrating dynamic clouds and private data centers a service provider would have to increase the size of its local resources far ahead to handle abruptly increasing user numbers [14]. Through integration, local resources may be provisioned more optimistically. If their number is insufficient, dynamic resources are provisioned temporarily. Note that this setup it still profitable if not all components of the application, such as data storage, may be hosted in the public cloud.

B. *Static Resources handling many Users per Instance*

Resources with high QoS such as the gold service in the motivating scenario are unlikely to be substituted by other resources. Therefore, the provider has to assure that increasing user numbers may be handled by provisioning more of these resources than are needed. Even if they were more flexible, such high-performance resources also tend to allow serving of very large user numbers. However, a tenant may sign up with an arbitrary small amount of users.

Both factors may lead to a bad utilization of powerful resources. Therefore users requesting weaker service qualities can be assigned to such badly utilized resources if possible to increase the overall utilization. A similar approach is taken by air lines when travelers are granted business class seats even though they only booked economy class and the economy class has been overbooked.

C. Equal Resource serving different Amount of Users

Available resources may display equal QoS while being unequally powerful and thus allowing to serve different amounts of users per instance. For example, when requesting an EC2 instance, one may specify processor speed, number of processors, size of the main memory, hard disk size, etc. in form of pre-defined classes such as small, large, or extra large. Also resources may contain multiple, different services for better utilization. For example, one server resource could contain only a Web server, only a database, or it could contain both.

Depending on the amount of users requesting services and the ratios between requests for different services types, resources should be provisioned which are best fitting to achieve a good utilization.

D. Factors of Profiting Systems

These generic cases allow to identify well suited candidates for optimization. A set of factors was deduced which a target system should display. The more these factors are present, the more likely it is that the system will profit from the optimization mechanisms introduced in Section VI:

- The system is used by tenants who demand different QoS.
- Based on QoS, resources can be identified which may substitute others.
- Managed resources differ in flexibility.
- Instances of resources may serve different user amounts.

V. Resource and Tenant Modeling

In order to optimize the user distribution among a heterogeneous set of computing environments, certain prerequisites have to be established. First, properties to evaluate user distributions have to be defined. We differentiate between properties of resources and system properties emerging from them. Resource properties may include price per instance, processing power, or service level agreements. Derived system properties could be overall cost, throughput, response time, utilization, flexibility, etc. Also it has to be described which resources may be used to serve a specific tenant and resource dependencies have to be modeled.

If this is established, the user distributions can be optimized while respecting tenants' demands and optimizing the overall system properties. Note that while the most obvious overall system property is cost, optimization is not limited to it.

A. Defining Resource and System Properties

In [6] a general cost schema is introduced to associate Grid resources with a certain amount of money required for their operation. This amount can be based on the time the resource is reserved, on the number of messages exchange etc. This scheme may also be used to describe other properties such as required computing time per request, by altering the unit of the associated costs. In this way, more complex system properties can be formulated, such as response time. In another scenario tenants might specify the time of day during which they usually access the system. Optimization might then target an evenly utilization of resources. Computation and evaluation of system properties is further described in Section VI.

Description of the other prerequisites, tenants' resource requirements and the dependency relationship between resources, is handled with a set of models on which the optimization algorithm operates.

B. Tenant oriented Deployment Diagram

In the following the combination of concepts from deployment diagrams, introduced in Cafe [15], and the *tenant aware resource model*, introduced in [16], is described. These concepts are combined to enable the modeling, deployment planning, and provisioning of the application introduced as motivation scenario in Section III. Cafe's deployment diagram describes software components and their deployment dependencies among each other. A component is represented by a box and may contain other components meaning that the contained component is deployed on the containing one. In the upper left corner of a box a solid rectangle represents so-called *visible properties* of the component. Those properties change depending on the provisioning of a component instance, such as host names and IP addresses. Those visible properties are used to fill *variability points* of other components which are represented by an open rectangle in the upper right corner of the component boxes. The host name of the DBMS for example is used to fill a variability point of the user interface which specifies where to access the data handler.

The tenant aware resource model is used in this paper to model tenants and their users. A tenant owns multiple *usage partitions* which contain his users. Further the model contains so-called *offering groups*. Each offering group is considered to offer a certain service. To combine the two models these offering groups are replaced by components of the deployment diagram. The extended deployment diagram of the example application is shown in Figure 4. The user interface is implemented as a PHP script. The data handler is realized as a set of tables in a database. The user interface has a deployment dependency on a Web server, whereas the data handler depends on a DBMS. Both hosting components are installed in a virtual machine. Every usage partition of a tenant is associated with a component based on the selection a tenant makes during the registration process. If a tenant registers for the full application, usage partitions are created for both, the user interface and the data handler component. If a tenant chooses to register for only one of the components the number of his usage partitions is reduced. Additionally a tenant specifies required QoS during the registration process

[17]. Those may include service availability or allowed tenant patterns [5].

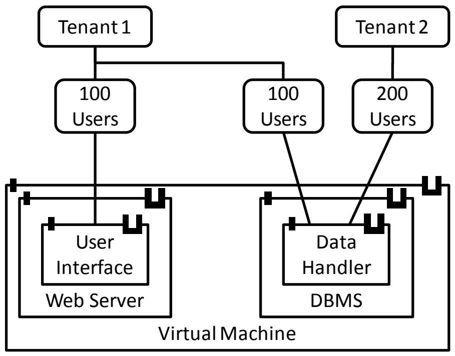

Figure 4. Extendend Deployment Diagram of the Example Application

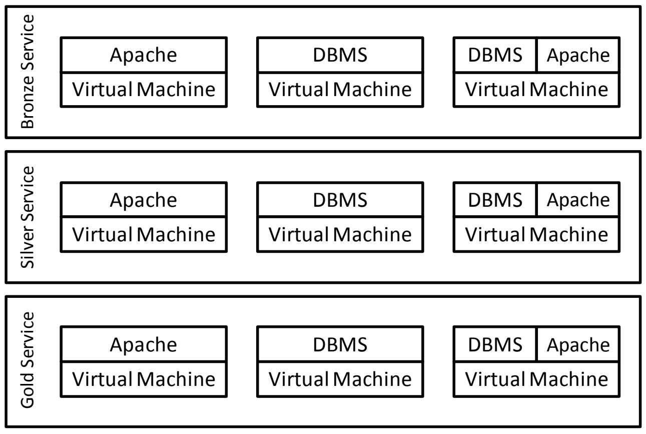

Figure 5. Deployment Alternatives of the Example Application

Since the QoS often depend on resource properties, the deployment diagram must be instantiated in different computing environments. The resulting deployment combinations are called *deployment alternatives*. A subset of all deployment alternatives of the motivating scenario is depicted in Figure 5. In this diagram, usage partitions are associated with all deployment alternatives which may serve their users, again according to the QoS provided by tenants. To increase visibility these associations were omitted from the diagram. In this way the diagram also shows influences of some QoS, which were not visible before. A tenant may, for example, specify that his DBMS must be hosted on a dedicated system which would render the deployment alternative in which the DBMS shares a virtual machine with the Web server unusable. The offered service levels are represented as separate stacks of the deployment alternatives. Note that offering groups cannot be mapped directly to all deployment alternatives, since they only offer one service while deployment alternatives may contain multiple services. The following section shows how the optimization algorithm introduced in [16] may be adjusted to operate on deployment alternatives.

VI. System Optimization

Since resources, tenant demands, and overall system properties can now be modeled, algorithms searching optimal user distributions are needed to make use of these models. Finding the optimal distribution of users among sets of variable resources is proven to be a np-hard problem in [16]. Therefore it is very unlikely that an algorithm exists that always find the optimal solution in polynomial time. A class of algorithms to find a good, even though not always the optimal, solution for such problems are local search algorithms [18]. In this paper a hybrid algorithm, called *smarter simulated annealing* [16], based on simulated annealing [19] and hill climbing [18] is used. It computes possible user distributions starting from an initial one and rates their performance based on a fitness function. This function utilizes the resource properties and derived system properties to compute a comparable value, the fitness, for each possible user distribution.

Originally the smarter simulated annealing algorithm operates on offering groups among which users are distributed. Algorithm 1 shows how deployment alternatives may be used in the same manner. It starts with one legal user distribution. For example, this could be obtained by randomly assigning users to a deployment alternative of the initially requested service level. By making small rearrangements of users the algorithms then searches an optimal solutions stepwise. It computes so-called neighbors of the initial user distribution by moving users between deployment alternatives. Then it selects one of these distributions according to a selection strategy as a next step. The selection strategies form the main differences between the simulated annealing and hill climbing approach.

Throughout the runtime, a random user distribution is computed at the beginning of every step. If its fitness is better than that of the user distribution selected in the last step, it is chosen as the next step. If it is worse then it is still chosen with a certain probability. This is the simulated annealing selection strategy. The probability decreases the longer the algorithm runs by subtraction of a so-called cooling rate. The algorithm terminates, once the probability is zero. If a worse user distribution is not chosen by the simulated annealing selection strategy, a hill climbing selection is performed. The algorithm looks at all possible rearrangements of the current user distribution and selects the one having the best fitness value.

Following this approach, the algorithm combines advantages of simulated annealing and hill climbing. By selecting weaker user distributions it is unlikely to get stuck in local optimum which is an advantage of simulated annealing. A local optimum is a user distribution for which all possible rearrangements display a degraded fitness, but another user distribution exists which performs better. If only the hill climbing selection strategy was used the algorithms would terminate once such a local optimum is found. Towards the end of its runtime, when it can be assumed that the algorithms is close to finding a good user distribution, hill climbing is performed to move towards this optimum faster.

Algorithm 1 Smarter Simulated Annealing Algorithm.

```
$result \leftarrow$ assign users to initial deployment alternatives (i.e. randomly)
$probability \leftarrow 1$
$coolingrate \leftarrow number \in [0 \ldots 1]$ {very small number}

while $probability > 0$ do
  $probability \leftarrow probability - coolingrate$
  $neighbor \leftarrow$ move users from one deployment alternative to others randomly
  if $fitness(neighbor) > fitness(result)$ then
    $result \leftarrow neighbor$
  else
    {Even though the neighbor is weaker it is chosen by chance (simulated annealing).}
    $random \leftarrow random\ number \in [0 \ldots 1]$
    if $random < probability$ then
      $result \leftarrow neighbor$
    else
      {The weaker element was not chosen, a better one is searched (hill climbing).}
      for $neighbor \in$ all possible user movements do
        if $fitness(neighbor) > fitness(result)$ then
          $result \leftarrow neighbor$
        end if
      end for
    end if
  end if
end while
```

A. Optimization Targets

For system performance it is very important to carefully consider the amount of users considered by optimization. Two different optimization targets are introduced in [16], *global* and *request-based optimization*. A global optimization considers and therefore rearranges all users of the application. This is likely to result in major system reconfiguration. Request-based optimization only considers the users which are new to the system. It leaves large portions of the resources untouched.

The adequate optimization target is chosen based on the characteristics of the managed system and behavior of tenants. If a large number of tenants signs up to the application in short time intervals, performing a global optimization after each request is likely to result in too many restructuring activities degrading the overall system performance. If system resources are however very dynamic it might still be feasible.

Since the time needed for system alterations is usually much larger than the time needed by optimization algorithms both optimization targets can be evaluated prior to changing the resources. This way, global optimization may only be reflected in system changes if a certain threshold respecting the system property to be optimized is exceeded.

VII. Management System Architecture

To reflect the optimized user distribution the number of required resources has to be provisioned and tenants' requests have to be routed properly. Figure 6 shows a service oriented architecture in which a central management and optimization component performs all of the required tasks. First it handles user registration and stores the required QoS and values for variability points. Based on this information, optimization is performed, required resources are provisioned, and routing information is made available to a tenant context-based router as well as load balancers.

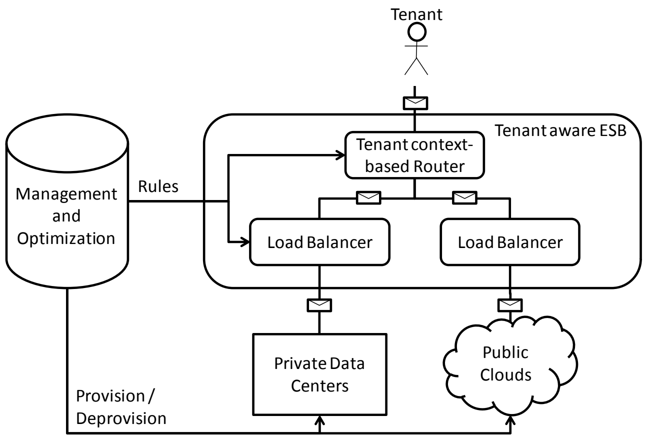

Figure 6. Architecture of the Management System [16]

The necessary tenant awareness of the system was realized as a component of Apache ServiceMix [20], an Enterprise Service Bus (ESB). Tenants access system resources residing in data centers and clouds through this middleware. Each request has a tenant context [5] attached that is used by the tenant context based router to select the appropriate computing environment for the tenant. Routing rules are passed to this router by the management component. Load balancers distribute requests among the resources of a certain computing environment. They also receive rules so that the different capabilities of resources, described in Section IV-C, can be respected in distribution ratios.

VIII. Evaluation of Different Provisioning Strategies

In this section the impact of different environmental conditions on the success of a set of provisioning strategies is investigated. A subset of the motivating scenario is chosen to reduce the number of possible side effects during the evaluation.

A general factor influencing optimization capabilities is the degree to which resources may be moved between different computing environments. Good candidates are usually all stateless resources, short running processes, and resources working independent of others. Bad movement candidates are resources handling large data volumes or long running processes. Their movement requires a lot of state information to be transferred between computing environments which is likely to decrease the system performance and may interrupt services.

A. Provisioning Strategies

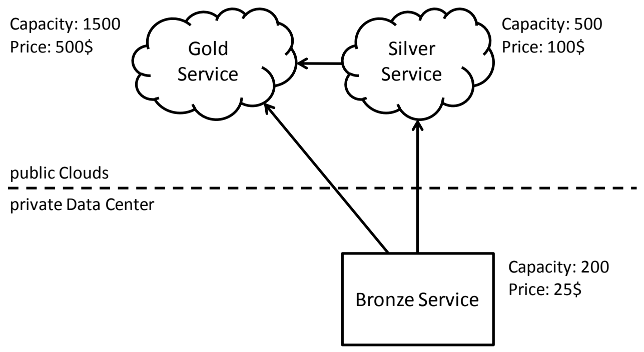

Figure 7. Scenario for the Evaluation.

The architecture of this framework allows the usage of arbitrary provisioning strategies based on the used metamodel. Some possible provisioning strategies are now evaluated using the user data of Ooyala shown in Figure 3 as a basis. Within the scope of this evaluation, all resources are associated with a certain amount of money that it costs to operate the resource for one month. The overall system property to optimize is the sum of individual costs of all required resources. The scenario depicted in Figure 7 still includes three service offerings: gold, silver, and bronze. The gold and silver offerings are considered to be hosted in dynamic cloud environments. The bronze service is hosted in a private data center and may not be provisioned dynamically meaning that the amount of resources of this type has to be fixed at the beginning of each month. Four strategies are evaluated:

1) Bronze always correct: provides a reference value by treating bronze resources as equally dynamic as silver and gold resources and thus always provisions the required amount.
2) Bronze always correct with optimization: performs an additional optimization considering all resources as equally dynamic.
3) Pessimistic prediction: bronze resources are provisioned in the amount needed plus an additional 30% to handle unexpected increasements. No optimization is performed.
4) Optimistic prediction: the number of needed resources is predicted by computing the change in user numbers of the last two months. Optimization allows that bronze resources are substituted by others if the predicted amount is insufficient.
5) Delayed provisioning: no prediction is performed at all. At the end of a month the number of bronze resources which would have been optimal for the last month is provisioned. Therefore all newly required bronze resources will be substituted to others at first.

In Figure 8, months February and March were omitted since their user numbers were used for predictions. Performance of the others is expressed as a percentage by which they optimize the overall system cost respectively to the pessimistic approach. For the first two strategies the performance of an optimized user distribution is only slightly better than that of the unoptimized one. This shows that optimization opportunities mainly originate in the different dynamicity of resources. If all resources are equally dynamic, optimization may only target the boundary users not filling complete resource instances. If the cost for bronze resources were higher, optimization might however have a larger impact. Further it can be observed that delayed provisioning performs well when the user number alternates within a certain corridor as is the case in the beginning of the observed time frame. However only resources which are easily movable are within the scope of this approach. Prediction tends to fail in such an environment, since there is no steady increase or decrease of user numbers. Towards the end of the time frame, the increase becomes more steady and therefore prediction is showing better results. Again the movability of resources has to be taken into account, especially when a decrease is predicted. In such a case resources which are hard to move should be provisioned with a more pessimistic strategy.

IX. Conclusion

Distribution of tenants has now been identified as a opportunity for system optimization. While leveling the load among similar resources is well established, the motivating scenario has shown that the need for heterogeneous resources introduces new challenges as well as opportunities for the service provider. General system characteristics were defined which allow identification of systems likely to profit from optimization. Existing models were extended to include a notion of tenants additional to resources and their deployment dependencies. Those new models form the basis

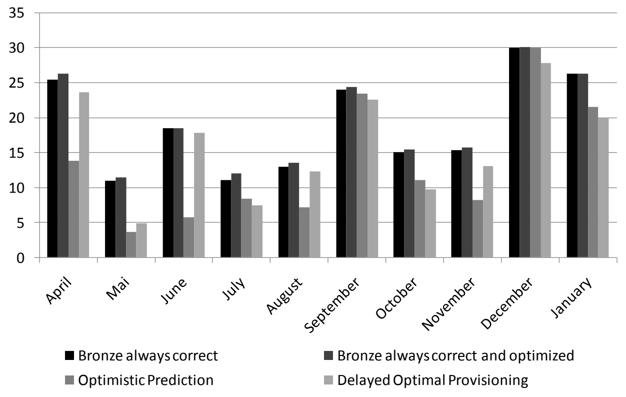

Figure 8. Improvement over Pessimistic Prediction (%)

for optimization and a set of general distribution strategies. Evaluation of these strategies has shown that successful optimization is highly dependent on the properties of the system to be optimized, and dynamicity of resources is the central factor.

The presented framework allows service providers to use distribution strategies as well as define them on their own. Optimization may be performed regarding individually modeled resource and system properties. The defined characteristics of profiting systems allow providers to identify optimization opportunities. The optimization results may be utilized through the described management system architecture handling the necessary provisioning and request routing.

REFERENCES

[1] R. Buyya, C. 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 Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] F. Chong and G. Carraro, "Building Distributed Applications Architecture Strategies for Catching the Long Tail," 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa479069.aspx

[3] L. Tao, "Shifting Paradigms with the Application Service Provider Model," *Computer*, pp. 32–39, 2001.

[4] C. Guo, W. Sun, Y. Huang, Z. Wang, and B. Gao, "A Framework for Native Multi-Tenancy Application Development and Management," in *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, 2007, pp. 551–558.

[5] R. Mietzner, T. Unger, R. Titze, and F. Leymann, "Combining Different Multi-tenancy Patterns in Service-Oriented Applications," *2009 IEEE International Enterprise Distributed Object Computing Conference*, pp. 131–140, 2009.

[6] A. Caracas and J. Altmann, "A Pricing Information Service for Grid Computing," in *Proceedings of the 5th International Workshop on Middleware for Grid Computing held at the ACM/IFIP/USENIX 8th International Middleware Conference - MGC*, 2007, pp. 1–6.

[7] D. Menasce and V. Almeida, *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2002.

[8] Amazon.com, "Amazon Elastic Compute Cloud (EC2)," http://aws.amazon.com/ec2/.

[9] F. Leymann, "Cloud Computing: The Next Revolution in IT," in *Proceedings of the 52th Photogrammetric Week*, 2009, pp. 3–12.

[10] Compete.com, "Siteanalytics of ooyala.com from 01.2009 to 01.2010," http://siteanalytics.compete.com/ooyala.com/.

[11] C. Kopparapu, *Load Balancing Servers, Firewalls, and Caches*. Wiley, 2002.

[12] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The Cost of Doing Science on the Cloud: the Montage Example," in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. IEEE Press, 2008, pp. 1–12.

[13] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep., Feb 2009.

[14] J. Varia, *Architecting for the Cloud: Best Practices*. White Paper of Amazon.com, 2010.

[15] R. Mietzner, T. Unger, and F. Leymann, "Cafe: A Generic Configurable Customizable Composite Cloud Application Framework," *On the Move to Meaningful Internet Systems: OTM 2009*, pp. 357–364.

[16] C. Fehling, "Provisioning of Software as a Service Applications in the Cloud," Master's thesis, University of Stuttgart, 2009.

[17] T. Unger, R. Mietzner, and F. Leymann, "Customer-defined service level agreements for composite applications," *Enterprise Information Systems*, vol. 3, no. 3, pp. 369–391, 2009.

[18] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach (Third Edition)*. Prentice Hall, 2010.

[19] S. Brooks and B. Morgan, "Optimization using simulated annealing," *Journal of the Royal Statistical Society. Series D (The Statistician)*, vol. 44, no. 2, pp. 241–257, 1995.

[20] Apache Software Foundation, "Apache ServiceMix," http://servicemix.apache.org.