Proactive Automated Dependable Resource Management in Cloud Environments

Anna Schwanengel and Gerald Kaefer
Siemens AG
Corporate Technologies / Industry Sector
Munich / Nuremberg, Bavaria, Germany
Email: {anna.schwanengel.ext, gerald.kaefer}@siemens.com

Claudia Linnhoff-Popien
Ludwig-Maximilians-University Munich
Institute for Informatics
Munich, Bavaria, Germany
Email: linnhoff@ifi.lmu.de

***Abstract*—Cloud Computing comes along with easy and self-managed resource provisioning and releasing. However, booting and shutting down of instances still means dealing with latencies, administrative efforts and supplementary costs. Considering that, scaling of required resources needs to be well-scheduled, especially, as resources in Clouds are often highly and complexly dependent among each other. The challenge, thereby, is to manage Cloud services with less overhead while fulfilling negotiated SLAs. To automatically provide the exact amount of required instances, our approach enables the detection of resource dependencies and the automated scaling of them without the need for observing the utilization of every single instance. For that reason, we introduce a model addressing resource dependencies and a self-calibration process of the dependency graph used by a regulation method for the dynamic management of dependent resources.**

Keywords*—*Resource Management, Dependencies, Cloud.

I. Introduction

Although Cloud Computing has set new standards regarding redistribution of virtual machines [1], especially dynamic integration of physical resources, some challenges remain [2], [3]. In particular, the allocation and shut down of these resources as well as their distribution on the same hardware requires a minimum of time [4]. Since a Cloud service does not know in advance when a client plans its usage, an efficient resource planning is not fully automated until now [5] and rule-based mechanisms need to be conducted by the Cloud user [6]. However, in industrial environments, deterministic behaviour is a fundamental requirement and resource availability should be guaranteed for every service, even though they share a pool of physical hardware. Negotiated Service Level Agreements (SLAs) have to be fulfilled and high quality of service should be ensured to satisfy Cloud user interests in time. There is still an enormous need of automated and efficient reaction upon variable resource demands to reduce the amount of precautionary allocated machines, which may then be underutilized most of the time in normal operation causing unnecessary costs.

In this context, resources of Cloud service deployments often show high complex dependencies among each other, as they have a multi-layer structure [7] and avail themselves of other services on the same layer (i.e., composite services). Thereby, a service normally consists of external facing nodes (e.g., Web servers) and internal dependent resources which are required to provide the service. Additionally, resources are shared between the single services, for the purpose of an optimized service-oriented architecture. These dependencies are essential to be considered in service offerings.

Consequently, the question raises how to manage these services while causing less administrative overhead in complex Cloud environments. The goal is to offer a proactive automatic instance management of dependable resources. To achieve this, we enable the automated detection of resource capacity dependencies for supplying the requesting clients with an exact amount of resources required during operation. Then, the scaling process can be done based on the dependency model without having to observe the utilization of each instance.

The paper is organized as follows: Section II gives an overview about research on load management and dependencies in Clouds. The fundamental developed protocol for reservation and feedback based load management through a Service Load Manager (SLM) is pointed out in Section III. The construction of the treated environment and its dependency model is described in Section IV. Section V explains our approach for deducting the capacity demands for dependable resources with the self-calibration and resource regulation methods. Section VI outlines the implementation and results and Section VII concludes and demonstrates future work.

II. Related Work

Load management is important in Clouds and studied by many researchers. The ‘SigLM’ system, e.g., concentrates on exact resource allocation in shared Cloud environments through fine-grain signatures [8] and Chen et al. use patterns to forecast load (not automated by now), which means trusting historical data and predicting the future [9]. However, without considering dependencies of resources which cover emerging loads on service usage, they waste potential for cost savings.

Brandic [10] wants to support applications according to predefined schedules. While minimizing user interaction with services, she focuses on failure minimization instead of efficiency and performance – we intent to improve the latter. A resource provisioning algorithm for the generation of reservation plans and for the reduction of total provisioning costs is formulated by Chaisiri et al. [11] However, cost factors are here the driving force and again performance issues are disregarded.

Takahashi et al. [7] identify issues emerging under the multi-layered resource environment of Clouds, while concentrating on problems caused by the damage of a single resource

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-290-5

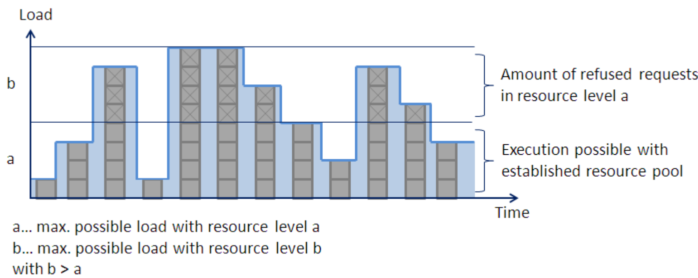

Fig. 1: Load progress of aggregated client requests without delay

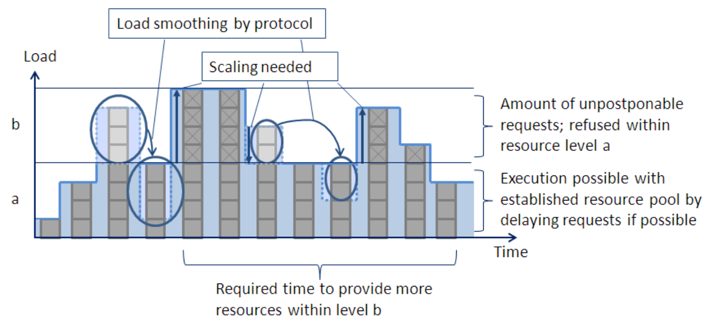

Fig. 2: Load progress with delay of client requests and scaling processes

affecting other resources, which (in-)directly use the faulty one. While creating a dependency graph, as we do, they focus on the aspect of failure tolerance when parts of the system break down. Furthermore, Takahashi et al. build on a very inflexible basis that requires an administrator to monitor all resources and their dependencies. In contrast, we implement active booting and shut down by automatic processes to save these efforts.

The SWAP system, developed by Zheng and Nieh [12], enables automatic dependency detection, too. They use system operation histories to determine resource dependencies among processes and consider them in scheduling. However, their scheduler only has fixed resource pools and scaling processes, which we are concentrating on, are not considered. Also the analyzed algorithms in [13], which are implemented to support optimal provisioning for multiple a priori known tasks, do not consider that the resource pool is changed during operation. In [6] auto-scaling scheduling is proposed, which finishes submitted jobs within specified deadlines considering costs. However, they need to constantly monitor workload changes and, again, newly submitted jobs are ignored in their test bed. Though an automated scheduling architecture managing changes in the Cloud workflow, especially in peak-load situations, is presented by [14], this work lacks the scheduling of tasks regarding their dependencies – unlike we do.

The optimization of scheduling mechanisms have been studied for decades, as in [15]–[17], etc. However, on these approaches, relatively static directed acyclic graphs [18] are assumed, which are given by administrators and elasticity and fast changing environments as common in Cloud Computing are not supported. To sum it up, load management still lacks of fully automated and highly efficient scaling processes, and more studies are needed within this research field.

III. Load Smoothing by the Protocol ReFeLoMaP

In the proposed solution for automatic scaling in Cloud environments, we base on our developed environment already described in [19] and [20], and extend this approach by the possibility to deduct capacity demands for dependable resources. In previous work, we created a technique to dynamically manage the load of Cloud services based on resource reservation and service feedback and a method to influence the behaviour of service usage by the service itself. The implemented light-weight protocol ReFeLoMaP handles the communication between the Cloud services and their clients via a service load manager, which coordinates the actual resource demands and their availability. In the following, we build on this procedure and define a self-calibration method before the process as well as a resource regulation algorithm of the SLM during active operation.

As outlined in [20], the mediating SLM initially aggregates all incoming client requests, and in this way, the load process can be depicted in a graph as shown in Figure 1. In highly distributed systems, client requests which exceed the actually provided resource capacity are normally refused in high load peak times in order to remain operational until additional resources are allocated. To avoid this ineffective manner of operation our protocol ReFeLoMaP delays specific requests about a defined time interval as agreed upon with the corresponding clients. After having delayed these client requests, one can observe less underutilized machines. Respectively, scaling processes are less frequent needed in order to cover all clients' demands. Consequently, we are enabled to smooth load peaks, which leads to better instance utilization and less frequent scaling demands, as illustrated in Figure 2.

The dependencies among every single resource need to be well known in order to be capable of deciding automatically which and how many virtual instances can be booted or shut down on these resources. More precisely, the ratio of running machines on each specific layer must be computable. That way, it is possible to scale the instances of a service with its respective dependable resources at once without having to evaluate the utilization of each resource in case of load variations and unpredicted load peaks.

IV. Construction of Service Environment and its Dependency Model

In order to get a better understanding of the typical multi-layered composite service architecture of Clouds, imagine a service environment consisting of several Web and Worker instances as well as databases for storing persistent information. Instances of the same type can be clustered to groups - called roles. Such a system may look as depicted in Figure 3 (a), wherein a shared database and a composite service are located - the latter comprising one Web role, two Worker roles and exactly that database. More abstractly, in this example, the service environment exists of five different services S_1 to S_5.

At first, we define which services are related to which others in order to build up a dependency graph for scaling processes. For that purpose, every service indicates its relationships to others on the initial registration at the SLM. The SLM

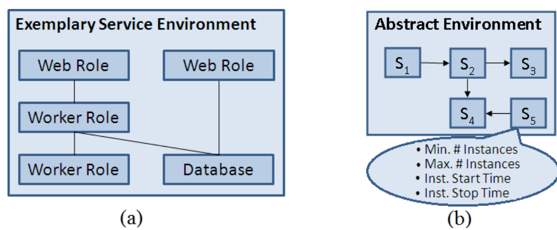

Fig. 3: Service environment (a) and its abstraction stored at the SLM (b)

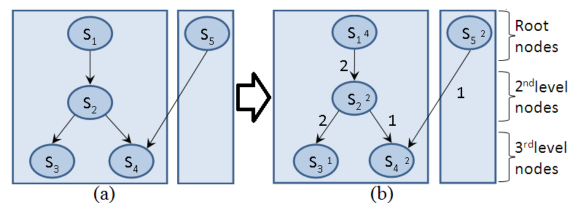

Fig. 4: DAG of the service environment at the SLM

stores these connections and sets up attributes about the passed on minimal and maximal amount of instances per each scalable resource (Min./Max. # Instances) and the time, a service needs for starting and stopping of new instances (Inst. Start/Stop Time). With reference to our example, on registration at the SLM, the Web role (represented by service S_1) indicates a dependency to the Worker role (service S_2), S_2 a relation to another Worker role (service S_3) as well as to the shared database (service S_4). Another Web role (service S_5) states a direct dependency to this database S_4, too (see Fig. 3 (b)).

In order to organize the dependencies of all listed services at the SLM, we start up from common Directed Acyclic Graphs (DAG) $G = (V, E)$, where V is a set of v_i nodes and E is a set of e_i directed edges. In [18], DAG-nodes represent tasks and the weight w of a node n_i is called the computation cost $w(n_i)$. An edge is represented by (n_i, n_j) and its weight (communication cost) is given by $c(n_i, n_j) = \frac{w(n_i)}{w(n_j)}$. Based on this, we apply the traditional DAG and modify its usage. In our implementation the nodes are the different Web, Worker or database roles registered at the SLM and their weight is the number of running instances of a role. In our example, we have a node set of $V = \{S_1, ..., S_5\}$. The weight of an edge represents the ratio between the two associated roles. If the Web Role S_1 has, e.g., four running instances and the underlying Worker Role S_2 needs two instances, their ratio of running instances is 4:2 with leads to a ratio of 2.

For the automated graph generation, shared resources must not be influenced by other related ones, because the instances can not be used simultaneously by different roles at the same time. Therefore, in the detection process, resource S_5 should be inactive when defining the graph of S_1, S_2, S_3 and S_4. Vice versa the services S_1 to S_4 must be idle on the graph creation of S_5, because, with using the same instances, the different sequence threads can not be parallelized. When defining the capacity demands for the resources, we then can guarantee always the same workload and no sum of load at each resource.

V. Deduction of Capacity Demands for Dependable Resources

Having defined the essential fundamentals of the distributed system, we got a starting point for the further solution. Our following approach consists of two procedures to automate the scaling of dependable resources in service provisioning of Cloud Computing systems. First, we implemented a self-calibration process at the SLM before productive operation. This technique for generating the dependency graph as well as setting up the initial amount of resources on each system layer is explained in Subsection V-A. Secondly, we describe the resource regulation method of the SLM during the active operation in Subsection V-B.

A. Self-calibration before Operation at the SLM

In order to be capable of scaling the whole composite service without monitoring every single instance utilization, the SLM sets up the dependencies between the resource-instances on the different layers in a DAG (see Figure 4(a)). The ratio between the instances of dependable resources can be either defined manually by an administrator or, as in our approach, automatically with boundary conditions. With this management by the generated dependency graph, organizational efforts can be minimized, as we do not have to constantly observe each resource utilization by an administrator. For automatic ratio detection of the instance amount per each dependable resource, the SLM specifies a target value for each resource-instance regarding, e.g., the average CPU or RAM utilization. For instance, it can be determined that the average CPU load must not exceed 50%. Although, in real scenarios, the server utilization in a datacenter is estimated to range only from 5% to 20% [21], [22], we are able to offer a higher utilization because of better knowledge of future resource demands based on our developed protocol ReFeLoMaP [20].

In a second step, the SLM defines a default value in the configuration file for each service and boots according to that value the amount of resources. The minimal amount of instances of service S_1 (Min. # Instances S_1) is, e.g., two virtual machines: $w_{min}(S_1) = 2$. Then the regulation process starts: a fictional load is generated on the first node of the graph (root node), which is the external facing resource of the service, and it passes, thereby, the depending load to behind nodes. As stated before, the load of shared resources is only generated at the first node in order to not falsify the measurement, and then, passed on by the first service node. So, we can exclude accumulated load values and guarantee reproducible test scores. This implies that other services are not active during the automated detection of the amount of needed instances for achieving the target value.

Afterwards, the SLM allocates instances on the according resource (role) until the previously set target value is reached and stores the corresponding amount of running instances ($w_{act}(n_i)$) in the nodes as its weight ($w(n_i)$). Analogous to that, the SLM proceeds for all depending resources on lower layers and adjusts the instance amount for each of them. In our implemented test scenario (see section VI), the Web role S_1 holds four instances and the underlying Worker role S_2 needs two instances. This Worker role requires one active instance at the lower Worker role S_3 and two running machines at the database service S_4 in order to achieve the corresponding target

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-290-5

values regarding CPU utilization and storage occupancy. In a second iteration, the required amount of instances in Web role S_5 can be set on two with an according process. By means of the list of the instance amount of every single layer, the role ratios can be defined. In our case, Web role S_1 holds four instances ($w(S_1) = 4$) and the underlying Worker role required two ($w(S_2) = 2$). Consequently, their ratio is 2, which is also the weight of the corresponding edge:

$$c(S_1, S_2) = \frac{w(S_1)}{w(S_2)} = \frac{4}{2} = 2 \quad (1)$$

This Worker role has a weight of 2, similar to the other Worker role S_3 ($c(S_2, S_3) = \frac{2}{1} = 2$) and a ratio of 1 to the database ($c(S_2, S_4) = \frac{2}{2} = 1$). The ratio of the Web role S_5 to the shared database S_4 is 1, too ($c(S_5, S_4) = \frac{2}{2} = 1$). As shown in Figure 4(b), the ratio and the amount of instances are inscribed at the edges and in the nodes of the dependency graph. Following this, the basic dependency graph is completed with concrete values by the SLM after the registration period.

After this completion, the regulation process is stopped and the required instances per each resource can be scaled during operation based on the dependency graph. The self-calibration procedure can be triggered periodically in order to correct the specified dependency values as appropriate.

B. Resource Regulation Process of the SLM

With this dependency graph, the SLM is able to regulate the resource amount during operation, described as follows. First, all composite services (meaning all depending resources) are reduced on one virtual resource. As a result, the corresponding limits regarding the minimum and maximum of instance amount and their booting and shut down time are well known for all composite services. In the case, there are shared resources (as database S_4), the maximum aggregated limit between the dependable resources is additionally defined.

In every single regulation step, the SLM:

1) accumulates all of the client requests on each virtual provided service, then,
2) controls the limit checks on basis of the external facing node accessed by the client and
3) checks the aggregated limits of the dependable services in case of shared resources.

The SLM modulates the role instance counts according to the relations between the roles and their defined weights of nodes and edges, as long as the aggregated instance counts stay within defined limits. Thereby, the SLM does not deceed the minimal amount of resource-instances predefined by the services themselves before their actual execution – nor exceed their corresponding maximum. For instance, we define a minimal amount of least two and maximal ten instances at resource S_1. Then S_1 will always run with at least two instances, even though, one would be sufficient to cover the actual load. That way, it is possible to react on unexpected load peaks if needed. If anon in S_2 we have at least one and maximal four instances, the maximum of four instances in S_2 is not extended, even though S_1 is scaled up to five and higher instance amounts.

In the case, aggregated limits of shared resources would be exceeded, a load release occurs according to the determined

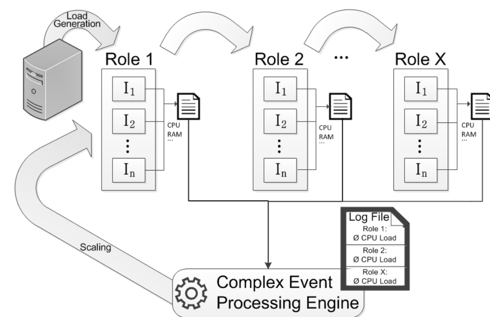

Fig. 5: Implementation of the Automated Scaling Process

service priority defined by the protocol ReFeLoMaP. On equal priority the sharing is defrayed in equal proportions. Thereby, we can additionally focus on maximization of throughput, meaning the service delays specific client requests about a defined interval. This is possible even then the client is running in a higher priority class of the service level agreement as long as the client allows its procrastination by the manipulation of the service load manger. Thereby, also lower prioritized client requests need not to be dropped. For further information about the concrete implementation and the benefits of the protocol ReFeLoMaP, see our paper [19].

If no aggregated limits are exceeded but particular service limits of virtual external facing nodes, the client load management feedback is generated on basis of this maximum load in order to distribute the load in a way that load dropping through request refusing can be avoided.

VI. Implementation and Results

Having defined the dependency graph and regulation method, the SLM is now able to regard dependable resources during the instance management process. Figure 5 provides an implementation overview of this control algorithm. As shown, an external server, which runs the SLM, generates a fictional load on $Role\ 1$. $Role\ 1$ now boots new instances $(I_1, ..., I_n)$ until the target value of 50% CPU and RAM workload is achieved and stores the number of running instances with their average utilization in an internal log file. The load is passed on to all subsequent roles ($Role\ 2, ..., Role\ X$) and they proceed accordingly. After having completed the log files, they are transferred to a complex event processing engine, which analyses the data streams from the resources about the happening events, consolidates all values and passes it on to the SLM. The SLM integrates this data in the dependency graph and starts scaling corresponding to this. If an instance brakes down, it is replaced by a self-healing routine of the roles.

In a first simulation for testing the time constraints about the generation of a corresponding dependency graph, we could make the following observations. We successively simulated the registration of 5, 10, 20, 30 and 50 services at the SLM. All of them indicated random dependencies to each other. As shown in Figure 6, we took the measurements after which time the complex event processing engine announces the completion of the dependency graph generation. With an amount of 5 services it took 3 seconds until all dependencies were stored in the graph with their corresponding weights regarding instance

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-290-5

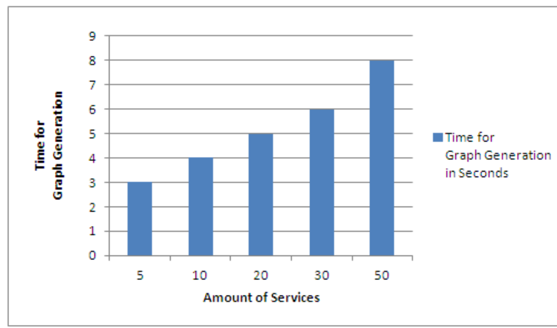

Fig. 6: Simulation of the Graph Generation Process

amount and ratios to each other. With 10 services, the time-demand linearly rises to 4 seconds, and with 20 services, to 5 seconds. On the registration of 30 services at the SLM, we measured 6 seconds and with 50 services we had 8 seconds for the graph creation. On the first glance, these values increase more and more. However, we assume that after the initial graph generation the time demand for dependency detection even with more services will not increase too extensively as the values do only rise sparsely. Furthermore, since we are concentrating on services with no real-time requirements, we can hazard these consequences and accept this delay to benefit from the more efficient service provisioning as a whole.

In a further realized scenario, we simulated an rising load requirement at the external facing node of the clients (i.e., Web service S_1), which results in a new demand of two additional instances on the resource of S_1 ($w_{act}(S_1) = 6$). Consequently, an upregulation of the related services S_2, S_3 and S_4 (Worker roles and database service) is needed, too. Following the regulation by the dependency graph, the SLM initiates the booting of an additional instance in the Worker role S_2 on the second layer, because the initial defined weight of their relation is 2 (see Formula 1) and the weight with the additional two instances on service S_1 is now 3:

$$c_{act}(S_1, S_2) = \frac{w_{act}(S_1)}{w_{act}(S_2)} = \frac{6}{2} = 3 \neq 2 = c(S_1, S_2) \quad (2)$$

That means the actual running virtual machines of the Worker role S_2 need support by a further resource-instance in order to achieve the given weight of 2 ($w(S_2) = 3$).

Since the load is passed on to the lower lying database service S_4 also in this resource a new instance is started in order to fulfill the determined weights of the dependency graph ($w(S_4) = 3$). Although the load is also transferred to the Worker role of layer three (S_3), here no additional instances are required: with three instances in S_2 and one already running instance in S_3 the ratio is yet sufficient because its weight is smaller then four (see Formula 3), on which an up-scaling of the resource-instances would be necessary ($w_{act}(S_3) = w(S_3) = 1$).

$$c_{act}(S_2, S_3) = \frac{w_{act}(S_2)}{w_{act}(S_3)} = \frac{3}{1} = 3 < 4 \quad (3)$$

With a second scenario, we covered the case that S_5 is the most stressed service in the system. The high load generated on service S_5 leads to an up-scaling of the database service S_4 through S_5 and S_1 according to the process described before. After a specific time, the internal maximal limit of instances is reached. As a consequence, the maximal amount of instances in the back-end reduces also directly the maximal instance amount of the front-end, and additional instances in S_1 as well as in S_5 cannot be supported by the fully exhausted resource pool of S_4. This situation is recognized by the complex event processing engine with the aggregated information of the log files from each resource and a corresponding warning is send to the SLM in order to take countermeasures. The SLM is now able to interrupt on its higher management level and can counteract an inefficient service provisioning which only supports one part of the composite service while neglecting the majority of the entire resource environment. For that reason, it throttles service S_5 in order to keep the whole composite service functional and reliable.

So, a predictive detection of resource bottlenecks can be conducted. With the aid of the defined resource limit values, the SLM is able to proactively respond to congestions by delaying client requests in times of overload. This prescient bottleneck identification is also possible with shared resources by using the computation of the accumulated resource loads of the composite service. On this basis the SLM can decide which service is postponed according to the specified service priority when exceeding the limit of the shared resource.

By means of the anticipatory recognition of the composite resource limitations caused by dependencies, it is possible to make an assumption of the limits for the virtual external facing node directly accessed by the client. In our example, at its registration, the external facing node S_1 in the root level states that it needs at least two and maximal ten instances ($w_{min}(S_1) = 2$ and $w_{max}(S_1) = 10$). This resource S_2 has anon at least one and maximal four instances ($w_{min}(S_2) = 1$ and $w_{max}(S_2) = 4$). As calculated before, the dependency ratio of S_1 to the hidden resource S_2 is 6:3 (proved by Formula 2). So, with the defined dependency ratio $c(S_1, S_2) = 2$ (see Formula 1), S_1 is allowed to double the instance amount of S_2 as long as it does not exceed its own maximum. Following this reasoning, we can conduct these minimum evaluations:

$$min\{w_{min}(S_1), [2 * w_{min}(S_2)]\} = min\{2, [2 * 1]\} = 2 \quad (4)$$

$$min\{w_{max}(S_1), [2 * w_{max}(S_2)]\} = min\{10, [2 * 4]\} = 8 \quad (5)$$

Consequently, S_1 is restricted in its instance amount by its dependency to service S_2 which has less capacity. So, the virtual resource S_1 has the actual instance limit range of minimal two (see Formula 4) and maximal eight running instances (see Formula 5). Thereby, it is recognizable that an utilization of the technically possible amount of ten instances in S_1 would not create added value.

Another resulting benefit of our approach is the possibility to make estimations, until when the additional capacity is available at the highest level on booting new instances. How long the entire starting time interval is, can be calculated already before based on a maximum evaluation along the dependency graph. In our scenario, booting new Web role instances in S_1 and S_5 needs a time interval of three minutes. An extra Worker role instance in S_2 and S_3 is available after five minutes each and the start of an additional database instance takes nine minutes for installing the base image and the particular software of the tenants. Consequently, in our scenario, it requires a start time consideration of nine minutes for the up-scaling of the whole service, which is the maximum

Copyright (c) IARIA, 2013. ISBN: 978-1-61208-290-5

of these parallel booting instances. Thereby, we are able to give estimations about the time point when the required instances are available earliest and the entire composite service is highly functional, again. On the other hand, it is calculable how long the release of redundant instances takes until unnecessary expense decreases.

In summary, we could observe an enhanced pro-active reaction of system resources upon proactively derived load demands. This is achieved by deriving load for dependable hidden resources from front-end resources instead of following an approach where roles manage their instance counts on local utilization monitoring. Hidden resources can be adjusted directly based on the metrics of the front-end nodes (dependency root nodes) and the dependency graph. This results in a significant improvement of the overall system load reaction.

VII. Conclusion

Load management is a driving force for comprehensive research in the area of Cloud Computing. Although this field already has its roots in basic Utility Computing and ranges from High Performance Computing (HPC) over Grid Computing to the today's era of Cloud Computing, there are still unsolved problems showing space for improvements regarding cost savings and resource efficiency. In Cloud Computing environments, providers offer a theoretically unlimited pool of resources, which Cloud services can benefit from. Thereby, a lot of heterogeneous workloads emerge and one has to react on enormous load variations in time in order to comply with the SLAs concluded with the Cloud clients. For this reason, Cloud systems provide elasticity meaning an easy booting and release of instances while relaxing strong SLAs.

On closer examination, in Clouds sophisticated and intricate dependencies among the involved resources can be observed while offering specific services both on its one and in its entirety as a composition. This is because of the multi-layer structure of Cloud services and leads to considerable efforts regarding administration, costs and time. Within this approach, we offer a management of Cloud services with less administrative overhead within complex Cloud environments. Thereby, we want to offer a proactive automatic instance management of dependable resources. For this purpose, we detect resource dependencies automatically to supply requesting clients the exact amount of required resources. The corresponding scaling process is conducted on basis of the dependency model without the need for observing each instance utilization.

While building on the previously developed protocol for reservation and feedback based load management through a SLM, we constructed a Cloud service environment and described its dependency model. Our approach deducts the capacity demands for dependable resources, while introducing a self-calibration mechanism before the standard operation and a resource regulation procedure by the SLM during operation. We proof the concept by an implementation and simulation. In future, we aim to expand the described load management to offer a functional entity for adding and releasing instances while guaranteeing low costs and SLA compliance by reliable service provisioning. Furthermore, we will prove our approach by additional evaluation with a practical use in Cloud Computing environments. Afterwards, we will compare our results with cloud scaling algorithms based on utilization values.

References

[1] M. Assunção, A. Di Costanzo, and R. Buyya, "Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters," in *18th ACM Int'l Symposium on High Performance Distributed Computing*, 2009, pp. 141–150.

[2] M. Armbrust and et al., "Above the clouds: A berkeley view of cloud computing," University Berkley, USA, pp. 141–150, 2009.

[3] M. Armbrust, et al., "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, 2010.

[4] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," in *5th Int'l Conference on Cloud Computing*. IEEE, 2012, pp. 423–430.

[5] K. Alam, E. Keresteci, B. Nene, and T. Swanson, "Multi-tenant applications on windows azure: Dokumentation," http://cloudninja.codeplex.com/releases/view/65798, 2011, last access 20.6.2013.

[6] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2011, pp. 1–12.

[7] T. Takahashi, Y. Kadobayashi, and H. Fujiwara, "Ontological approach toward cybersecurity in cloud computing," in *3rd Int'l Conference on Security of Information and Networks*. ACM, 2010, pp. 100–109.

[8] Z. Gong, P. Ramaswamy, X. Gu, and X. Ma, "Siglm: Signature-driven load management for cloud computing infrastructures," in *17th Int'l Workshop on Quality of Service*. IEEE, 2009, pp. 1–9.

[9] J. Chen, W. Li, A. Lau, J. Cao, and K. Wang, "Automated load curve data cleansing in power systems," *Transactions on Smart Grid*, vol. 1, no. 2, pp. 213–221, 2010.

[10] I. Brandic, "Towards self-manageable cloud services," in *33rd Software and Applications Conference*, vol. 2. IEEE, 2009, pp. 128–133.

[11] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *Transactions on Services Computing*, vol. 5, no. 2, pp. 164–177, 2012.

[12] H. Zheng and J. Nieh, "Swap: A scheduler with automatic process dependency detection," in *1st Symposium on Networked Systems Design and Implementation*. USENIX Association, 2004, pp. 145–158.

[13] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," in *Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2012, pp. 1–11.

[14] T. Dornemann, E. Juhnke, and B. Freisleben, "On-demand resource provisioning for BPEL workflows using amazon's elastic compute cloud," in *9th Inte'l Symposium on Cluster Computing and the Grid*. IEEE, 2009, pp. 140–147.

[15] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *3rd Symposium on Operating Systems Design and Implementation*. USENIX Association, 1999, pp. 145–158.

[16] F. Dong and S. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School of Computing, Queen's University, Kingston, Ontario, Tech. Rep., 2006.

[17] M. Aggarwal, R. Kent, and A. Ngom, "Genetic algorithm based scheduler for computational grids," in *19th Int'l Symposium on High Performance Computing Systems and Applications*, 2005, pp. 209 – 215.

[18] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[19] A. Schwanengel and G. Kaefer, "Light-weight load management protocol based on reservation and feedback loops," in *23rd Int'l Conference on Parallel and Distributed Computing and Systems*. ActaPress, 2011, pp. 165–172.

[20] A. Schwanengel, G. Kaefer, and C. Linnhoff-Popien, "Improved throughput and response times by a light-weight load management protocol," *Journal of Parallel and Cloud Computing*, vol. 1, pp. 1–9, 2012.

[21] K. Ragan, "The cloud wars: $100+ billion at stake," Tech. Report, Merrill Lynch, 2008.

[22] L. Siegele, "Let it rise: A special report on corporate it," *The Economist*, vol. 389, pp. 3–14, 2008.