Utility Driven Elastic Services *

Pablo Chacin and Leando Navarro

Departament d'Arquitectura dels Computadors, Universitat Politénica de Catalunya, Barcelona, Spain,
pchacin@ac.upc.edu

Abstract. To address the requirements of scalability it has become a common practice to deploy large scale services over infrastructures of non-dedicated servers, multiplexing instances of multiple services at a fine grained level. This tendency has recently been popularized thanks to the utilization of virtualization technologies. As these infrastructures become more complex, large, heterogeneous ad distributed, a manual allocation of resources becomes unfeasible and some form of self-management is required. However, traditional closed loop control mechanisms seems unsuitable for this platforms.

The main contribution of this paper is the proposal of an Elastic Utility Driven Overlay Network (eUDON) for dynamically scaling the number of instances of a service to ensure a target QoS objective in highly dynamic large-scale infrastructures of non-dedicated servers. This overlay combines an application provided utility function to express the service's QoS, with an epidemic protocol for state information dissemination, and simple local decisions on each instance to adapt to changes in the execution conditions. These elements give the overlay robustness, flexibility, scalability and a low overhead.

We show, by means of simulation experiments, that the proposed mechanisms can adapt to a diverse range of situations like flash crowds and massive failures, while maintaining the QoS objectives of the service.

Keywords: Web Services; QoS; Epidemic Algorithm; Overlay; Self-adaptive

1 Introduction

Modern large scale service-oriented applications frequently address unexpected situations that demand a rapid adaptation of the allocated resources, like flash crowds –that require a quick allocation of additional resources– or massive hardware failures –that require the re-allocation of failed resources. At the same time, applications are expected to maintain certain QoS objectives in terms of attributes like response time and execution cost [14].

To address these requirements, it has become a common practice to deploy services over large scale non-dedicated infrastructures – e.g. shared clusters –

* This work has been partly supported by Spanish MEC grant TIN2010-20140-C03-01.

on which servers are dynamically provisioned/decommissioned to services in response to workload variations. In comparison, in a traditional enterprise infrastructure the scale up process takes a long time and requires manual intervention, and therefore over-provisioning services to handle such situations is the common practice.

Chandra et al. [7] demonstrated that fine-grained multiplexing at short time-scales – in the order of seconds to a few minutes – combined with fractional server allocation leads to substantial multiplexing gains over coarse-grained reallocations. To accomplish this fine grained multiplexing, it is necessary to count with mechanisms to allocate/deallocate servers efficiently and then be able to manage those servers in a very dynamic environment with a high turn-over of servers.

As these infrastructures become more complex, large, distributed and heterogeneous, some sort of self-adaptive [18] (also known as autonomic [12]) capabilities are needed to stabilize its performance within acceptable limits despite variations in the load and resources, recover from failures, and optimize them according to business objectives.

However, as noted in [15] traditional closed loop self-adaptation approaches are of limited applicability in the scenarios described above, as they made a set of restrictive assumptions: a) The entire state of the application and the resources are known/visible to the management component b) the adaptation ordered by the management component is carried out in full and in a synchronized way, and c) the management component gets full feedback of the results of changes made on the entire system. In contrast, in a large scale, wide-area system getting a global system knowledge is infeasible and coordinating adaptation actions is costly. Additionally, each server may belong to different management domains – different sites in an organization, external providers – with different optimization objectives. Two additional problems arise for the self-management of non-dedicated servers: the complexity of eliciting a model to predict the effect of the adaptation decisions and drive the adaptation process, and the need for an isolation mechanism to prevent servers to interfere with each other's performance.

The main contribution of this paper is the proposal of an Elastic Utility Driven Overlay Network (*e*UDON) for dynamically scaling the number of service instances used to support a service, ensuring a target QoS objective in highly dynamic large scale shared infrastructures.

Based on UDON [5], *e*UDON combines a) an application provided utility function to express the service's QoS in a compact way; b) an epidemic protocol for scalable, resilient and low overhead state information dissemination; c) a model-less adaptive admission policy on each service instance to ensure a QoS objective; and d) mechanisms for the elastic assignment of instances to adapt to fluctuations in the load and recover from failures. All these mechanisms act autonomously on each instance based on local information, making the systems highly scalable and resilient.

We show by means of simulations experiments how eUDON adapts to diverse conditions like peaks in the workload and massive failures, maintaining its QoS and using efficiently the available resources.

The rest of the paper is organized as follows. Section 2 presents the general model for eUDON and describes in detail the two main mechanism used to achieve elasticity. Section 3 describes the simulation based experimental evaluation that explores its behavior under diverse scenarios. Section 4 presents relevant work in the field to put the proposed work into context. Finally, section 5 presents the conclusions and outlines the future work.

2 *e*UDON: an Elastic Utility Driven Overlay Network

The model for eUDON which is shown in Fig 1.

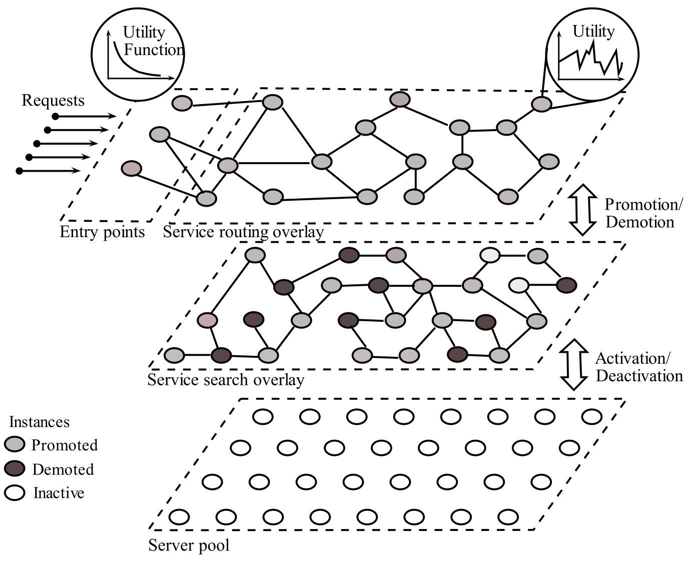


Fig. 1: Elastic service overlay model.

Requests coming from users are processed through a set of entry-points, which correspond to segments of users with similar QoS requirements, and must be routed to service instances that offer an adequate QoS. Requests are evenly distributed over the multiple entry points for the same user segment using traditional DNS or L4 level load balancing techniques [4]. It is important to notice that in our work we concentrate in the web application layer and assume that the data access, including consistency requirements, are handle by a separated data layer as proposed in modern highly scalable web architectures [13].

Each service has a utility function that maps the attributes and execution state of a service instance (e.g. response time, available resources, trustworthiness, reliability, execution cost) to a scalar value that represents the QoS it provides. Utility functions allows a compact representation of the QoS requirements for services and facilitate the comparison of the QoS that different instances provide, even if they run on very heterogeneous nodes [11]. The QoS required by a request is defined as the minimum acceptable utility that a service instance must provide to process it. The QoS offered by an instance may vary over time due to, for example, fluctuations on the load or the available resources [1] of the non-dedicated server it runs on.

There is large pool of servers available for diverse services. At any given time, on a subset of those servers there are instances activated to process requests for a service. The active instances are organized in a **Service Search Overlay**, whose objective is to facilitate finding an instance offering adequate QoS. Among the active instances, a subset capable of processing the current workload is **promoted** – as described later in section 2.2 – to join the **Service Routing Overlay**. This overlay is used by the entry points to route requests preserving QoS and achieving load balancing. Instances which are underutilized, are **demoted** and leave the routing overlay but remain in the search overlay. Eventually, instances can be deactivated from the search overlay.

The number of active instances in the search overlay depends on the expected service demand and the level of replication required to ensure resilience to failures as well as to handle short time increases in the demand. To estimate this number approaches like those proposed in [17] [9] can be applied. In *e*UDON, this problem is part of our ongoing research and diverse options are being explored. In the rest of this paper we assume the number of active instances is fixed even when the proposed approach can accommodate variations in this number. The main problem we address is how to maintain to a minimum the number of instances in the routing overlay to keep the number of hops needed to process requests low while still offering an adequate QoS.

It is important to notice that in shared clusters, the active instances which are not promoted for processing requests add little overhead to the cluster. In a cloud scenario, it makes sense to have instances activated for some time even if idle, because of the activation overhead and because computing resources are paid by hours – and therefore a 15 minutes activation cost the same than a full hour activation.

The routing and search overlays use a push style epidemic algorithm to maintain their topologies, find new (activated, promoted) instances, remove unavailable (failed, demoted, deactivated) instances, and spread the current state of instances. Periodically each instance selects a subset of its neighbors (the exchange set) and sends a message with its local view (the neighbor set) and its own current state. When a instance receives this message, merges it with its current neighbor set and selects the subset with the more recently updated entries

[1] For example, modern virtualization technologies make feasible to change resource assignments to execution environments on the fly.

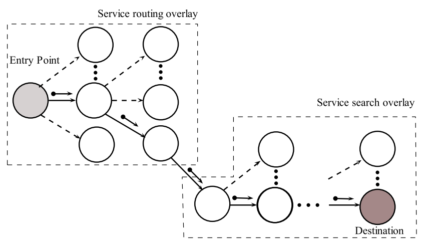
as the new neighbor set. In this way, each instance keeps a local view with the most up date information among all the information received from neighbors.

Next sections describe in detail the two main elements of the model outlined before that provides the attributes of self-adaptiveness and elasticity to *e*UDON.

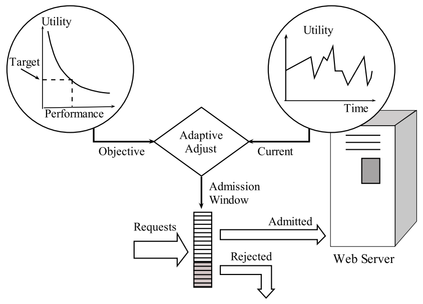
2.1 Routing

The objective of the routing mechanism is to deliver each request to a service instance that satisfies its QoS requirements, with high probability and the minimal amount of routing hops. In *e*UDON Requests are routed using the unicast algorithm shown in Fig. 2a. On the reception of a request, the routing algorithm uses an *admission* function to determine if the instance can accept the request for processing. If the request is not accepted, then a *ranking* function is used to rank the neighbors and select the most promising one as the next hop in the routing process.

*e*UDON uses an adaptive admission function – inspired by the one proposed in the Quorum system [2] – summarized in Fig. 2b. The utility of the service instance is periodically monitored and compared with a target QoS objective and the size of the admission window is increased or decreased as needed to correct deviations. The only assumption made by this process is that the utility is non-increasing with respect of the load. That is, that increasing/decreasing the load lowers/rises the utility, given that the rest of the utility related attributes remains equal. One significant advantage of this adaptive admission function is that it does not require any model to estimate the future performance of the service. Moreover, it works even when the resources allocated to a service cannot be reserved and therefore the available capacity fluctuates.



(a) Routing process



(b) Adaptive admission.

Fig. 2: Adaptive routing.

Every overlay uses a different ranking function. The routing overlay uses a round-robin ranking which has been shown to offer an acceptable performance in this context – see [6] for details. When a request is routed beyond a predefined number of hops, it is routed using the search overlay, looking for an active (but not promoted) instance capable of serving it. This search uses a greedy ranking based on the (last known) utility of the nodes which was shown in [5] to be highly effective in a wide range of conditions and scenarios.

2.2 Promotion and demotion

The decision to join/leave the routing overlay is taken by each instance autonomously based both on its local information, like the rate of request being processed or the server's utilization, and aggregate non-local (potentially global) information like the total workload of the system or the average service rate of other instances.

*e*DUON uses a probabilistic adaptation mechanism implemented by two rules based on the service rate (the number of service requests processed by time interval). An instance promotes itself if its service rate is close to the average of the system and demotes itself if it is offering a service rate below the 25 percentile of all instances. These rules where chosen for their simplicity and because they could be easily traceable to the system's status, more than for their optimality. However, they exhibit a very acceptable behavior as shown in the experimental results. The probabilistic nature of the rules leads to a progressive adaptation preventing that many instances take simultaneously the same decision, overreacting to a situation and leading to oscillations in the system.

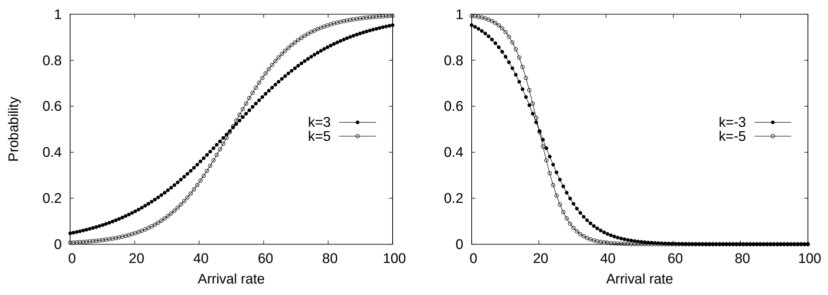
An estimate of the global service rate can be obtained by an epidemic aggregation process embedded into the overlay maintenance algorithms [10] [8]. In the simulation described in section 3.1, each instance gets an estimated of these values perturbed by an error factor to simulate the estimation error of the distributed aggregation algorithms.

The probability for promotion/demoting an instance is given by:

$$P(S_\epsilon) = \frac{1}{1 + e^{kS_\epsilon}}, \tag{1}$$

Where $S_\epsilon = (\bar{S} - S)/\bar{S}$ is the deviation of the node's service rate S from the target service rate $\bar{S}$, and k is a parameter that adjust the sensitivity of probability to the service rate. When calculating the probability for demoting, $k > 0$ and for promoting $k < 0$. Fig. 3 shows this probability for various values of k, and and $\bar{S} = 50$ for promotions and $\bar{S} = 20$ for demotions.

As the promotion and demotion rules behaves independently of each other, we have added an additional condition to prevent an instance to be continuously promoted/demoted without having the change to stabilize: instances will not run these rules again for the 3 cycles following a promotion/demotion. This number was empirically obtained by testing multiple options and found to work well in different situations.



(a) Promotion probability. (b) Demotion probability.

Fig. 3: Promotion/Demotion probability function for diverse values of k.

3 Experimental Evaluation

In this section we describe the simulation model and the metrics used for the evaluation, and summarize the results of different experiments developed to analyze how the system adapts to different scenarios.

The results presented correspond, unless the contrary is explicitly indicated, to the average over 10 simulation runs. Each run simulates 200 seconds (300 for the peak load scenario).

3.1 Simulation model

We implemented a discrete event simulator for the detailed simulation of the processing of requests allowing us to capture very detailed measurements of the system's behavior. Table 1 summarizes the more relevant simulation parameters.

Overlay. We have simulated an idealized network that mimics a large cluster, on which nodes can communicate with a uniform delay. The base experimental setup was 128 overlay nodes, with a 8 entry points and the 120 service instances (a 1:15 ratio). There is an ongoing work – with promising preliminary results for up to 2048 nodes– to experiment with several thousand instances. However, as the instances work exclusively with local information, we expect that the results will hold for larger scales, as was previously shown in [5].

All Service instances are initially members of the search overlay, but only a fraction initially join the routing overlay according to a join probability parameter. The adaptation process dynamically adjust this fraction accordingly to the conditions (e.g. load).

Each instance maintains a neighbor set of 32 nodes and contacts 2 of them periodically to exchange information. These values correspond, to the optimal trade off between information freshness and communication costs as discussed in [5].

Table 1: Simulation parameters.

Parameter	Values	Description
Servers	**128**, ... 2048	Number of instances
Entries ratio	1:15	Ratio of entry points, with respect of the number of instances
Neighbor set	32	Number of neighbors maintained per node in the overlay
Update cycle	1	Frequency of information dissemination (in seconds)
Exchange set	2	Number of neighbors contacted per node on each update cycle
Adaptation cycle	3	Frequency of the adaptation process (in seconds)
Join probability	.60	Probability of an service instances to join the routing overlay at initialization
Load Maximum	.5	Maximum fraction of a server capacity used by background load
Load variability	0.10	Maximum variation of background load per second
QoS	0.7	Target utility for requests
K	-3 (promotion) 3 (demotion)	Adaptation probability adjust constant
$\bar{S}$	50 (promotion) 20 (demotion)	Target service rate for promotion/demotion

Service Instances. Each service instance dispatches requests using a processor sharing discipline. This model fits well for web servers like Apache, a well-known and widely used multi-threaded web server, and is amenable to analytical evaluation using a $M/G/1/k*PS$ queuing system [3]. This facilitates the comparison of simulation results with analytical estimates. For instance, the maximum arrival rate that can be processed by the system maintaining a target response time – as explained below – was estimated using this model.

Arrivals. The service requests arrive following a Poisson distribution and are evenly distributed among the entry points. The arrival rate is calculated using the analytical model for service instances considering the average background load of servers to ensure that the allocation of the workload is feasible, but demands all the available capacity. Therefore, the maximum theoretical allocated demand is of 1.0 and the expected utilization is around 0.95. All requests generated in the tests have the same expected QoS.

Utility Function. In our experiments we use a utility function that relates the utility to the deviation of the response time RT from a target maximum response time RT_0:

$$U(RT) = \left[\frac{RT_0 - RT}{RT_0}\right]^{\alpha} \tag{2}$$

As shown in Fig. 4 the coefficient α controls how quickly the utility decreases as the response time approaches the maximum. This function was selected because it can easily be related to metrics obtained from both the analytical and simulation models, making it straightforward to predict and measure the impact of the adaptation decisions in the resulting utility. The evaluation of more complex utility functions is subject of future work. However, it is important to stress that the adaptation process considers only the utility function and makes no explicit reference to the underlying response time. This allows us to generalize the results to other utility functions given that they satisfy the basic assumption of being non-increasing with the load of the system, as discussed in section 2.1.

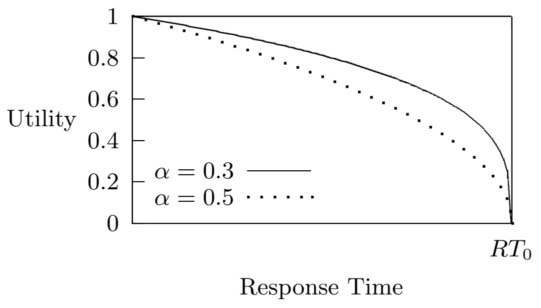


Fig. 4: Utility Function.

Background Workload One important aspect in our experiments is the evaluation of the impact of background load in non-dedicated servers, which impacts the utility that an instance can provide. This load (defined as a faction of the node's computing power) is modeled as a random variable whose value varies along the time following a random walk with certain *Variability.* This model is consistent with the observed behavior of the load on shared servers [16, 20].

3.2 Metrics

In the evaluation of the proposed mechanisms we have considered the following metrics, which are related to its main objectives of efficiently delivering requests to an appropriate node to maintain an adequate QoS:

Allocated Demand: measures the fraction of the demand that is actually allocated to a server, before being dropped due to the expiration of its TTL (set to 8 hops in our experiments). This metric measures how effective is the systems in allocating requests.

QoS Ratio: is the ratio between the target QoS expected by a service request and the actual QoS received. A ratio below 1.0 means target was not met, while a ratio over 1.0 means the target was exceeded (which is not necessarily desirable, as it may indicate the server is underutilized).

Utilization: Measures the percentage of the node capacity being used, considering both the background load and the load produced by the service requests. This metric is relevant as measures how efficiently resources are used.

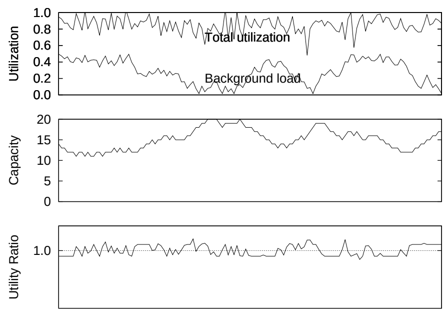
Routing hops: measures the number of hops (or retries) needed to allocate a request to a server with an adequate utility. It measures how efficient is the mechanism in allocating requests.

In the graphics of experimental results in section 3.3 the percentiles 25, 50 and 75 of the QoS Ratio, Utilization and Routing Hops are presented to show the variability of these metrics. Percentiles 25 and 75 are drawn as a filled curve and percentile 50 as a continuous line.

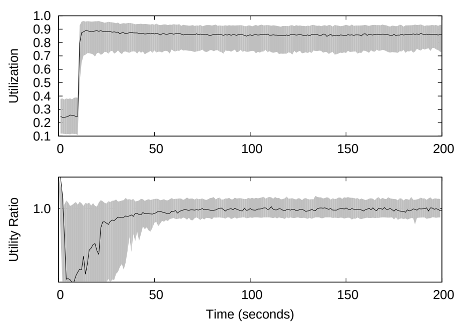
3.3 Results

In this section we describe the different experiments we made to test the behavior of the system under diverse conditions and usage scenarios.

Base scenario. In the base scenario, a steady workload is submitted to the system that demands all the available capacity to achieve the QoS objective.



(a) Evolution of an Instance.

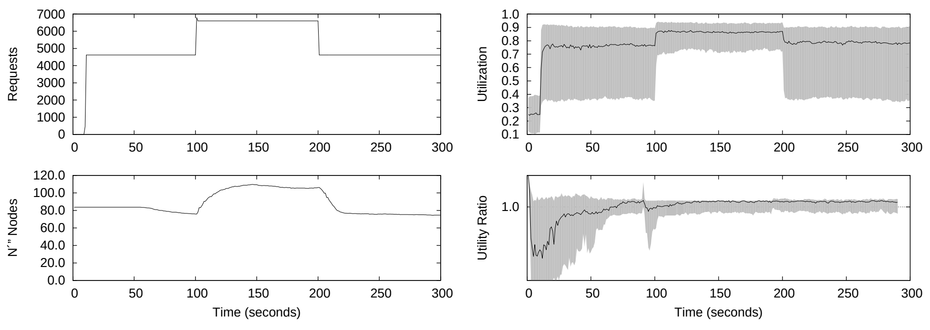


(b) Utilization and QoS Ratio.

Fig. 5: Behavior for base scenario.

Fig. 5a shows how the utility driven adaptation occurs for an instance. As the background load fluctuates so does the capacity of the instance – following the adaptive admission described in section 2.1 – to compensate the change in the available CPU and maintain the QoS ratio close to 1.0. With respect of the overall behavior of the system, as shown in Fig. 5b it quickly converges to a utility ratio of 1.0, with a small variability. The adaptation process also achieves a high level of system efficiency, with a total utilization of system capacity around *0.9*. Additionally, 90% of the maximum theoretical workload is allocated – this figure is maintained in all scenarios – and a 75% of request needs at most 3 hops to be allocated (graphics for these results not shown for brevity). These results show that the system is effective in achieving the QoS goals, efficient in the utilization of resources and imposes a low overhead in terms of the number of hops needed for allocation.

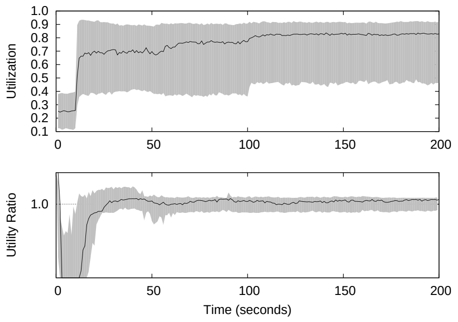
Peak load scenario. In this scenario, the system is initially submitted to a steady workload that demands 70% of the the available capacity, but at time 100s, an additional load is injected. As can be seen in Fig. 6, the systems quickly reacts to the load by promoting more instances. The overall utilization of the system also is increased - the percentile 25 of the Utilization rises significantly – but the QoS Ratio is maintained during this adaptation process. At time 200s, the additional load is removed and the systems returns to the previous state, demoting the instances not longer needed.



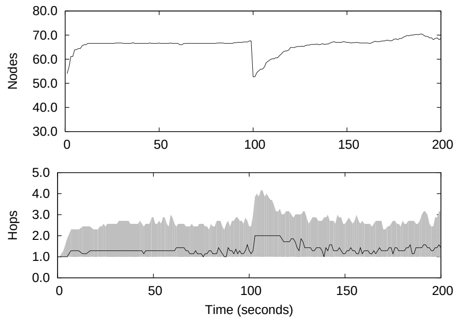
(a) Injected load and number of instances (b) Utilization and Utility Ratio.

Fig. 6: Behavior for peak load scenario.

Failure scenario. In this scenario, the system is submitted to a steady workload that demands a fraction of its capacity. At time 100s, 20% of the promoted instances fail – a correlated failure as expected in clusters. Fig. 7 shows how the system reacts, incorporating more instances until the system stabilizes. As can be seen, the utility ratio is maintained along this process – except for a short period just after the failure – as requests are routed to nodes in the search overlay; as a consequence, routing hops increase until all the required nodes are promoted to the routing overlay.



(a) Utilization and Utility Ratio.



(b) Number of instances and Number of Hops.

Fig. 7: Behavior for failure scenario.

4 Related Work

The elasticity in the allocation of resources to web applications has attracted significant attention from different perspectives.

In [9] the dynamic placing of the instances of multiple applications on a set of server machines is formulated as a two dimensional packaging problem and several heuristics are proposed to optimize the solution by minimizing the number of placement changes while maximizing the balancing of the load across nodes. However solving this problem has a high computational complexity, severely limiting its scalability.

VioCluster [17] uses both machine and network virtualization techniques which allow a domain in a shared cluster to dynamically grow/shrink based on resource demand by negotiating the borrowing/lending of virtual machines with other domains. This approach has, however, a significant overhead for both the negotiation process and the need to create and start machines.

In [19] a utility related performance metric is used by a request scheduler to order the processing of requests from multiple services classes, so that the resulting aggregate utility is maximized. The main drawback of the proposed schema is its dependency of a cluster level centralized load balancing, making it unpractical to the scales of our systems of interest. Also, it requires the on-line elicitation of the resource consumption profile for each service –using application supplied metrics– to adjust the resource allocation, while our approach uses a model-less adaptation.

Closer to out work, in [1] nodes are self-organized and sliced according to an application defined metric and the group that represents the "top" slice are selected to form the application's overlay. Its main drawback is that as the nodes' attributes may change continuously the slicing must also be continuously updated, and operation that require the execution of protocols that run over "epochs" of several update cycles in the order of several seconds. This requirements make this approach unsuitable to the scenarios of interest. In our approach, we integrate this process of updating the set of active nodes into the routing process, making it more responsive to changes. Besides, there's no empirical evidence of the actual performance of the proposed model.

5 Conclusions

We have presented *e*UDON an overlay for dynamically scaling services on large scale infrastructures of non-dedicated servers. The evaluation of the different scenarios shows the ability of *e*UDON to adapt to changing conditions using only local information and local decisions, while maintaining the QoS objectives and a high utilization level. More importantly, the system is highly scalable and resilient to failures, two characteristics that are critical for systems based on commodity hardware clusters.

A salient feature of *e*UDON is its model-less adaptation approach, which can be used in scenarios where there is not a model to predict the QoS of a service, or applying such a model is not feasible due to the dynamism of the environment. Moreover, the adaptation does not require any isolation between competing services.

One additional advantage of the proposed model is that it unifies different events, like service failure, saturation or demotion under a single set of simple adaptation mechanism, simplifying the system design.

The results presented are part of a work in progress and there are still diverse aspects to develop. As already mentioned, the continuous adaptation of the number of active instances (the activation/deactivation mechanism) is still an open issue which is being actively researched. We envision using a mechanism similar to the one used for promotion/demotion but triggered at the servers to decide which services to activate/deactivate.

Additionally, we use only instantaneous measurements of an instance's utility for the various adaptation decisions and in particular, for the admission acceptance window. As a result, even when the average utility ration is around 1.0 and shows little variability, no guarantees of the type "95% of request will have a certain QoS" are currently possible. We are exploring the utilization of a form of summarization of the recent history to offers such guarantees.

References

1. Babaoglu, O., Jelasity, M., Kermarrec, A.M., Montresor, A., van Steen, M.: Managing clouds: a case for a fresh look at large unreliable dynamic networks. ACM SIGOPS Operating Systems Review 40, 3 (2006)
2. Blanquer, J.M., Batchelli, A., Schauser, K., Wolsk, R.: Quorum: Flexible quality of service for internet services. In: 2nd Symposium on Networked Systems Design and Implementation (NSDI '05) (May 2–4 2005)
3. Cao, J., Andersson, M., Nyberg, C., Kihl, M.: Web server performance modeling using an m/g/1/k*ps queue. In: 10th International Conference on Telecommunications (2003)

4. Cardellini, V., Casalicchio, E., Colajanni, M., Yu, P.S.: The state of the art in locally distributed web-server systems. ACM Computing Surveys 34(2), 263–311 (June 2002)
5. Chacin, P., Navarro, L., Lopez, P.G.: Utility driven service routing over large scale infrastructures. In: In Proceedings ServiceWave Conference (2010)
6. Chacin, P., Navarro, L., Lopez, P.G.: Load balancing on large-scale service infrastructures. Technical Report UPC-DAC-RR-XCSD-2011-1, Polytechnic University of Catalonia, Computer Architecture Deparment. Computer Networks and Distributed Systen Group. (2011)
7. Chandra, A., Goyal, P., Shenoy, P.: Quantifying the benefits of resource multiplexing in on-demand data centers. In: First ACM Workshop on Algorithms and Architectures for Self-Managing Systems (2003)
8. Jelasity, M., Montresor, A., Babaoglu, O.: Gossip-based aggregation in large dynamic networks. ACM Transactions on Computer Systems 23(3), 219–259 (2005)
9. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic placement for clustered web applications. In: Proceedings of the 15th international conference on World Wide Web. pp. 595–604. ACM (2006)
10. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03) (Octuber 11–14 2003)
11. Kephart, J.O., Das, R.: Achieving self-management via utility functions. IEEE Internet Computing 11(1), 40–48 (January 2007)
12. Kephart, J., Chess, M.: The vision of autonomic computing. Computer 31(1), 41–50 (2003)
13. Kossmann, D., Kraska, T., Loesing, S.: An evaluation of alternative architectures for transaction processing in the cloud. In: oceedings of the 2010 international conference on Management of data SIGMOD'10. pp. 579–590 (2010)
14. Menasce, D.A.: Qos issues in web services. IEEE Internet Computing 6(6), 72–75 (2002)
15. Nallur, V., Bahsoon, R., Yao, X.: Self-optimizing architecture for ensuring quality attributes in the cloud. In: Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. (2009)
16. Oppenheimer, D., Chun, B., Patterson, D., Snoeren, A.C., , Vahdat, A.: Service placement in a shared widearea platform. In: USENIX Annual Technical Conference. p. 273–288 (2006)
17. Ruth, P., McGachey, P., null Dongyan Xu: Viocluster: Virtualization for dynamic computational domains. In: IEEE International Conference on Cluster Computing (2005)
18. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems 4(2), 42 (May 2009)
19. Shen, K., Tang, H., Yang, T., Chu, L.: Integrated resource management for cluster-based internet. In: 5th Symposium on Operating Systems Design and Implementation (2002)
20. Yang, L., Foster, I., Schopf, J.: Homeostatic and tendency-based cpu load predictions. In: Proceedings. International Parallel and Distributed Processing Symposium. p. 9 (2003)