Conceptual Approach for Performance Isolation in Multi-Tenant Systems

Manuel Loesch[1] and Rouven Krebs[2]
[1]*FZI Research Center for Information Technology, Karlsruhe, Germany*
[2]*SAP AG, Global Research and Business Incubation Karlsruhe, Germany*
loesch@fzi.de, rouven.krebs@sap.com

Keywords: Performance; Isolation; Architecture; Multi-Tenancy; PaaS

Abstract: Multi-tenant applications (MTAs) share one application instance among several customers to increase the efficiency. Due to the tight coupling, customers may influence each other with regards to the performance they observe. Existing research focuses on methods and concrete algorithms to performance-isolate the tenants. In this paper, we present conceptual concerns raised when serving a high amount of users. Based on a load balancing cluster of multiple MTAs, we identified potential positions in an architecture where performance isolation can be enforced based on request admission control. Our discussion shows that different positions come along with specific pros and cons that have influence on the ability to performance-isolate tenants.

1 INTRODUCTION

Cloud computing is a model that enables ubiquitous and convenient on-demand access to computing resources (Armbrust et al., 2009) via the Internet, offered by a central provider. Economies of scale reduce costs of such systems. In addition, sharing of resources increases the overall utilization rate and allows to distribute static overheads among all consumers.

The NIST defines three service models for cloud computing (Mell and Grance, 2011). Infrastructure as a Service (IaaS) provides access to hardware resources, usually by levering virtualization. Platform as a Service (PaaS) provides a complete runtime environment for applications following a well-defined programming model. SaaS offers on-demand access to pre-installed applications used remotely.

Multi-tenancy is used in SaaS offerings to share one application instance between different tenants, including all underneath layers, in order to leverage cost saving potentials the most. At this, a tenant is defined as a group of users sharing the same view on an application. A view includes the data they access, the configuration, the user management, particular functionality, and non-functional properties (Krebs et al., 2012a). Typically, a tenant is one customer such as a company. This way, multi-tenancy is an approach to share an application instance between multiple tenants by providing every tenant a dedicated share of the instance which is isolated from other shares.

1.1 Challenges

Since MTAs share the hardware, operating system, middleware and application instance, this leads to potential performance influences of different tenants. For potential cloud customers, performance problems are a major obstacle (IBM, 2010) (Bitcurrent, 2011). Consequently, it is one of the primary goals of cloud service providers to isolate different customers as much as possible in terms of performance.

Performance isolation exists if for customers working within their quotas, the performance is not affected when aggressive customers exceed their quotas (Krebs et al., 2012b). Relating this definition to Service Level Agreements (SLAs) means that a decreased performance for the customers working within their quotas is acceptable as long as their performance is within their SLA guarantees. Within this paper we assume SLAs where the quota is defined by the request rate and the guarantees by the response time.

In order to fully leverage benefits of multi-tenancy, the goal is to realize an efficient performance isolation which means that a tenant's performance should only be throttled when (1) its quota is exceeded, and (2) he is responsible for performance degradation of other tenants. If violating the quota were the only criteria, free resources would unnecessarily be wasted.

Since customers have a divergent willingness to pay for performance, SaaS providers are furthermore interested in product diversification and providing different Quality of Service (QoS) levels when sharing application instances. This is only possible when having a mechanism to isolate the performance.

On the IaaS layer mutual performance influences can be handled by virtualization. However, on the SaaS layer where different tenants share one single application instance, the layer discrepancy between the operating system that handles resource management and the application that serves multiple tenants makes performance isolation harder to achieve. Multi-tenant aware PaaS solutions handle issues related to multi-tenancy transparent for the application developer in order to increase the efficiency of the development process. However, nowaday's PaaS solutions do not address the introduced performance issues.

When solving the problem of mutual performance influences in practice, it has to be considered that multi-tenant aware applications have to be highly scalable since typical use cases aim at serving a very large customer base with a huge number of simultaneous connections. Hence, one single application instance running on a dedicated server may not be enough and it is likely that more processing power is needed than a single server can offer.

1.2 Contribution

The introduction of a load balancing cluster where a load balancer acts as single endpoint to the tenants and forwards incoming requests to one of several MTA instances, results in the need for an architectural discussion. In addition to the development of algorithms that ensure performance isolation, it is also necessary to provide solutions that show how they can be applied in real-world environments. Hence, this paper identifies two essential conceptual concerns for performance isolation in multi-tenant systems with regards to the request distribution in a load balancing cluster. We defined three positions in an architecture where performance isolation can be enforced based on admission control. The discussion of their pros and cons with respect to the elaborated concerns helps to apply existing solutions in real-world environments.

The remainder of the paper is structured as follows. The related work presents an overview of existing isolation mechanism as well as the the current architectural discussions, and we outline the missing points in the ongoing research. Section 3 introduces the conceptual concerns related to the distribution of request. Section 4 evaluates various positions to enforce performance isolation in a load balancing cluster and the last Section concludes the paper.

2 RELATED WORK

The related work is twofold. The first part focuses on concrete methods and algorithms to isolate tenants with regards to the performance they observe, and the second part discusses conceptual issues. Following, we first give an overview of the first part of related work.

Li et al. (Li et al., 2008) focus on predicting performance anomalies and identifying aggressive tenants in order to apply an adoption strategy to ensure isolation. The adoption strategy itself is not addressed in detail, but it reduces the influence of the aggressive tenant on the others.

Lin et al. (Lin et al., 2009) regulate response times in order to provide different QoS for different tenants. For achieving this, they make use of a regulator which is based on feedback control theory. The proposed regulator is composed of two controllers. The first uses the average response times to apply admission control and regulate the request rate per tenant, the second uses the difference in service levels between tenants to regulate resource allocation through different thread priorities.

Wang et al. (Wang et al., 2012) developed a tenant-based resource demand estimation technique using Kalman filters. By predicting the current resource usage of a tenant, they were able to control the admission of incoming requests. Based on resource-related quotas, they achieved performance isolation.

In (Krebs et al., 2012b) four static mechanisms to realize performance isolation between tenants were identified and evaluated. Three of them leverage admission control, and one of them uses thread pool management mechanisms.

All of the above approaches miss to discuss architectural issues that become relevant when they have to be implemented. Furthermore, no solution discusses scenarios where more than one instance of the applications is running as a result of horizontal scaling. After this overview of concrete methods, subsequently the second part of related work is presented which addresses MTAs and isolation on a conceptual level.

Guo et al. (Guo et al., 2007) discuss multiple isolation aspects relevant for MTAs on a conceptual level. Concerning performance isolation they propose Resource Partitioning, Request-based Admission Control and Resource Reservation as mechanisms to overcome the existing challenges. However, the paper does not focus on situations with several application instances.

Koziolek (Koziolek, 2011) evaluated several existing MTAs and derived a common architectural style. This architectural style follows the web application pattern with an additional data storage for tenant-related meta data (e.g., customization) and a meta data manager. The latter uses the data stored in the meta data storage to adopt the application to the tenants' specific needs once a request arrives at the system. However, Koziolek's architectural style does not support performance isolation.

In (Krebs et al., 2012a) various architectural concerns such as isolation, persistence, or the distribution of users in a load balancing cluster are presented and defined. Furthermore, an overview of the mutual influences of them is presented. The paper defines various aspects relevant for the following section. However, it does not discuss in detail the information that are needed to ensure performance isolation. Further, the position of a potential admission control in a load-balanced cluster is not addressed.

3 CONCEPTUAL CONCERNS IN MULTI-TENANT SYSTEMS

In this section two major conceptual concerns are presented that are of interest for performance isolation in the context of a load balancing cluster of multiple MTA instances.

3.1 Tenant Affinity

The need to horizontal scale out by using multiple processing nodes (i.e. real servers or virtual machines) to run application instances of the same application leads to different ways to couple tenants and application instances. For this purpose, the term affinity is used. It describes how requests of a tenant are bound to an application instance. Various types of affinity might be introduced because of technical limitations, or to increase the performance since it is likely to increases the cache hit rate when the users of one tenant use the same instance. However, sharing a tenant's context among application instances that are running on different processing nodes requires a shared database, or the use of synchronization mechanisms. Since this might be inefficient, tenants may be bound to certain application instances only. In (Krebs et al., 2012a), four different ways are described of how such a coupling of tenants and application instances can be realized:

1. *Non-affine*: Requests from each tenant can be handled by any application instance.
2. *Server-affine*: All requests from one tenant must be handled by the same application instance.
3. *Cluster-affine*: Requests from one tenant can be served by a fixed subgroup of all application instances and one application instance is exactly part of one subgroup.
4. *Inter-cluster affine*: Same as cluster-affine, but one application instance can be part of several subgroups.

3.2 Session Stickiness

Independent of tenant affinity, requests can be stateful or stateless. Stateless requests can always be handled by each available application instance. However, maintaining a user's temporary state over various requests may be required, especially in enterprise applications. This is described by the term session. A session is a sequence of interaction between a tenant's user and an application in which information from previous requests are tracked. For load balancing reasons, it makes sense that requests of one session can still be handled by different application instances depending on the processing nodes' load. Hence, when dealing with stateful requests, it can be distinguished between two kinds of sessions:

1. *Non-sticky sessions* are sessions where each single request of a tenant's user can be handled by all available (depending on the tenant affinity) application instances. Single requests are not stuck to a certain server.
2. *Sticky sessions* are sessions where the first request and following requests of a tenant's user within a session have to be handled by the same application instance.

When using non-sticky sessions, the session context must be shared among relevant application instances. This results in an additional overhead. Consequently, it might be beneficial to use sticky sessions to avoid sharing of session information.

4 TOWARDS PERFORMANCE ISOLATION

In this section two aspects of performance isolation in a load balancing cluster of multiple instances are discussed. First, the information availability at different positions of the requests processing flow, and second, the consequences of tenant and session affinity.

4.1 Possible Positions

An intermediary component such as a proxy will get different information at different positions in the process flow of a request. In Figure 1, three possible positions to enforce performance isolation based on request admission control are depicted.

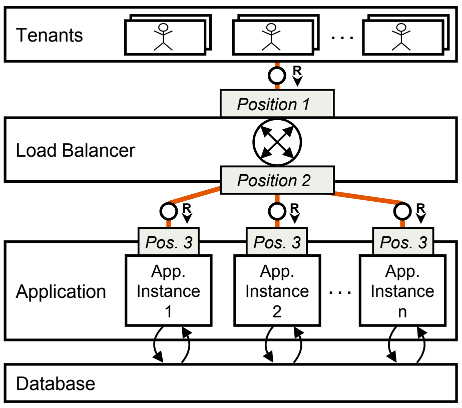

Figure 1: Positions to enforce performance isolation.

In front of the load balancer (Position 1) an intermediary has access to requests from all tenants, it can determine the response times and also whether a tenant is within its quota. The latter is relevant since performance isolation is based on the overall amount of requests from a tenant. However, it is not know which request is executed by which application instance since this is decided by the load balancer. The independence of the request distribution is a motivation for this position since it can allow for easier admission decisions. When fine-grained information about a processing nodes's internal state should be used by the isolation algorithm (e.g., resource utilization), access to this data is only possible with a notable communication overhead.

Directly after or included in the load balancer (Position 2), the information available is a superset of the information available at Position 1. In addition to the access to all requests and their response times, at this position, access to their distribution is given as well. It is known which application instance is responsible for which request and the overall amount of requests from each tenant is known as well. Again, the use of fine-grained information about a processing nodes's state comes along with a notable communication overhead.

In front of the application (Position 3) an intermediary has no information about other processing nodes, such as the number of requests they processed for a given tenant or their utilization. However, information about response times of the respective processing node are available. Compared to the other positions, fine-grained access to a processing node's internal state is possible with significantly less overhead since the component can be placed directly on the respective processing node. Further, no global knowledge of the other instances exists. If the information of all intermediaries is shared, this position would offer the same information as Position 2.

4.2 Comparison of Different Positions

In this section, the suitability of performance isolation at the three introduced positions is discussed with respect to tenant and session affinity. It is shown that the kind of tenant affinity and support of sticky sessions is a major decision for horizontal scalable MTAs. Besides load balancing, synchronization of data and support for session migration, it has big impact on performance isolation.

We assume that requests from each tenant are always homogeneously distributed over all available application instances if possible. Hence, accumulations of requests from a tenant to a single application instance are avoided and a clear separation of server-affinity and the other cases of affinity is given. From an information-centric point of view, it has to be noted that the required information for performance isolation and QoS differentiation is the same. Whenever it is possible to performance isolate tenants, it is also possible to give precedence to certain tenants by adding weighting factors when isolating them.

4.2.1 In Front of Load Balancer

Server-affine: In this scenario, performance isolation is not possible. An increase in response times and request rates can be measured. However, it can not be determined which request will be processed at which application instance since this information is maintained in the load balancer. Although it is known that requests from a tenant are always served by the same instance, the tenants that influence each other's performance by being bound to the same instance are not known. This makes it impossible to efficiently separate tenants. Sticky sessions do not in influence this since they do not answer which tenants are influencing each other.

Non-affine: In this scenario, it depends on the session stickiness whether performance isolation is possible. Using non-sticky sessions, performance isolation is possible. In front of the load balancer it can be determined whether a tenant is within his quota since the full number of requests from a tenant is known. Furthermore, in case of non-sticky sessions, the load balancer can homogeneously distribute the requests. Hence, the more aggressive a tenant is, the more he is contributing to a bad performance of any tenant. With this knowledge, it is possible to performance-isolate tenants. However, when using sticky sessions, requests are bound to an unknown instance. In this case, interfering when one or more tenants experience a bad performance is not possible since requests are not uniformly distributed to the different instance, and hence not necessarily the most aggressive tenant is responsible for bad performances. While initial requests will be distributed homogeneously, it might end up with a significant number of sessions that spend more time than others. Thus, it is possible that the most aggressive tenant is bound to a processing node with no further load whereas a less aggressive tenant has to share a processing node's capacity and thus is responsible for the bad performance of other requests.

Cluster-affine: In this scenario, performance isolation is not possible. The behavior in terms of request allocation is the same as described in the non-affine case with sticky sessions: the underling problem is that the request allocation information is missing and the uniform distribution of request workload could no longer be assumed since the available instances are limited to a subset which is not known at this position. This is not changed by sticky sessions since they only make existing request-to-instance allocations fix.

4.2.2 Directly After/ Included in Load Balancer

Server-affine, Non-affine, Cluster-affine: At this point, performance isolation is possible in all three cases of affinity. The load balancer maintains state to enforce tenant affinity and the stickiness of sessions in order to allocate requests to instances. Hence, at this position the available information about tenant affinity and session stickiness is a superset of the information available at the two other positions. The information about the request allocation and the ability to measure response times allow to interfere and performance-isolate tenants. However, as already stated, access to a processing nodes's state which may increase quality of performance isolation is complicated and comes along with communication overhead.

4.2.3 In Front of Application

Server-affine: In this scenario, performance isolation is possible. Given server-affinity, requests are always processed by the same instance and at this point we are directly in front of the respective processing node. Thus, any information about other processing node does not come along with benefits. Since requests of a tenant are not spread over multiple instances, other processing nodes do not influence this tenant and it is possible to completely measure all information related to the specific tenant's performance. Since requests are already bound to a specific instance, it is irrelevant whether sticky sessions are used or not.

Non-affine: In this scenario, performance isolation is not possible without further information. Since requests of tenants can be served by all instances, the load balancer is free to distribute the requests of all tenants. Hence, it can be assumed that requests of each tenant are homogeneously distributed over all instances. However, performance isolation is not possible as the information about the total number of request send by each tenant is not available. This way, it can not be determined whether a tenant's quota is exceeded. The use of sticky or non-sticky sessions does not change this since requests from a single tenant are still distributed over various instances. However, in the case of non-sticky sessions, performance isolation is possible when the processing capacity of the processing nodes is equal and the total number of instances is considered. Then, the overall request rate can be determined since a homogeneous distribution of the requests can be assumed. Hence it is possible to determine whether a tenant's quota is exceeded. But in the case of sticky sessions, performance isolation is still not possible since a homogeneous distribution of requests cannot be assumed any more.

Cluster-affine: Again, in this scenario, it is not possible to realize performance isolation without further information. The behavior in terms of request allocation is the same as in the non-affine case with the limitation that the available set of instances is a smaller subset. Similar as in the former case, the problem is missing information about requests that are processed at other instances, which makes it impossible to determine quota violations. Again, there is no difference when non-sticky or sticky sessions are used since the latter only make the request-to-instance allocation fix. However, like in the non-affine case, performance isolation is possible in the case of non-sticky sessions when all processing nodes have the same processing capacity and the cluster size is known. Then, information can be projected from one processing node to another by assuming a homogeneous distribution of the load balancer. This allows to determine whether a tenant is within its quota and thus performance can be isolated since access to response times is given as well.

4.3 Summary and Implications

Table 1 summarizes the above discussion and shows the elaborated differences based on different kinds of tenant and session affinity. The stickiness of sessions is only influential in some cases. In the presence of a non-affine behavior and session affinity, a central management of request processing information with access to the allocation of requests to instances as well

Table 1: Positions and feasibility of performance isolation.

Tenant Affinity	Session Stickiness	Pos. 1	Pos. 2	Pos. 3
affine	no	no	yes	yes
	yes	no	yes	yes
non	no	yes	yes	yes
	yes	no	yes	no
cluster	no	no	yes	yes
	yes	no	yes	no

as the overall amount of requests is required in order to guarantee performance isolation. It was explained why, in many scenarios, performance isolation is not possible without information about the request distribution (Position 1), or directly in front of the application instance (Position 3). Offering a superset of the information available at the two other positions, Position 2 is the only one that allows to realize performance isolation for all affinity combinations.

5 CONCLUSION

It was shown that performance isolation between tenants is an important aspect in multi-tenant systems, and that serving a huge amount of tenants requires the existence of several application instances and a load balancer that distributes requests among them. While existing work focuses on concrete algorithms and techniques to enforce performance isolation, this paper focuses on a conceptual realization of performance isolation in a load-balanced multi-tenant system.

We were able to outline that, from an information-centric point of view, the best placement strategy for a performance isolation component that leverages request admission control is directly after the load balancer. At this position, information about the allocation of requests to processing nodes as well as the overall amount of requests from a tenant is given. It was shown that the positions before the load balancer, or directly before the applications have disadvantages which make it impossible to realize performance isolation in every scenario. However, the use of fine-grained information about a processing node's state may increase the quality of performance isolation and this is best possible when the component is placed at the respective processing node. Consequently, data has to be transmitted via the network in the other cases, which leads to a trade-off decision depending on the concrete scenario.

Our future research focuses on providing a complete architecture to enforce and evaluate performance isolation based on the here presented results.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement Nº 258862.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Technical report, EECS Department, University of California, Berkeley.

Bitcurrent (2011). Bitcurrent Cloud Computing Survey 2011. Technical report, Bitcurrent.

Guo, C. J., Sun, W., Huang, Y., Wang, Z. H., and Gao, B. (2007). A Framework for Native Multi-Tenancy Application Development and Management. In *Procceedings of the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*.

IBM (2010). Dispelling the vapor around cloud computing. Whitepaper, IBM Corp.

Koziolek, H. (2011). The SPOSAD Architectural Style for Multi-tenant Software Applications. In *Procceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA 2011)*.

Krebs, R., Momm, C., and Kounev, S. (2012a). Architectural Concerns in Multi-Tenant SaaS Applications. In *Proc. of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012)*.

Krebs, R., Momm, C., and Kounev, S. (2012b). Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. In *Proceedings of the 8th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2012)*.

Li, X. H., Liu, T., Li, Y., and Chen, Y. (2008). SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment. In *Proc. of the 6th International Conference on Service-Oriented Computing (ICSOC 2008)*.

Lin, H., Sun, K., Zhao, S., and Han, Y. (2009). Feedback-Control-Based Performance Regulation for Multi-Tenant Applications. In *Proc. of the of the 15th International Conf. on Parallel and Distributed Systems*.

Mell, P. and Grance, T. (2011). The NIST definition of cloud computing (Special Publication 800-145). *Recommendations of the National Institute of Standards and Technology*.

Wang, W., Huang, X., Qin, X., Zhang, W., Wei, J., and Zhong, H. (2012). Application-Level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications. In *Proc. of the 2012 IEEE 5th International Conf. on Cloud Computing*.