Scalability analysis of the OpenAM access control system with the Universal Scalability Law

Thomas Heyman
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
thomas.heyman@cs.kuleuven.be

Davy Preuveneers
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
davy.preuveneers@cs.kuleuven.be

Wouter Joosen
iMinds-DistriNet, KU Leuven
3001 Leuven, Belgium
wouter.joosen@cs.kuleuven.be

***Abstract*—The scalability of a software system is greatly impacted by the scalability of the underlying access control system, which makes analyzing the scalability of that access control system paramount. However, this is not trivial, as contemporary access control systems have a myriad of architectural deployment variations, each of which has a potentially large impact on overall system throughput. There is a need for a systematic approach to map these architectural variations to a reference model which allows to make comparisons and to identify trade-offs. This work provides a piece of the puzzle by demonstrating how this can be achieved by systematically applying the Universal Scalability Law (USL). We illustrate our approach by performing a rigorous scalability analysis of the OpenAM access control system for various deployment alternatives in the domain of authentication. We conclude that the approach is able to provide both qualitative and quantitative results which can be translated into practical operational recommendations for the envisioned types of system deployments.**

I. Introduction

Identity management and access control are crucial supporting services in contemporary online software systems—it is hard to imagine an interactive web site that does not depend on authentication, authorization and even federation with third party identity providers such as Facebook or Google. When viewed from a business perspective, these services need to be as transparent as possible, and it should be avoided at all costs that they become a bottleneck. Moreover, access control services need to be always available in order for users to be able to log into protected software systems, instead of rendering those protected systems inaccessible. Consequently, the scalability and performance of access control systems are crucial system qualities to consider, and scalability is often mentioned as one of their key requirements [1].

The question of whether a certain access control system is scalable, is not a simple one: It is exacerbated by the many architectural variations in which these systems can be deployed. For instance, traffic composition, stateful (i.e., "sticky") load balancing, session replication strategies, and configurations of the data backend all have a potentially huge impact on the overall access control system performance. Even though there is a need to take architectural deployment variations into account, there are currently very few studies that document these systematic comparisons and verify whether the correct trade offs have been made.

This paper documents our experiences in analyzing and characterizing both the scalability and performance of OpenAM[1], a widely used open source access control system, deployed globally by government agencies, financial institutions, telecom operators and IT services. Our analysis is based purely on benchmarking data and treats the system as a black box: We identify and characterize the impact of a number of architectural variations by applying Gunther's Universal Scalability Law (USL) [2], [3], [4], and analyze the quality of service by characterizing the distribution of residence times. As deployment variations, we consider the dimensions of user load, traffic composition, number of deployed service instances, used replication strategies in data backend systems, and session persistence options. For scoping reasons, we limit ourselves to the authentication functionality of OpenAM.

The rest of this paper is structured as follows. Section II provides an overview of both OpenAM and quantifying scalability. Section III documents our experimental setup and benchmarking methodology. Section IV documents our results. Section V provides a discussion of these results and summarizes some lessons learned for software architects and deployers of access control systems such as OpenAM. Section VI gives an overview of related work. We conclude in Section VII.

II. Background

This section provides a brief overview of authentication and the OpenAM access control software in Section II-A, increasing capacity through clustering in Section II-B, and of scalability as a non-functional requirement and its analysis in Section II-C.

A. Authentication

Authentication, or the act of confirming the identity of a user, is an essential part in the larger picture of access control and identity management. OpenAM is an open source software product that offers authentication, authorization, identity management and federation functionality. It is maintained by ForgeRock, and is the continuation of Sun's OpenSSO[2]. OpenAM is part of the ForgeRock Open Identity Stack[3], which includes software such as the OpenDJ LDAP server, and OpenIDM, which offers user identity lifecycle management. OpenAM depends on OpenDJ, which it uses as a data backend—both configuration and user details are stored in a different OpenDJ instance.

[1] http://openam.forgerock.org/
[2] http://en.wikipedia.org/wiki/OpenSSO
[3] http://forgerock.com/products/open-identity-stack/

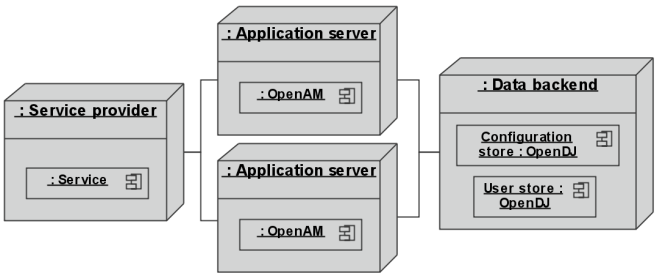

Figure 1. A deployment view on a simple OpenAM instantiation.

An example OpenAM instantiation is shown in Figure 1. It shows a service deployed at a service provider which uses one of two OpenAM instances deployed on separate application servers for authentication. These instances can be clustered, as documented in Section II-B. The OpenAM instances, in turn, depend on two distinct instances of the OpenDJ LDAP service for its data backend—one, the user store, contains all relevant user account details, while the other, the configuration store, contains the current OpenAM configuration and active session data.

There are two main ways to interact with OpenAM: Via the built-in web interface, or via the REST interface. For the remainder of this work, we focus on the REST interface, as it is the most flexible and extensible interface, and therefore more likely to be used. As an example, given an OpenAM service deployed at the endpoint http://mydomain.org/openam/, a user can log in given a username 'joe' and password 'secure' by calling http://mydomain.org/openam/identity/authenticate?username=joe&password=secure. The specified credentials are checked against the user account details in the user store. If successful, that call returns a token (i.e., an alphanumeric string) which represents the newly created user session. The validity of a session can later be established by calling http://mydomain.org/openam/identity/isTokenValid?tokenid=token, with token the alphanumeric string returned earlier. Finally, a user can log out of a session by passing the corresponding token to http://mydomain.org/openam/identity/logout?subjectid=token. Similar REST APIs exist to check user authorization and perform identity management functions.

B. Clustering

Clustering is an often used technique to increase the capacity of a service by deploying it on multiple servers that collaborate to handle a workload. OpenAM supports clustering out of the box. If one OpenAM instance fails to meet the volume of incoming authentication requests, more instances can be deployed on different access control servers, each of which uses the same configuration and user stores. Similarly, each of the data backend services can be replicated in order to distribute the load of the OpenAM cluster, and decrease the risk of data loss. OpenDJ supports best effort (i.e., eventually consistent), assured write (i.e., data is guaranteed to be replicated to at least one other machine) and assured read (i.e., data reads are guaranteed to be consistent) replication options.

When a user logs in, the newly created session is only local to the current OpenAM instance. Upon subsequent user requests, the validity of the token has to be established by that issuing OpenAM instance. When multiple OpenAM instances are deployed behind a load balancer, this implies that the other instances constantly have to query the originator to establish the validity of a token. Similarly, when one instance becomes unavailable, the validity of the sessions created by that instance can no longer be verified, and those users need to log in again. The first issue can be handled by stateful (or 'sticky') load balancing, in which subsequent user requests are always directed to the same OpenAM instance. The second issue can be resolved by making sessions persistent. In that case, all session state is stored in the configuration store, which is shared between all OpenAM instances of the same cluster. All these features have an impact on scalability, as we will see in Section IV. First, we elaborate on what we mean with 'scalability'.

C. Scalability

In this work, scalability means the ability of a service to handle various user loads. In the strict interpretation, scalability is the function $C(p)$ of how service capacity C varies in function of user load p. In this context, user load is defined as the number of concurrent user processes that periodically issue requests to a service. For instance, if a user process issues two requests per second, then a user load of 50 implies that 100 requests per second will be issued to the service. The relative capacity $C(50)$ shows how many of those 50 user processes are effectively served by the service.

However, scalability also has a more qualitative aspect—it can be seen as the function of how the quality of service depends on that user load. In that sense, scalability is the function $R(p)$ of how residence times R vary in function of the user load. We begin by detailing the service capacity interpretation, after which we deal with residence times and quality of service.

1) Service capacity and the Universal Scalability Law: Our approach is based on that presented in [5]. In order to analyze the capacity of the OpenAM service under varying user loads, we generate simulated user requests at a fixed rate and calculate the rate at which those requests are successfully handled, i.e., the throughput X. Based on this data, we can calculate the capacity ratio $C(p) = X(p)/X(1)$, which is the ratio of the throughput of the system for a load of p, compared to the baseline of its throughput for a load of 1. In practice, for a user load p, a system configuration with relative capacity $C(p) = 1.5$ has 50% more throughput than a system configuration with relative capacity $C(p) = 1$. The relative capacity curve $C(p)$ is a good indicator for how much the observed behaviour diverges from the ideal linear scalability $C_L(p) = p$. The closer $C(p)$ is to $C_L(p)$, the better the scalability of that system configuration. Note that the relative capacity of real systems is never larger than the ideal linear scalability, or $C(p) \leqslant C_L(p)$.

One way to model the capacity of a system is by applying the universal scalability law (USL) [2], [4]. The universal scalability model of a system takes into account both the serial nature of the workload of that system (i.e., how much of the workload can be parallelized in theory) and coherency

costs (i.e., the costs incurred when waiting for data to become consistent between different instances of a system that share the same workload). The universal scalability model takes the form of the following curve:

$$C(p) = \frac{p}{1 + \sigma(p-1) + \kappa p(p-1)}$$

Here, κ denotes the impact of coherency on the system performance, and σ denotes the serial fraction, which is the fraction of the workload that cannot be parallelized. When the coherency factor κ is negligible, the maximum performance of the system is bounded only by the serial fraction. When κ is non zero, the performance model of a system will have a specific maximum, achieved for a load $p^* = \left\lfloor \sqrt{(1+\sigma)/\kappa} \right\rfloor$. Beyond p^*, the throughput of a system will decrease.

We can find values for σ and κ for a specific OpenAM deployment by measuring specific values for $C(p)$, after which the measurements are fitted to the USL model.

2) Residence time and quality of service: The residence time of a service is the time that a request resides in the system, i.e., the time between issuing a request and receiving an answer. While the service capacity considers the business view (i.e., "How many servers do I need to handle this many concurrent users?"), residence time considers the end user's perspective (i.e., "How long do I have to wait before my request is handled?"). Residence time R is often seen as the sum of the queueing time Q, i.e., the time that the request spends waiting to be serviced, and the actual service time S. However, as we are only interested in the end user experience, we do not make this distinction.

Characterizing the residence time in function of the user load is important to anticipate how responsive or fast the end user will experience that service. However, simply knowing the average residence time does not suffice in most cases—quality of service is usually expressed in a policy that states that $X\%$ of requests need to be handled within Y (milli-) seconds. In other words, we need to know the distribution of residence times too. This distribution can be analyzed by collecting residence times for sufficient requests and plotting them in a histogram.

III. Experimental setup

We begin by outlining the goals of our experiments in Section III-A. As mentioned in Section II, in order to characterize the scalability of a deployed system, we need to collect data on both throughput and residence times in function of user load. The experimental OpenAM deployment on which the benchmarks are run, as well as the setup we use to generate user load, are documented in Section III-B. The benchmarking methodology followed is documented in Section III-C.

For completeness, the software versions used are OpenAM 11.0 deployed on Apache Tomcat 7.0.47, and OpenDJ 2.6.0.

A. From stakeholder concerns to experimental goals

The goals of the experiment as documented in this section are distilled from actual stakeholder concerns encountered in an industrial research project. The goal of that project was to investigate the feasibility of a proposed authentication framework for online media in Belgium, which implies a user base of up to 11 million users. The stakeholder concerns mainly related to validating the proposed access control infrastructure (e.g., "Is the proposed cluster architecture a good idea?", "Will we be able to scale out further?"), but also involved use case related questions (e.g., "Should we strive for longer session durations, or let sessions time out faster?").

In order to provide a quantified answer to these concerns, they were subsequently mapped to configurable architectural deployment properties (e.g., varying the OpenAM cluster size, varying the OpenDJ replication model, varying session persistence and load balancing strategies), resulting in a list of quantifiable experimental goals. That last step of mapping stakeholder concerns to deployment alternatives also shows that many of the initial stakeholder questions are actually asking about the impact of a small list of architectural properties, which can then be prioritized based on how many stakeholder concerns they relate to. The end result is the following list.

a) Traffic composition: Not all requests are handled equally. Both login and logout requests imply mutating session data, while token verification requests serve as inspectors and do not mutate state. Determining how many times tokens are verified, or establishing guidelines for session longevity clearly impact the traffic composition. But how large is this impact?

b) Cluster size: Depending on the nature of the workload, adding more and more nodes to a cluster quickly runs into the wall of diminishing returns. But where is this wall, and how high is it?

c) Load balancing strategy: Is stateful load balancing essential for achieving a scalable deployment, or is its effect limited?

d) Replication strategy: Whether session persistence should be enabled, and what replication strategy should be used in the backend, greatly impacts service availability and user convenience by not requiring users to log in again unnecessarily. But what is the cost of these features on scalability?

For each of these dimensions, we want to express the capacity of the OpenAM service in function of user load, and extrapolate additional scaling capacity by applying the universal scalability law.

B. Distributed deployment

Our testing setup that we use to generate and collect data is shown in Figure 2. Load generators simulate a variable number of users that each generate a fixed load. The users begin by logging in, repeatedly verify their token, and have a configurable probability of logging out after every token verification. A user that is logged out, will log in automatically upon the next request. A user generates a request every 250ms, i.e., 4 requests per second. User load is increased by spawning more concurrent user threads. Note that, while the 'thinking time' per user is constant, i.e., 250ms, the different user processes are started in a staggered fashion with a random delay. If the number of concurrent user processes is large with respect to the time between two requests for one user, the request inter-arrival rate at the service follows a Poisson distribution.

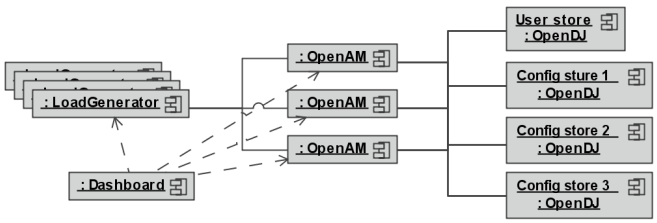

Figure 2. The experimental setup used for obtaining benchmark data.

The load generators communicate directly to one or more OpenAM instances. In order to avoid an intermediate load balancer as a potential bottleneck, load balancing functionality is emulated by the load generators: Depending on whether stateful load balancing is configured, every simulated user will send all requests to the same, resp. a random OpenAM instance. The OpenAM instances are configured as a cluster. They all use one user store[4], and multiple configuration stores. The configuration stores are all in the same replication topology, their replication strategy is configurable.

Overall system health is monitored, and measurement data is gathered, by the dashboard. Load generators send their results directly to the dashboard. The OpenAM instances are monitored by a JMX client that sends intermediate data to the dashboard as well. All data is visualised in real time to check results and identify anomalies, and is stored for later reference.

Every component is deployed on a dedicated machine. All machines are off-the-shelf Dell Optiplex 755 desktops with Intel Core2 duo processors and 4GB of RAM. They are connected via a dedicated gigabit ethernet network.

C. Load generation methodology

We are only interested in system throughput and residence times for various user loads and architectural configurations. This is a significant advantage, as it allows us to treat the system as a black box, and avoids the need to create intricate models (e.g., queueing networks in the case of queueing theory).

For every system configuration under test, the following benchmarks are performed. The load generators begin generating a constant load p, wait for 15 seconds until the OpenAM services stabilize, and then begin to record request residence times for 60 seconds. The specific time to wait (i.e., 15 seconds) is established beforehand by monitoring residence times and CPU utilisation: Due to class loading and just-in-time compilation, there is a clear peak in both CPU utilisation and residence times when a newly started service is placed under load, which abates and stabilises on average after 10 seconds (to which we added 50% to be on the safe side). After the measurement period, the load generators stop, system throughput is calculated, the state of the dashboard is reset, and the OpenAM services are allowed to cool down for another 30 seconds. Again, the exact duration of this cool down period has been experimentally obtained by monitoring CPU utilisation and residence times of a low number of sample requests.

[4]Initial experiments have shown that, for our experiments, the load on the user store is negligible. For simplicity, we have only deployed one user store.

The values for p are chosen after initial experimentation so that they are sufficiently interesting and cover the region of optimal load p^*, if possible. In total, around 10 different p values are used to ensure a proper USL fit. Finally, as an upper bound on the p value region, we take the load p_{max} for which the service starts returning a significant (i.e., more than 1%) amount of errors—initial experiments showed that once this point is reached, the relative frequency of errors quickly increases, rendering the measured relative capacity useless as a significant fraction of requests are not handled properly.

IV. Results

This section documents the results of the OpenAM scalability experiments. Section IV-A documents the throughput of one OpenAM instance in function of various traffic compositions. Section IV-B documents the throughput of an OpenAM cluster of various sizes. Section IV-C documents the impact of stateful load balancing. Section IV-D documents the impact of various session replication models.

In all the subsequent graphs, dots denote measured data points, and lines denote instances of the universal scalability model fitted to that data as per Section II. Load is expressed in concurrent users that each issue a request every 250ms—to get the load in requests per second, multiply the indicated user load by 4. Fitting is done with gnuplot, with the additional constraints that the resulting values for σ and κ are positive; a small example script is provided in Appendix A.

A. Throughput in function of traffic composition

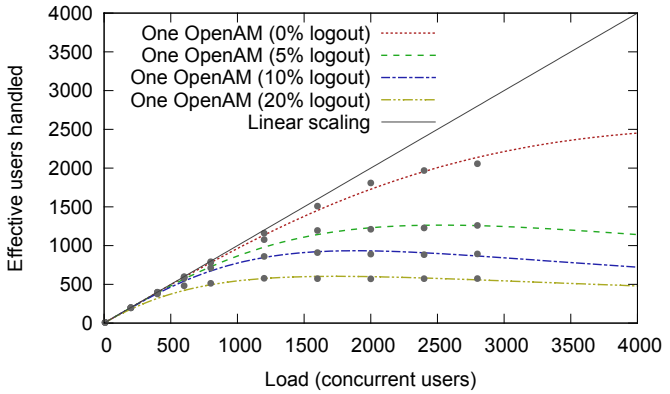

Figure 3. Comparing OpenAM throughput in function of traffic composition. *0% logout:* $\sigma = 2.41 \times 10^{-9}, \kappa = 3.95 \times 10^{-8}$*; 5% logout:* $\sigma = 2.47 \times 10^{-9}, \kappa = 1.56 \times 10^{-7}$*; 10% logout:* $\sigma = 1.14 \times 10^{-5}, \kappa = 2.81 \times 10^{-7}$*; 20% logout:* $\sigma = 5.05 \times 10^{-4}, \kappa = 3.32 \times 10^{-7}$.

Figure 3 shows the throughput of one OpenAM instance, in function of increasing user load, for different traffic compositions. Clearly, the number of requests that one OpenAM instance is able to handle, is highly dependent on the relative frequency of login and logout requests with respect to token verification requests: Login and logout requests impart more overhead on the OpenAM service as credentials need to be verified and sessions need to be created, resp. destroyed. Token verifications are relatively more light weight.

Note that for token verifications, the throughput of one OpenAM instance is very good up to 1200 concurrent users,

at which point its capacity starts to deviate measurably from linearity. At this point, however, an OpenAM that handles traffic consisting of 20% logout requests only has half (49%) the throughput of the ideal case where no logouts are performed.

B. Scaling to multiple instances

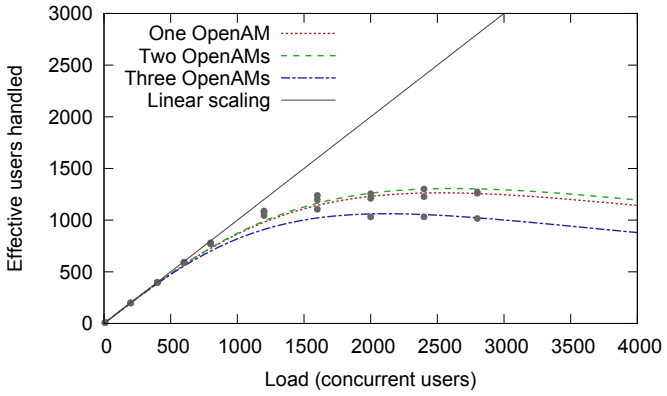

Figure 4. Comparing scalability for multiple OpenAM instances in a cluster. The load is normalized per OpenAM instance. *One:* $\sigma = 2.47 \times 10^{-9}, \kappa = 1.56 \times 10^{-7}$*; Two:* $\sigma = 1.60 \times 10^{-9}, \kappa = 1.47 \times 10^{-7}$*; Three:* $\sigma = 2.08 \times 10^{-9}, \kappa = 2.22 \times 10^{-7}$.

Figure 4 shows the relative capacity of OpenAM instances in a cluster of size one, resp. two and three. Note that the data shows the throughput normalized by the cluster size, so that the contribution of individual OpenAM instances can be compared. The OpenAM instances were configured to use persistent session data which is actively replicated over three configuration stores, to allow the data backend to scale together with the OpenAM service and avoid it becoming the bottleneck. A traffic configuration of 5% logout requests was used.

The data shows that the difference in throughput of individual nodes in a cluster of size one and two is negligible—in other words, a cluster of size two has twice the capacity of a cluster of size one. In fact, this cluster of size two has slightly more than twice the capacity of a cluster of size one, however, this difference is small enough to be attributable to experimental variance. Adding a third OpenAM instance to this cluster decreases the relative throughput of the cluster significantly. For instance, while one OpenAM instance is capable to effectively handle 1224, resp. 1296 users when confronted with a load of 2400, adding a third instance lowers this capacity to 1024 users. We expect the penalty of adding additional instances to the cluster to be higher still. However, in the range where individual OpenAM throughput still scales linearly, i.e., below approximately 1000 concurrent users per instance, the overhead of adding extra OpenAM instances to a cluster is negligible.

C. Impact of stateful load balancing

Figure 5 shows the impact of stateful load balancing on an OpenAM cluster throughput. As in Section IV-B, traffic is configured to consist of 5% logout requests, and the configuration store backend consists of three OpenDJ instances in active replication.

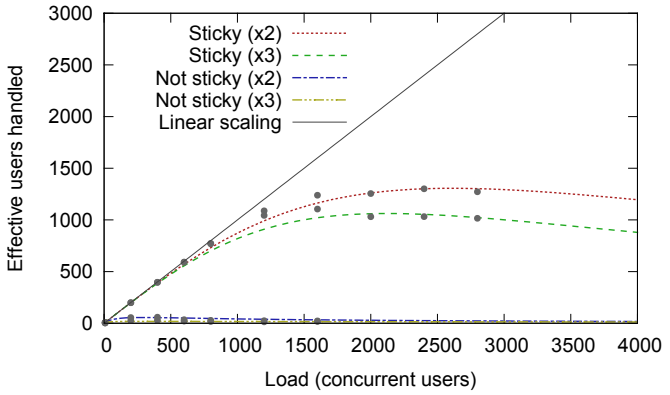

Figure 5. Assessing impact of stateful load balancing on OpenAM throughput. The load is per OpenAM instance. *Sticky (x2):* $\sigma = 1.13 \times 10^{-2}, \kappa = 1.16 \times 10^{-5}$*; Sticky (x3):* $\sigma = 4.97 \times 10^{-2}, \kappa = 7.85 \times 10^{-6}$*; Not sticky (x2):* $\sigma = 1.60 \times 10^{-9}, \kappa = 1.47 \times 10^{-7}$*; Not sticky (x3):* $\sigma = 2.08 \times 10^{-9}, \kappa = 2.22 \times 10^{-7}$.

Clearly, the overhead incurred from intra-cluster communication to verify whether tokens are valid, is unacceptable. Once multiple OpenAM instances come into play, stateful load balancing is a necessity to achieve even a modest throughput.

D. Impact of session replication

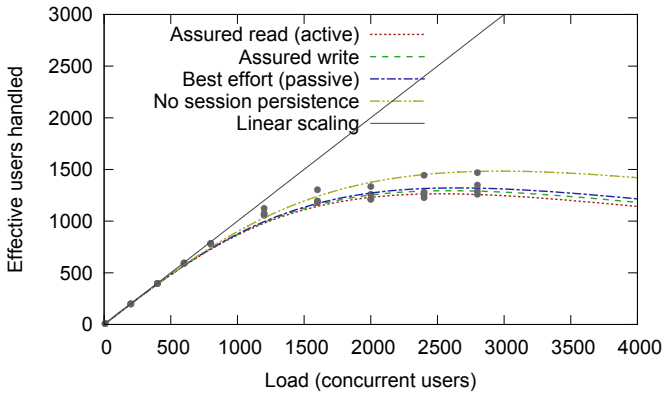

Figure 6. Comparing scalability for different session replication models. *Assured read:* $\sigma = 2.47 \times 10^{-9}, \kappa = 1.56 \times 10^{-7}$*; Assured write:* $\sigma = 2.55 \times 10^{-9}, \kappa = 1.49 \times 10^{-7}$*; Best effort:* $\sigma = 1.25 \times 10^{-8}, \kappa = 1.43 \times 10^{-7}$*; No persistence:* $\sigma = 2.37 \times 10^{-9}, \kappa = 1.14 \times 10^{-7}$.

Figure 6 shows the impact of replication on a deployment with 5% logout traffic, and one OpenAM instance that is connected to a configuration store consisting of a cluster of 3 OpenAM instances. The figure compares the performance penalty incurred by the different replication strategies with the situation where session persistence is disabled (and session data is not stored in the configuration store).

Disabling session persistence is the most performant configuration, but also offers the least guarantees. The other replication models incur a systematically higher performance penalty, but all methods are generally equivalent in the range where OpenAM scales linearly, i.e., below approximately 1000 concurrent users.

E. Quality of service

In order to analyze the quality of service offered by OpenAM, we investigate the distribution of residence times. Figure 7 shows the quality of service for the default single OpenAM configuration with active session replication. Note that of the three request types, token verification is the most efficient, followed by logout requests, and finally login requests, which have the longest residence times.

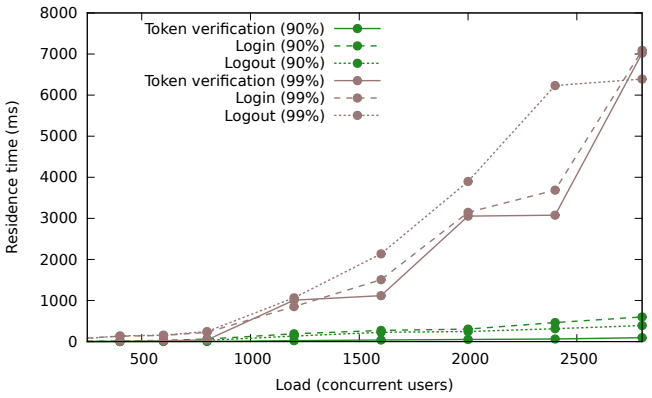

Figure 7. Quality of service for one OpenAM, 5% logout requests, active session replication.

Note that at higher loads, the 90th percentile of request residence times remains relatively stable, while the 99th percentile grows very rapidly. This is due to the nature of the distribution of residence times, which is generally modelled as an Erlang distribution: the majority of requests will be handled efficiently, even under high loads, but the 'tail' of the distribution becomes arbitrary large. Therefore, under high load, the majority of requests will still be handled fast. However, the 'tail' of the request distribution will grow to arbitrary size.

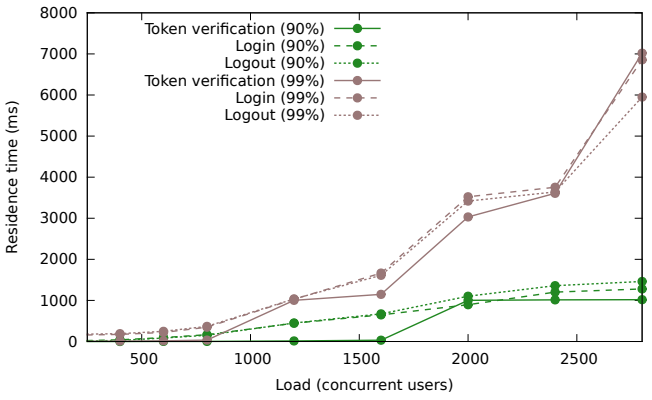

Figure 8. Quality of service for a cluster of three OpenAMs, 5% logout requests, active session replication.

Figure 8 shows the quality of service for a cluster of three OpenAMs. Note that these values are largely comparable to the single node cluster, with the difference that login and logout requests take noticeably longer. On average, token verifications happen very fast for low loads, but at higher loads token verification takes significantly longer than in the single node case.

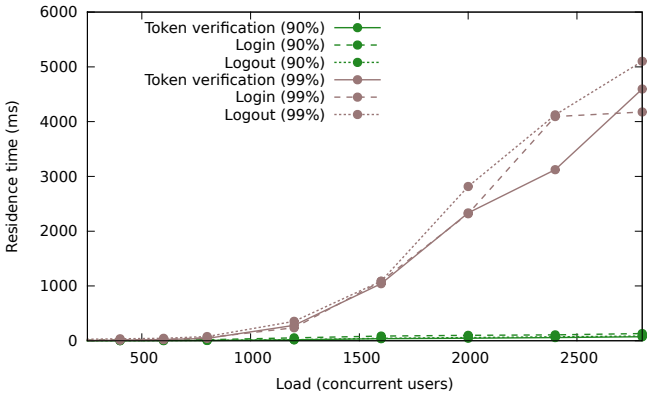

Figure 9. Quality of service for one OpenAM, 5% logout requests, no session replication.

Finally, Figure 9 depicts the quality of service for one OpenAM instance without persistent sessions. Of all configurations, this exhibits the best quality of service. First, the difference between login, logout and token verification requests is insignificant, and all three happen extremely fast in the majority of cases. Second, even under higher loads, the 'tail' of the residence time distribution does not grow as fast as in the other configurations.

V. Discussion

We summarize the main lessons learned in Section V-A. We briefly discuss alternative scalability models and motivate our choice for the USL in Section V-B.

A. Lessons learned

Clearly, the results of a scalability assessment are relative to whether a modelled user process accurately models real user behavior. In the case of this study (which was performed before the eventual service roll out), no data on typical user behaviour was available. Therefore, a general user model was created that issued a relatively high number of requests per second, to err on the side of caution, at the risk of underestimating the real service capacity of the eventual roll out. Given a more representative user model, this analysis could be redone to obtain data that is fine-tuned to the deployment at hand.

We have tried to avoid bottlenecks that are external to the system under test. The load generators themselves have been benchmarked to verify that they can handle the number of user threads without significant slowdown. For a simulated benchmark with dummy users that only perform empty requests, we achieve $\kappa = 2.31 \times 10^{-13}$. This extremely low value (six orders of magnitude below the values for κ in the other experiments) shows that the load generating framework is not hindered by coherency and its throughput is bounded only by the serial fraction of the workload.

Similarly, the network was monitored to exclude it from being the main bottleneck, as were the OpenDJ nodes. Finally, the system throughput and residence times were visualised (and recorded) by the dashboard to identify and analyze potential abnormalities during data capture. The cool down period between two different experiments is chosen to be large

enough for the system CPU activity to return to idle, so that the risk of two subsequent experiments interfering with another is low.

Based on the experimental data gathered, the experimental goals are sufficiently quantified to formulate an answer to the original stakeholder concerns. The following lessons were learned from the experimental results.

e) Sticky load balancing is essential: Not employing stateful load balancing imposes an unacceptable cost on both the throughput and the quality of service of token verification requests.

f) When session persistence is important, use active replication: The extra overhead of active replication compared to the assured write or best effort replication models is negligible in practice. If session replication is required, active replication offers the strongest guarantees.

g) Keep sessions as long lived as possible: In the case of persistent sessions, the percentage of login and logout traffic has an enormous impact on overall throughput and quality of service. Therefore, users should be motivated to stay logged in as long as possible.

h) Avoid login floods: When a substantial number of sessions become invalidated at the same time, the subsequent higher login traffic will have a significant impact on overall system performance, which in turn might impact overall system stability and give rise to more sessions becoming invalidated. This feedback loop could have major consequences on overall system health. In order to avoid login floods, the OpenAM setup should support throttling the number of incoming login requests to allow the system to gracefully recover after a major service interruption.

i) For optimal throughput, disable persistent sessions: The impact of session persistence on both throughput and residence time is noticeable. Disabling session persistence increases both, at the cost of requiring users to log in again when an OpenAM instance becomes unavailable. In times of high peak loads (i.e., the "Slashdot effect"), media services often fall back to a non interactive low-fidelity version. Temporarily disabling persistent sessions could alleviate the higher peak load, and act as an intermediate solution before falling back to this low-fidelity version.

Note that these lessons only take scalability into account, and might negatively impact other non-functional aspects. For instance, active replication imparts a potentially high performance cost on the system when a new node enters a replication topology and needs to be synchronized. Additionally, active replication negatively impacts recovery time as it takes longer for timeouts to be generated in the case of failure. However, these concerns are out of scope for this work.

B. Alternative scalability models

The main contender as an alternative scalability model, is Amdahl's law [6]. That model only considers how the throughput of a system is constrained by the serial fraction σ—if the longest non serializable part of a workload is a fraction σ of the total workload, then the scalability of that system is inherently limited to the size of that non serializable fraction of the workload. Therefore, scalability models that are only bound by Amdahl's law tend to show a monotonically increasing capacity that approaches the limit imposed by the serial fraction asymptotically.

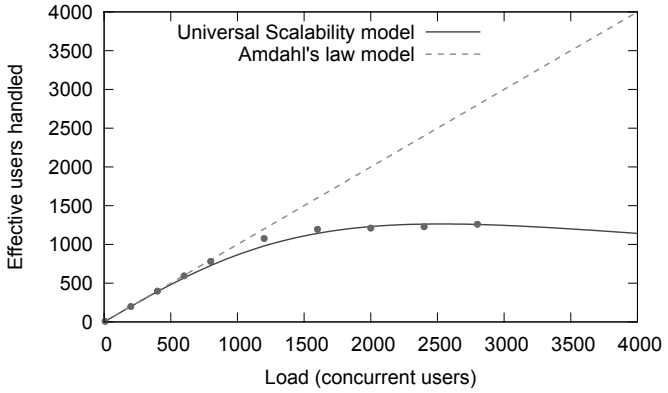

Figure 10. Comparing the Universal Scalability Law with Amdahl's law for modelling the scalability of one OpenAM instance, with 5% logout probability.

In fact, the Universal Scalability Law reduces to Amdahl's law when contention is not considered, i.e., when κ is 0. In cases where contention is not negligible, however, Amdahl scalability models clearly overestimate the capacity: A comparison of Amdahl's law with the Universal Scalability Law for one OpenAM instance with 5% logout probability is shown in Figure 10. A discussion on other scalability models, and their comparison with the Universal Scalability Law, can be found in [5].

VI. Related work

Related work on benchmarking exists on many different system levels, ranging from low-level benchmarks of execution environments and embedded software, to high-level distributed system and business process benchmarks. Guthaus et al. [7] perform benchmarking on embedded programs and provide a comparison with the industry standard benchmark suite SPEC2000. Ghosh et al. [8] analyze the performance of WiMax networks. Uskov [9] provides a comprehensive study of the performance of authentication and encryption algorithms for virtual private networking. Rashwan et al. [10] study the performance of message authentication codes for mobile networks, for both residence time and power consumption. Dayarathna et al. [11] document their results of comparing the performance of three complex event processing engines via benchmarking. Carvalho et al. [12] document a method to analyze scalability of running systems from the data center viewpoint, by only measuring CPU utilization.

For the domain of authentication, some related work exists on benchmarking authentication methods. Tirel et al. [13] document a benchmark for multimodal authentication methods, implemented as a prototype GUI in Matlab. Poh et al. [14] systematically compare 22 fusion systems for multi-modal biometric authentication. Shaikh et al. [15] document their performance benchmarking results for fingerprint-based biometric authentication. However, none of that work considers performance analysis of these authentication methods in a large-scale distributed setup. To the best of the authors' knowledge, there

are little to no reports of scalability and performance studies for large-scale access control deployments.

VII. Conclusion

We have presented a rigorous analysis of the scalability of the OpenAM access control system for the domain of authentication. Starting from specific stakeholder concerns, our study characterizes the scalability of an OpenAM deployment in function of traffic composition, cluster size, load balancing strategy and replication strategy. The scalability analysis observes the system as a black box and does not depend on internal system modelling. By systematically gathering benchmarking data and applying the Universal Scalability Law, we were able to provide quantitative answers to the initial stakeholder concerns: We find that OpenAM with session persistence disabled and stateful load balancing does not incur a high coherency penalty and scales well when sessions are long lived. On the other hand, session persistence and a higher percentage of login and logout requests impact that scalability measurably. Stateful load balancing, however, is essential to achieve even modest throughput. The outcome of this analysis had a real impact on stakeholder decision making, and allowed the stakeholders to better evaluate OpenAM deployments proposed by the contracted hosting company.

Acknowledgment

This research is partially funded by the Research Fund KU Leuven.

Appendix

Regression was performed with gnuplot, a minimal gnuplot configuration script is provided. Input is expected to be in a file 'data', with the generated load p in the first column, and the measured throughput $X(p)$ in the second column.

```
# Fit to find a^2 and a^2 + b^2, so we can
# constrain sigma and kappa to be positive
# (as required by the USL).
f(x) = a*a * x**2 + (a*a+b*b) * x
fit f(x) 'data' using ($1-1):(($1/$2)-1) \
    via a,b

# The serial fraction.
sigma = b*b
# The coherency factor.
kappa = a*a
# The optimal load point p*.
p = floor(sqrt((1+sigma)/kappa))

# The universal scaleup model.
C(p) = p / (1+sigma*(p-1) + kappa*p*(p-1))

# Linear scaling, for comparison.
L(p) = p
```

References

[1] A. D. Keromytis and J. M. Smith, "Requirements for scalable access control and security management architectures," *ACM Transactions on Internet Technology (TOIT)*, vol. 7, no. 2, p. 8, 2007.

[2] N. J. Gunther, "A simple capacity model of massively parallel transaction systems," in *CMG-CONFERENCE-*. COMPSCER MEASUREMENT GROUP INC, 1993, pp. 1035–1035.

[3] ——, "A general theory of computational scalability based on rational functions," *CoRR*, vol. abs/0808.1431, 2008.

[4] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 2013.

[5] N. J. Gunther, *Guerrilla capacity planning - a tactical approach to planning for highly scalable applications and services.* Springer, 2007.

[6] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 483–485.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

[8] A. Ghosh, D. R. Wolter, J. G. Andrews, and R. Chen, "Broadband wireless access with wimax/802.16: current performance benchmarks and future potential," *Communications Magazine, IEEE*, vol. 43, no. 2, pp. 129–136, 2005.

[9] A. V. Uskov, "Information security of ipsec-based mobile vpn: Authentication and encryption algorithms performance," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. IEEE, 2012, pp. 1042–1048.

[10] A. Rashwan, A. Taha, and H. S. Hassanein, "Benchmarking message authentication code functions for mobile computing," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 2585–2590.

[11] M. Dayarathna and T. Suzumura, "A performance analysis of System S, S4, and Esper via two level benchmarking," in *Quantitative Evaluation of Systems*. Springer, 2013, pp. 225–240.

[12] N. A. Carvalho and J. Pereira, "Measuring software systems scalability for proactive data center management," in *On the Move to Meaningful Internet Systems, OTM 2010*. Springer, 2010, pp. 829–842.

[13] M. Tirel, E. O. Sahin, G. C. Silvestre, C. Roche, K. Mıhçak, S. Kesici, N. J. Hurley, N. Gerek, and F. Balado, "Benchmark for multimodal authentication," 2009.

[14] N. Poh, T. Bourlai, J. Kittler, L. Allano, F. Alonso-Fernandez, O. Ambekar, J. Baker, B. Dorizzi, O. Fatukasi, J. Fierrez *et al.*, "Benchmarking quality-dependent and cost-sensitive score-level multimodal biometric fusion algorithms," *Information Forensics and Security, IEEE Transactions on*, vol. 4, no. 4, pp. 849–866, 2009.

[15] S. H. Shaikh, K. Saeed, and N. Chaki, "Performance benchmarking of different binarization techniques for fingerprint-based biometric authentication," in *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013*. Springer, 2013, pp. 237–246.