Dynamic Scaling of Call-Stateful SIP Services in the Cloud

Nico Janssens, Xueli An, Koen Daenen, Claudio Forlivesi

Dynamic Scaling of Call-Stateful SIP Services in the Cloud

Nico Janssens, Xueli An, Koen Daenen, and Claudio Forlivesi

Service Infrastructure Research Dept.
Alcatel-Lucent Bell Labs, Antwerp, Belgium
{nico.nj.janssens,xueli.an,koen.daenen,claudio.forlivesi}@alcatel-lucent.com

Abstract. Many cloud technologies available today support dynamically scaling out and back computing services. The predominantly session-oriented nature and the carrier-grade requirements of telco services (such as SIP services) complicate the successful adoption of dynamic scaling in a telco cloud. This paper investigates how to enable dynamic scaling of these telco services in an effective manner, focusing in particular on call-stateful SIP services. First, we present and evaluate two protocols to transparently migrate ongoing sessions between call-stateful SIP servers. These allow to quickly shutdown call-stateful SIP servers in response to a scale back request, removing the need to wait until their ongoing calls have finished. Second, instead of responding to load changes in a reactive manner, this paper explores the potential value of pro-active resource provisioning based on call load forecasting. We propose a self-adaptive Kalman filter to implement short-term call load predictions and combine this with history-based predictions to anticipate future call load changes. We believe that session migration and call load forecasting are two important elements to safely reduce the operational expenditure (OpEx) of a cloudified SIP service.

Keywords: cloud, telecommunication, elasticity, dynamic scaling, session migration, load prediction, SIP

1 Introduction

Cloud computing has gained substantial momentum over the past few years, fueling technological innovation and creating considerable business impact. Public, private or hybrid cloud infrastructure shortens customers' time to market (new hosting infrastructure is only a few mouse-clicks away), and promises to reduce their total cost of ownership by shifting the cost structure from higher capital expenditure to lower operating expenditure. One of the fundamental features of cloud computing is the ability to build dynamically scaling systems. Virtualization technologies (including XEN, KVM, VMware, Solaris and Linux Containers) facilitate computing services to automatically acquire and release resources. This enables to dynamically right-size the amount of allocated resources, instead of statically over-dimensioning the capacity of such services. Dynamic scaling thus enables to reduce operational costs and to gracefully handle unanticipated load surges, all without compromising the performance and correct functioning of the affected services.

Although the majority of existing (cloud) scaling solutions have been targeted at web and enterprise applications [10, 14, 18, 20], telco services can also benefit significantly from dynamic scaling. To guarantee carrier-grade service execution, telco operators typically over-provision the employed resources to handle sporadic unanticipated load surges (e.g. caused by events with a significant social impact) or anticipated load spikes (e.g. caused by New Year wishes). This reduces their resource utilization ratio and raises their operational cost.

This paper investigates the application of dynamic scaling in telco services, focusing in particular on call-stateful SIP servers. While stateless web applications or RESTful [4] web services can scale back immediately without breaking ongoing interactions, this is not the case for call-stateful SIP servers. Before removing a call-stateful SIP server from an elastic SIP cluster, one needs to ensure that all ongoing sessions processed by that server have ended[1]. To fully exploit the potential of dynamic scaling for call-stateful SIP services, this paper explains how to transparently migrate the processing of ongoing sessions to peer servers. Hence a call-stateful SIP server can be released quickly in response to a scaling back event.

Our second contribution builds upon the observation that successful adoption of dynamic scaling support for telco services highly depends on its ability to preserve the services' stringent availability requirements. Instead of responding to load changes in a reactive manner, this paper explores the value of pro-active resource provisioning based on call load forecasting. We observed that the daily call variations of a local trunk group adheres to recurring patterns. This allows to formulate load predictions (and the consequent decisions to increase or decrease the amount of virtual resources) from a history of load observations. To handle also sporadic unanticipated load surges that significantly diverge from these recurring patterns, we combine history-based forecasting (based on time series spanning multiple days) with limited look-ahead predictions (taking into account only on a few prior observations).

The remainder of this paper is structured as follows. Section 2 provides the required background information on SIP services and discusses related work. Section 3 elaborates on how to transparently migrate sessions between elastic SIP servers. Next, Section 4 presents our algorithms to predict call load variations and simulates a dynamically scaling communication service to evaluate these prediction algorithms. Finally, conclusions and future work are presented in Section 5.

2 Background and Related Work

SIP (Session Initiation Protocol) is an IETF-defined signaling protocol for creating, modifying and terminating sessions (including Internet telephone calls, multimedia distribution and multimedia conferences) between two or more remote participants over Internet Protocol (IP) networks. Although SIP is essentially a peer-to-peer protocol (more details on the protocol can be found in [6, 17]), a SIP telco service includes servers to help routing requests to a user's current location, to authenticate and authorize users for services, to implement provider call-routing policies and to provide extra features to users [17]. Such SIP servers can operate either in a stateless or stateful mode. If *stateless*, a SIP server processes each message as unrelated to any previous messages – hence simply forwarding SIP requests and responses straightaway. Because of their very nature, such stateless servers (like RFC3261 proxies) can be safely added to or removed from a server farm without compromising ongoing calls. A *stateful* SIP server, in contrast, remembers information about each incoming request and any request it

[1] An experimental analysis of Skype® usage [5] indicated that the average length of a Skype® call was 12m 53s, while the longest call lasted for 3h 26m. Although Skype® is not a SIP service, both technologies offer similar Voice over IP (VoIP) services.

sends as a result of processing incoming requests, and uses this information to affect the processing of future messages associated with that request [17]. In the remainder of this section we further clarify the difference between *transaction-stateful* SIP servers, retaining only transaction state, and *call-stateful* SIP servers, retaining both transaction and session state.

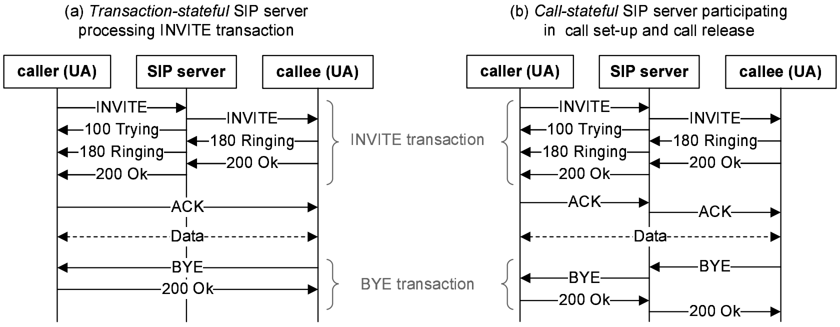

(a) *Transaction-stateful* SIP server processing INVITE transaction
caller (UA)
SIP server
callee (UA)
INVITE
100 Trying
180 Ringing
200 Ok
ACK
Data
BYE
200 Ok
INVITE transaction
BYE transaction
(b) *Call-stateful* SIP server participating in call set-up and call release

Fig. 1. Transaction-stateful vs. call-stateful SIP servers.

Waiting until a stateful SIP server holds no more execution state (and therefore can be removed safely) is only suitable if this condition can be met in bounded time. This is the case, for instance, when stateful SIP servers retain *only transaction state* and *no session state*. As an example, Figure 1(a) illustrates a transaction-stateful SIP proxy participating solely in INVITE transactions. RFC 3261 [17] specifies that the default timeout window of an INVITE and a non-INVITE transaction equals 32 seconds and 4 seconds, respectively. Since ongoing transactions will be canceled if they did not complete after this timeout, a transaction-stateful SIP server reaches a safe removal state in bounded time after preventing the initiation of new transactions.

This is not the case for *call-stateful* SIP servers, such as back-to-back user agents (B2BUAs) or SIP proxies controlling middle boxes that implement firewall and NAT functions. As illustrated in Figure 1(b), call-stateful SIP servers retain session state (like dialogs) during the entire call – that is, from the initiating INVITE to the terminating BYE transaction. Since sessions (in contrast to transactions) typically do not complete in bounded time, waiting until all ongoing sessions have terminated before removing a call-stateful SIP server can significantly delay scaling back operations.

Related work has focused on various aspects of dynamic scaling. The work presented in [8, 11, 12] defines feedback loops to dynamically right-size the amount of provisioned resources. In [18], Seung et al. discuss how to scale enterprise applications over a hybrid cloud (including both private and public resources). The authors of [20], in turn, propose a dynamic provisioning technique for scaling multi-tier Internet applications. Furthermore, today's commercial public clouds (including Amazon Web Service®, Google App Engine® and Heroku®) offer support to automatically scale out and back cloud applications. To the best of our knowledge, however, none of these general purpose solutions take into account the session-oriented nature and the stringent availability requirements of telco applications.

More closely related to our work, [3] explains how IMS functionality can be merged and split among different nodes without disrupting the ongoing sessions or calls. An important difference with our work is that the presented IMS scaling solution is *not* transparent to the SIP UAs. To be precise, SIP UAs need to re-REGISTER and re-INVITE when the IMS functionality has been merged or split among different nodes. Since changing the behavior of SIP UAs complicates the successful adoption of elastic SIP services, our solution seeks to be transparent to both SIP UAs and (non-elastic) SIP servers that belong to different domains.

Finally, we briefly compare our work both with SIP session mobility and SIP handover (as discussed for instance in [2]). SIP session mobility targets the transfer of an ongoing session from one device to another, while SIP handover aims to preserve ongoing SIP sessions when roaming between different networks. In both cases, the UAs initiate and participate in the migration process. This paper, in contrast, presents a solution to migrate the processing of ongoing sessions between SIP servers transparently to the UAs.

3 SIP Session Migration

Safe and transparent migration of SIP sessions between servers can be decomposed into two sub-problems. First, *referential integrity* must be preserved while and after executing a session migration. Since call-stateful SIP servers by nature share their state with their clients (which can be phones as well as other SIP services), these clients are tightly coupled to a specific server during the entire call. Migrating the processing of that particular session to another server, therefore, requires preserving at all times the client's reference to the server that is actually processing its session.

Second, safe and transparent migration of session state from SIP SERVER A to B must be coordinated properly. First, SERVER A must be put into a quiescent execution state. Goudarzi and Kramer describe in [13] that such a quiescent execution state is reached when a service (1) is currently not involved in *ongoing* transactions, and (2) will not participate in any *new* transaction. When applying these prerequisites to call-stateful SIP services, a SIP server reaches a quiescent execution state once (1) all ongoing transactions that belong to the affected SIP dialogs have been completed or terminated, and (2) no new transactions will be started on that server unless to complete other ongoing transactions. The latter occurs, for instance, when a PRACK transaction (to exchange a provisional ACK request [16]) is executed as part of an ongoing INVITE transaction.

To put SERVER A into a quiescent execution state, all SIP requests creating new dialogs (such as INVITE and SUBSCRIBE requests) must be redirected to other SIP servers. Additionally, all requests starting new transactions on confirmed dialogs [17] processed by SERVER A (such as re-INVITE and BYE requests) must be buffered. All other messages, including requests that are sent within an early dialog [17] such as CANCEL and PRACK requests, need to be delivered to SERVER A in order to complete ongoing transactions. Once a quiescent execution state is reached, all remaining session state can safely be captured from SERVER A to be reinstated in SERVER B. Finally, intercepted messages must be released again, but should be redirected to SERVER B.

The remainder of this section presents two stateless "elasticity gateways" to preserve referential integrity while and after migrating SIP sessions. Additionally, we describe and evaluate two protocols coordinating the migration scenario presented above.

3.1 Architectural Overview

We developed a stateless SIP *Client Elasticity Gateway* (CEG) to decouple SIP User Agents (UAs) from the call-stateful SIP servers that process their calls (as illustrated in Figure 2). Configured as the UA's outbound proxy, a CEG conceals the elastic SIP servers from a UA by acting as a single SIP server. It includes load balancing support based on the weight and priority tags of DNS Service (SRV) records and can also be equipped with fail-over support to cope with SIP server crashes. Additionally, the SIP CEG terminates elasticity control messages originating from the elastic SIP cluster, hence concealing the dynamics of the elastic SIP cluster from the UA. We note that traditional load balancing support processes only incoming messages. Since the SIP CEG seeks to control all communication between the UAs and the elastic SIP cluster, it also forwards outgoing messages to the UA. Hence, the CEG can enforce the UA to send back responses to itself instead of to the actual SIP server that previously processed this message. This enables the CEG to transparently redirect requests and responses when the associated dialog has been migrated.

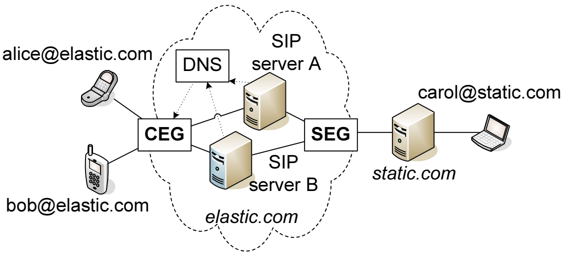


Fig. 2. Architecture of a dynamically scaling (call-stateful) SIP cluster

To achieve this behavior, the CEG combines stateless proxy functionality with registrar support. To enforce that all communication directed to a SIP UA passes the UA's CEG first, the CEG updates all REGISTER requests to replace the UA's contact address with its own - and thus publishes itself as a contact on the client's behalf. This way the CEG will intercept all SIP messages directed to the UA's contact address. To dispatch these incoming messages to the target UA, the CEG can store the UA's original contact address locally, or it can encode this address in the updated contact header. Additionally, when processing non-REGISTER requests, the CEG adds a VIA header and a RECORD-ROUTE header [17] to the request to make sure responses and subsequent requests pass the CEG as well.

As a potential drawback of this architecture, one could argue that these CEGs may become the system's choke point – thus shifting the scaling problem of stateful SIP servers to another SIP component. We want to stress that the functionality of the CEG is stateless and very lightweight – in our current implementation a single CEG consumes approximately 10% of the CPU load a stateful SIP server consumes when processing the same amount of Calls Per Second (CPS). Furthermore, by deploying (multiple) CEGs close to the client[2] instead of only a few CEGs close to the elastic SIP servers, the CEGs have to meet less strict scalability and high-availability requirements. Since only a few UAs depend on their functionality, the impact of a failure is limited, as is the probability of the CEG to become a choke point.

In addition to the SIP CEG, a stateless SIP *Server Elasticity Gateway* (SEG) has been developed to decouple elastic SIP servers from peers that belong to different (non-elastic) domains (see Figure 2). The role of the SEG is similar to the CEG; it redirects

[2] CEGs can be deployed on home gateways, femto-cells or as a separate service on the UAs.

incoming messages to the appropriate server when the associated dialog has been migrated, it terminates elasticity control messages originating from the elastic SIP servers, and it forwards outgoing messages to the target domain (hence concealing the elastic SIP server that actually processed this message and ensuring responses are sent back to the SEG). Although the objectives of the CEG and the SEG are similar, their implementation is slightly different. SEGs do not receive REGISTER requests, for instance, and must be registered with DNS to intercept requests sent to the domain. These and other implementation differences have been the main reason to distinguish between the CEG (which decouples elastic SIP servers from SIP UAs) and the SEG (which decouples elastic SIP servers from peers that belong to different domains).

Finally, we note that in contrast to this application layer solution, network technologies such as MIP [15] or dynamic NAT could be considered as well to transparently redirect messages between SIP servers. By holding together all SIP session migration functionalities at the application layer, however, we seek to limit dependencies to technologies that are not omnipresent – which complicates large scale deployment. Besides, we note that in addition to redirecting messages, these elasticity gateways also participate in the actual session migration process, as we further explain in the next section.

3.2 Migration Protocols

This section introduces two protocols to safely coordinate a session migration between two SIP servers. The first protocol (further referred to as the Gateway Intercept Protocol – GIP) imposes a quiescent execution state by intercepting messages on the elasticity gateways. Figure 3(a) illustrates the various steps of this protocol, exemplified with the migration of an ongoing session from SERVER A towards SERVER B. This migration starts by acquiring the dialog specifications of the affected session (see step 1 in Figure 3(a)), including the addresses of the CEGs and SEGs participating in this session. Next, the Scaling Logic (coordinating the execution of the migration) instructs these CEGs and SEGs to intercept requests starting new transactions on a confirmed dialog (as explained in the beginning of Section 3) and to temporarily buffer these messages in a waiting queue (step 2). The elasticity gateways keep on forwarding all other messages to SERVER A such that any ongoing transaction can complete. Next, SERVER A must be monitored until all ongoing transactions have been completed or terminated (step 3). At this point, a quiescent execution state is reached and the remaining dialog state (as well as all other session state) can safely be transferred from SERVER A towards SERVER B[3] (steps 4 and 5). After the dialog state has been transferred, the CEGs and SEGs are instructed to release all intercepted requests and to redirect them to SERVER B (step 6). If this dialog migration is preceding a shutdown of SERVER A, the latter should also be prevented from receiving dialog-creating requests. This can be accomplished by deregistering SERVER A before executing the dialog migration (step 0). When using DNS to implement load balancing, for instance, removing the records associated with SERVER A prevents the arrival of new dialog-creating requests on that server.

The second protocol (further referred to as the Local Intercept Protocol – LIP) builds upon the same principles of the previous one, but intercepts and buffers requests at the

[3] State migration to transfer dialogs can be implemented by extending the APIs of the affected servers to capture and reinstate state data, or by exploiting a service's high availability support that periodically stores state data to recover from failures (if present).

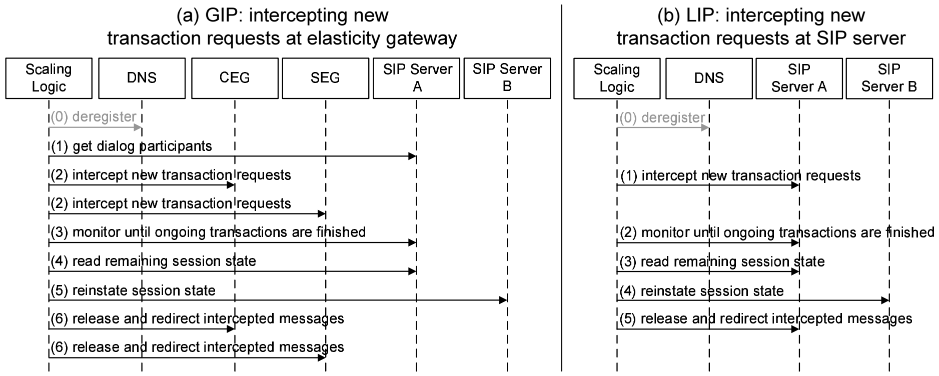

(a) GIP: intercepting new transaction requests at elasticity gateway
Scaling Logic
DNS
CEG
SEG
SIP Server A
SIP Server B
(0) deregister
(1) get dialog participants
(2) intercept new transaction requests
(2) intercept new transaction requests
(3) monitor until ongoing transactions are finished
(4) read remaining session state
(5) reinstate session state
(6) release and redirect intercepted messages
(6) release and redirect intercepted messages
(b) LIP: intercepting new transaction requests at SIP server
Scaling Logic
DNS
SIP Server A
SIP Server B
(0) deregister
(1) intercept new transaction requests
(2) monitor until ongoing transactions are finished
(3) read remaining session state
(4) reinstate session state
(5) release and redirect intercepted messages

Fig. 3. SIP Session Migration Protocols

SIP server instead of the elasticity gateways. As illustrated in Figure 3(b), the protocol starts by instructing SERVER A to intercept requests starting new transactions on a confirmed dialog (see step 1 in Figure 3(b)). Next, SERVER A must be monitored until all ongoing transactions are completed or terminated (see step 2), which indicates that a quiescent execution state is reached and the remaining dialog state (as well as all other session state) can safely be transferred from SERVER A towards SERVER B (steps 3 and 4). After the dialog state has been migrated, SERVER A acts as stateless proxy, forwarding intercepted messages as well as messages that were still in transit during the migration towards SERVER B. We note that although the elasticity gateways are not involved in this session migration process, they are updated indirectly after the Scaling Logic added or removed server instances to/from DNS (step 0).

3.3 Experiments and Evaluation

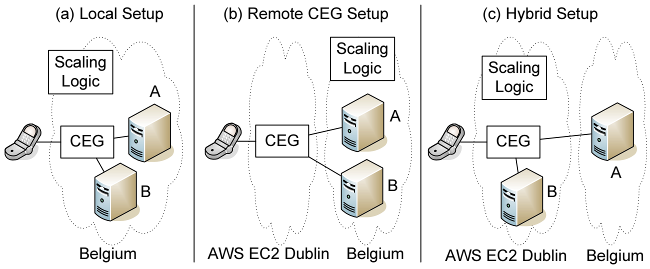
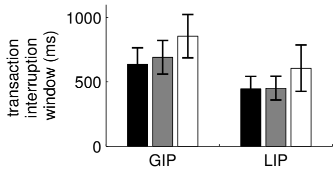


Fig. 4. SIP Session Benchmark Scenarios

In this section we compare the protocols presented above, focusing in particular on their *transaction interruption window*. This interval quantifies the maximum amount of time that requests starting new transactions may be buffered in the course of a migration. Delaying these messages for too long may cause redundant retransmissions or even cancel the affected transaction. To compare the potential transaction interruption window of both protocols, we benchmarked the migration of a single session between two elastic SIP servers in three different settings. A first benchmark was executed while our Scaling Logic (coordinating the execution of both protocols), a single CEG prototype and both elastic SIP servers[4] (Servers A and B) were deployed on a local private cloud platform. This scenario, depicted in Figure 4(a), is further referred to as Local. To measure the impact of deploying a CEG outside this private cloud (which impacts the communication latency between the Scaling Logic and the CEG), we performed a second benchmark while the CEG was running on Amazon's EC2 cloud computing platform in Dublin, Ireland[5]. This scenario, illustrated in Figure 4(b), is further referred to as Remote CEG. Finally, to measure the impact of a session migration when the affected SIP servers are deployed on a hybrid cloud, we performed an additional benchmark with the CEG, Scaling Logic and one elastic SIP server (Server A) running on Amazon's EC2 data center in Dublin, while Server B was deployed on the private cloud platform in Antwerp. This scenario, depicted in Figure 4(c), is further referred to as Hybrid.



	GIP	LIP
Local Setup (black bars)	637 ms	446 ms
Remote CEG Setup (gray bars)	691 ms	452 ms
Hybrid Setup (white bars)	855 ms	607 ms

Fig. 5. Result from benchmarking the transaction interruption window of GIP and LIP.

For each scenario, we benchmarked 200 session migrations using a prototype implementation of both protocols presented above. The results of these benchmarks are depicted in Figure 5. We can deduce from Figure 5 that LIP has a smaller transaction interruption window than GIP. This can easily be explained by the fact that LIP involves only the SIP servers participating in the migration, while GIP includes the elasticity gateways in the migration process as well (see step 2 and 6 in Figure 3(a)). A different deployment of the CEG only impacts the transaction interruption window when using GIP, for the same reason. Finally, to understand the impact of the measured transaction interruption windows one must take into account that a (transactional) SIP entity starts a retransmission timer with a default value of 500 ms when transmitting a message over an unreliable transport protocol. The results shown in Figure 5 indicate that the measured transaction interruption window of GIP may potentially cause a client's retransmission timer to expire once, resulting in a redundant retransmission of the buffered message. When benchmarking LIP, however, the measured transaction interruption window for Local and Remote CEG turns out to be smaller than the default retransmission timeout. Hence, in the absence of significant communication delays between UA and CEG, LIP has the lowest probability to cause redundant message retransmissions. We also note that the actual transaction interruption window can be further reduced by optimizing our prototype implementation.

We conclude this section with some final remarks. First, the benchmark results discussed above may create the perception that LIP is in general a better solution than GIP to implement SIP session migration. This is indeed the case if we focus exclusively on the transaction interruption window of both protocols. One of the main benefits of

[4] The employed prototypes of the elasticity gateways and the elastic SIP servers are developed in Java, using the JAIN-SIP stack version 1.2

[5] The measured average round-trip time between the private cloud platform located in Antwerp and the employed VMs from Amazon's EC2 located in Dublin was around 32 ms.

GIP over LIP, however, is that it can be integrated more easily into existing SIP infrastructure. To apply GIP, existing SIP servers must (1) provide access to the state and specification of the servers' ongoing transactions and dialogs, and (2) enable reinstating the state of migrated dialogs. All remaining support to buffer and redirect messages is handled by separate elasticity gateways. When integrating LIP, in contrast, the affected SIP servers must accommodate this functionality as well.

Finally, instead of intercepting and buffering messages to safely migrate sessions, one can also exploit SIP message retransmissions when using an unreliable transport protocol. In this case, messages are not buffered but become discarded instead. This could be particularly useful when implementing GIP, as the benchmarks indicate that the execution of this protocol may potentially cause a retransmission of these buffered messages anyway.

4 Call Load Forecasting

Migration protocols enable to quickly shutdown call-stateful SIP servers in response to scale down requests, excluding the need to wait until these servers' ongoing calls have finished. In this section, we explore the potential value of pro-active resource provisioning to support dynamic scaling of telco services without compromising their stringent availability requirements.

The call capacity of conventional telephony systems is typically designed to meet the expected Busy Hour Call Attempts (BHCA). Figure 6 depicts the average amount of call attempts per 15 minutes, collected from a trunk group in Brussels from May 2011 until October 2011. Based on these data, we deduce that static peak load dimensioning in this case results into an average call capacity usage of only 50% when averaged over a day. This resource utilization ratio is even lower if the system needs to be dimensioned to handle sporadic unanticipated load surges, such as in case of natural disasters, or anticipated load spikes caused by events with a significant social impact.

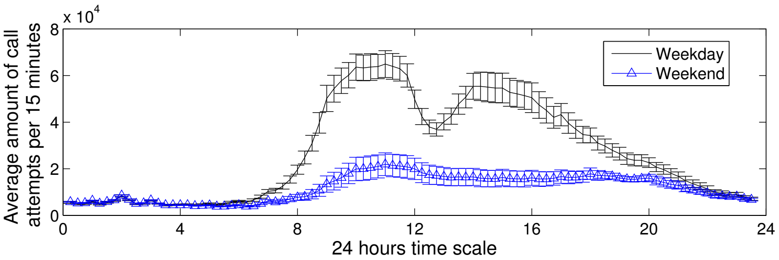


Fig. 6. Average number of call attempts/15 minutes, collected from a local trunk group

Although dynamic scaling enables telco services to optimize their resource utilization ratio, it also increases the risk to compromise their availability requirements. Insufficient resource provisioning to handle load raises, for instance, may cause SLA violations and increases the risk of losing customers. This section explores the potential value of *pro-active* scaling based on call load forecasting to preserve availability requirements while dynamically scaling a telco service.

4.1 Forecasting Algorithms

We present a lightweight limited look-ahead prediction algorithm to forecast short-term call load variations. Furthermore, we combine the short-term forecasting mechanism with history-based forecasting (based on measurements spanning multiple months) to improve the accuracy of call load forecasting by exploiting recurring load variation patterns.

History based predictions. Usage variations of communication systems typically represent iterative patterns, resulting from the end users' daily activity routines. The weekday call pattern shown in Figure 6, for instance, includes a peak in the morning around 10 am (when a business day has started for most employees) followed by another peak around 2 pm (after lunch time). Brown's statistical analysis of a telephone call center shows similar call patterns [1]. Additional to weekdays, recurring call patterns can also be observed on weekend days (although the amount of call attempts typically is much lower than on weekdays). These recurring patterns enable to predict call load expectations for each type of day based on a history of measurements. One possible technique to accomplish this involves the use of a Kalman filter, which is an established technique in control systems for noise filtering and state prediction [7]. For every time k, a Kalman filter is trained using a history of measurements collected on previous days at the same time. Based on today's measurements at time k, this Kalman filter can be used to estimate tomorrow's expected call load at the same time [9].

Limited look-ahead predictions. An important limitation of history based predictions is the inability to handle irregular or unexpected events (such as natural disasters or popular sports events) triggering significant load surges. Short-term forecasting (also referred to as limited lookahead control) aims to cope with such unexpected surges by making predictions based solely on a set of recent measurements. Short-term forecasting is a well-studied subject [8,19]. In this paper, we propose a lightweight *Self-adaptive Kalman Filter* (SKF) to anticipate call surges without the knowledge of history data.

A Kalman filter is a recursive estimator. It estimates a new state $x(k+1)$ based on both the current measurement $z(k)$ and the estimation of the previous state $x(k)$. The call load $x(k+1)$ at time $k+1$ can be described as a linear equation $x(k+1) = Ax(k) + w(k)$ with a measurement $z(k) = Hx(k) + v(k)$, in which A represents the relation between the call load from the previous and the current time. $z(k)$ is the measured value at time k, which is related to the load state $x(k)$ multiplied with a factor H. The normally distributed variables w and v represent the process and measurement noise, respectively. Furthermore, we assume that w and v have zero mean and have variance Q and R, respectively. $\widehat{x}$ and $\widetilde{x}$ are defined as the a priori and a posteriori estimations, where $\widetilde{x}(k) = \widehat{x}(k) + K(k)\,(z(k) - H\widehat{x}(k))$, $\widehat{x}(k+1) = A\widetilde{x}(k)$ and $K(k)$ is the Kalman gain. $\widehat{P}$ and $\widetilde{P}$, in turn, are the a priori and posteriori estimation error variance, where $\widehat{P}(k+1) = A\widetilde{P}(k)A^T + Q$ and $\widetilde{P}(k) = (I - K(k)H)\,\widehat{P}(k)$. Taking all this into account, the Kalman gain is obtained as $K(k) = H\widehat{P}(k)(H^2\widehat{P}(k) + R)^{-1}$. Assuming that w and v have negligible influence on the system, the Kalman gain is dominated by H. We define that H is within the range $(0, 1]$. When H approaches 1, the system trusts the measurement more. When the system under-predicts the load, H should be decreased to proportionally increase the estimation for the next time. Hence, to enhance the accuracy of our prediction system, we propose to let H self-adapt according to the estimation error as

$$H(k+1) = \begin{cases} \mathrm{I}_{e_k<0}\left(H(k)-\tau\right) + \mathrm{I}_{e_k>e_{th}}\left(H(k)+\tau\right) & , 0 \leq H(k+1) \leq 1 \\ 0 & , H(k+1) < 0 \\ 1 & , H(k+1) > 1 \end{cases} \qquad (1)$$

where $e_k = \widetilde{x}(k) - z(k)$ is the estimation error at time k. I_A is an indicator function returning value 1 if condition A is true, while otherwise value 0 is returned. As expressed in equation (1), when at time k the call load is under-predicted we increase the value of H at time $k+1$ with a small pre-defined value τ. Otherwise, if the call load is over-predicted, we decrease the value of H by τ. Reducing H will be more harmful than increasing H since under-provisioning resources might violate the service availability. Hence we decrease H only if the error is higher than a threshold e_{th}.

Hybrid call load forecasting. By using the history call load measurements, we can calculate the mean ν and standard deviation σ for a certain time. In this section, we propose two algorithms that use this information to limit abnormal predictions resulting from limited look-ahead forecasting. Hence, we aim to further improve the prediction accuracy.

Hybrid algorithm 1. At time $k-1$, we perform both a limited look-ahead prediction $\widehat{x}(k)$ and a history based forecasting $\overline{x}(k)$. At time k we calculate $e_s = \widehat{x}(k) - z(k)$ and $e_l = \overline{x}(k)(1+\sigma) - z(k)$. If $e_s < e_l$, the resulting prediction for time $k+1$ relies exclusively on the limited look-ahead prediction, while otherwise the history based forecasting is used. The motivation of this algorithm is to give more credibility to the algorithm that has the lowest prediction error at the current time.

Hybrid algorithm 2. If the previous measurement $z(k-1)$ is within the range $(\overline{x}(k-1)(1-\sigma), \overline{x}(k-1)(1+\sigma))$, the predicted load for time k is set to $\overline{x}(k)(1+\sigma))$. Otherwise, we fall back to a limited look-ahead prediction. This algorithm gives more credibility to the history data. The limited look-ahead prediction is adopted only if the measurement does not fall in the history range.

4.2 Safety Margin

Due to the intrinsic cost and risk of under-provisioning telco services, we apply a safety margin δ to the predicted call load x when calculating the amount of required resources. By provisioning enough resources to handle x times $(1+\delta)$ call attempts, we seek to reduce the possibility of under-provisioning.

4.3 Evaluation

We simulated the behavior of a dynamically scaling communication service to evaluate the prediction algorithms presented above. This simulation uses real-life call attempt measurements, collected from a local trunk group during 66 weekdays (similar to the data depicted in Figure 6). The simulation implements a control function that periodically updates the number of server instances based on the call load prediction. The simulation assumes these instances can be removed quickly, for instance by using the session migration techniques presented in Section 3. To evaluate the forecasting algorithms, the simulation calculates the amount of server instances that are over-provisioned as well as the amount of missing instances to handle the current load (under-provisioning). We define over-provisioning as the ratio between (1) the amount of over-provisioned instances during a single day using call load forecasting and (2) the amount of over-provisioned instances to handle the BHCA of the same day. If the incoming call load is higher than the overall capacity of all provisioned instances, in contrast, a number of requests will be dropped (under-provisioning). We define the Successful Call Processing Rate (SCPR) as the ratio between (1) the total amount of processed call attempts and (2) the total amount of offered call attempts. Both parameters help to understand and evaluate the effectiveness of our forecasting algorithms.

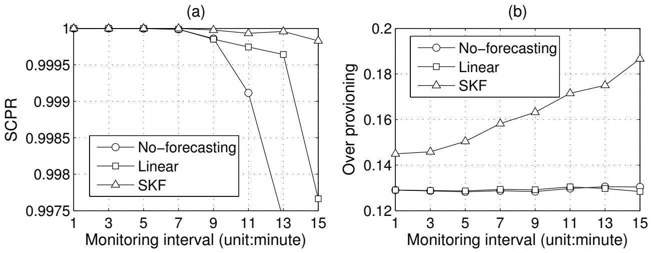


Fig. 7. Limited look-ahead predictions with varying monitoring interval. SKF has initial values $H(1) = 1, \tau = 0.1, \delta = 0.15, e_{th} = 100$.

First, we analyze the correlation between the monitoring interval and the SCPR when using limited look-ahead predictions. We compare our SKF algorithm with limited look-ahead predictions based on linear extrapolation and with a scenario that does no forecasting at all. As illustrated in Figure 7(a), using SKF results into the highest SCPR. Furthermore, for all tested monitoring intervals SKF achieves a SCPR $>$ 99.95%. When the monitoring interval is smaller than 13 minutes, SKF can even achieve a SCPR $> 99.99\%$. These experiments also indicate that SCPR $> 99.99\%$ could be achieved without call load forecasting if the monitoring interval is smaller than 8 minutes. Although using a small monitoring interval indeed enables the system to quickly respond to under or over-provisioning, frequently scaling may compromise the stability of the system as well as the overall OpEx reduction depending on the cost associated with every scaling action [11]. Additional to the SCPR, Figure 7(b) depicts the over-provisioning ratio of the tested limited look-ahead prediction algorithms. Although SKF generates the highest over-provisioning, we can observe that when the monitoring interval is 15 minutes SKF safely reduces the provisioned call capacity to 18.66% of the capacity needed without dynamic scaling.

We also compare linear and SKF predictions with history-based Kalman predictions and width both hybrid call load forecasting algorithms. During these simulations the monitoring interval was set to 15 minutes. The measured SCPR and over-provisioning rate are depicted in Figures 8(a) and 8(b), respectively. From these results we can deduce that only SKF and Hybrid 1 can realize a SCPR above 99.9%, while Hybrid 1 generates less over-provisioning than SKF. To further compare these two algorithms, we depict their cumulative distribution function (cdf) in Figure 9 by using a different buffering ratio δ. It is easy to understand that increasing δ results in a higher SCPR. Based on this experiment we can also observe that the performance of SKF and Hybrid 1 is very similar for all tested δ values.

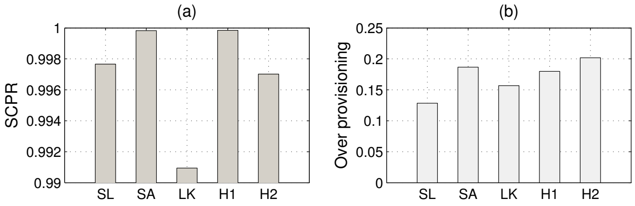


Fig. 8. Call load forecasting comparison, $\delta = 0.15$, SL: linear extrapolation, SA: SKF predictions with $e_{th} = 100$, LK: history-based predictions with Kalman filter, H1: Hybrid 1 with $\tau = 0.1$, H2: Hybrid 2 with $\tau = 0.1$.

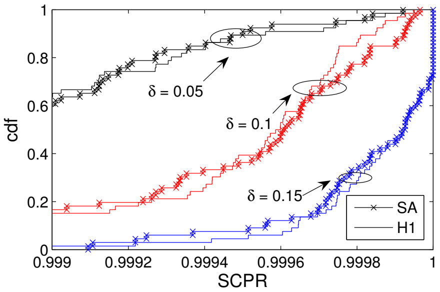


Fig. 9. CDF for SCPR comparison between SA and H1 by using different δ, $e_{th} = 100$.

In the previous simulations we restricted the capacity of a single instance to 100 calls per minute due to the low BHCA of the employed trunk group measurements (around 5k calls per minute). When increasing this capacity to 500 calls per minute, we observe similar results in terms of SCPR and over-provisioning rate. The only difference worth mentioning relates to the safety margin δ. Since increasing the capacity of a single instance reduces the probability to cause resource under-provisioning, it also decreases the impact of safety margin δ. Our simulations indicate that when the instance capacity is increased to 500 calls per minute, δ can be set to 0 to achieve similar results as shown in Figure 8 (which uses $\delta = 0.15$).

5 Conclusion and Future Work

In this paper, we investigate the feasibility of applying dynamic scaling to cloudified telco services. We present and evaluate two protocols for transparently migrating ongoing sessions between call-stateful SIP servers. This enables to quickly release a server in response to a scale down request, instead of unnecessarily wasting resources by waiting until all ongoing sessions on that server have ended. Additionally, we propose a self-adaptive Kalman filter to implement limited look-ahead call load predictions and combine this with history-based Kalman predictions to reduce the amount of resource over-provisioning. We believe that both techniques enable to reduce the OpEx of a cloudified SIP service and to increase the resource utilization ratio of a telco cloud provider without compromising service availability.

Future work focuses on how to protect a dynamically scaling SIP service against malicious load surges. Additionally, we are studying the influence of server capacity variations caused by the underlying virtualization technology on the employed SIP scaling feedback system. By combining these results with the findings presented in this paper, we seek for dedicated SIP scaling solution to optimize the amount of employed cloud resources in a safe manner.

References

1. Brown, L., Gans, N., Mandelbaum, A., Sakov, A.: Statistical analysis of a telephone call center: a queueing science perspective. Journal of the American Statistical Association (2005)
2. Chen, M.X., Wang, F.J.: Session mobility of sip over multiple devices. In: Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities. pp. 23:1–23:9. TridentCom '08 (2008)
3. Dutta, A., Makaya, C., Das, S., Chee, D., Lin, J., Komorita, S., Chiba, T., Yokot, H., Schulzrinne, H.: Self organizing IP multimedia subsystem. In: Proc. of the 3rd IEEE Int. Conf. on Internet multimedia services architecture and applications. pp. 118–123. IMSAA'09
4. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
5. Guha, S., Daswani, N., Jain, R.: An experimental study of the skype Peer-to-Peer VoIP system. In: 5th International Workshop on Peer-to-Peer Systems. Microsoft Research (2006)
6. Hilt, V., Widjaja, I.: Controlling overload in networks of SIP servers. In: IEEE International Conference on Network Protocols, 2008. ICNP 2008. pp. 83–93 (Oct 2008)
7. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transactions of the ASME, Journal of Basic Engineering pp. 35–45 (1960)
8. Kusic, D., Kephart, J.O., Hanson, J.E., Kandasamy, N., Jiang, G.: Power and performance management of virtualized computing environments via lookahead control. In: Proceedings of the 2008 International Conference on Autonomic Computing. pp. 3–12. ICAC '08 (2008)
9. Li, J., Moore, A.: Forecasting web page views: methods and observations. Journal of Machine Learning Research 9, 2217–2250 (Oct 2008)
10. Lim, H.C., Babu, S., Chase, J.S., Parekh, S.S.: Automated control in cloud computing: challenges and opportunities. In: Proceedings of the 1st workshop on Automated control for datacenters and clouds. pp. 13–18. ACDC '09 (2009)
11. Lin, M., Wierman, A., Andrew, L.L.H., Thereska, E.: Dynamic right-sizing for power-proportional data centers. In: Proceedings IEEE INFOCOM 2011. pp. 1098–1106 (2011)
12. Mao, M., Li, J., Humphrey, M.: Cloud auto-scaling with deadline and budget constraints. In: 11th IEEE/ACM International Conference on Grid Computing (Grid 2010) (2010)
13. Moazami-Goudarzi, K., Kramer, J.: Maintaining node consistency in the face of dynamic change. In: Proceedings of ICCDS'96. pp. 62–69 (1996)
14. Padala, P., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Salem, K.: Adaptive control of virtualized resources in utility computing environments. SIGOPS Oper. Syst. Rev. 41, 289–302 (March 2007)
15. Perkins, C.: RFC 5944:IP Mobility Support for IPv4, Revised (2010)
16. Rosenberg, J., Schulzrinne, H.: RFC 3262:Reliability of Provisional Responses in Session Initiation Protocol (SIP) (2002)
17. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: RFC 3261: SIP: Session Initiation Protocol (2002)
18. Seung, Y., Lam, T., Li, L.E., Woo, T.: Cloudflex: Seamless scaling of enterprise applications into the cloud. In: Proceedings IEEE INFOCOM 2011. pp. 211–215 (april 2011)
19. Trudnowski, D.J., McReynolds, W.L., Johnson, J.M.: Real-time very short-term load prediction for power-system automatic generation control. IEEE Tran. Control Systems Technology 9(2), 254–260 (2001)
20. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P.: Dynamic provisioning of multi-tier internet applications. In: Proceedings of ICAC 2005. pp. 217–228 (june 2005)