Auto-scaling Emergency Call Centres using Cloud Resources to Handle Disasters

Srikumar Venugopal[1] Han Li[1] Pradeep Ray[2]

[1]School of Computer Science and Engineering,
University of New South Wales, Australia
{hli,srikumarv}@cse.unsw.edu.au

[2]School of Information Systems, Technology and Management
University of New South Wales, Australia
p.ray@unsw.edu.au

Technical Report
UNSW-CSE-TR-1107
May 2011

Abstract

The fixed-line and mobile telephony network is one of the crucial elements of an emergency response to a disaster event. However, frequently the phone network is overwhelmed in such situations and is disrupted. It is not cost-effective to maintain an over-provisioned IT infrastructure for such rare events. Cloud computing allows users to create resources on-demand and can enable an IT infrastructure that scales in response to the demands of disaster management. In this paper, we introduce a system that uses the Amazon EC2 service to automatically scale up a software telephony network in response to a large volume of calls and scale down in normal times. We demonstrate the efficacy of this system through experiments based on real-world data.

1 Introduction

Natural disasters demand immediate, targeted and large-scale response from the government and relief agencies so that the loss of life and the disruption to the community is minimised. In the connected world of today, Information Technology (IT) plays a central role in disaster response and emergency management [2]. Therefore, it is imperative that IT systems be robust and scalable enough to meet the challenges of disaster response. Frequently however, the IT infrastructure is overwhelmed by the load of managing the disaster response and fails. However, it would be impossible for agencies to maintain a large IT infrastructure that is over-provisioned to respond to infrequent and unpredictable calamities. Therefore, the need for such agencies is to have a infrastructure that can scale out to respond to emergencies effectively and then shrink back in times of normal operation.

Cloud computing [6, 1] is based on enabling users to access applications, platforms and even the underlying computing infrastructure as services. The dominant pricing scheme adopted by cloud computing providers is that of pay-as-you-go, that is, the users only pay for the services for the time that these have been used. Also, users can add or remove capacity on demand using a self-service interface, usually through a website or an Application Programming Interface (API) supplied by the provider. Thus, cloud computing makes it possible to implement an IT infrastructure for disaster response that can scale in response to increasing demands of the relief operations.

One of the elements of an IT infrastructure is the communications network. The traditional fixed line and mobile telephony networks are considered highly dependable and reliable but are occasionally overwhelmed in the face of natural disasters [11]. For example, in the immediate aftermath of a disaster, the local emergency call response center is swamped with calls from the affected population and comes under severe strain. Failure of emergency communications is a feature in many disasters around the world, such as Hurricane Katrina in 2005 [8] and the fires in the state of Victoria, Australia in 2009 [7].

Extending the telephony infrastructure is expensive, even more so in developing nations which suffer a greater loss during natural disasters. Fixed-line telephony is currently undergoing a fundamental transition from a circuit-switched network to an packet-switched one. In the latter, the voice traffic is delivered using the Internet and is therefore known as VoIP (Voice over IP) as well.

Currently, the dominant protocol used for signalling and control in VoIP is the Session Initiation Protocol (SIP) [15]. In this paper, we use the features of SIP and resources from the Amazon Elastic Compute Cloud (EC2)[1] service to implement a mechanism that automatically scales a VoIP-based call centers to respond to a disaster or an emergency. Our primary contribution has been to develop a control system that automatically adds (or removes) capacity from cloud providers in response to increasing (or decreasing) call volumes. We evaluate the framework through tests involving call volumes based on actual data from the Victorian fire disaster and demonstrate the benefits derived from this approach.

The rest of this paper is organised in the following manner. In the next section, we will discuss how cloud computing can be used to scale emergency call

[1]http://aws.amazon.com/ec2/

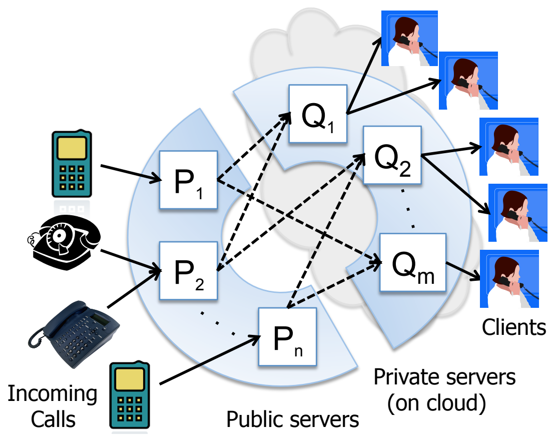


Figure 2.1: An auto-scaling emergency call centre.

centers. Section 3 describes the components of the system while Section 4 details the operational aspects. We report the results of our experimental evaluation using actual cloud resources in Section 5. We compare our approach to those in the literature in Section 6. Finally, we present our conclusions and discuss some of our thoughts for future work.

2 Scaling via clouds

A disaster management agency or non-governmental organisation (NGO) operates a call centre that accepts calls from telephones connected to the public switched telephone network (PSTN) and to the cellular (mobile phone) network. The connections from PSTN to the agency's internal telephone network are handled by devices known as Private Branch eXchanges (PBXs). While PBXs used to be hardware devices, recent developments have brought about PBXs implemented in software that can be installed on any commodity machine. Asterisk[1] is one such software that is open-source and can be easily installed on computers running Linux, MacOSX or Windows operating systems. Asterisk also supports VoIP through SIP and can function as a gateway between the PSTN and the IP-based internal phone network.

Initially, Asterisk is installed on computers belonging to an emergency response agency. These PBX installations, thereafter called *public servers*, serve calls coming from PSTN or mobile networks and are configured to handle the average volume that the agency receives. These calls are handled by operators connected to the PBX via internal phones. Hereafter, we term these phones as *clients*.

As the agency may not have more server capacity, this expansion can be achieved by installing Asterisk on virtual machines (VMs) sourced from cloud providers on demand. But, the callers are still only able to contact through lines connected to the public servers, and hence, the new instances are therefore termed as *private servers*. Calls are routed to the private servers transparently and the call centre is able to cope with the increased demand.

[1]http://www.asterisk.org/

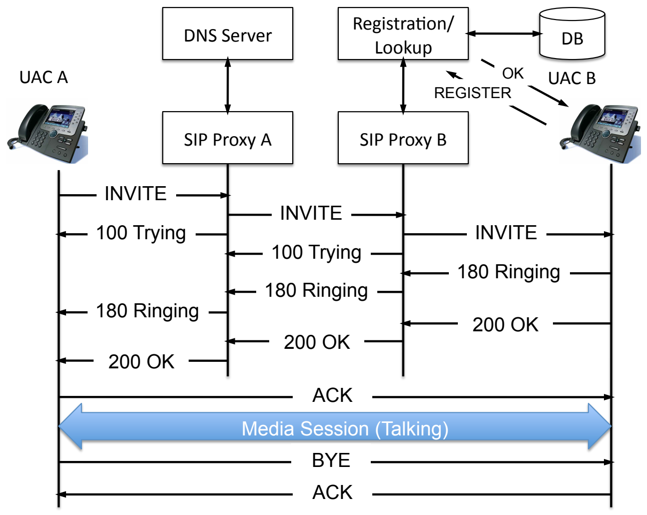


Figure 3.1: Set up and teardown of a SIP call.

Disasters are sudden events and therefore, the phone network should be able to scale up or down on-demand without human intervention so that it can dynamically react to the changing environment as quickly as possible. This property of the system wherein its size increases and decreases dynamically is termed *elasticity*, or *auto-scaling* [21]. Figure 2.1 illustrates the evolution of the phone network under auto-scaling. However, this ability introduces the following challenges :

- The need to react quickly as well as avoid overreactions through positive feedback loops. Also, when the volume of calls drop, the system has to shrink down to the original set of public servers in order to save on infrastructure costs.
- To maintain a consistent quality of service, represented by the ability to serve as many calls as possible with a limit on the maximum time a call has to wait.
- To transparently connect and disconnect PBXs to and from the network without affecting the performance of call handling.

3 The Auto-scaling System

3.1 Call Handling in SIP

SIP is a lightweight, stateless protocol for controlling voice and audio-based communication over IP networks. SIP is only used to setup a multimedia session between two endpoints while the actual voice and video traffic is transported using other protocols. The most important element of a SIP network is a *User Agent (UA)* that either creates or terminates a SIP session and sends or receives SIP messages. In the course of a session, the UA can assume the role of a Client (or UAC) or of a Server (UAS) or both at the same time. The UAC is

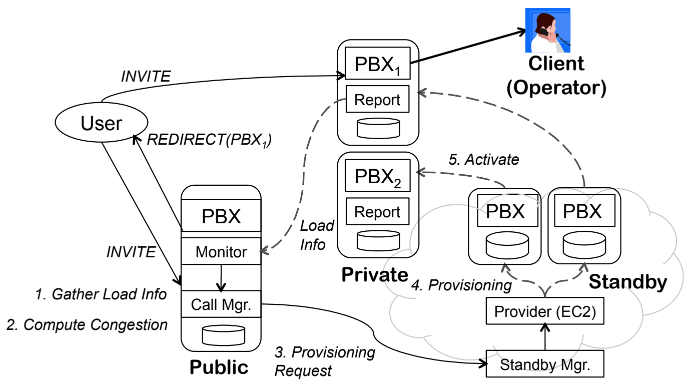


Figure 3.2: Auto-scaling system components and their operation.

implemented in SIP phones, either in hardware (e.g., dedicated VOIP phones) or in software (e.g., softphones such as Ekiga on Linux). Other roles that resources can have in a SIP network are: *proxies* that take care of call routing, authentication and authorization of users, and enforcing network policies; *registrars* that accept registration requests and maintain location information; and *redirect servers*. The redirect server is a user agent server directing the client to contact an alternate set of servers. The redirect server allows proxy servers to direct SIP session invitations to external domains. It is important to note that an element can assume more than one role at any time in the network.

Figure 3.1 shows the interaction between various elements of a SIP network for setting up a call between two UAs. SIP network elements communicate with each other via request-response message pairs. A `REGISTER` request registers contact information for a UAC with a UAS. A UA sends an `INVITE` message with a SIP URI to initiate a SIP session with another user or service. The outgoing proxy looks up the destination via DNS or other methods and sends it to the inbound proxy on the destination node. The latter refers to the registrar and location service to discover the destination UA which sends a `200 OK` signal if it has accepted the call. The two UAs then agree on a protocol to set up a peer-to-peer media session. To terminate the call, one of the UAs sends a `BYE` message.

A redirect server accepts SIP `INVITE` requests and returns redirection (`30x`) responses that direct the client to contact an alternate set of SIP addresses. The caller UA must then send an `INVITE` request to the alternate addresses. Redirect servers are stateless while proxy servers are mostly stateful. This means that SIP redirects can be used as a simple means for load-balancing calls among a set of proxy servers.

The call-center in Figure 2.1 is therefore realised through a combination of SIP redirect and proxy servers. Initially, the public servers are PBXs that act as SIP proxies and registrars, meaning existing clients are registered with the public servers. In the event of a disaster, as the call volume goes up, new SIP proxies are introduced as private servers, and new clients (operators) are registered with them. The public servers use SIP redirects to route calls to the private servers. The following sections detail the design of the individual components of the system.

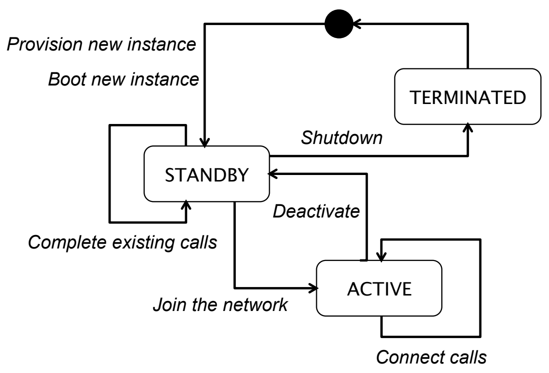


Figure 3.3: State Diagram for a private server.

3.2 Servers

Figure 3.2 shows the auto-scaling phone infrastructure and the control flow for handling incoming phone calls. The servers, whether private or public, share the same software base that is saved in a Virtual Machine (VM) image. A machine can assume the role of a public or private server whenever required. Initially, there can be multiple public servers representing the phone numbers that callers can dial into. One of the public servers assumes the role of a manager. A Distributed Hash Table (DHT) provides the lookup functionality for the network of public and private servers.

A public server is bound to a static IP address and acts as a SIP redirect and proxy server for inbound and outbound calls in an organisation. It is responsible for load balancing by diverting incoming calls to appropriate private servers. A public server that is a call manager also determines when to provision a new server to take care of excess load. Both of these are accomplished on the basis of load information gathered from other servers by the monitoring component. A private server acts only as a SIP proxy server that connects calls redirected by the public servers. Each server maintains a buffer used to queue calls if all operators are busy. The private server joins the network and announces its availability. It calculates the congestion at its end and reports this value to the network periodically.

The call centre capacity is increased by adding more private servers dynamically. This would involve starting new VM instances to host new PBXs. However, the time taken for a VM to boot up and join the network (approx. 3 - 5 minutes) would significantly impact the speed at which the call centre can respond to rapid increases in call volumes. Hence, there is a provision for servers to be kept on standby so that they can be switched into the network on-demand.

Figure 3.3 shows the state transitions for a private server. When a server is provisioned, that is the VM instance is created and booted, it is added to the standby pool. The standby manager activates the server on a signal from the call manager. The server joins the network and receives calls from the public servers. On deactivation, the server is returned to the standby pool. While calls that are active on it are allowed to complete, those that are queued on the server are redistributed to other active servers. The server no longer accepts calls or reports its load to the network. The server may be made active again or will be shut down in order to reduce costs. This is determined by the standby manager and is covered in more detail in Section 4.2.

Each server is initialised with a preset maximum time for which calls can be queued while waiting for an operator. This measure, called *maximum queue time*, influences the quality of service that the call centre network is able to deliver to the callers. As will be shown in Section 4.1, this measure also determines the rate at which the network will expand during the course of operation. The manager can also broadcast a changed queue time to all the servers during the course of operation of the network. For example, when a disaster strikes, the queue time could be reduced so that emergency calls are answered faster and subsequently, it can be increased to meet normal operating requirements.

4 Auto-scaling in Operation

The steps for auto-scaling at a high level are shown in Figure 3.2 and are as follows:

1. Every server computes the load at its end and advertises it to the network. The manager uses these measures to compute the overall congestion (Steps 1 and 2).
2. If the congestion is too high, the manager requests the standby manager to provision a new private server (Step 3).
3. The standby manager periodically monitors the size of the pool and signals the provider to create a new instance if it's falling short. When it receives a new provisioning request from the manager, it activates one of the machines in the pool. The new private server then joins the network (Steps 4 and 5).
4. For the next incoming call, a public server sends a SIP redirect to the caller to contact an available private server. The caller then sends a SIP invite to the private server and the connection is established. If the server is busy, the caller is put into a queue by the private SIP proxy server.

The next few subsections will present in detail how these operations are carried out.

4.1 Congestion Management

The call manager has to periodically examine the state of the network and determine if a new PBX instance has to be provisioned or if an existing PBX instance has to be deactivated. Each server computes a congestion index at its end based on the sigmoid function:

$$y = f(x, T_q, T_h) = 1 - e^{-\left(\frac{kT_h x}{(2T_q + T_h)N}\right)} \tag{4.1}$$

where x is the sum of queued and connected calls at the server, T_h is the average time for which a call is answered (talk time), T_q is the maximum queue time, N is the number of clients (operators) connected to the server and k is a

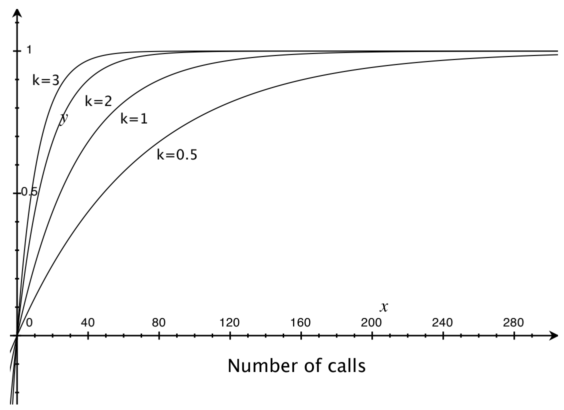


Figure 4.1: Congestion Calculation Function.

scaling coefficient. This function belongs to the class of sigmoid functions that have been used heavily in artificial neural networks for mapping outputs to a consistent scale [20]. This function maps the variables that determine demand to a scalar value that is used to predict the capacity required to handle the load.

Figure 4.1 shows curves of the congestion index against an increasing number of calls on the x-axis for $N = 10$ clients, $T_q = 300$ seconds, $T_h = 240$ seconds, and for different values of k. The congestion index is dimensionless, and lies between 0 and 1 for positive values of all parameters.As shown in Figure 4.1, when the call volume increases, the index could rise rapidly or slowly towards 1 depending on the value of k. Each private server periodically reports this value to the DHT network, hashed by its IP address.

The call manager performs a DHT lookup to obtain the congestion indices of all the servers in the network. There are two thresholds associated with the index. If a majority of the congestion indices are above the upper threshold, then the call manager determines that the network is congested. In response, it initiates the creation of a new private server by signalling the standby pool manager. If a majority of the congestion indices are below the lower threshold, then the phone network is deemed over-provisioned and the server with the lowest congestion index is signalled to deactivate.

4.2 Standby Pool Management

The standby manager interfaces with the cloud provider and determines the number of instances that should be kept provisioned to meet the needs of the call center. It has two objectives, to be cost-effective in using the infrastructure, and to respond quickly to any request for new servers. The manager's strategies are determined by the billing policies of the provider. For instance, Amazon EC2 compute services are billed on an hourly basis. Therefore, if a server has been up for just over an hour, then it makes sense for the server to be available for the next hour as it does not incur extra cost until the next billing interval comes up.

The standby manager periodically computes the *uptime* for each instance in the pool. The uptime is the interval since the time the instance was booted up and available. It then calculates the difference between the next billing interval for a server and its uptime (denoted by ΔT), and shuts down the server if this difference is less than 15 minutes. This is to account for the time taken to shut down the server before the next billing interval starts. To illustrate, a server which has an uptime of 1 hour and 20 minutes (i.e. $\Delta T = 40$ minutes) will be allowed to stay while that with an uptime of 46 minutes will be signalled to shut down immediately.

When the standby manager receives a request for a private server, it will select the instance with the largest ΔT. A server with a smaller ΔT is likely to be a candidate to be shut down sooner. In this manner, the billing cycle is exploited to lower the cost of scaling the system. The manager periodically checks if the pool is empty and if so, creates a new instance so that one standby server is available. However, if no requests for a private server are received for a long time (e.g., 24 hours), then the standby pool is allowed to remain empty after all instances are shut down.

4.3 Role of the DHT

Our system employs the Kademlia DHT [13] which plays an important role in ensuring reliability and scalability of the network. The private servers calculate the congestion index at their end and push this data into the DHT network. The public servers perform a DHT lookup periodically. When a call is received, the public servers use the congestion data in order to redirect the call to the least congested private server. The effect of this is to avoid an exponential increase in the number of messages with the increase in the size of the network. Also, if a private server leaves the network, the performance of the lookup is not affected. The public servers can also determine the status of all private servers via Kademlia PING message and avoid routing calls to deactivated servers.

4.4 Operating Issues in Implementation

The call volume is represented as the number of active channels in Asterisk. A call, whether queued or active, occupies one active channel. The call volume data is queried by the private server in order to compute the congestion. This is done by querying the Asterisk database directly as well as the PBX itself. The Asterisk Manager Interface allows a client program to connect to an Asterisk PBX instance and issue commands or read events over a TCP/IP stream. It is particularly useful when trying to track the state of a telephony client inside Asterisk, and directing that client based on custom and possibly dynamic rules. For example, the Asterix configuration was modified dynamically so that any call that arrived at a private server but was not able to be allocated to an operator was placed in a queue for a specific time period (T_q).

Asterisk PBX, along with our custom auto-scaling software, was installed into a Fedora Linux image on Amazon EC2. Since an EC2 instance does not have any audio output hardware, we had to use a dummy kernel audio module to enable the talk session. This base image was used to create the VM instances to host both public and private servers. The standby manager uses the Amazon Web Services Java API[1] to launch VMs when the pool is empty, and to terminate VMs when required. Our framework uses Mojito[2], an implementation of

[1]http://aws.amazon.com/java/

[2]http://wiki.limewire.org/index.php?title=Mojito

Kademlia DHT in Java, for peer-peer communication. Nodes in Mojito communicate with each other via User Datagram Protocol (UDP), and therefore, each private server reports its congestion index at least thrice so as to guard against packet loss.

Asterisk uses the PostgreSQL database for storing persistent data such as call records, and user agent registrations. When a private server instance is terminated, all the data contained in it, including the information stored in its database, is lost. This data is important for evaluating the state of the network. Therefore, when the signal to terminate is received at a server, the first task is to transfer the database contents over to the call manager before commencing the shutdown process.

5 Experimental Evaluation

5.1 Setup

Automated testing of the auto-scaling phone network is challenging as one needs to duplicate the conditions in an emergency call centre at the time of a disaster. In particular, we needed to simulate the call traffic during disaster occurrence and its aftermath. Also, the calls are answered by human operator for varying time periods.

For these set of experiments, a single public server was started to represent the initial status of the emergency call centre. The standby manager was co-located on the public server as well. As described in the previous sections, the standby manager initiated new private servers using the Amazon AWS API. We used the EC2 small (m1.small) instance type which represented a machine with 1 processor core, 1.7 GB of memory and 160 GB of storage. Each server was statically initialised with 10 software VoIP phones (or softphones) as clients.

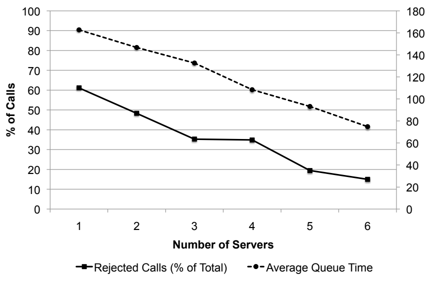
Linphone[1], a Linux softphone implementation, was used as a SIP User Agent in our experiments. Linphone was installed on Linux machines in the UNSW Computer Science Labs to emulate callers, and on the EC2 private server images to emulate the clients or operators connected to the PBXs. We were able to emulate a conversation by controlling Linphone to "answer" a call for a specific period of time by setting up a SIP media session.

Two sets of experiments were performed. In the first set, the auto-scaling was turned off and a static number of private servers were used for each iteration. This attempted to discover the effect of increasing the network size on the call centre's quality of service. The second set of experiments were conducted under dynamic conditions and were used to understand the behaviour of the network under auto-scaling. We modeled these experiments based on the data obtained from the Victorian fires [7].

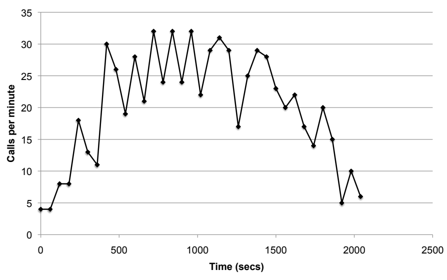
5.2 Static

For this experiment we generated an input data set of 700 calls offered in a space of 30 minutes which corresponds to the call rate at the peak of the crisis. The peak number of calls per minute increased from 4 to 40 calls per minute and decreased back according to a normal distribution. The call duration, or the

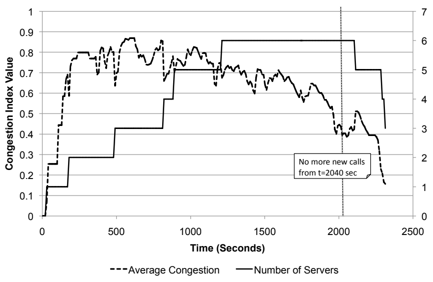
[1]http://www.linphone.org/



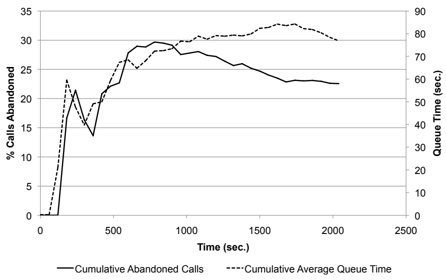
(a) Static Evaluation Results



(b) Input Call Rate



(c) Effect of Congestion Index



(d) Cumulative Performance

Figure 5.1: Evaluation Results

sum of the time spent in the PBX queue and in "conversation", was randomly generated between 180 and 600 seconds using a normal distribution. The input data set was generated in advance and the same set was used in each iteration of the experiments. If a call could not be connected to an operator, then it was queued for up to a maximum queueing time of 180 seconds (3 minutes), after which it was disconnected and considered abandoned. The value of k for the congestion management function was set to 2.75 ensure a rapid reaction to increasing call volumes.

In this case, each iteration was carried out with a fixed number of private servers. Figure 5.1a shows the graphs of abandoned calls and average queueing time generated for each iteration of the experiments. The results show a steady improvement in the handling of calls with the expansion of the network with an approximately 9% drop in the percentage of abandoned calls and 17.8 second drop in the average call queueing time with the introduction of one server at a time.

5.3 Dynamic

In this case, the auto-scaling capability was turned on and the system was allowed to provision private servers to meet the increased number of calls. We used the same volume of 700 calls but the call duration was varied from 60 to 360 seconds. The value of k was retained as 2.75 and the maximum queueing time as 180 seconds. This also implies that a call which was queued at the private server and had a duration less than 180 seconds could be disconnected by the calling phone before the server could allocate an operator. This would simulate the real-world behaviour of a caller "hanging up" before being allocated to an emergency call operator. The calls were offered over 2,040 seconds (34 minutes) according to a normal distribution. The input call rate (number of calls received per minute) over the experiment period is shown in Figure 5.1b. After calls stopped arriving, the existing calls that were connected or queued were allowed to complete. This was to test the scaling down property of the system.

The behaviour of the system is determined by the average congestion index computed periodically by the call manager. Figure 5.1c shows the graph of the average congestion index overlaid with the graph of the number of servers in the system. As can be clearly seen, the index closely tracked the increased demand on the system. Consequently, the new private servers were created to deal with the demand. Every time a new server was introduced, there was a corresponding drop in the congestion index. But the index rose again thereafter. However, once the system size reached 6 private servers (at 1,250 seconds), the congestion index started dropping even though the call demand remained high. After 2,040 seconds, when there were no new calls, the congestion index dropped rapidly and the system started deactivating servers.

Figure 5.1d shows the cumulative performance of the system over the experiment period in terms of the average queueing time and the ratio of abandoned calls to total calls. We chose the cumulative measurement as both these measures were dependent on the calls that had arrived not only at a particular time but also in the intervals prior to that. As can be seen from the graphs, both the percentage of abandoned calls and the average queueing time increase sharply until the system provisions enough servers to take care of the influx. With 5 servers (Figure 5.1c), both the measures plateau and with one more server, the percentage of abandoned calls decreases substantially.

Overall, 77.3% of the calls were connected and completed in this experiment. Nearly 7% of the abandoned calls were caused by the callers hanging up and the rest due to exceeding the queue time at the private servers. Among the servers, the lowest acceptance percentage was 71% and the highest was 95%. The variation is due to the distribution of calls with different answering times at different servers. A server that receives longer calls is prone to longer call queues leading to high abandonment rates. The average call queueing time for the entire experiment was 85.35 seconds which is substantially lesser than the preset maximum queueing time of 180 seconds.

6 Related Work

The importance of IT has prompted new research in ensuring resilience of the infrastructure to cope with the demands of disaster response. Arnold et al. [2] discuss challenges at all levels of IT infrastructure – including communications, security, applications and integration of humans – in such situations. The authors also suggest solutions such as peer-peer application architectures and adhoc routing to meet these challenges. Manoj and Baker [12] highlight the difficulty of introducing new communication systems into disaster-struck areas where people are dependent on existing infrastructure and recommend using existing networks to deliver disaster relief services. While there are ongoing efforts to introduce support for emergency communications in VOIP [14], the research in this paper does not focus on this aspect. Instead, we aim to autoscale SIP-based telephony so as to increase the capacity of an emergency call centre to deal with increased volume of calls following a disaster.

SIP itself is mostly decentralised except for a few elements such as registration [4]. Digium, Inc., the developers of Asterisk, have proposed a more decentralized VoIP protocol called IAX (Inter-Asterisk eXchange) [19]. Another proposed protocol, Distributed Universal Number Discovery (DUNDi) [18], aims to bring a DNS-like system to VoIP phone numbers. However, DUNDi and IAX depend on centralised directories for maintain caller and peer information within a single domain. Other efforts have led to P2P-based SIP (P2P SIP) systems where information about SIP peers is maintained in a Distributed Hash Table rather than a centralised database [5, 16, 9]. Skype[1] is a well-known VoIP provider that uses a combination of SIP and a proprietary P2P protocol.

Others have considered aspects of scalability and fault-tolerance of SIP systems in different ways. Gurbani et al. [10] developed analytical models for characterising the performance and reliability of SIP networks and evaluated these using accepted benchmarks for PSTN networks. They conclude that SIP can give equivalent performance and is equally reliable to PSTN if the SIP servers are replicated. Singh and Schulzrinne [17] address failover of SIP servers through a combination of techniques such as Domain Name Service (DNS) records, IP address takeover and database replication. Balasubramaniyam et al. [3] introduce SERvartuka, a system that improves SIP scalability by executing a SIP session across 2 servers where one of the servers performs stateless part of SIP operations (e.g., lookup) while stateful operations, such as maintaining call records, are offloaded to the other server. However, auto-scaling not only requires decentralisation, but also the ability to add and remove nodes from the network without ill-effects on the participants or network performance. This has not been achieved for VoIP-based systems yet.

Auto-scaling or elasticity is considered as one of the differentiators between cloud computing and related paradigms. Nearly all cloud service providers such as Amazon Web Services[2], Google App Engine[3], and Microsoft Azure[4] have elastic computing features that can be leveraged by developers who use their APIs. However, auto-scaling has side-effects that can be disadvantageous [1]. Firstly, the developer has to ensure that their distributed applications are able to take advantage of the auto-scaling feature. That is, the applications have to be designed to be decentralised and scalable. Secondly, if the triggers for auto-scaling are not set up correctly or if there's a positive feedback loop, the developer can end up incurring a large cost due to inappropriate invocation of new instances. Our system is a decentralised implementation specific to a SIP-based VoIP infrastructure and avoids positive feedback via a conservative congestion function.

There are installations of Asterisk PBX that have been customised to run on resources leased from Amazon EC2 but there is none that we are currently aware of that scales automatically to accommodate a large volume of calls. Our approach integrates the SIP protocol and cloud provisioning using peer-to-peer mechanisms. Using a custom congestion management function, our system is able to scale up to meet excess demand and then scale back down when the

[1]http://www.skype.com
[2]http://aws.amazon.com/autoscaling/
[3]http://code.google.com/appengine/
[4]http://www.microsoft.com/windowsazure/windowsazure/

demand is lower. To our knowledge, there is no SIP system that is capable of this elasticity.

7 Conclusion and Future Work

In this paper, we have introduced a system that is able to scale VoIP infrastructure using resources from cloud providers, in order to meet the demands of increased calls to an emergency call centre during and after a disaster. We were able to meet the requirements of disaster response by integrating congestion management, peer-to-peer mechanisms, and the SIP protocol with the cloud provisioning model. Evaluation using data modelled on an actual disaster situation shows that the system is able to react to increasing call volumes by provisioning enough private servers. The benefit of this is a reduction in the number of abandoned calls, and in the time spent in waiting to be connected to an operator.

However, extra human call-centre operators are needed as well to take advantage of the extra server capacity provided by the auto-scaling function. While this issue is out of scope of the aims of this paper, the system proposed here can be used to relieve the pressure on the existing call-centre operators. For instance, during the Victorian fires, callers would ring up emergency services in order to gather information on the location, speed, and direction of the fires. This could have been just as well conveyed by automated telephone-based information services. Such services could have freed up the human emergency services operators to focus on helping callers who were in danger or were injured.

The system discussed in this paper also motivates the organisation of a disaster relief agency's phone infrastructure in a virtualised model in which capacity can be added or removed on-demand. In effect, a *private cloud* is created within the agency. This model is suitable for organisations that are not able to use commercial (or public) cloud providers due to regulatory or security considerations.

At the moment, the PBX participates only in connecting the caller to the operators. It is not involved in the actual talk session which is conducted as peer-to-peer between the calling parties. However, it is a regulatory requirement in most countries that all calls to emergency services be recorded in which case the PBX would have to be an intermediary in the actual talk session. This will not only increase the load on the actual VM but will also involve higher data transfer and storage costs while using cloud services. There is a need to further fine-tune the auto-scaling system in order to take into account these factors.

The phone network is only one of the elements of the IT infrastructure. In the aftermath of a disaster, hospitals and other health services are put under severe pressure, and this is translates to a high demand on the health IT infrastructure [2]. We are in the process of extending this work to auto-scaling systems that store and serve electronic health records [22] and hope to look at other IT systems involved in disaster response as well.

Bibliography

[1] Michael Armbrust et al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009.

[2] J. L. Arnold et al. Information-sharing in out-of-hospital disaster response: the future role of information technology. *Prehospital and Disaster Medicine*, 19(3):201–207, September 2004. PMID: 15571195.

[3] Vijay A. Balasubramaniyan, Arup Acharya, Mustaque Ahamad, Mudhakar Srivatsa, Italo Dacosta, and Charles P. Wright. Servartuka: Dynamic distribution of state to improve sip server scalability. In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS '08)*, pages 562–572, Washington, DC, USA, 2008. IEEE Computer Society.

[4] David A. Bryan and Bruce B. Lowekamp. Decentralizing SIP. *Queue*, 5(2):34–41, 2007.

[5] David A. Bryan, Bruce B. Lowekamp, and Cullen Jennings. SOSIMPLE: a serverless, standards-based, P2P SIP communication system. In *Proceedings of the First International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications*, pages 42–49. IEEE Computer Society, 2005.

[6] Rajkumar Buyya et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, 2009.

[7] Jane Louise Elkington. Statement of Jane Louise Elkington. Technical Report Exhibit No. 0117, 2009 Victorian Bushfires Commission, June 2009. Filed on behalf of Telstra Corporation.

[8] Ali Farazmand. Learning from the Katrina Crisis: A Global and International Perspective with Implications for Future Crisis Management. *Public Administration Review*, 67:149–159, 2007.

[9] Ali Fessi, Heiko Niedermayer, Holger Kinkelin, and Georg Carle. A cooperative SIP infrastructure for highly reliable telecommunication services. In *Proceedings of the 1st international conference on Principles, systems and applications of IP telecommunications*, pages 29–38, New York City, New York, 2007. ACM.

[10] Vijay Gurbani, Lalita Jagadeesan, and Veena Mendiratta. Characterizing session initiation protocol (SIP) network performance and reliability. In *Service Availability*, volume 3694 of *LNCS*, pages 196–211. Springer Berlin, 2005.

[11] D. Richard Kuhn. Sources of failure in the public switched telephone network. *Computer*, 30(4):31–36, 1997.

[12] B.S. Manoj and Alexandra Hubenko Baker. Communication challenges in emergency response. *Commun. ACM*, 50(3):51–53, 2007.

[13] Petar Maymounkov and David Mazires. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer Berlin / Heidelberg, 2002.

[14] Francesco Moggia, Mudumbai Ranganathan, Eunsook Kim, and Doug Montgomery. Emergency telecommunication support for IP telephony. In *Operations and Management in IP-Based Networks*, volume 3751 of *LNCS*, pages 1–8. Springer Berlin, 2005.

[15] J. Rosenberg et al. SIP: Session Initiation Protocol. RFC 3261, Internet Engineering Task Force (IETF), June 2002.

[16] Kundan Singh and Henning Schulzrinne. Peer-to-peer internet telephony using sip. In *Proc. Intnl. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '05)*. ACM Press, New York, NY, USA, 2005.

[17] Kundan Singh and Henning Schulzrinne. Failover, load sharing and server architecture in SIP telephony. *Comput. Commun.*, 30(5):927–942, 2007.

[18] M. Spencer. Distributed Universal Number Discovery (DUNDi) and the General Peering Agreement (GPA). Technical report, Digium, Inc., October 2004.

[19] M. Spencer et al. IAX: Inter-Asterisk eXchange Version 2. RFC 5456 (Informational), Internet Engineering Task Force (IETF), Feb 2010.

[20] V.N. Vapnik. An overview of statistical learning theory. *Neural Networks, IEEE Transactions on*, 10(5):988–999, 1999.

[21] Jinesh Varia. Architecting for the Cloud: Best Practices. Technical report, Amazon, Inc., January 2010.

[22] Srikumar Venugopal et al. Autoscaling infrastructure for electronic health records: A disaster management perpsective. In *Proceedings of International Mobile Health Workshop:12th IEEE International Conference on E-Health Networking, Applications and Services (Healthcom '10)*, Washington, DC, USA, 2010. IEEE Computer Society.