Journal of Applied Computing Research, 2(1):50-59
January-June 2012

A Distributed Presence Service over Epidemic Multicast

Peterson Wilges, Taisy S. Weber, Sérgio L. Cechin, Regina L.O. Moraes
Universidade Federal do Rio Grande do Sul. Av. Bento Gonçalves, 9500, 91509-900, Porto Alegre, RS, Brazil
pwilges@inf.ufrgs.br, taisy@inf.ufrgs.br, cechin@inf.ufrgs.br, regina@ft.unicamp.br

Abstract. In a dynamic distributed system with a very large number of nodes, such as a cloud, it is sometimes useful to discover the nodes that are up in the system at a given time. The number of those nodes changes continually along the operation time, as some nodes crash and some join the system. In this paper we introduce a presence service that was implemented over a gossip structure using an epidemic multicast protocol. Unlike other common presence services, our service is fully distributed. Due to epidemic dissemination and inherent redundancy provided by the multicast protocol, the service is resilient against message loss and link crashes. In a scenario we developed to evaluate the efficiency and scalability of our presence service, we show how presence notifications propagate to reach all nodes in the group and we also show how adjustments for the gossip configuration can benefit the efficiency and resilience of the notification dissemination. The results of the experimental evaluation show that following a distributed approach over epidemic communication leads to a resilient and scalable presence service.

Key words: Presence service, epidemic protocols, resilience, fault tolerance, clouds

Introduction

A presence service aims to manage presence information about computer nodes in a given network. In order to be efficient and resilient in a dynamic large scale environment, it is necessary that the service overcomes scalability problems.

This paper introduces the development of a scalable and resilient presence service over a gossip structure. Our service, called PingCloud, follows a different approach than those described in the IETF standards through RFCs 2778 (Rosenberg and Day, 2000) and 2779 (Day *et al*., 2000).

Currently, the existing presence services are centralized, as suggested by the RFCs. We choose a fully distributed approach to avoid a single point of failure. Each node in a given group collects and saves presence information of the entire group of nodes. Each node is responsible for managing the presence information extracted from notifications it receives from other nodes of the system.

Considering time propagation and service availability, the distributed approach we propose can be more efficient than the centralized one that is commonly used for this kind of network service. The penalty for the high availability and efficiency we can achieve with our approach is the great number of messages the nodes disseminate to their neighbors. A certain amount of these messages is redundant, which means that some nodes can receive multiple replicas of the same message.

PingCloud was built on using an epidemic multicast protocol (Pereira *et al*., 2003, 2004). Epidemic multicast is highly scalable and resilient to communication faults (Alvisi *et al*., 2007). It does not provide strong guaranties (Birman, 2003), but ensures a high probability that the disseminated messages reach all the destination nodes (Eugster *et al*., 2004).

When using epidemic multicast, each presence notification sent from one node to its neighbors is retransmitted from these nodes to their own neighbors, which in turn retransmit the message to other neighbors, and so on. The neighbors of each node are chosen randomly among the nodes in the group.

We select the NeEM, Network-friendly Epidemic Multicast (Pereira *et al*., 2003), as the protocol to disseminate presence information.

NeEM provides mechanisms for an application to manage the ideal number of nodes to disseminate a message during epidemic multicast. This number, called *fan-out*, is important to ensure a high probability of message delivery.

Epidemic protocols are inherently redundant. This redundancy is the main reason why these protocols are resilient to message loss. However, NeEM has some mechanisms to mitigate unnecessary retransmissions (Leitão *et al*., 2007) but preserving resilience. We need also to avoid that a message remains being disseminated forever between the members of a group. NeEM limits retransmission by using a protocol parameter called *time-to-live* (TTL). TTL determines the number of rounds a message can be retransmitted.

In this paper, we show the tests we applied to demonstrate how efficient the dissemination of information is. We developed a demonstration scenario using a single computer, since the synchronization of events based on a single clock simplifies monitoring the test experiment. In this scenario, one node disseminates a notification using epidemic multicast. Each node in the sequence floods the notification message to other nodes within the group.

For each experiment using this scenario, we evaluated the time of dissemination by varying the number of nodes. In a second analysis we evaluated the use of different fan-outs. Combining fan-out, time-to-live and number of nodes allows determining the probability that a message can reach the entire network. The results show that a distributed approach over epidemic communication leads to an efficient, resilient and scalable presence service.

The paper is organized as follows. The section "Epidemic Multicast" presents its basic concepts and the section "NeEM" describes the used epidemic multicast protocol. In the section "Presence Service Proposal" we describe how our service operates and explain some "Implementation Issues". In section "Scalability and Resilience" we discuss issues related to the main properties of Epidemic Multicast. The test experiments are described in section "PingCloud Evaluation". In section "Related Work" we present some existing presence services. Finally, we present our "Conclusions" and possible future works.

Epidemic Multicast

Epidemic multicast is a special case of group communication protocol that is also called gossip-based or probabilistic protocol. It is easy to see that the epidemic protocol is inherently redundant. We can do an analogy with a gossip being disseminated in a group of friends. One person tells a story to some group members, possibly chosen at random, that in turn send the message to some other friends. Each one that hears the story will tell it again to other group members. Each one of these groups can overlap with the group of friends of their friends that already know the story. And so we observe that each person would probably hear the same story more than once.

An epidemic dissemination (Eugster *et al*., 2004) obeys the following mode of operation: each node connects itself with a number of nodes k, forming the overlay. For each node j and for each received message, the node j retransmits the message to f (fan-out) nodes, where $f < k$. In other words, the node j retransmits each message to a number of neighbors that is less than the number of established connections in the overlay.

The Figures 1 to 4 illustrate the behavior of an epidemic protocol in a group with k = 18 nodes and fan-out f = 3. Figure 1 shows an originator node (the seed) disseminating a message to 3 neighbors. It is the only one infected, i.e. it is the only node that knows the message.

At the end of the first round 4 nodes are infected (the seed and its fan-out nodes).

Figure 2 shows the second round. All nodes infected in the first round flood the message to their fan-out nodes.

Some messages can be received two or more times during the second round. In our example this will occur on node N of Figure 2. Some messages, as for example *m*, can be sent to nodes that had already received the

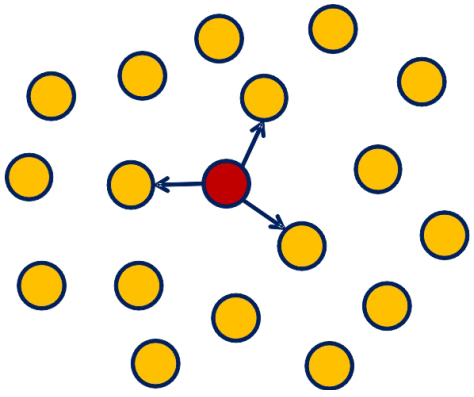

Figure 1. Epidemic multicast with fan-out of 3.

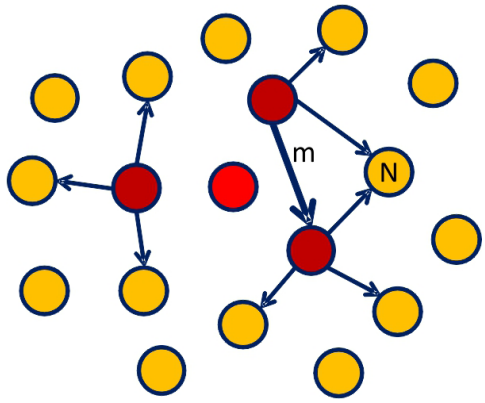

Figure 2. Second round of an epidemic multicast.

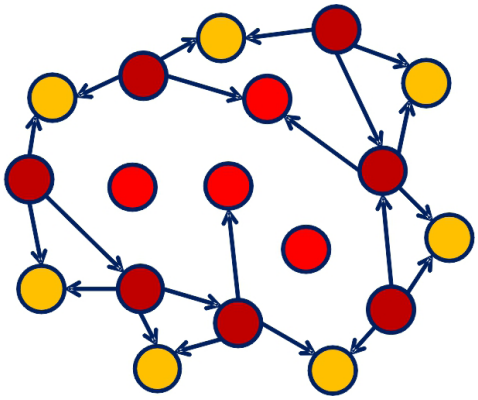

Figure 3. Third round of an epidemic multicast.

message in an earlier round. At the end of this round 11 nodes become infected.

Figure 3 shows the third round. The number of messages grows exponentially. 21 messages are now flooding through the network. But the probability that a node receives a redundant message also increases. Out of 21 messages, 7 will reach nodes that are already infected, 14 will reach seven new nodes (7 messages are redundant).

All nodes in Figure 3 become contaminated at the end of the third round. However, a situation as the one shown in Figure 4 could also happen. The grey node does not receive any message during three flooding rounds. It was not randomly chosen from any other node to form its fan-out set during the three rounds.

To enhance the chance to infect the grey node, one solution could be increasing the number of rounds. Other solution is increasing the fan-out. But the nodes that form the fan-out set are chosen randomly at each node in a round and it is not possible to guarantee that all nodes will receive a given message, except statistically.

Increasing the number of rounds also increases the propagation time to multicast a message. Increasing the fan-out increases the traffic through the network. So, a proper configuration of f and TTL must be carefully chosen.

Redundancy allows for enhanced fault tolerance (Birman, 2003). If one or more nodes crash, some nodes are guaranteed to receive the message and flood it away.

For each retransmission round different nodes among the k nodes of the overlay are chosen (f) at random. Periodically new connections are established, changing the overlay. This enables new users to join the group and also current users to leave the group. This characteristic also enhances the group resilience against faults: faulty nodes can be easily isolated and the group reconfigures itself automatically and dynamically.

Epidemic multicast protocols show a high performance. The performance is independent of the number of k nodes in the overlay. These protocols are also scalable to a large number of participants. They are also resilient against network and nodes failures. However, these protocols generate a large message traffic in the network due to their inherent redundancy.

Several epidemic protocols can be found in the technical literature (Frey *et al.*, 2009; Lim *et al.*, 2011; Wuhib *et al.*, 2012). They vary in the way they achieve efficiency using the network.

NeEM

As already mentioned, our service uses the Network-friendly Epidemic Multicast (NeEM) protocol. This protocol runs into every node of an overlay and is implemented over TCP connections.

The NeEM protocol can also mitigate the high usage of network capacity due unnecessary retransmissions: a large number of message retransmissions is inherent in gossip protocols. To avoid it, NeEM can operate in two modes: eager and lazy. The first one, eager, is automatically chosen for short messages. In this mode the message is retransmitted once it is received.

By contrast, when the message is long, the lazy mode becomes active. In this mode, the protocol sends a short advertisement message to the connected nodes asking whether they wish the complete message. The complete message is only sent when an acknowledgment is received (Leitão *et al.*, 2004).

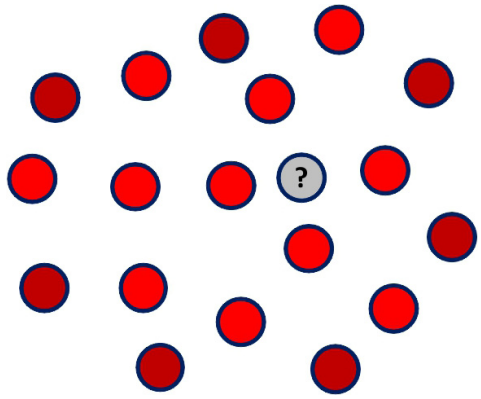

Figure 4. Incomplete multicast.

Presence Service Proposal

To manage the information of which nodes are present in a given group, we built a presence service called PingCloud (Wilges *et al*., 2012). This name was chosen because the group of overlay nodes can compose a cloud, but this service is not restricted to clouds. It can also be used to manage membership for large groups in group communication middleware or a simple presence service in any kind of network.

Usually presence services aim to manage information about entities that are present in a network, which can be users, devices or an application.

The configuration of three main parameters is essential to the proper operation of the presence service: transmission periodicity, information aging and resolution.

The first parameter (transmission periodicity) sets the time interval between two notification messages. Each node that wishes to inform its presence sends a message to the group notifying this. To ensure that this information is up to date, these notifications must be sent periodically and the shorter this time interval is, the more accurate is the presence information.

The second parameter (information aging) is used to determine when presence information has become obsolete and has to be removed from its presence list. In practical terms, this parameter determines how long the presence information of a node can be used without being updated. To be useful, this parameter should be greater than the "transmission periodicity" parameter.

Finally, the third parameter (resolution) determines how often a node scans its presence list looking for obsolete information. This parameter should be shorter than the other two parameters.

To join the group and use the presence service, a node needs to know at least one node present in the overlay group.

In PingCloud, there is just one presence type. Other presence services, such as instant message applications, identify whether a user is available, away and busy, for instance. The presence of a computer in a network has otherwise only two possible states: present or not present. To inform its presence, a node simply sends a presence notification. If it is not present or if it wishes to leave, a node simply no longer sends any notifications.

The presence notification message also carries a timestamp to allow an ordering of events.

The main difference between PingCloud and other related services is the maintenance of the presence list. The common presence services follow a centralized approach, where a central server is responsible for managing the presence information of the entire network. The central server is clearly a single point of failure. It also presents scalability issues that jeopardize the growth of the group.

In contrast the distributed architecture of PingCloud each node manages its own presence list. Each list starts empty and is updated within established time intervals (transmission periodicity), removing the nodes that do not send notifications within another interval (information aging) and adding new nodes that wish to join the overlay group.

Implementation Issues

We can break down the management of the presence list in two tasks: add nodes and remove nodes.

A node inserts presence information into a presence list when it receives a notification message. The task of removing presence information is accomplished by a special thread, which runs periodically, according to the "resolution" parameter.

We have considered performance issues in the design of the presence service. We have realized that two parameters determine the CPU load: the number of nodes in the group and the "resolution" parameter. The number of nodes in the group induces the size of the presence list and the time spent to scan it. The parameter "resolution" determines how often this list is scanned.

We have used two data structures to manage presence information: a hash table and an ordered linked list. This list is ordered according to the forecast removal time.

Each element of the list contains the node identification (presence information) and the due removal time. The hash table key is the node identification and the value stored is a pointer to the associated element in the linked list.

To include a node's presence information we search for its identification in the hash table, which returns the pointer to the associated element in the linked list, if it exists. In case the element exists, it will be removed from the hash table and the linked list. Then, the new node identification will be inserted in the linked list and the hash table. The procedure described avoids scanning the linked list. Thus, the time spent is mainly determined by the hash table operations.

When the special thread runs, the scanning of the linked list is started. The older information is checked first. The expired elements are removed and the process stops when a non-expired element is found. Then, a full scan of the list is not necessary. The CPU time is spent to process the elements that were actually changed.

Figure 5 shows the two data structures: linked list and hash table. Also, we can see the cross-reference between the linked list and the hash table.

Scalability and Resilience

An important issue in distributed systems is scalability. Birman (2003) states that strong properties, as for example fault tolerance, consensus and consistency, do not scale well. A cloud is a huge system with scalability problems. We can see a cloud as a group of processes or nodes that need to communicate to reach an agreement or to know about the existence and states of other nodes in a large dynamic group.

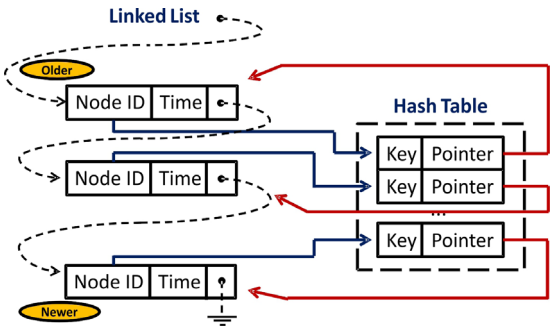

Figure 5. PingCloud data structures.

Of course, not all applications and services need to provide strong properties. But when properties such as reliability or consistency are required, we cannot ignore scalability issues. In this arena the natural redundancy provided by epidemic multicast seems to be a natural choice for communication in a large scale computer group.

Epidemic multicast is highly scalable and resilient to communication faults. However, it depends on an appropriate choice of parameters, such as TTL and fan-out values (Eugster *et al*., 2004). The configuration of the parameters mentioned above may decide if the dissemination is really successful.

Some authors (Carvalho *et al*., 2007; Alvisi *et al*., 2007) propose ways to get both resilience and efficiency. Resilience is achieved by adding redundancy; to get efficiency, in general, one tries to eliminate redundancy.

When using epidemic protocols, resilience can be measured by a probability (in case of messages, this means a message delivery probability). The higher the probability is, the higher is the number of messages: thus, higher network traffic and a worse efficiency.

Thus, to validate the service we built, we must consider varying the protocol parameters to analyze the protocol behavior and track the flow of messages to see whether a notification reaches all the nodes in a group.

PingCloud Evaluation

We built a scenario to validate PingCloud. The scenario aims to demonstrate PingCloud's operation and also to test the service.

This scenario consists of a single computer which simulates several nodes. One node, the seed, sends the notification. This notification propagates through the network to the other members of the group.

Using this scenario we made two evaluation experiments. Both evaluations aim at showing the dissemination of notifications through the group. The first evaluation disregards the propagation delay and varies the number of nodes that compose the group.

In Figure 6, we can see the three configurations used in the evaluations: (a) 15 nodes, (b) 25 nodes and (c) 40 nodes.

For the second evaluation we kept the total number of nodes unchanged and set the delay propagation to about 100 ms to all packets.

Then, we used a different fan-out value for each configuration. One of these scenarios (fan-out = 3) can be seen in figure 7, where we represent two rounds (three snapshots). In the figure we can see the seed node (red node in t=0ms) and the nodes "infected" at time 200ms (red nodes in t=200ms).

For the second evaluation we used a fault injection tool, FIRMAMENT (Drebes *et al*., 2006.) The tool allows emulating network communication faults like message loss, message corruption as well as delay of packets. These faults are commonly used to evaluate the fault coverage and resilience of communication protocols (Siqueira *et al*., 2009) in fault scenarios.

For both evaluations we simulated a network with a fixed number of nodes and different fan-out configurations.According to this scenario and settings, some test experiments were done. The results are described in this section.

Dissemination Time of a Notification

The first evaluation checked how long one presence notification takes to reach all present nodes in a group. Here we disregarded the propagation time of the notification from one node to the next one. We built a scenario with different numbers of nodes in the group to evaluate the service.

The experiment runs as follows: one node, the seed, sends just one single notification message using epidemic multicast and then we collect, through generated logs, the time this message arrives in all other nodes of the group.

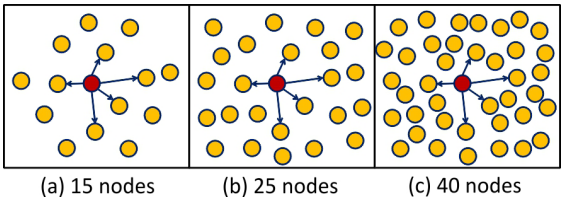

Figure 6. First evaluation.

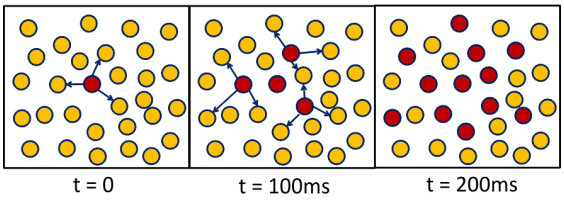

Figure 7. Second evaluation.

The NeEM parameters were adjusted to a fan-out of 5 and time-to-live of 6. These parameters mean that each node retransmits the message to 5 other nodes and the message will be ignored after 6 rounds.

To send just one message from the seed node (the originator), we adjust the parameter corresponding to the notification interval to allow the seed to send only one message during the experiment. The seed node does not send two or more messages during the experiment because we set a large time interval. This guarantees that we can follow the propagation of this message.

The experiment shows, as expected, that the dissemination time of presence information does not grow linearly with the increasing number of network nodes. This is because PingCloud uses an epidemic dissemination protocol.

At the first moment, when just a few nodes have the message, there is little redundancy of retransmission, and so the transmission to f nodes is complete, or practically complete. Later, when a large number of nodes have received the notification, there is much useless retransmission; almost all messages that are received are duplicate ones.

In the last hops the retransmission is less efficient, because few of the messages that the nodes received are actually new, which means not already received in those nodes. The redundant messages are silently dropped.

The redundancy is an important characteristic in epidemic protocols, because the protocol's fault tolerance relies on it. Redundancy enhances the resilience against node crashes, link crashes and loss of messages. Redundancy also allows for very fast propagation and scalability.

To follow the dissemination of a message in the scenario just described, we performed simulations for 15, 25 and 40 nodes forming the group. With 40 nodes we reach the limit using only one computer. With more than 40 nodes, the result might be affected by the large number of simultaneous threads.

Figure 8 shows an example where it is possible to visualize the fast dissemination of a notification in its initial phase and its loss of performance over time.

Figure 8 displays the time to reach all nodes in a group using an epidemic multicast protocol with fan-out of 5 and TTL of 6.

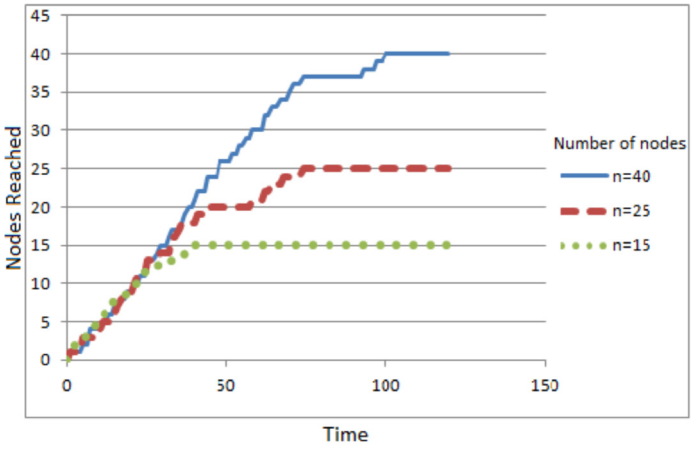

Figure 8. Time to reach all nodes in ms

The number of nodes in the group varies from 15 to 24 and to 40.

Dissemination time for different fan-outs

In this evaluation experiment we run two simulations, both with 30 nodes belonging to the group. The first simulation uses a fan-out of 3, and the second a fan-out of 7.

For this evaluation, we use a fault injection tool called FIRMAMENT to set 100 ms delay in all transmitted packets. Thus, with fixed delays it is possible to analyze each round of message transmission. The introduced delays permit emulating a scenario that is closer to real network environments, thus compensating for the fact that the experiments are running in a single computer.

First the node which wishes to transmit a notification, the seed, sends the notification to f (fan-out) other nodes in the group. As none of these nodes has received the notification before, all transmissions are effective, which means it transports a new notification, not a duplicate one.

In a second round, each one of the f nodes which received the message will retransmit it to f other nodes. Some of the destination nodes in this round may have received the message before, so this message can be potentially redundant. In the following rounds, this can happen again, with the difference that the probability of receiving a redundant message grows in each round.

Now, analyzing the simulation to fan-out 3, it is possible to observe in the first round that 3 notifications are going to be sent. In the second round, are going to be there will be 9 notifications, where there is a probability that some messages are redundant. In the third round, 3 times the number of nodes which received the message in the previous round will be transmitted minus those that receive redundant message, since these nodes simply discard the message. Thus, it is important to observe that if in a round all retransmitted messages are redundant, all will be dropped and the propagation will stop. None of the nodes retransmits a message it has already retransmitted before.

The experiment for a fan-out of 7 is very similar to the last experiment that uses a fan-out of 3. But now each node must retransmit the message to 7 other nodes. It is easy to see that the dissemination is faster and less rounds will be needed for the notification message that was originated in the seed to reach all the nodes in the group.

According to this, in Figure 9 it is possible to observe the different rounds of retransmission, separated by 100ms of interval. We can see also that for larger fan-outs less rounds of retransmission are needed to reach all nodes.

It is important to emphasize that the fault injector was used in this scenario just to compensate for the fact that the evaluation runs in a single computer and this is not representative of a real network environment. The delay was injected to emulate the network's natural delays, and it cannot be considered as a fault emulation.

Related work

We describe some examples of presence services. As we are going to see, the services described here follow a centralized approach

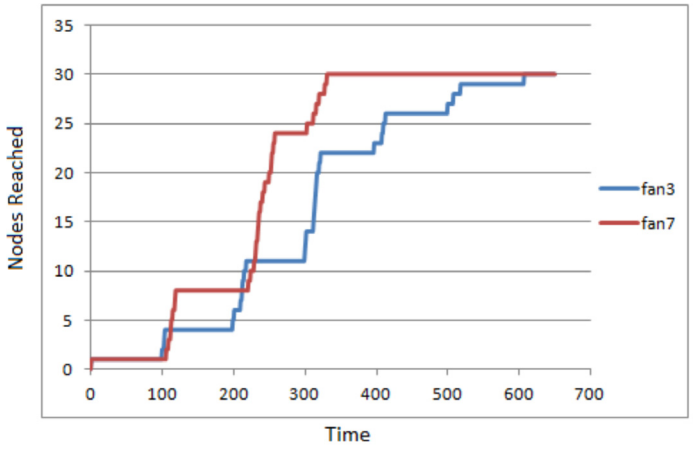

Figure 9. Multicast time with different fan-outs.

except for the last one (Celesti *et al*. 2010), where distributed agents collect presence information and publish them in a central location.

Comparing these services with PingCloud, the main point to emphasize is that our service has no single point of failure. This problem is typical of centralized approaches. PingCloud can also scale better than any centralized approach. The reliance and scalability reached by our proposal came with a penalty: the flooding of notification messages can affect the traffic load of the network.

Extensible Message and Presence Protocol

XMPP, Extensible Message and Presence Protocol (Saint-Andre, 2004, 2011) is an open-source protocol for instant messaging, presence information and contact list maintenance. Its architecture follows a centralized approach. The presence list resides on servers which mediate all the communication between the users. All the nodes that belong to the group are addressed through a single identifier. Any two users wishing to notify their presence and communicate with each other must, each one, connect to the XMPP server. These servers send the presence information and exchange messages between these two users. The communication is done exchanging messages in XML format.

Simple

SIMPLE (Niemi, 2004) is a presence service protocol and instant messaging suite based on SIP protocol (Session Initiation Protocol) (Johnston, 2009), which is a protocol for creating, modifying, and terminating multimedia sessions like, for example, voice application over IP. SIMPLE adds three methods to SIP to exchange messages and presence information between users:

- Subscribe: method invoked when a node wishes to receive presence information.
- Notify: method used to send presence information
- Message: method invoked to send messages.

The communication between two users can be done by P2P, since the nodes know each other in advance. If the users do not know each other, they use a central server which holds the presence list and the addresses of the users.

Wireless Village

Wireless Village – WV (OMA, 2001) was conceived by OMA – Open Mobile Alliance. It was developed to provide a set of universal specifications for mobile instant messaging and presence services. It uses the client-server architecture. The clients are any mobile devices, and the server is the Wireless Village Server. The WV server is responsible for managing the presence information as well as other features, like exchanging messages, group management, group membership and file sharing.

CCFM Discovery Agent

Celesti *et al*. (2010) propose an architecture for cross cloud federation management in a federate environment composed of agents, and

one of them is a discovery agent. It manages the discovery process among all the available clouds. The purpose here is not to find available nodes but the presence of foreign clouds.

The discovery process is implemented in a distributed way using the publisher-subscriber approach. An application publishes its own set of information at a centralized location from which only a set of authorized (subscribed) entities are able to retrieve it. Even though such location is logically centralized, it can be implemented in a distributed way, and it also reaches resilience against central node crash.

Conclusions

We can conclude that it is advantageous to use a distributed approach associated with epidemic multicast to build a presence service. The results show that this approach allows for good performance and better scalability and resilience against network faults in comparison to traditional centralized approaches. The results also demonstrate that the NeEM protocol used to support the epidemic multicast was an opportune choice for the dissemination of presence notification.

It is also possible to observe that configuring appropriately the protocol parameters is fundamental for its correct operation. The parameters can be modified by an application according to its needs. For example, in a network with a large percentage of message loss, the application can increase the redundancy by increasing the fan-out, thus guaranteeing a higher probability of message delivery.

Some issues are not explored in this first version of the presence service, including some security mechanism to join the group. Other important point to be investigated is improving the application so that it manages the NeEM parameters at runtime. Parameters as fan-out and time-to-live are related to the number of nodes present in the group and, thus, must be managed by the application.

For future works, we are planning tests with a larger number of computers to simulate overlay groups in larger scales. We also want to complement the tests with communication fault injection, dropping messages, delaying some messages, crashing nodes and links. Testing in this way, besides the service efficiency, we can also extract better performance measures under faults, fault coverage and service availability.

Acknowledgements

This work has been funded by the JitCloud project with support of the RNP.

References

ALVISI, L.; DOUMEN, J.; GUERRAOUI, R.; KOLDEHOFE, B.; LI, H.; VAN RENESSE, R.; TREDAN, G. 2007. How Robust Are Gossip-based Communication Protocols? *ACM SIGOPS Operating Systems Review*, **41**(5):14-18. http://dx.doi.org/10.1145/1317379.1317383

BIRMAN, K.P. 2003. The Surprising Power of Epidemic Communication. *In:* A. SCHIPER (ed.), *Future Directions in Distributed Computing*. LNCS 2584. Berlin, Springer-Verlag, p. 97-102. http://dx.doi.org/10.1145/1556154.1556172

CARVALHO, N.; PEREIRA, J.; OLIVEIRA, R.; RODRIGUES, L. 2007. Emergent Structure in Unstructured Epidemic Multicast. *In:* ANNUAL IEEE/IFIP INT. CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 37, Edinburg, 2007. *Proceedings*... Los Alamitos, p. 481-490. http://dx.doi.org/10.1109/DSN.2007.40

CELESTI, A.; TUSA, F.; VILLARI, M.; PULIAFITO, A. 2010. How to Enhance Cloud Architectures to Enable Cross-Federation. *In:* IEEE INTERNATIONAL CONFERENCE ON CLOUD COMPUTING (CLOUD), 3, Miami, 2010. *Proceedings*… Los Alamitos, p. 337-345. http://dx.doi.org/10.1109/CLOUD.2010.46

DAY, M.S.; AGGARWAL, G.; MOHR, J.; VINCENT, 2000. Instant Messaging/presence Protocol Requirements. RFC, 2779.

DREBES, R.; JACQUES-SILVA, G.; TRINDADE, J. da; WEBER, T. 2006. A Kernel-based Communication Fault Injector for Dependability Testing of Distributed Systems. *In:* S. UR; E. BIN; Y. WOLFSTHAL (ed.), *Hardware and Software, Verification and Testing. Lecture Notes in Computer Science 387*. Berlin, Springer-Verlag, p. 177-190. http://dx.doi.org/10.1007/11678779_13

EUGSTER, P.T.; GUERRAOUI, R.; KERMARREC, A.-M.; MASSOULIE, L. 2004. Epidemic Information Dissemination in Distributed Systems. *Computer*, **37**(5):60-67. http://dx.doi.org/10.1109/MC.2004.1297243

FREY, D.; GUERRAOUI, R.; KERMARREC, A.; KOLDEHOFE, B.; MOGENSEN, M.; MONOD, M.; QUÉMA, V. 2009. Heterogeneous Gossip. *In:* ACM/IFIP/USENIX INTERNATIONAL CONFERENCE ON MIDDLEWARE, 10, Urbana Champaign, Illinois, 2009. *Proceedings*… New York, Springer-Verlag. Inc., **3**:1-3:20. http://dx.doi.org/10.1007/978-3-642-10445-9_3

JOHNSTON, A.B. 2009. *SIP: Understanding the Session Initiation Protocol*. London, Artech House Publishers, 395 p.

LEITÃO, J.; PEREIRA, J.; RODRIGUES, L. 2007. HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast. *In:* ANNUAL IEEE/IFIP INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 37, Edinburg, UK, 2007. *Proceedings…* Los Alamitos, p. 419-429. http://dx.doi.org/10.1109/DSN.2007.56

LIM, J.; LEE, J; CHIN, S.; YU, H. 2011. Group-Based Gossip Multicast Protocol for Efficient and Fault Tolerant Message Dissemination in Clouds. *In:* J. RIEKKI; M. YLIANTTILA; M. GUO (ed.), *Advances in Grid and Pervasive Computing*. *Lecture Notes in Computer Science*. Berlin/Heidelberg, Springer, **6646**:13-22. http://dx.doi.org/10.1007/978-3-642-20754-9_3

NIEMI, A. 2004. Session Initiation Protocol (SIP) Extension for Event State Publication. RFC 3903.

OMA 2001. The Wireless Village Initiative: System Architecture Model, 2001-2002. Available at: http://www.openmobilealliance.org/tech/affiliates/wv/wv_architecture_v1.0.pdf Access on: 01/10/2012

PEREIRA, J.; RODRIGUES, L.; MONTEIRO, M.J.; OLIVEIRA, R.; KERMARREC, A.-M. 2003. NEEM: Network-friendly Epidemic Multicast. *In:* INTERNATIONAL SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, 22, Florence, 2003. *Proceedings…* Los Alamitos, p. 15-24. http://dx.doi.org/10.1109/RELDIS.2003.1238051

PEREIRA, J.; RODRIGUES, L.; PINTO, A.; OLIVEIRA, R. 2004. Low Latency Probabilistic Broadcast in Wide Area Networks. *In:* IEEE INTERNATIONAL SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS, 23, Florianópolis, 2004. *Proceedings…* Washington, DC, p. 299-308. http://dx.doi.org/10.1109/RELDIS.2004.1353030

ROSENBERG, J.; DAY, M. 2000. A Model for Presence and Instant Messaging. RFC 2778.

SAINT-ANDRE, P. 2004. Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM). Available at: http://tools.ietf.org/html/rfc3922. Access on: 01/10/2012.

SAINT-ANDRE, P. 2011. Extensible Messaging and Presence Protocol (XMPP): Core. Available at: http://tools.ietf.org/html/rfc6120.txt. Access on: 01/10/2012.

SIQUEIRA, T.; FISS, B.; WEBER, R.; CECHIN, S.; WEBER, T. 2009. Applying FIRMAMENT to Test the SCTP Communication Protocol Under Network Faults. *In:* LATIN AMERICAN TEST WORKSHOP, 10, Búzios, 2009. *Proceedings…* Los Alamitos, p. 1-6. http://dx.doi.org/10.1109/LATW.2009.4813793.

WILGES, P.; LOVISON, H.D.C.; CECHIN, S.L.; WEBER, T.S.; MORAES, R.L.O. 2012. Serviço de Presença sobre uma Estrutura Gossip em Cloud. *In:* M.L. PILLA (ed.), *Escola Regional de Redes de Computadores*. Pelotas, SBC, p. 61-64.

WUHIB, F.; STADLER, R.; SPREITZER, M. 2012. A Gossip Protocol for Dynamic Resource Management in Large Cloud Environments. *IEEE Transactions on Network and Service Management*, **9**(2):213-225. http://dx.doi.org/10.1109/TNSM.2012.031512.110176

Submitted on October 15, 2012.
Accepted on December 11, 2012.