University of Tromsø

Distributed Observation Worm (DOW)

INF-3203 Advanced Distributed Systems

Camilla Stormoen & Sigurd Thomassen

Department of Computer Science

May, 2017

1 Introduction

Computer worms were described in 1982[1]. Worms were regarded as distributed computations, and several issues with such computations were analysed, including how they sometimes can go wrong and create problems for many computers.
This report describes the implementation and design of a distributed observation worm and discusses the issues and difficulties keeping the worm stable across a distributed system.

2 Technical Background

2.1 Computer Worm

A computer worm is a standalone malware computer program that replicates itself in order to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target computer to access it [2].

A worm consists of one or several worm segments. A segment is a program and is located on multiple computers, typically one segment per worm per computer. A worm has also a payload. The payload can collect data from the environments where the segments are executing, increasing robustness for an application, harvesting CPU cycles, and can be used by public or private purposes.

3 Design

The worm is built on the concept of having no centralized orchestration. Which means that it does not have a specific node as an orchestrator. It does however have temporary leaders, where the nodes elect a leader to broadcast commands.

When the worm starts, it starts out with one segment. It will then wait until it gets an update on target segments, which is the wanted number of segments. When the target segments is updated, the one segment will grow the worm up to the amount of segments. Whenever a segment is started, it also gets a heartbeat. A heartbeat is a routine that runs parallel with the segment, and pings all the other segments to check for life. In this way the worm can have a relatively good idea on which segments are "online". And when a segment does not answer, it is declared dead by the heartbeat. It is then removed from the list of active segments, and the leader will be notified with this new list, and act accordingly. If the segment was removed from the list, the leader will broadcast to everybody that the specific segment is dead, and should be removed from the active segments list.

The leader election is done by simply hashing the hostnames. The hostname with the highest hash value will be the leader. So every segment hashes all active segments and compares them to their own hash value. In this way everybody knows who the leader is. And this check is done often, so if the leader should die, they should be able to quickly elect a new leader.

When the leader dies, and some other segments notices that, the segment will figure out who the new leader is and notify him. This is done by checking the hashes, and then using a post request on the new leader's address, telling him he "is the man". The new leader will then check if the number of segments is satisfactory to the target number of segments. And if not, it will grow or shrink the worm.

When there is a deviance between actual segments and target segments, the leader will bootstrap or kill new segments. This is done by selecting an available address, and then invoking a startup routine on that address. It will also add this new address to active nodes, and broadcast it to all other alive nodes, as well as broadcasting the target segments number. As this might have changed and some segments might not know about it.

As everything goes through the leader, it is enough that the leader is broadcasting the target segments, and active nodes. Everyone else will tell the leader of changes if they are detected, and the leader will handle it.

4 Implementation

The program is written in the programming language Go, and it is tested as a distributed system on the uvrocks cluster

4.1 Cluster setup and experience

The cluster contains 84 nodes and are divided using these computers: *Dell Precision T3600 with Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz, 36GB RAM and 1.00 TB DISK*, *Dell Precision T3500 with Intel(R) Xeon(R) CPU E5520 @ 2.27GHz, 12GB RAM and 500GB DISK* and *HP Compaq 8000 SFF with Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz, 8GB RAM and 250GB DISK*.

Testing on the cluster proved difficult because of contension when many people tried to test their implementation. This also caused this implementation to be unstabile each time it was tested.

5 Discussion

The implementation is simple and does not support adding or removing worms as intended. One problem arrives when the worm is trying to increase or decrease its segments with a double "+" or "-" in the visualizer, for example from 5 to 7 segments or 6 to 4 segments. The problem arise mostly because pinging the other nodes takes too long and not everybody gets the target segments number which tells how many nodes that should be up and running. The implementation works great if the worm get some time to stabilize before it gets another update by increasing og decreasing the number of segments spreading on the cluster.

There is also a problem when trying to spread over 7 segments, the program

crash. This is also because there are much communication over the network and to much contention causing the segment to not receive the broadcasts sent. When the control center is trying to kill the worm, it does not survive. Occasionally it tries to survive by starting new nodes when the worm register that they are too few compared to the target segment. This periodically survival is mostly dependent on the network contention and how much delay it is when every segment is broadcasting.

5.1 Improvements

As this is a simple solution that have flaws, one could have improved the worm with many approaches. First, instead of hash each segment every time the hash is checked, one could make a list with hashed segments, for example every time an update occurs. Then the host segment could just check with the hashed lists instead and improve time and performance.

Another improvement can be the pinging function. In this solution, every segment pings every segment. Alternately, one could make the leader do all the pinging and if the leader dies, the next leader will continue the pinging. This could cause a lot of work for the leader, besides starting and shutting down other segments. Therefore, a solution could be that almost everybody is pinging, but not everybody.

6 Conclusion

This report concludes the implementation and design of a distributed observation worm and discusses the issues and difficulties of keeping the worm stable across a distributed system.

References

[1] M. Barwise, "What is an internet worm?," 2010. [Online; accessed 1-May-2017].

[2] Wikipedia, "Computer worm — wikipedia, the free encyclopedia," 2017. [Online; accessed 1-May-2017].