A Token Based Approach Detecting Downtime in Distributed Application Servers or Network Elements

Sune Jakobsson

Telenor ASA Global Development, Otto Nielsens vei 12,
7004 Trondheim, Norway
sune.jakobsson@telenor.com

Abstract. This paper defines and outlines and proposes a heuristic method to detect unavailability on a practical service created on the Internet. It uses a token that is passed between all the nodes. The algorithm is under trial in a set of servers spanning over multiple administrative domains. This paper confirms that using a simple token passed through the contributing applications servers, is able to detect unavailability without intruding or changing the service that one wants to monitor.

Keywords: Application availability, Web Services.

1 Introduction

Application availability is becoming an increasing concern for users. They use or create services composed from servers origination on different administrative domains, and have hence lost the possibility to observe or monitor the real availability of a given service. Today services built on the Internet make use of data, processing nodes, and services spread on a number of nodes, that belong to different administrative domains, and even spread on multiple continents and countries. The Internet is now of such scale and size, that it is no longer possible to have administrative control over all parts of a service. Virtualization of hardware is common with provides like Google, Sun, Microsoft and others, where users can acquire resources on a pay-per-use basis. Typical resources are application servers and storage, but also access to more specialized services like maps. With the introduction of virtualization all low level faults are masked by the virtualization, but that does not mean that all the challenges with availability are met. Communication, network, routing and node name look-up are still hampering even simple services if their usage is infrequent. Additionally Web Services are becoming the de facto model of interaction on the Internet, where software components are loosely coupled together using standard communication protocols. The lack of firm bindings between the components hence complicates the capacity and resource monitoring. User's perception of services with input and response, is that if there is no response in approximately 8 seconds say, the users perceive the service to be non-operational.

An example of such a service would be an application, showing the location of individuals on a map, as shown in Fig 1. In this example server labeled A, runs the

F.A. Aagesen and S.J. Knapskog (Eds.): EUNICE 2010, LNCS 6164, pp. 209–216, 2010.

application, and use data or services from the other servers. Server A has an application server that ties the entire service together and the result can be browsed or dispatched to the mobile terminal as a multi-media message. The Server labeled B provides maps, the server labeled C provides the registration and group functions, and relies on the server labeled E, that is a database where the users are registered. The assumption is that all the servers are all managed independently of each other and possibly in different administrative domains. If the usage of such a service is infrequent, and at random times, the challenge is to know when parts of the nodes are unavailable, due to maintenance or failure, or due to network changes or outages. When this knowledge is available, a user could be given the option to use an alternative node configuration, either manually or automated in some way.

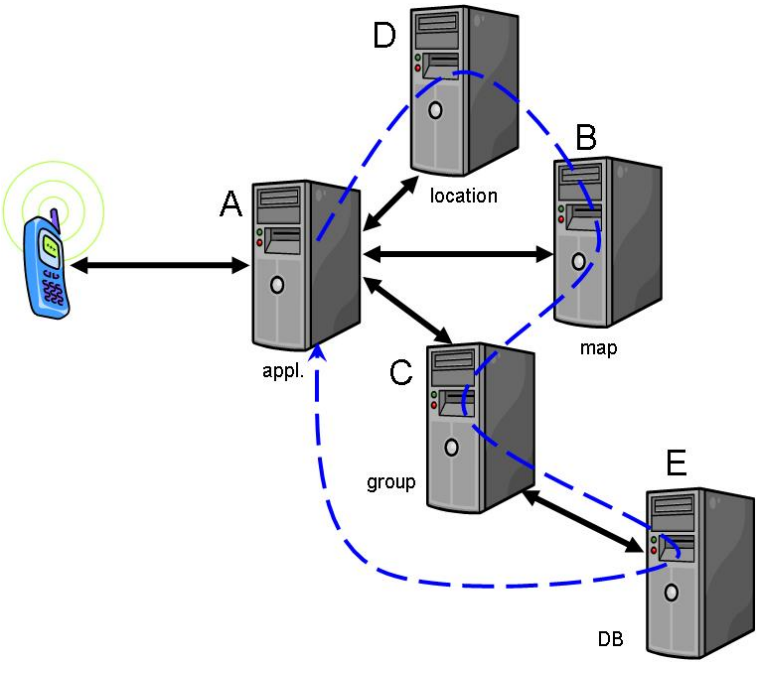

Fig. 1. One example service on the Internet consisting of five application servers

The obvious way would of course to try to use the service at some predetermined interval, but that might involve manual procedures, and could also come at a cost, if there is a fee for the usage of the nodes or their services. It is reasonable to assume that the nodes use some higher level communication protocol, like HTTP and that the nodes expose themselves as Web Services.

The paper starts with a high level background of application availability, and then describes the approach in detail. Traditional approaches to this subject have often focused on solving this on a local domain, but this paper suggest a alternative approach to solve the problem both in a local domain and in multi administrative domains. This is done by circulating a small data structure, referred to as a token.

Then the algorithm is explained, backed by some practical results, showing occasions where a given service would suffer form either application or network outages.

2 Related Work

Application availability and SW availability has and is studied in detail by many researchers and organizations, and there are many techniques to accomplish this. However most of them are focused on redundancy, watchdogs, probing or monitoring on a local domain or cluster of nodes [3]. In virtualization schemes HW resources are distributed and clustered across many users. At the lowest levels like physical storage, disk drives provide magnetic storage, but in order to obtain high availability they are often configured in a redundant configuration like for example RAID, where the data elements always exist in multiple copies in case of failure. At higher levels like operating systems, or application serves, the resource usage is distributed with load balancing techniques, and monitored for their responses, in order to detect failures, and necessities for switching over to other resources. Porter [1] gives a good overview of challenges to pinpoint failures in huge server parks on the Internet.

3 The Approach

Given the Internet picture and typical usage, like the example outlined in the introduction, there is a need for a mechanism or service that is a subset of the real service, yet using or exercising the basic properties of the connectivity and functionality used in the service. Assuming that the nodes are able to communicate and capable of for example HTTP communication patterns, passing a small data object around like shown with the dotted arrow in Fig. 1 would reveal if the contributing servers and the network communicate. Here a node, in this example node A, that also has the information of the participating servers or nodes, creates a list of http addresses, and some administrative data, and passes this to the next server or node on the list. The receiving server or node does the same but uses the next entry on the list. The final entry on the list can be the originating server or node, but this is only necessary in particular cases, where trip time measurements are desirable.

3.1 Assumptions

The “TCPDUMP” tool [6], which is part of most operating system, or can be installed, would be able to capture any or selected traffic between two IP endpoints, and pass the log files to a central node for further processing and analysis. However the capture process can be resource intensive, and would not cover the aspects of the application level availability. In order to be able to monitor a given node at application level, some software must be installed. Another alternative is use of tools like “PING” or “TRACEROUTE” to verify connectivity between nodes, but this does not detect application server internal issues. There are also tools available that require installation of probes, that in turn sends the information to a central or distributed management platform, but this requires commercial agreements across many administrative domains. The traffic generated by the probes, might get lost, and also have a

volume that consumes unnecessary bandwidth on the communication links. Some of the probe mechanisms do require dedicated communication ports, requiring changes to firewalls and communication links, and in some administrative domains this is not a feasible path, due to restrict policies on configurations of firewalls and communication links.

The Web Service world on the Internet makes extensive use of servlet techniques, to be able to split applications in multiple tiers. Typically they are deployed with a HTTP front-end, an application logic layer, and a database layer, hence the term 3 layer architecture. The easies way to accomplish this is with WAR files, which are deployed into J2EE servers. Making the token passing as a simple service in this way will make use of the front-end and the application logic, and hence cover a minimalistic, yet functional set of that particular server. This works equally well with virtualized servers. One reason for choosing HTTP as token bearer is that most firewalls are open for HTTP traffic, and no network configuration is required.

3.2 The Token

The data structure that is passed around is a list of stings and a time reference, a reference to the next node, which is circulated. The data structure is shown in Fig. 2. Where the first server is Server D and the last server on the route is Server A.

- Token identity number
- Token dispatch time
- Pointer to next URL
- http://serverD.example.com/Echo_Token
- http://serverB.example.com/Echo_Token
- http://serverC.example.com/Echo_Token
- http://serverE.example.com/Echo_Token
- http://serverA.example.com/Receive_Token

Fig. 2. The internal structure of a token

This data structure is the passed around, and the pointer is incremented with one, for each node that receives the list. A typical length of this list is 5-10 URL's, that make up the service in question, that one wishes to monitor.

3.3 Token Flow

The token is created in the generator, and then passed through a set of nodes called the echo instances, according to the list, and finally ends up in a receiver node, see Fig. 3. The current version of the algorithm is implemented in Java, and writes data to log files locally on each node. The data contains a timestamp, available free memory in the Java virtual machine, HTTP status of the request, and the time it used to do the trip end to end. The data collected on the sender side is shown in Fig. 6.

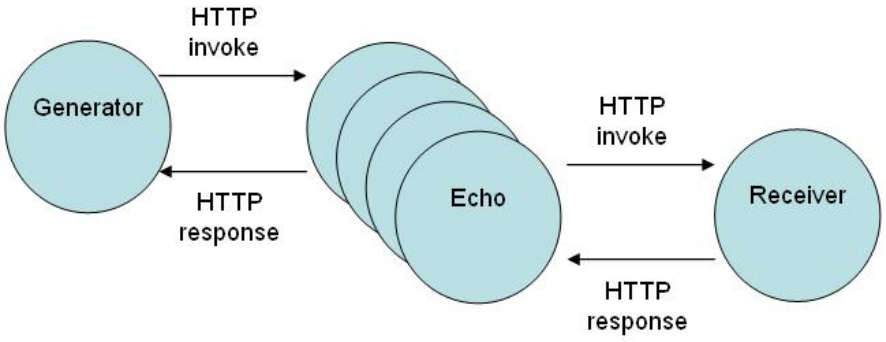

Fig. 3. Passing the token end-to-end

3.3.1 Generator

The token is dispatched at fixed 30 second intervals, in order to minimize the bandwidth consumption of the token itself. Since the assumption is that 10 seconds would be the maximum tolerable delay for a user, we measure the trip time, as part of the token passing. The use of the HTTP protocol built on top of TCP protocol, means that the invocation is robust and capable of internal retries in case of network faults. However if the IP link does not return to a operational state, the link will eventually time out, and signal to the invoking service that communication could not be established, and resulting in error messages on the generator side.

3.3.2 Echo

As shown in Fig. 1, servers are marked B, C, D and E, have Web Services capable of the executing the Echo function. These intermediate nodes will then upon reception of a token inspect the data structure shown in Fig. 2, and increment the pointer, and then forward the token to the next server on the list, by invoking its URL in the token. This means that the TCP connection is held open over all nodes, until the last node on the token list is accessed.

3.3.3 Receiver

The receiving endpoint could be any HTTP capable node, but in order to measure the trip time for the token, the receiver resides on the same physical node as the sender, and hence is using the same time reference. It is therefore easy to compute both the round-trip time, as well as the end-to-en time.

3.4 Experiment Evaluation

From the individual log files that are collected, the interested data can be extracted and plotted. In Fig. 4 shows a plot for the month of January. The tokens have a sequence number included, and if we plot the difference between two consequent tokens, we get a plot on how many have been lost. The plot shows multiple occasions where large numbers of token are lost, and this translate several minutes of down time in a given service. Since the token arrival time is known, a watchdog mechanisms dispatches SMS's to the author when a few consequent tokens are lost.

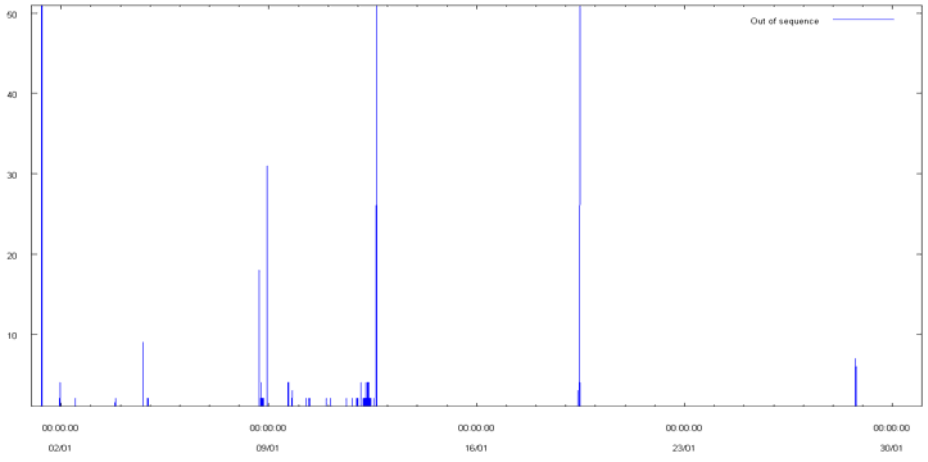

Fig. 4. Tokens on receiver side

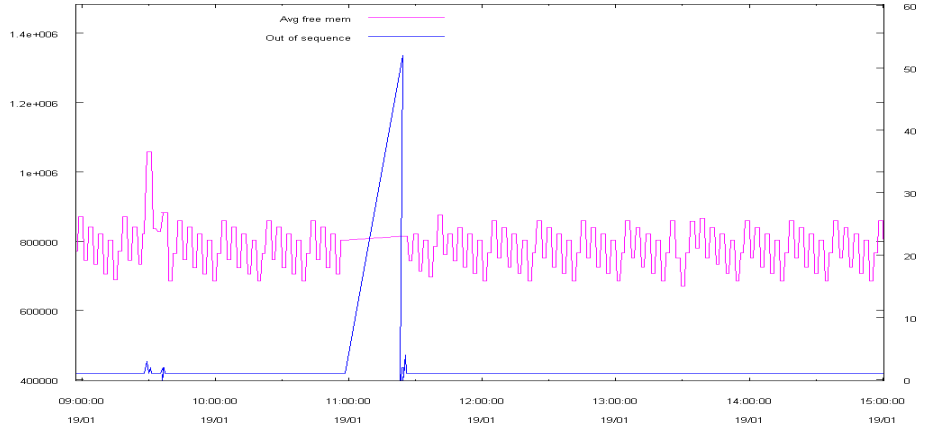

Fig. 5. Zoom in on token reception

Since log files are created at all nodes, one can do post processing of the data and analyze in grater detail on what goes wrong. Zooming in on the incident on the 19th of January, as shown in Fig. 5, the plot shows in blue that at 11:00, 50 tokens are lost.

A few tokens are also lost at 9:30. The cyan plot is the available free memory. The saw tooth shape of the cyan plot illustrates the garbage collection in the JVM. However in a real situation the data from the end node might not be accessible if it resides in a different administrative domain. Therefore it is of interest to look at the data generated at the sender side.

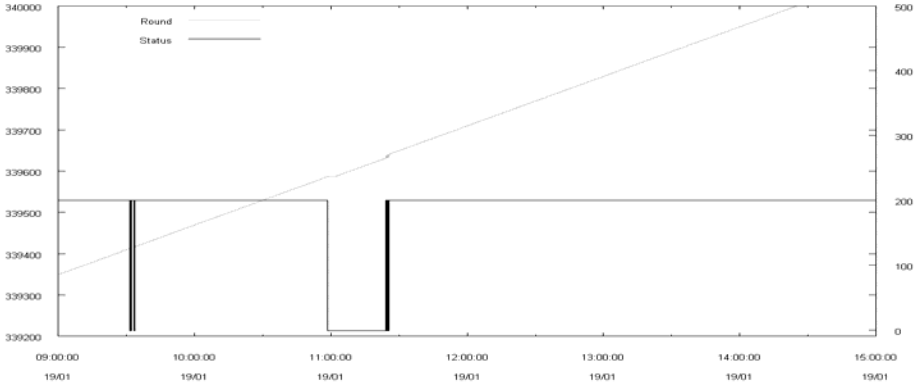

Fig. 6. Status at sender side

The black graph is the HTTP status, and the gray graph is the token sequence number. When the black graph stays at the value "200", we have normal operation, otherwise there is some kind of failure, like when the graph shows "-1" indicating that there is no route or connection to the next application server. The large dip in the plot at 11:00 indicates a particular incident, which was due to human failure. A system administrator replaced a network cable to a switch at 9:30, and then later at 11:00 decided to reconfigure a router, but accessed the wrong unit, causing the network to fail, and then it took 27 minutes to restore the operation.

4 The Trial System

The system is build from a set of servers running various flavors of Unix operating systems, Tomcat application servers [5] and JVM [6]. The system has run continuously since November 2008. The servers span over three different administrative domains, and in one of the domains the server is installed a virtualized node. None of the organizations have automated monitoring of their servers. Due to the sequential behavior of the algorithm, all port numbers are cycled in a short time period, and miss configurations in routers and firewalls are quickly detected, and administrative action can be taken. For the month of January 2010, 85116 tokens were dispatched from the generator. Of these 1459 tokens experienced a delay for more than 5 second, and 253 tokens failed, while only 3 tokens got other error messages. The incident on the 19th of January accounts for 52 lost tokens. Of all the tokens in the period 215 tokens were out of sequence, meaning that they were delayed more than 30 seconds. This is due to the robustness of TCP protocol, the tokens are eventually retransmitted, and not lost until the timeout of roughly 200 seconds is exceeded. None of the servers had any downtime in the entire period, so the unavailability is only due to network issues.

5 Conclusion and Further Work

This paper has briefly outlined a method to detect unavailability, and described one trial system. Only a small excerpt of the data is shown in this paper, but there are several interesting observations in the data that are under investigation. The variations in available free memory around the times when tokens are lost, and the distribution of the round trip time, need further investigation. Also when one has detected that the service indeed is down, different service recovery strategies should be trial as well.

However this approach shows that it is indeed possible to detect downtime across different administrative domains, and as a result maintenance action can be taken. Further work is to investigate how alternative servers or nodes could improve the availability, and how the knowledge of the unavailability could be distributed in order to trigger reconfiguration at the different administrative domains.

Acknowledgments

I would like to thank Professors Rolv Bræk and Bjarne Helvik at Department of Telecommunication at NTNU, for their advice and guidance in my research work.

References

1. George, P.: Improving Distributed Application Reliability with End-to-End Datapath Tracing, PhD at Electrical Engineering and Computer Sciences, University of California, Berkeley. Technical Report No. UCB/EECS-2008-68, May 22 (2008) http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-68.html
2. Bouricius, W.G., Carter, W.C., Schneider, P.R.: Reliability Modeling Techniques for Self-Repairing Computer Systems. In: Proceedings 24th National Conference ACM (1969)
3. Avižienis, A., Laprie, J., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. Dependable and Secure Computing 1(1), 11–33 (2004)
4. Java virtual machine, http://java.sun.com
5. Tomcat, http://tomcat.apache.org/
6. TCPDUMP tool, http://en.wikipedia.org/wiki/Tcpdump