# The Hydra: A layered, redundant configuration management approach for cloud-agnostic disaster recovery

Ke Huang
Institute of Informatics
University of Oslo
kehuang@ifi.uio.no

Kyrre Begnum
Department of Computer Science
Oslo and Akershus University College
of applied sciences
kyrre.begnum@hioa.no

***Abstract*—This paper demonstrates a bottom-up approach to developing autonomic fault tolerance and disaster recovery on cloud-based deployments. We avoid lock-in to specific recovery features provided by the cloud itself, and instead show that tools used in system administration today can provide the foundation for recovery processes with few additions. The resulting system, Hydra, is capable of detecting failures in instances and can redeploy any instance at a new location without human intervention. The layered design of configuration management tools enables separation of recovery processes and the actual service, making the Hydra applicable to a wide range of scenarios. The implementation was tested and an analysis of the recovery time is provided, demonstrating that the Hydra is capable of completely rebuilding a new site in 15 minutes.**

Keywords: Cloud Computing, Autonomic management, Fault tolerance, Fail-over, Configuration management

## I. Introduction

With the massive growth of cloud-based services, there are many options for storing and sharing information on the Internet. Components such as servers and network management solutions have been delivered as services by many cloud providers. With the obvious benefits in flexibility, high availability and pay-as-you-go cost models, virtual infrastructures have been purchased by many companies as resources to run their sites rather than hosting them on their own. The accelerating development of cloud services and technologies has been accompanied by several critical issues, such as the danger of unpredictable connectivity and unexpected outages. Even the most popular hosting providers, such as Amazon Web Services and Rackspace, still occasionally suffer from service interruptions or failures [1], [2], [3], [4], [5], [6].

This paper aims to address global outages from the ground up. Using established frameworks and tools in the system administration profession, we demonstrate that fault-tolerant behavior can be reached with little additional effort and while keeping solutions within the domain of system administration in order to limit complexity. Node health as well as deployment information can be handled by tools already in use in real life. This paper explores a novel mechanism to take advantage of cloud services and recover automatically from unforeseeable system failure, without manual interaction. The design enhances the reliability of cloud services by providing a distributed, self-organizing structure built from standard configuration management and storage tools that can detect and heal failures spontaneously. Moreover, it provides a degree of portability and usability as it is built to be free of local/proprietary cloud-based features and thereby enables cloud-agnostic adoption with little modifications.

## II. The Hydra Model

The terminology used is explained as following:
**Zone:** A set of sites which locate in the same availability region of a cloud.
**Site:** A set of servers which provides a service to users. A site normally consists of one master server and several agent servers which have different functions related to the purpose of the site.
**Master Server:** A server that controls the behaviors of agent servers and sends out reports of their status regularly.
**Agent Servers:** Servers in a site that host a set of designated services.
**Controller Server:** A server which supervises other servers in a site and manages their responsibilities. It also communicates with other controllers and issues regenerating process if one of the sites/zones it communicates with appears to be down.

### A. Architecture of Hydra

In our design for Hydra, a fully redundant system, combines several sites which can run in parallel over multiple geographical locations (zones).

The zones are located in physically distinct regions to ensure that a disaster occurring in one geographic region would not cause service disruption of the whole system. However, such a geographic distributed system may introduce network latency if the users are served by the servers that are not in the same geographic region. Hydra avoids such situation at the zone creation stage by ensuring that whenever a new zone is needed, it is always established in a region where no other zones are present.

Figure II-A shows an example of the Hydra architecture in a zone. In this figure, multiple sites located in the same zone are directly monitored by a controller. In the controller, the monitoring process keeps checking the status of every site, and failed nodes are automatically detected. Once a site failure

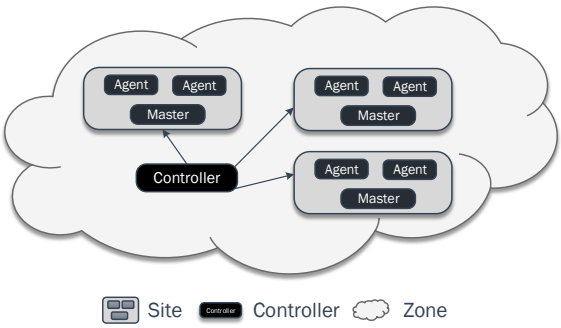


Fig. 1. Architecture of a Zone

is confirmed, the regeneration process will then be triggered, and one of the remote controller servers will be in charge of rebuilding the site somewhere else. The remainder of the malfunctioning site will then be terminated and replaced by a new site that is created at a location chosen by the controller. All sites are assumed to be active and receive part of the traffic. Therefore, instead of having backup sites sitting idly in cloud waiting for activation, a new site will only be created and brought online after the outage of an old site has been confirmed.

Data synchronization is essential requirement for cooperation between Hydra nodes and providing data availability. This synchronization is taken care by a cluster file system. In the design, the controllers are sharing a directory that is provided through the cluster file system. Whenever there is a change in the directory, such as a modification or upgrade of the configuration files, it propagates to the rest of the cluster in a short time.

When a new site is created in an availability zone, it automatically synchronizes and replicates the most recent metadata of the whole system amongst all the controllers. This means that any controller can rebuild any other controller, which in turn can rebuild any master and agent server. This ability comes from the integration with configuration management. Configuration management tools store policies on each type of system. This policy directs the automatic installation and configuration of any system which is part of the Hydra. The policy descriptions are relatively small, consisting of a collection of text files. They also tend to stay very stable, so there is little bandwidth needed for synchronization. This strategy gives the Hydra its needed robustness. That is, even if there is only one controller, this single controller will have enough information and ability to replicate itself and then rebuild the entire Hydra system.

Through the use of configuration management as the deployment mechanism, the kinds of operating systems or the services that run on the sites are largely irrelevant. The intention of this design is to support any kind of operating system and service using the same Hydra model. The individual details of each agent would be manifested in the concrete configuration policies for that agent, and not interfere with the overall configuration of the Hydra. The approach taken here is to layer the configuration management into the controller, a master and an agent. The controller will provide a configuration management service to all masters in its zone. The master will have its own configuration management server which is configured by the client based on instruction from the controller. The agents connect their client to the master and receive the instructions on how to configure their respective service (see Figure 3).

There are several reasons for this choice. This layered approach will limit the number of client connections to the controller. Each master server will handle the configuration of its agents. Also, in line with common system administration practices, it is wise to separate responsibility of the site maintainers and the Hydra maintainers. A functioning Hydra controller is critical and should be kept as simple as possible as any syntax error in the policy would halt the entire functioning. Should it go down due to a crash, the zone would be unsupervised. Furthermore, the controller can detect the failure of a master and its site and rebuild it locally, providing extra robustness against local crashes.

Aside from managing their local zone, controllers keep communicating with each other, tracking the availability of entire zones in other regions. If a zone fails, such failure will be detected by a controller from another arbitrary zone, which then sends out an error signal and propagates it to controllers in all zones. The controller with the lowest ID is in charge of enacting any changes, such as terminating the remainder of a zone and rebuilding it somewhere else. More details of how the Hydra behaves in certain error situations can be found in [15].

## III. Implementation

A Hydra implementation has been built to prove the concept as well as provide data on the speed of rebuilding a zone based on the layered approach. In order to have the most global spread, Amazon EC2 was used as the cloud platform. In the implementation, the Hydra uses multiple zones with one controller and one site at each zone. Every site has a master and three agent servers. The controllers are all nodes in a cluster filesystem, and thereby create replicated storage of configuration management policies across them.

GlusterFS is used as the cluster file system in the Hydra. GlusterFS is a well known open source platform for cluster cloud storage file system. It is widely used and easy to manage. Furthermore, it allows all the nodes to be clients of the cluster and requires no meta-servers. This means that all the controllers are GlusterFS cluster nodes as well as the clients of the same cluster. The storage volume is set up in a fully replicated mode, meaning that all the files are replicated in all nodes. When a new controller is deployed, it will connect first as a new cluster node, obtaining a full local copy of the volume, and next as a client, mounting the filesystem for local use.

The Hydra makes use of MLN to deploy, start and stop virtual machines in Amazon EC2. MLN (Manage Large Networks) is a tool to build, configure and manage large groups of virtual machines based on Amazon EC2 and Eucalyptus[13]. Its plugin system allows it to be adapted to other cloud technologies as well.

Configuration management is handled through Puppet[14]. The architecture of Puppet is a master/client design, utilizing

a centralized Puppet master and any number of clients which pull their updated configuration from the master at a defined interval. In order to make it work with the layered model of the Hydra, Puppet is set up in the following way: Puppet is installed on controllers, masters and agents. The controller will only provide its configuration to master servers. The configuration manifests are located in the cluster filesystem and available to all controllers. On the master, Puppet behaves both as a client and a server. The client will get its instructions from the controller, which also instructs it to set up its own Puppet service. The configuration policies are installed from the controller, and they also reside on the cluster filesystem. However, since the master does not have direct access to it, this installation is done via Puppet. The master's Puppet service will serve the Puppet clients on the agents.

The monitoring of the controllers is based on script-based inspection of GlusterFS storage health as well as Amazon EC2 API to get status information. This is the only interaction with the cloud service. Monitoring of the master and agents is done by investigating Puppet client intervals and status reports from the APIs. If a virtual machine is unresponsive for a defined period of time, meaning its Puppet client stops connecting, or is reported as terminated or shutdown by the cloud itself, it is considered down by the controller. This requires that the masters report Puppet logs back to the controller, which is handled by the regular operation of their own Puppet client.

### A. Controller Server

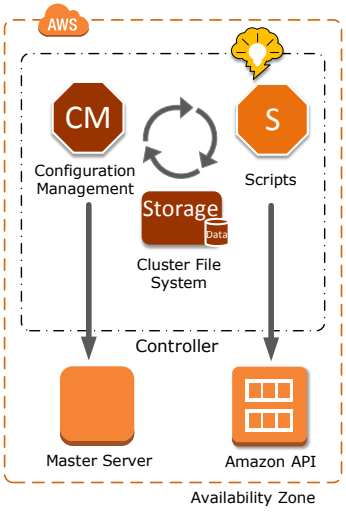


Fig. 2. The architecture of a Controller.

Figure III-A shows the architecture of a controller server. Local scripts that run on the controller compare status information and make decisions on whether all nodes are operational. If a controller detects the failure of a local agent or master server, it will attempt to rebuild it directly. If another controller is unresponsive or the whole site does not respond to service requests, it will be considered down and rebuilt by one the controllers. It is important to note that although all the controllers monitor all the agents and masters, only one controller will take action. When a controller is being rebuilt, information about it will be updated in the shared filesystem, allowing all controllers to see its new location and start to monitor it.

### B. Rebuild process

The procedure of rebuilding a zone is the following:

1) Terminate the remainder of the damaged zone $Z_{damaged}$
2) Find a new location for the zone $Z_{new}$
3) Deploy and boot a controller $C_{new}$ at $Z_{new}$. $C_{new}$ will receive instructions for how to connect to the Hydra.
4) The $C_{new}$ will connect to the cluster filesystem and get access to configurations
5) $C_{new}$ sets up a Puppet master and becomes a part of the Hydra
6) $C_{new}$ detects that master and agents which should be in $Z_{new}$ are missing and starts recovery of them
7) $C_{new}$ deploys and boots a master $M_{new}$
8) When $M_{new}$ is ready, the agents $A1_{new} \dots An_{new}$ are deployed and booted

It may be the case that a network outage disconnects a zone long enough for the Hydra to rebuild it somewhere else. The API of that cloud may also be unavailable, preventing the zone from being terminated. The old controller then will notice that it has been replaced because it has been removed from the storage cluster. It will then proceed to terminate all masters and agents in its zone and finally itself.

## IV. Analysis

A set of experiments were carried out to test the features of the Hydra. Of particular interest is the overall stability of the recovery process and the time it takes from when a full recovery starts until the service in the site is operational. A web site was chosen as the service to be run by all sites. The three types of disaster situations was (S1) the loss of a server, (S2) the loss of a site and (S3) the loss of a zone.

During each experiment, the failure was induced by manually shutting down one or more instances from the EC2 management console. The logs on the controllers then provided the necessary data for when the recovery steps were taken and how long they took.

For the sake of brevity, we will only discuss the results from scenario S3, since it encompasses S1 and S2. This same scenario was repeated seven times. Our initial findings showed that a complete recovery of a zone was achieved in about 25 minutes using the m1.small instance type. Using a stronger instance (c1.medium) will cut the time to 15 minutes from the time the old zone is terminated until a webserver agent in the new zone responds to clients. The performance gain is most pronounced from m1.small to c1.medium, indicating that the time of the API calls becomes the main factor as the virtual machine becomes more powerful.

We can see from the table that the stage that takes the longest is to create the controller server. In this stage the instance needs to install Puppet and GlusterFS software packages and dependancies, which are downloaded from repositories. The following state is to connect the new controller to the

| Stage | m1.small Average | c1.medium Average | c1.xlarge Average |
|---|---|---|---|
| Terminate a zone | 207 | 69 | 47 |
| Relocate a new controller | 42 | 46 | 24 |
| Controller up again | 694 | 510 | 463 |
| Rebuild Glusterfs | 217 | 135 | 138 |
| Create master and webservers | 236 | 92 | 77 |
| Master up again | 132 | 80 | 75 |
| Webserver up again | 16 | 10 | 9 |
| Sum | 1545 | 941 | 833 |

TABLE I. THE TIME SPENT IN SECONDS AT THE DIFFERENT STAGES OF THE RECOVERY PROCESS.

cluster filesystem and get a replicated copy of the files. Once this is in place, the next stage will deploy the master server and agents. It takes a while for the master server to get its Puppet server up and running. In the mean time, the agents have booted and will constantly try to connect their Puppet client to the master until it is ready to accept. The resulting time to install the webserver on the agents is consistently short regardless of hardware type chosen for the agents.

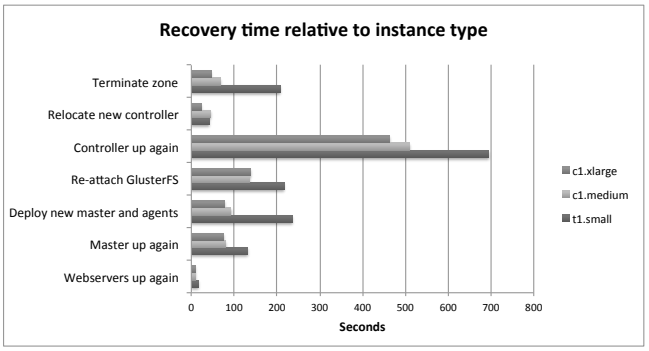


Fig. 3. The recovery time is affected by the performance of the master and controller node. Using a more powerful node will reduce the overall time by

## V. DISCUSSION AND CONCLUSION

One of the key concerns listed in the beginning was that if disaster recovery processes are provided by the cloud itself, they may create a lock-in for customers who become dependent on them. The Hydra only makes use of the API to get status information for each instance. The same functionality can be expected from every cloud provider with small variations in API technology. This means that the Hydra can be adapted to work on different clouds and also span multiple clouds at the same time.

Our data show that the recovery time will decrease with stronger hardware types. This is interesting and one would be tempted to automatically choose the most powerful alternative. However, the hourly price of m1.small ranges from USD $0.060 to $0.088 while the price for c1.medium is USD $0.140 to $0.200 and c1.xlarge varies from $0.580 to $0.740. This means that with three controllers and three masters the 10 minutes saved comes at an increased cost of almost USD$16 per day. For very large deployments, this added cost is perceived to be of little consequence. With more complex services running in the sites, the recovery process may take longer. The fastest recovery was with the strongest instance, the fastest recorded time was 718 seconds, about 11 minutes. It is unclear if the increase in cost would justify the performance.

The Hydra is a disaster recovery solution built on the tools Puppet, MLN and GlusterFS. It demonstrates that complex deployments can be rebuilt elsewhere using the established approach of configuration management and automated deployment. The layered design of the Hydra allows for a distributed group of controller nodes to manage each other and local sites. The sites are managed in a traditional local centralized architecture. Monitoring information is gathered from the the cluster storage, configuration management logs and cloud API. Recovery times have been observed in real-life experiments to be down to 15 minutes.

## REFERENCES


[1] H. S. Gunawi, T. Do, J. M. Hellerstein, I. Stoica, D. Borthakur, and J. Robbins, *Failure as a service (faas): A cloud service for large-scale, online failure drills*. University of California, Berkeley, Berkeley, vol. 3, 2011.

[2] M. W. Thomas, Rackspace resolves email outage following possible denial of service attack, *http://www.bizjournals.com/sanantonio/news/2013/01/14/rackspace-resolves-email.html,* Last accessed July 2013.

[3] Amazon, Summary of the december 24, 2012 Amazon ELB service event in the us-east region, *https://aws.amazon.com/message/680587/*, Last accessed July 2013.

[4] S. Ludwig, Amazon cloud outage takes down Netflix,Instagram, Pinterest, *http://venturebeat.com/2012/06/29/amazon-outage-netflix-instagram-pinterest/*, Last accessed July 2013.

[5] J. Scott, Amazon and microsoft cloud services hit by lightning strike, *http://www.cloudpro.co.uk/cloud-essentials/1453/amazon-and-microsoft-cloud-services-hit-lightning-strike* Last accessed July 2013.

[6] B. Butler, Amazon outage one year later: Are we safer? *http://www.networkworld.com/news/2012/042712-amazon-outage-258735.html,* Last accessed July 2013.

[7] V. Sarathy, P. Narayan, and R. Mikkilineni, Next generation cloud computing architecture: *Enabling real-time dynamism for shared distributed physical infrastructure, in Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE),* 2010 19th IEEE International Workshop on. IEEE, 2010, pp. 48-53.

[8] C. S. U. Omar H. Alhazmi, Taibah University Yashwant K. Malaiya, *Evaluating disaster recovery plans using the cloud*,Software Reliability Engineering, 2012.

[9] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker and I. Stoica, *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing*, Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, 2012, USENIX Association

[10] E. Juhnke, T. Dornemann and B. Freisleben, *Fault-tolerant BPEL workflow execution via cloud-aware recovery policies*, 35th Euromicro Conference onSoftware Engineering and Advanced Applications, 2009. SEAA'09, p. 31 - 38, 2009,IEEE.

[11] J. Deng, SC-H. Huang, Y. Han and J.H. Deng, *Fault-tolerant and reliable computation in cloud computing*, GLOBECOM Workshops (GC Wkshps), p. 1601-1605, 2010, IEEE

[12] N. Bonvin, T.G. Papaioannou and K. Aberer, *A self-organized, fault-tolerant and scalable replication scheme for cloud storage*, Proceedings of the 1st ACM symposium on Cloud computing, p. 205-216, 2010, ACM

[13] K. Begnum. *Simplified cloud-oriented virtual machine management with MLN*. Journal of Supercomputing, 2010

[14] L. Kanies, *Puppet: Next-generation configuration management*,The USENIX Magazine. v31 i1, p.19-25,2006, The USENIX Association.

[15] K. Huang, *The Hydra: An automated disaster recovery solution for the cloud*, MSc Thesis, University of Oslo, Norway, 2013