ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645
Indian Journal of Science and Technology, *Vol 8(S9), 181–187, May 2015*
DOI: 10.17485/ijst/2015/v8iS9/65567

Coalition of Cloud Monitoring Systems Postulating Anti-Fragility

G. Akshaya*, K. Subha, R. Tamizh Baggiya, P. Padmakumari and A. Umamakeswari

Department of Computer Science and Engineering, School of Computing, SASTRA University, Thanjavur - 613401, Tamil Nadu, India;
akshayagopalakrishnan16@gmail.com, vaishubama@gmail.com, tamizh.rtb@gmail.com, padmalec.sastra@cse.sastra.edu, aum@cse.sastra.edu

Abstract

The objective of our project is to build an anti-fragile system where in a proactive measure build is done to handle errors that can occur and is infused into separate databases for every client by server. A method similar to fault injection is deployed, to make the system more efficient in handling errors that occur. Certain of the faults are built and are infused inside the database; this technique employed here is a proactive method. Foreseeing certain errors and creating those errors manually for every of the client so that a notification is sent when those faults occur. Building an efficient system that reduces the down time in predicting the errors that occurs and helps improve performance of the cloud. Using Vm's the current working state can be saved and thus enhancing more of back up and ensuring no data loss at any point of time since a redundant server can be made to take up the original server using zero gigabyte transformation. Aiding the system performance with monitoring helps build a strong and anti-fragile system monitoring every activity a client does. The whole of monitoring is done by server. The server supports n clients simultaneously.

Keywords: Anti-Fragile, Notification, Proactive, Redundant, Zero Gigabyte

1. Introduction

Cloud computing plays an important role in the distributed environment. It relies on sharing of computing resources rather than having inadequate local storage or personal devices to handle applications. It is a general term that involves delivering hosted services over the Internet. Hence user can access any services without knowing where the services are hosted and how they are delivered. The complex and scalable nature of cloud makes system prone to errors. To manage these errors without affecting system performance we need an eye which continuously monitors the system and helps to make system more strong responding to errors. Here key concept is anti-fragility which strengthens a system from the occurring faults instead of weakening it. With this technique system can handle unpredictable and irregular events and resist spikes. Adapting failure induction technique for cloud system enhances anti-fragility. This paper focuses on how to learn from failure and builds an invincible system.

2. Related Works

An approach based on fault tolerance, measures the ability of a system to respond to an unexpected failure. Main benefit of this approach is the fault recovery but it is a reactive measure taken after the occurrence of error[4]. Challenges in implementing fault tolerance techniques in cloud computing and comparison of various tools used for it is studied in detail[6]. Window-based violation monitoring, state monitoring and multi-tenant state monitoring

*Author for correspondence

techniques are discussed which helps to gain a strong foundation on the monitoring[3]. Another approach on real time monitoring for cloud resources proposes an adaptive algorithm for scalable and reliable cloud monitoring. It balances amount and quality of monitored time series and reduces monitoring costs significantly without affecting data quality[1]. But the limit of the proposed system cannot match the exponential growth of system components and data size. Fault Tolerance policies in cloud platforms are discussed in a system which covers fault detection and management techniques. It deals with two important aspects like exclusive FT management and collaborative FT management[5]. But FT techniques based on check pointing is lacking which needs to be improved by providing new solutions. An important approach based on failure induction technique, monitoring and learning mechanism was proposed where we increase the frequency of failure to better prepare the system in case of real time errors[2]. But this architecture does analysing of data only from virtual machines and not concentrated on physical machine. The paper concentrates on designing anti-fragile system in cloud environment by generating errors and maintaining them in error logs.

3. Concept of Anti-Fragility

Everything requires monitoring; similarly the cloud that handles a lot of clients as well as multiple servers also requires monitoring. A system that tries to incorporate the monitoring scheme with a new technique called the anti-fragility. A system when is termed anti-fragile becomes more resilient to errors, the basic principle here is that it builds out a very strong system from the occurring errors. It takes a leading edge by not forcing the system to crash or respond slowly with the occurring faults but it builds a very strong proactive mechanism that can overcome the faults and increase the performance thus reducing the overhead. An anti-fragile system is built by inducing the errors manually before it even occurs in the real time. The methodology incorporated here is the fault injection.

3.1 Parameter Definition

The Monitoring algorithm is characterized by the trade-off with two parameters. The first parameter is pay-off which is defined as one minus the ratio between Induced Errors (IE) and Existing Errors (EE). Pay-off (p)takes up values from [0, 1].

$$p = 1 - \frac{IE}{EE}$$

The second parameter Q (Quality measure) quantifies the capability of an algorithm to depict the load spikes with high accuracy. It takes up two factors into consideration, the faults induced by the monitor using sampling intervals larger than t0 and the ability to substantiate the load spikes in the monitored data. Q is defined as the combination of normalized error which is calculated from normerr (Normalized Root Mean Square Error) and Mrate which characterizes the weighted average of generate and retrieve, where generate is the fraction of induced faults collected from the samples (λ), retrieve displays a message if the fault occurred exists in the log. Both take up values from [0, 1].

$$Q = \frac{Mrate + (1 - normerr)}{2}$$

3.2 Fault Induction

Server plays a vital role in configuring an anti-fragile system. Applications run using the tomcat server. It is called the "monitoring eye" since it takes complete hold of every single action. The tomcat server is powered on before starting the monitoring process. The tomcat server records all events a client does from the initial logon to the logout. As to provide an anti-fragile model faults are recorded to log and prepare a defensive system highlighting the proactive lead. Also mentionable monitoring eye induces the faults manually through a method of fault injection and stores them in a separate log. The faults that occur in network and database are concentrated on the server part. The network error mostly concentrates on connection time out and is induced by pinging to an unknown IP address and in return a proper establishment of connection with the IP is not possible and is recorded into the event log. Similarly a database error is induced when the appropriate database is not found or no such database exists. This was induced by manually crashing the existing database and generates the fault and is stored in the event log. Server in here induces errors and stores them.

Only the authorized clients are allowed to access the cloud. The client side monitoring is also done by the server to monitor any errors a client does, grouping all the application errors as the client side errors. An error that a client does after entering into an application is recorded

and stored in log. Basically a divide by zero error is introduced as client side application error. The client is not given any rights to view the error message or logs. Only notification messages are displayed to the client.

Algorithm 1. Failure Induction

1. F<=0
2. te->1
3. While te<5 do
4. if te is 1 then
5. F => Induce N (e)
6. Update NF
7. NF [] <- (d, m)
8. if te<-2 then
9. F <= Induce D (e)
10. Update DF
11. DF [] <-(d, m)
12. if te<-3 then
13. F <= Induce A (e)
14. Update AF
15. AF [] <-(d, m)
16. if te<-4 then
17. F <= Induce All (e)
18. Update ALF
19. ALF [] <-(d, m)

Initially the Failure (F) is set to zero (line1). An integer variable (te) is assigned (line2) to choose failures for induction (each te value represents a type of failure). When te is 1 a network error N (e) is induced and updated in the Network Fault (NF) database with parameters date (d) and error message (m) (line 4-line 7) and so all the other errors like Database D (e) (line 8-line 11), Application error A (e) (line 12-line 15) and all errors combined All (e) (line16-19) are induced and updated in the corresponding fault databases.

3.3 Event Log

Event log can also be termed as the main database for monitoring. This aids in maintaining an antifragile system recording all the faults that were induced by the server. The event log maintains the date of occurrence of the faults. The event log is created as an independent table under the main database and is only available to the server for viewing. All the individually generated errors are integrated and stored into a single console. Every fault that is generated is classified according to the type as network, database or application faults. Description of every error is elaborated for better understanding and for providing the accurate solution in case the client requests. Integration of all errors into one is a new technique under innovation. In case of non-induced errors monitoring system considers it as a new fault and sends it to Fault Tolerant Mechanism for further processing then get solutions are updated into event log for further maintenance.

Algorithm 2. Performance Evaluation

1. initial time<=t0
2. {P1, N1, P0, N0} <=0
3. normerr <=0
4. r= avg (ts) ± 1.5• std (ts)
5. while k < λ • t0 do
6. pk <= k
7. k<= k+ t0
8. ts0 <= ts [pk]
9. if ts[k] then
10. ts0<-TS[k]
11. $normerr <= normerr + \left(\frac{ts[k]-ts0}{max(|ts[k]|,|ts0|)}\right)$
12. {P1, N1, P0, N0} <= {P1, N1, P0, N0} + monit(r, ts[k], ts0) n
13. $normerr = \sqrt{\frac{normerr}{\lambda}}$
14. $generate = \left(\frac{P1}{P1+N0}\right)$
15. $retrieve = \left(\frac{P1}{P1+N0}\right)$
16. $Mrate = \left(\frac{2.generate.retrieve}{generate+retrieve}\right)$
17. $Q = \frac{Mrate+normerr}{2}$

Initial time is set to 0 (line 1) and the true or false positive counts are initialized to 0 (line 2). The Normalized Root Mean Square Error value (normerr) evaluates the sum of square errors which is initialized to 0 (line 3). The mean deviation value is computed (line 4). The time stamp value is calculated (line 5 – line 10). The normerr value is then updated (line 11). True or false positive counts are updated with monit function which limits the detection of errors (load spikes) within the mean deviation value

(r), using the initial and Kth time stamp values (line 12). The square root for the normerr of the induced samples is evaluated and the normerr is updated (line 13). Two variables generate and retrieve are designated to induce faults and displays the message if the fault already exists in one of the fault databases (line 14 and line 15). Q (Quality measure) is the combination of Mrate (monitor measure) and normerr where Mrate is computed using generate and retrieve (line 16, 17).

Algorithm 3: Anti-Fragile Monitoring

1. tmin <=0
2. max =0
3. trig <=N
4. Δ <=0
5. c <=0
6. pk <=λ
7. k <=pk+ tmin
8. max <=max+k
9. while True do
10. Δ <=Δ + (tsk-tspk)
11. if Δ ∈ retrieve then
12. tΔ << ts
13. set t>0.5
14. PE=t.pay off+ (1-t).Q1
15. else if tΔ exceeds max then do
16. set t=0.5
17. PE=t.pay off+ (1-t).Q1
18. trig++
19. λ++
20. FTM<=Δ

Monitoring interval t_{min} is set to 0 (line 1). A threshold value max is initialized to 0 (line 2). A variable trig is used to trigger the induction of sample failures with minimum time interval (line 3). Δ represents deviation in the server load (failure occurrence) (line 4). Another variable c is used to count the length of the time stamp (line 5). The variables pk and k holds the time interval values of the currently induced failures (line 6, 7). The maximum threshold value (max) is updated (line 8). An infinite loop of monitoring to identify the deviations begins (line 9). If any deviation occurs, the Δ value is updated along with the corresponding time stamps tsk and tspk (line 10). When the deviation Δ already exist in the database, the corresponding response time tΔ is negligible than the entire time stamp (line 11, 12). The administrator sets the value of t (tuning constant) to be greater than 0.5 so that while evaluating the Performance (PE) more importance is given to improve the pay-off value (line 13, 14). If the response time of the particular failure exceeds the maximum threshold value, the administrator sets the value of t to 0.5 to give equal importance to both the pay-off by collecting more samples (λ) of failures for inducing and the Q value by reducing the response time for a particular type of incoming fault if it is already present in the database (line 15, 16, 17) (Table 1). The trigger variable trig is incremented for the newly induced failure (line 18). More samples are taken from the new failure for inducing (line 19). Also this fault is sent to the Fault Tolerant Machine (FTM) (line 20).

Table 1. Performance evaluation

T	Pay-off	Q	PE
0.25	40%	60%	55%
0.50	75%	75%	75%
0.75	60%	40%	55%

4. Performance Analysis

The performance analysis is done with time as metric by comparing the performance of Cloud Service Provider (CSP) with or without monitoring service (M) refers Table 2.

Table 2. Performance P at time stamps ts

Cases	T min	P at ts1	P at ts2	P at ts3	P after Δt
Case-1	0	5.1	3.1	2.1	1.1
Case-2	0	5.0	3.2	4.6	6.0
Case-3	0	5.2	3.1	3.2	4.1

From the graph

Case-1: When a fault occurs in timestamp ts2 the load deviations are not monitored and thus reduce the performance by increasing the server's load making the system more fragile.

Case-2: A monitoring service M (IE) is considered which induces faults and stores it along with an error message in the server's database which reduces the down-

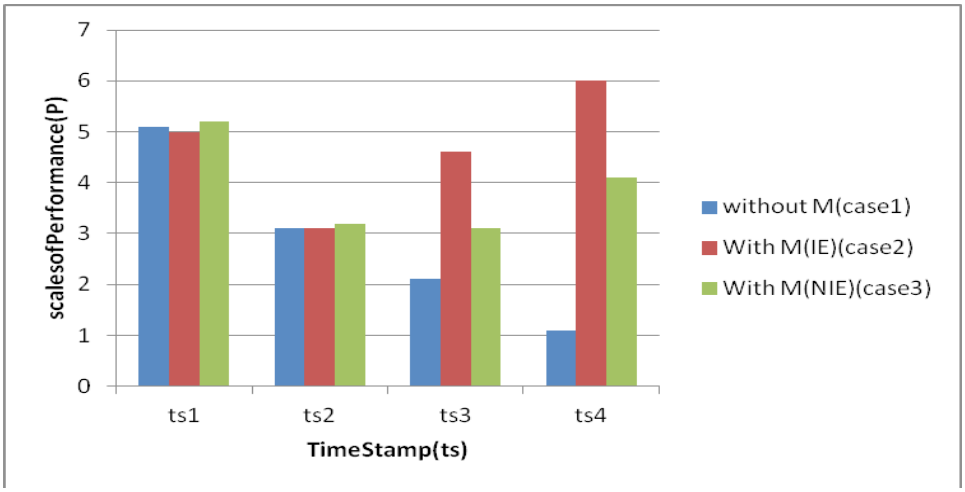

Figure 1. Scales of performance, P (y-axis) and time stamp, ts (x-axis).

time. If a fault occurs say in timestamp t2, the monitor will indicate the server about the fault and the server will respond to that particular fault and clears the spike within the same timestamp and helps in increasing the performance of the CSP.

Case-3: When a fault occurs in timestamp ts2 the monitor will indicate the server about the fault and if it's not in server's database, then the time taken (Δt) for responding will be more at the same time the server will send a notification to the client. In case of un-occurred faults it forwards the fault details to Fault Tolerant Mechanism (FTM).

5. Communications between PM's and VM's

Cloud today provides accessing resources within the same VM, and now the further enhancement is accessing the resources of PM. Communication between multiple PM's can be established by running our application in the tomcat server and accessing it from other PM machine is also possible; Similarly we can access PM form VM giving the PM's IP address and vice-versa. Hence enabling clients the present in their environment and can get the benefit of using the monitoring tool. Multiple client support is provided for a single server so scalability is achieved.

6. Performance Comparison

The graph depicts the comparison between the existing monitoring services like Monitos, Rack space, Kaseya with the proposed system Imon based on the monitoring arenas such as network monitoring, database monitoring, integrated (network and database combined) monitoring.

7. Backup Maintenance

Although the clients can access the resources and switch between using VM's and PM's, the only advantage of using through VM is having a potential backup. Which can't be done in the PM as PM does not enable us with an option of snapshot. The full backup of the running server can be taken while using the VM. VM snapshot of the OS is taken manually at periodic intervals of time, and is stored in a separate place without the interference of the main data. In case the main server crashes we can use the backup snapshot as a recover mechanism enhancing the performance of the monitoring tool.

8. Conclusion and Future Enhancements

This paper acquaints with a new monitoring service called Integrated Monitoring (IMON) which monitors both virtual and physical machine for network, database and application faults by using an advanced technique of failure induction for creating an anti-fragile cloud environment. By inducing failures the system can be strengthened to recover from failures without degrading its performance, thus promising high reliability and performance.

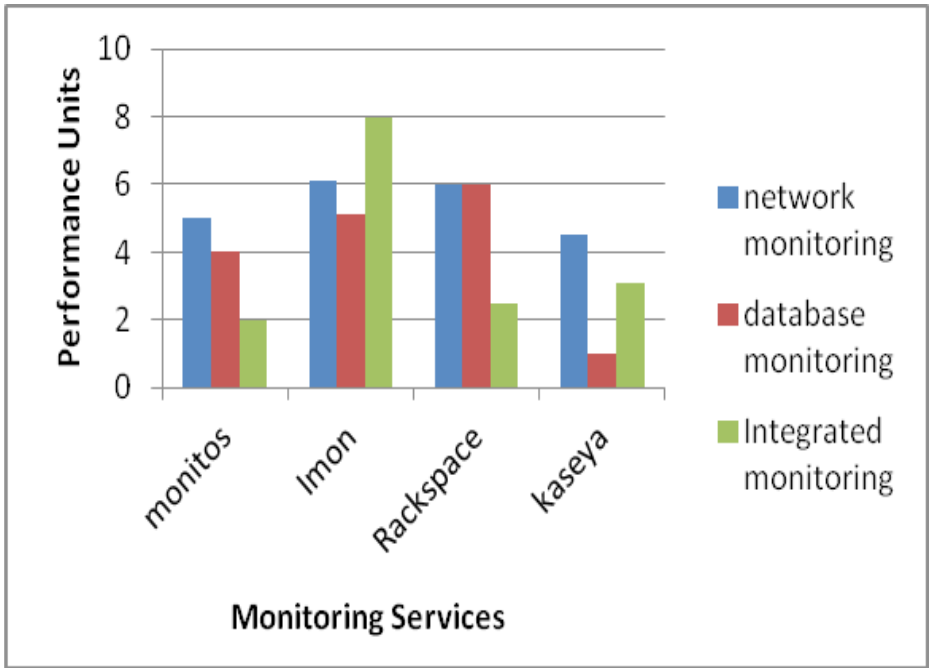

Figure 2. Comparison of monitoring services depicting services (x-axis) and units scale (y-axis).

As a future enhancement, Full automated maintenance of the backup is achievable using VMotion which uses the Gigabyte Ethernet transfer mechanism and keeps backing up the redundant server at every instant. In case the main server crashes it automatically switches the process to the redundant server with no loss in time and performance; hence bringing in no downtime to the monitoring tool even in the worst case of server crashes. This VMotion configuration can only be done on the 'bare metal' for better performance.

9. References

1. Sarma VAK, Rajendra R, Dheepan P, Kumar KSS. An optimal ant colony algorithm for efficient VM placement. IJST. 2015; 8(s2):156–9.
2. Abid A, Khemakhem MT, Marzouk S, Jemaa MB, Monteil T, Drira K. Toward anti-fragile cloud computing infrastructures. Procedia Computer Science. 2014; 32:850–5.
3. Meng S, Liu L. Enhanced monitoring-as-a-service for effective cloud management. IEEE Trans Comput. 2013; 62(9):1705–20.
4. Patra PK, Singh H, Singh G. Fault tolerance techniques and comparative implementation in cloud computing. International Journal of Computer Applications. 2013; 64(14):37–41.
5. Tchana A, Broto L, Hagimont D, editors. Fault tolerant approaches in cloud computing infrastructures. ICAS 2012, The Eighth International Conference on Autonomic and Autonomous Systems; 2012.
6. Zhao W, Melliar-Smith P, Moser LE, editors. Fault tolerance middleware for cloud computing. 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD); IEEE; 2010.
7. Bala A, Chana I. Fault tolerance-challenges, techniques and implementation in cloud computing. IJCSI. 2012; 9(1):288–93.
8. Aceto G, Botta A, De Donato W, Pescape A. Cloud monitoring: A survey. Computer Networks. 2013; 57(9):2093–115.
9. Andreolini M, Colajanni M, Pietri M, Tosi S. Adaptive, scalable and reliable monitoring of big data on clouds. J Parallel Distr Comput. 2014. In Press.
10. Brintha SR, Nalini C. An efficient cost model for data storage with horizontal layout in the cloud. IJST. 2014; 7(3S):45–6.
11. Pal AS, Pattnaik BPK. Classification of virtualization environment for cloud computing. IJST. 2013; 6(1):127–33.
12. Shetty JP, Kumar HK. Cloud computing: an exploratory study on adoption among SME clusters in Bangalore and Mysore. IJST. 2015; 8(S4):169–75.
13. Smit M, Simmons B, Litoiu M. Distributed, application-level monitoring for heterogeneous clouds using stream processing. FGCS. 2013; 29(8):2103–114.

14. Quarati A, Clematis A, D'Agostino D. Delivering cloud services with QoS requirements: Business opportunities, architectural solutions and energy-saving aspects. FGCS. 2015. In Press.
15. Wang S-S, Wang S-C. The consensus problem with dual failure nodes in a cloud computing environment. Inform Sci. 2014; 279:213–28.
16. Tarmidi M, Abdul Rasid SZ, Alrazi B, Abdul Roni R. Cloud computing awareness and adoption among accounting practitioners in malaysia. Procedia-Social and Behavioural Sciences. 2014; 164:569–74.
17. Jararweh Y, Jarrah M, Kharbutli M, Alshara Z, Alsaleh MN, Al-Ayyoub M. CloudExp: A comprehensive cloud computing experimental framework. Simulat Model Pract Theor. 2014; 49:180–92.