Journal of Theoretical and Applied Information Technology
10th June 2016. Vol.88. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

BUILDING FAULT TOLERANCE WITHIN CLOUDS FOR PROVIDING UNINTERRUPTED SOFTWARE AS SERVICE

[1]S. L. SUSHMITHA, [2] Dr. D. B. K. KAMESH, [3]Dr. J.K. R. SASTRY, [4]V. V. N. SRI RAVALI, [5]Y. SAI KRISHNA REDDY

[1,4,5]Student, Department of Electronics and Computer Engineering, K L University, Vaddeswaram
[2]Assoc. Professor, Department of Electronics and Computer Engineering, K L University, Vaddeswaram
[3]Professor,Department of Electronics and Computer Engineering, K L University, Vaddeswaram

E-mail: [1]sushmitha5264@gmail.com, [2]kameshdbk@gmail.com, [3]drsastry@kluniversity.in, [4]v.sriravali948@gmail.com, [5]syeruva989@gmail.com

ABSTRACT

Use of clouds for availing various IT based services (Software, Platform, Infrastructure platform) has been in rampage. The way IT computing is done has been in radical change. However, many challenges are thrown when one needs to use the clouds for their IT computing. The challenges include security and privacy of the Information stored on the cloud and to provide continued services during the occurrence of the faults within the clouds. This paper addresses architectural framework for implementing fault tolerance at software as service.

Keywords: *Cloud Computing, Virtual Machine(VM), Fault-Tolerance, Check Point, Software-As-A-Service, Redundancy.*

1. INTRODUCTION

With the swift growth in the technological and computational demands as well as high availability of Internet, Computing resources have become inexpensive, powerful and available ubiquitously than ever before. This latest drift has enabled a significant transformation in IT environment to emerge a new computing model known as "Cloud Computing", popularly called "Cloud". The advents in this latest model moved computation and data away from the desktops and portable PCs into massive data centres located elsewhere. It leases resources to the users for general utility purpose, in an on-demand fashion to release when not necessary.

The prime intention of cloud computing is the finer utilization of distributed resources, integrate to achieve exorbitant throughput and resolve large scale computational troubles. In this demanding world, the promising cloud, should support features like, dependability stability, flexible infrastructure, quick provisioning, scalability, reliability, green IT and like [4].

The footing concept of cloud computing was developed way back, opined to organize computation as a public utility. Also the key characteristics of cloud were explored to minimise human and home device efforts. With the massive proliferation of Internet across the world, delivering applications as services has become easier over the Internet. As a result, the overall costs are minimised.

In cloud environment, the conventional roles of providers is classified into: infrastructure providers who maintain the platform and rent resources in accordance with usage-based pricing model, and service providers who host one or many infrastructure providers to rent resources to serve the end users [1]. In cloud, the user's ingress the data, applications, or any other services over the internet through browser irrespective of the device or location of the user. This is possible because of the infrastructure that is usually provided by the third-party organizations. These vendor parties give access and lease their services following the usual delivery models of cloud-private, public, hybrid or community [3]. The Cloud providers deploy their services in accordance with the need of the end user. The vendor provides the computing services in three ways, namely, *Infrastructure-as-a-Service* (IaaS), *Platform-as-a-Service* (PaaS) and Software-*as-a-Service* (SaaS).

In general, the architecture of the cloud environment can be split into layers depending on the resources managed at each layer and the end

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

users [1]. Figure1 shows the general Architecture used for implementation of a Cloud.

The physical resources of the cloud are managed at the *hardware layer* which is typically maintained at the data centres. The data centre consists of several physical servers that are lined in racks and connected with routers, switches and other components. The hardware layer also maintains the power supply and cooling system for all the physical resources present at the data centre. The virtualization layer or *infrastructure layer* partitions the physical resources using virtualization technologies to create a lot of storage or computing resources. This layer is an eminent component of cloud as the important features like dynamic resource allocation are made available using virtualization technologies. The virtualization technologies are further layered upon by the *platform layer* which holds the operating system and the framework for applications. This layer reduces the burden of deploying the applications into the Virtual Machine(VM) containers. The *application layer* stands at the top most level of hierarchy which accommodates all the applications that the end users utilize from the cloud.

Unlike the legacy service hosting environment, the cloud architecture is more adaptive. Each layer is coupled with the higher and lower layers loosely enabling every layer to independently evolve. This architectural feature supports the increasing varied application demands whilst minimising management and maintenance expenses.

Software-as-a-Service or SaaS is made available to end users over internet on pay-as-you-go model. This sets the user free from actually installing and running the applications or software on the native desktop or PDAs. Also it spares the user from the distress of deploying and maintaining the software. The rumpus of managing large data is not necessary with the IaaS and PaaS services already provided by this service model. The software may be shared by several clients, auto-updated from the cloud, and additional licenses needn't be purchased [2]. Any desired features may be rolled out frequently with on-demand request. The service characteristics render SaaS to often be easily interoperable with other mashup apps. Google Apps is one of the popular known SaaS examples. Figure 2 shows architecture that demonstrates the use of software as service.

All the clients operating on different devices like PDA, mobile phones, laptops or desktops may reach out to the cloud for any service with the access of Internet. The client may only be subject to view the user Interface or virtually the cloud of the software as a service. The cloud communicates with the client and middleware which creates a connection between client and server. The Middleware switches the request to a server satisfying its necessary specification requirement. The several servers maintained by the middleware are deployed with the same software to be provided as service. However, each server maybe working on different platform or use database software but the service provided is the same as any other server linked with the middleware.

Though experienced and skilled professionals develop and deploy services into the cloud, there may always be the probability of some issues which are over-looked. Every aspect has its own pros and cons. Similarly, SaaS brings upon a few issues in its behalf too. Following are some of the issues that one will encounter when software has to be used as service:

- *A seamless update* of the software or application may be done in the cloud itself, which may vary the outlook or work so far done by the client in the application. This may create a problem when the client is dependent on a key feature of the application and the upgrade gets rid of that feature.
- *SaaS hosting* is another way of keeping the services available for the client by service vendor instead of service provider. The matter of concern here for the service provider is to make sure that all the services can be accessed by all the users across the globe and the software services provide constant availability.
- *Security* of data is one of the key aspects of any computing or data storage services. In SaaS, the client's data is stored in the large pool of storage which is shared by other client's too.
- The service providers cannot afford allocating separate storage area to each client so they follow *multi-tenancy*, i.e., storing data of different clients at one pool. The clients are unaware of the other users and their intentions so their confidentiality and safety of information may not be impact.
- Clients working on different platforms and servers also try to connect with the cloud for their services but cloud fails to support other

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

formats of data, thus raising *Interoperability Issue* [4].

- With the increasing dependability on the cloud, the number of clients requesting for services is reaching the critical stage and the provider may not be able to handle the *high load*.
- All the client requests may be *addressed by the same software* by the provider. Thus terminating all the requests upon a wrong usage of any one client.
- *Faults* caused by various reasons are one of the major issue of concern and reason for the termination or successful functioning of the software deployed in the cloud.

A mistake in the software programming or defect in the hardware stands responsible for any failure or fault that arises in a system or cloud. Faults are distinguished into hardware and software faults basing on their domain. There are several other factors, like creation phase, occurrence, system failure boundaries, persistence, intent or cause, relying on which classification of faults is done. The scope of this paper is to identify and analyse the causes and effects of *Software Faults*, thus deriving an optimum method for tolerance.

A program running on a system maybe hindered during its execution due to various faults that are beyond the proximity of prediction. A few software packages sustain the feature of recovery system which allows it to roll back the previous session status when a fault occurs. However, a few faults may not be predicted to recover or avoid such faults. Thus, the program fails entirely and terminates its operation abruptly. The reason for few such instances of occurrence of faults include Overflow, Lack of memory, Execution fault, Loss of files, CPU fault, Runtime Errors, Loss of Identification, too much of switching, Insufficient disk storage.

When any of these faults occur, the software will fail, and as a consequence the service to the end user will be disrupted. It is hard to recover from the failure and commence the service to the user from the point from where the service has been disrupted. Therefore, it is absolutely necessary to see to it that the software does not fail through incorporating fault tolerant measures.

Problem Definition

Several clients are dependent on cloud for various applications and software. The occurrence of any software fault may result in the containment of the general functioning of the client works. The clients send their requests to cloud that are directed to the Middleware which communicates between the end user and the service provider. A software fault may result in the termination of a service to the end user due to which releasing all the current work status, session details and uncommitted work done, drawing the client to a great loss. Therefore, there should be a provision of incorporating the fault tolerant measures such that the software will never fail.

2. RELATED WORK

Lakshmi Prasad Saikia et al.[8], studied the existence of Fault-tolerance and how the technologies evolved to imbibe the concept in the increasing requirements. Their survey shows that the concept of fault-tolerance came into existence and developing since about 1970. Several renowned journals, like IEEE transactions on Computer, IEEE transaction on Reliability, and IEEE transactions on cloud computing, have been publishing research or survey papers and other related works. In the period between 1971-1975, enough work was done to come out with microprocessors and computers to build the first fault tolerant computer for processing online transactions.

Sourabh Dave et al., [9] studied various techniques of fault-tolerance, specifically replication and check pointing. Different types of implementing fault tolerance have been studied which effect the functioning of a cloud. They carried out systematic investigation to identify the pros and cons of the fault tolerant techniques and suggested a much consistent fault-tolerant system by combining the replication and check-pointing. The different types of techniques have been perfectly amalgamated to address the overhead and crash of a process communicating with multiple processes.

Cloud Computing offers adaptable results for High Performance Computing applications by furnishing huge amount of virtual machines. Even in the presence of faults, Fault tolerance grants the execution of HPC systems by creating multiple nodes. Generally, check pointing is the widely used fault-tolerance technique for HPC clouds. In 2012, Ifeanyi et al., [12] have designed a new framework which uses Process Level Redundancy(PLR) along

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

with Proactive and Reactive fault tolerance techniques, and Live migration policy. Their basic observations show that the fault is decreased by 40% by their approach.

To reduce the faults that occur on the devices, the faults must be foreseen and managed such that the system works efficiently even during the failures. In the same year (2012), Anju Bala et al.,[11] explained the current fault tolerance techniques based on their behaviour and also proposed a Cloud Virtualized System architecture, which implements autonomic fault tolerance method. Their preliminary outcomes indicate that their suggested system accords different software failures for server programs. To implement their technique, they have used tools such as AZURE, Hadoop, *S*Help. To execute Linux based applications, Amazon EC2 presents a virtual environment.

The familiar technique in Cloud Computing is Virtualization, i.e., creating number of virtual machines with distinct operating systems, running on a particular system. In 2012, L. Arockiam et al., [13] proposed a system, wherein a middle layer is to be introduced between the application layer and virtualization layer to obtain maximum fault-tolerance. The objective of this middle layer is to tolerate node failure and is also user transparent. Fault Tolerance Manager is the middle layer, which is arranged between the two layers. This consists of Fault Detector, Replica Manager, Check Point Manager, Recovery Overseer and Communication Manager. The performance of these components can be improved by considering different algorithms.

In 2013, Ravi Jhawar et al., [10] have introduced a new system-level model to manage the faults that occur in Clouds. In this paper, they have considered a Resource manager that records the details and checks the working condition of the devices which are to be provided to the client. The system they have conferred is a two – stage delivery model, consists of *design stage*, and *run time stage*, to give effective results. Also they have explored a Conceptual Framework, namely Fault Tolerance Manager to embed fault tolerance as a service layer between client's applications and hardware working over the virtual machine manager at VM instances level.

In 2014, Jasbir Kaur et al., [14] carried out their survey on different types of fault-tolerances and their techniques. This paper includes the definition of Cloud computing based on NIST standards, important features, different service models namely SaaS, PaaS, and IaaS and deployment models. They have identified several mechanisms to avoid the faults that can be implemented either before or after their occurrence.

Generally, DBMS systems obtain fault tolerance by Replication. In 1999, Maitrayi Sabaratnam et al., [2] have proposed a system using Replicated Database. The security properties of DBMSs have serious threat, which can be evaluated by the DBMS fault tolerance. Their system comprises of two components, where the first part deals with impact of fault-tolerance in the database, if the variable data stored in database gets corrupted. The second part deals with finding the weak part in the buffer cache and propose the need to look after the components either independently or collectively.

3. CLOUD COMPUTING ARCHITECTURE WITH SOFTWARE AS A SERVICE

The SaaS model also facilitates the user with complete backend services along with the software, through IaaS and PaaS support. The clients make use of all the services by requesting their demands to the cloud which has many servers working for a software service. To address all the requests to the required server, keep track of all the requests and to maintain the details of every session, all the client-server transactions are managed by a primary server which is known as the *Middleware*. The Middleware accepts the requests from the client through a communication medium, virtually believed to be cloud, and assigns the request to the suitable server as well as maintains a log file. A log file holds the details of the client-server pair transaction, instance information and specifications of the server. Particular software is deployed in more than one server with different software specifications or the same one to balance high load or server failures. Each server creates instances for every client request, which enables to create a new environment such that failure of any instance does not affect the other instances and not utilise the complete server. The architecture that shows the software as a service is shown in the Figure 2.

4. FAILURE ANALYSIS OF SOFTWARE AS A SERVICE

Even though the architecture and components for delivering software services are employed

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

wisely, the occurrence of software failures as discussed earlier may tend an instance or server to terminate or fail completely. If the number of client requests exceeds the handling capacity of the software server it may fonder due to overflow, too much switching or CPU fault. The cause of an instance to fall apart immediately maybe execution fault, loss of files or any runtime errors. This drives the server to temporarily stop the execution of a client request until it redirects the request details and session details to any other server which now acts as the alternative server. However, this may not be possible if the backend database server itself is lost as it is the hub for all the sessions and plea details. If the underlying database server itself goes down, then the data may not be saved disabling the feature of diverting the traffic as the transaction details are essential for restarting the instance elsewhere would have been already wiped off. It is essential to address all these problems to enhance the system to be more efficient.

5. AN EFFICIENT FAULT TOLERANCE METHOD FOR ENHANCING THE AVAILABILITY OF SOFTWARE AS A SERVICE

Fault tolerance concept which began to gain its importance since two decades [5] has at least seen its success in imbibing this concept in the database servers[6][7].The probability that the underlying database server will crash is minimal. Nevertheless, with striking technological advancements, the software faults do have seen high inclination. To make available continuous database related services to the user, and also ensure that the software do not lose its essential data, two processes namely ***Mirroring*** and ***Buffer Copying*** have been considered and included into the overall architecture of cloud computing.

Even if a server is corrupted unfortunately, all the client request details will be diverted to other servers which will continue processing the request. Every process or software server maintains a transaction log file which incrementally updates the sequence of changes made by the transaction in the database. The *Redo Log File* maintained keeps the server informed about the changes made by the instance for every time interval [8]. The server creates a buffer for every instance and the instance updates it with the data simultaneously while it accesses it. The architecture framework for *Mirroring* and *Buffering* is shown in the Figure 3.

All client requests for database services for instance are first directed to the Middleware. The middleware maintains a table of details of the client-server transaction pair and the instances created for the client request as shown in Table 1.

The middleware sends the request to the suitable server which creates an instance for the request and send the details to the middleware. The server maintains the buffer in its RAM and, stores the data and redo log file in its database storage. The servers may also be allocated separate spaces in the common database which is connected to each server through a network instead of maintaining their private data storage.

The database resident on one server is made to be mirrored and stored in a different server especially on the server that has a different execution path. ***Mirroring*** leads to 100% data duplication but helps in recovering from the failures, if any, at a faster rate. These fundamental methods helps in connecting the data storage on all the servers such that any changes made in their respective data files or redo log files are immediately duplicated or mirrored into the other server's data space. Whenever an instance fails, the middleware carries out ***Buffer Copying*** i.e., accesses the buffer memory of that instance that failed and copies it to the alternate server to continue processing the request. A new instance is created on a different server for servicing the request of the user, in case of failure of an instance, on some other server and at the same the buffer related to the failed instance is also copied to the buffer created for the new instance. Thus, both the *Mirroring* and *Buffering* helps a new instance taking over from the instance that has failed. The reliability of such as arrangement greatly improves into 4 folds as one can design 4 alternative paths to access an instance created for accessing a particular service. The middleware ensures that the client request is fulfilled even if an instance of a server fails. This method improves the reliability, availability and minimal loss of time of the software as a service level in cloud by making the cloud service fault-tolerant.

6. ENHANCING FAULT TOLERANCE WITHIN CLOUDS WHEN SOFTWARE IS USED AS SERVICE

To make the service provided by the cloud more reliable, the cloud must be able to leverage its services in any kind of failure. The cloud must be

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

able to provide fault-tolerant services even when the complete software crashes down or the database server does not support instance creation. The cloud shall not be led back to provide a software service if the database software crashes down. The cloud creates copies of the software so that the client request may be processed even if the service deployed in a software tool fails. The architecture that considers the replication of service related program is shown in the Figure 4.

Every client request is first received by the middleware and the middleware which allocates the request to be processed by any of the idle copies of the replicated software and maintains the details of the software copies as shown in Table 2. The user request is assigned to one of the available processes for servicing the request. Software packages like Rationale Rose do not support the method of managing the user requests through instance creation.

Software packages like Rationale Rose can handle only a single client request at a time, and the request is serviced through a model file "mdl" that contains data about all the models that the rational rose software supports. The allocation of software copy to a user is also maintained by the Middleware. All the user requests as such are directed to the middle for processing, monitoring and reporting the status of processing to the client.

The Data files (mdl files) that are created for each of the user by the attached copy of the rational rose server are made sharable across several copies of the same software. The sharing is affected to all the copies of the same software that can be made to be resident on multiple servers. Since the data files are sharable, any of the software copy can access the file. If for any reason one of the copies of the software fails, some other copy of the software which is free can be initiated to provide the service using the data file related to the software copy that has filed. The newly selected copy of the software generally is picked from a different server to the extent possible as a matter of strategy. Find new and replace the failed with the new is the strategy followed in this case. The tolerance level of the rational rose software can further be increased by implementing mirroring of the "mdl" file itself. Thus the issue of an "mdl" file failing along with the software copy that handles the data file can also be taken care of by selecting a new copy of the software and the mirrored mdl file so that fault tolerance can be greatly enhanced.

7. COMPARATIVE ANALYSIS

Various authors have recommended different types of fault tolerance models. Table 3 shows the comparison of the methods proposed by several authors. Many of the authors have banked on implementing the fault tolerance using single criteria. Most of the authors utilised system level fault tolerance and as such no model has been presented that deals with fault handling outside the system level fault tolerance. It also seen that most of the software that are needed have not considered within its implementation the issues related to fault occurrence and handling. Database software is one kind that implements the system level fault tolerances. All the methods presented by others have not considered any issue of faults happening due to failure of data resources which are used for running of the software. **Sushmitha et al., model** presented in this paper considers both the system level failures and data level failures and built robust fault tolerance framework to make cloud computing much more versatile**.**

Mirroring and Buffering with process replication has been seen as most ruggedized fault tolerance methods as they lead to very high level of reliability**.**

8. IMPLEMENTING FAULT TOLERANCE ON A SAMPLE CLOUD

The fault tolerance methods proposed in this paper has been applied to university level cloud computing system, the architecture of which is shown in Figure 5. The mean time between the failures of the stream 10 times in 130 days works out to a reliability of 92.3%.

The university architecture has been modified to include the mirroring, buffering and replicating processes. The modified architecture has been shown in the Figure 6. The reliability of the KLU network has been greatly enhanced as we witnessed only 2 faults within 130 days of test trail made with modified KLU network which works out to a reliability of 98.2%. The model presented in this paper clearly improved.

9. CONCLUSIONS

Cloud computing technologies are making radical changes in the way computing is undertaken in support of several levels and sizes of the business establishments. The cost of computing is being

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

reduced quite drastically due to the use of cloud computing technologies. The clouding computing infrastructure while is leading many great economies but still are suffering from different kinds of fault due to which the users are not being given with satisfactory services. Mirroring, buffering and replication of the software addresses most of the fault related issues that happen when software is provided as service.

REFERENCES

[1] Jim Gray, Paul McJones, Mike Blasgen, Bruce Lindsay, Raymond Lorie, Tom Price, Franco Putzolu, Irving Traiger, "*The Recovery Manager of the System R Database Manager*", ACM Computing Survey, Vol. 3, No. 02, pp. 223-242, 1981.

[2] Maitrayi Sabaratnam, ØysteinTorbjørnsen, Svein-Olaf Hvasshovd, "*Evaluating the Effectiveness of Fault Tolerance in Replicated Database Management Systems*", Fault-Tolerant Computing, Twenty-Ninth Annual international symposium(IEEE), 1999.

[3] William Hodak, Sushil Kumar, Ashish Ray, "Oracle Database 11g High Availability",http://www.oracle.com/us/solutions/twp-databaseha-11gr1-134841.pdf

[4] Tharam Dillon, Chen Wu and Elizabeth Chang, "*Cloud Computing: Issues and Challenges*", 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 27-33, 2010

[5] Qi Zhang, Lu Cheng, RaoufBoutaba, "*Cloud Computing: State-of-the-Art and Research Challenges*", pp. 7-18, 2010

[6] YashpalsinhJadeja, Kirit Modi, "*Cloud Computing - Concepts, Architecture and Challenges*", International Conference on Computing, Electronics and Electrical Technologies [ICCEET], pp. 877-880, 2012.

[7] V. M. Sivagami, K. S. Easwara Kumar, "*Survey on Fault Tolerance Techniques in Cloud Computing Environment*", International Journal of Scientific Engineering and Applied Science (IJSEAS), Vol. 1, No. 9,pp.419-425, 2015.

[8] Lakshmi Prasad Saikia and YumnamLanglen Devi, "*Fault Tolerance Techniques and Algorithms in Cloud Computing*", International Journal of Computer Science & Communication Networks, Vol. 4, No. 1, pp. 01-08.

[9] Sourabh Dave and Abhishek Raghuvanshi, "*Fault Tolerance Techniques in Distributed System*", International Journal of Engineering Innovation & Research, Vol. 1, No. 2, pp. 124-130, 2012.

[10] Ravi Jhawar, Vincenzo Piuri, Marco Santambrogio, "*Fault Tolerance Management in Cloud Computing: A System-level Perspective*", IEEE Systems Journal, Vol 1, Iss. 7, pp. 1-7, 2013

[11] Anju Bala and Inderver Chana, "*Fault Tolerance – Challenges, Techniques and Implementation in Cloud Computing*", IJCSI International Journal of Computer science issues, Vol. 9, Iss. 1, pp. 288-293, 2012

[12] Ifeanyi P. Egwutuoha, Shiping Chen, David Levy, Bran Selic, "*A Fault Tolerance Framework for High Performance Computing in Cloud*", 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 709-710, 2012.

[13] L. Arockiam, Geo Francis E, "*FTM- A Middle Layer Architecture for Fault Tolerance in Cloud Computing*", Special Issue of International Journal of Computer Applications (0975 – 8887) on Issues and Challenges in Networking, Intelligence and Computing Technologies – ICNICT 2012, pp. 12-16, 2012.

[14] Jasbir Kaur, SupriyaKinger, "*Analysis of Different Techniques Used For Fault Tolerance*", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5, No. 3, pp.4086-4090,2014.

Journal of Theoretical and Applied Information Technology
10th June 2016. Vol.88. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

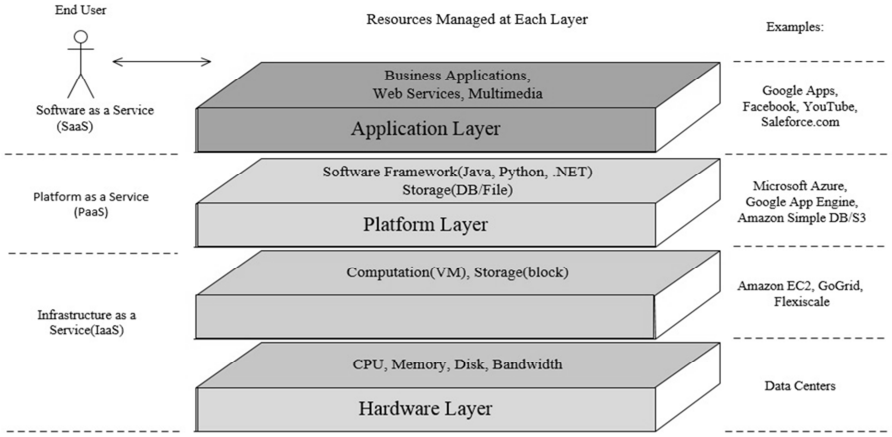

Figure 1: Architecture of Cloud Computing

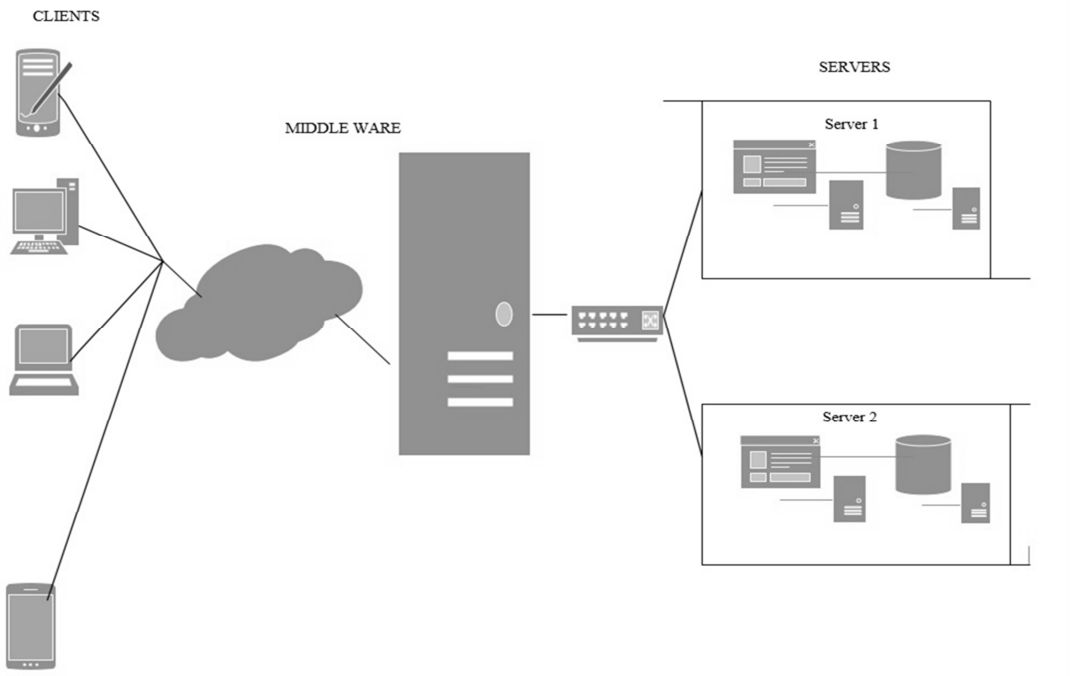

Figure 2: Architecture Of Software As A Service Within A Cloud

Journal of Theoretical and Applied Information Technology
10th June 2016. Vol.88. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

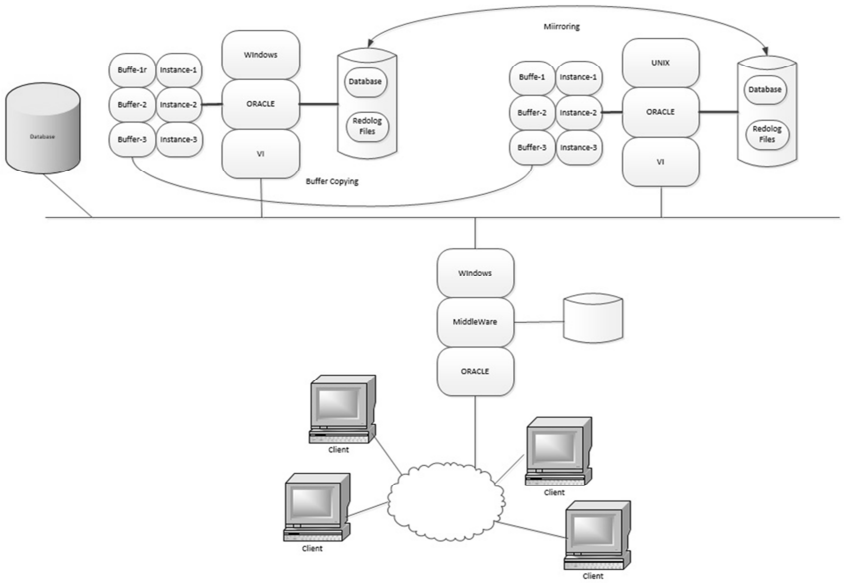

Figure 3: Fault Tolerance Of Software As A Service Using Multiple Instances Through Buffer Copying And Disk Mirroring

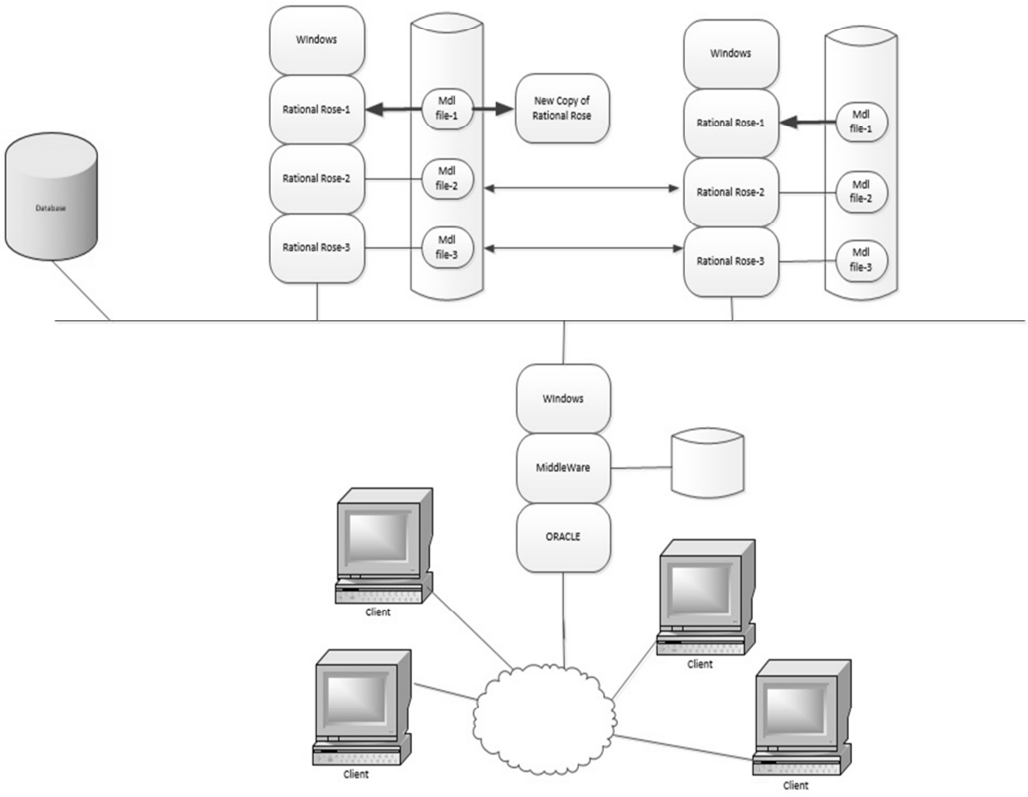

Figure 4: Fault Tolerance Of Software As A Service Using Multiple Copies Through Process Duplication

Journal of Theoretical and Applied Information Technology
10th June 2016. Vol.88. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

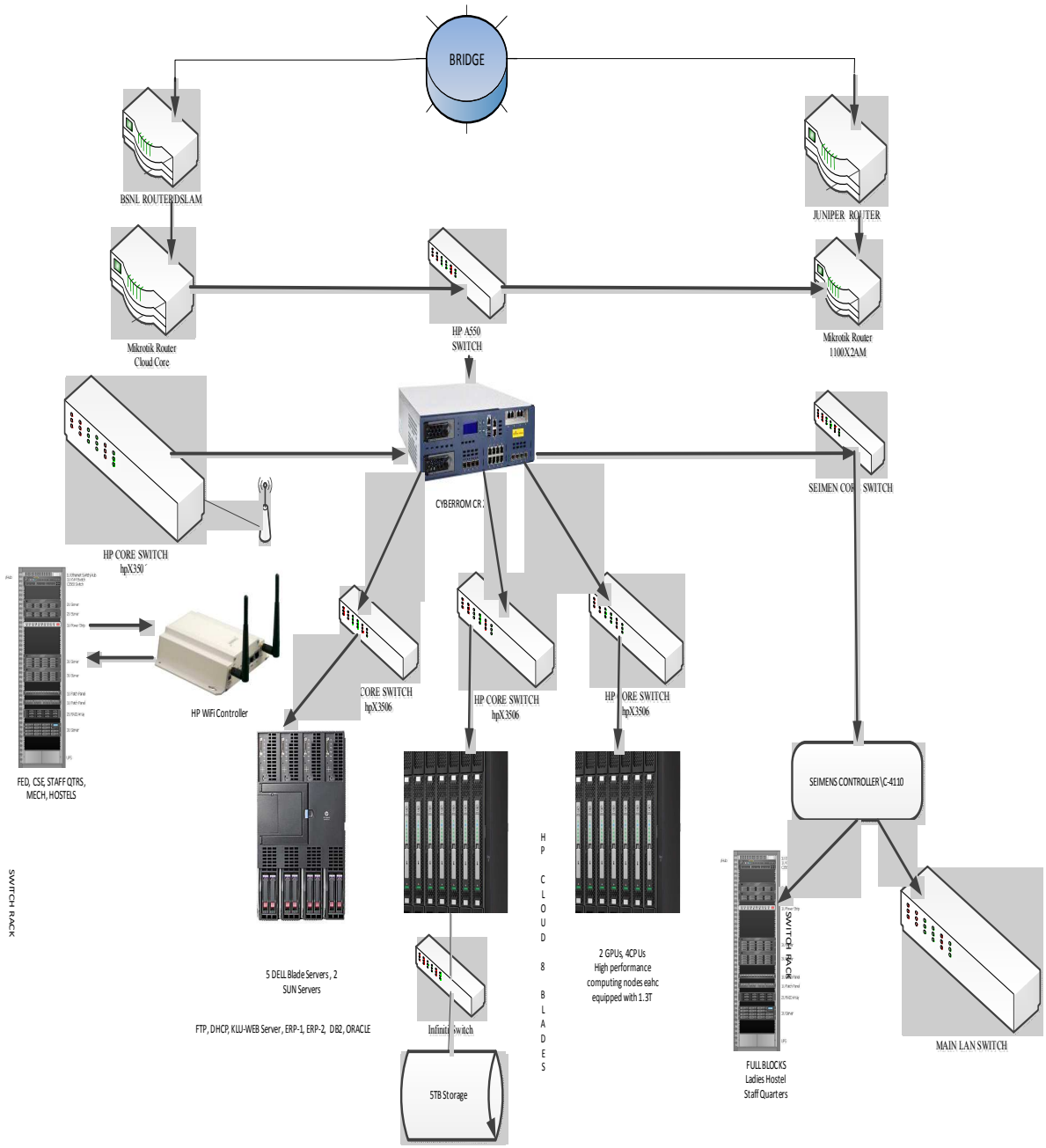

Figure 5: University Cloud Architecture

Journal of Theoretical and Applied Information Technology
10th June 2016. Vol.88. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

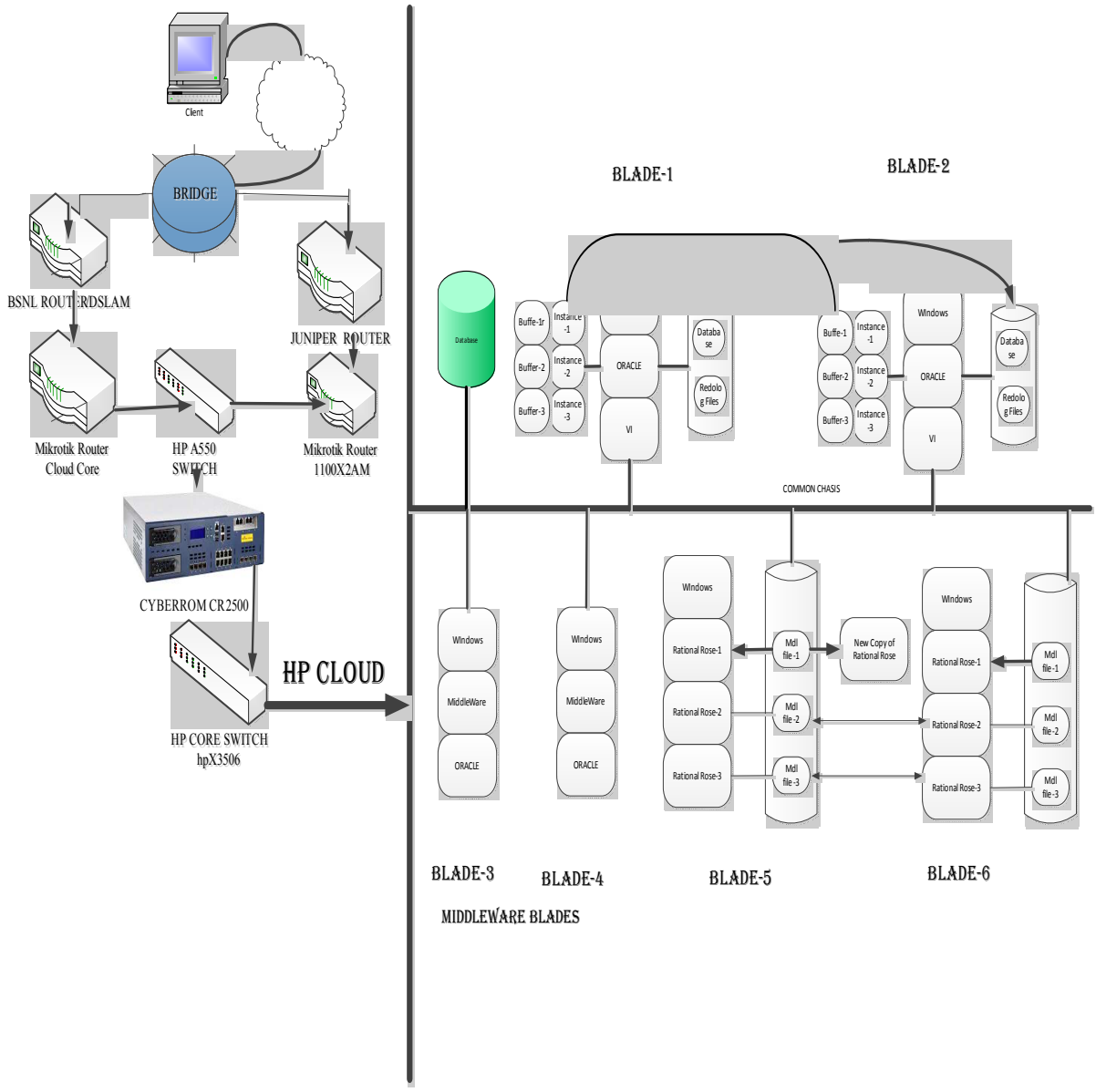

Figure 6: Fault Tolerance included University Cloud Architecture

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Instance Allocation Table

UID	**Server**	**Instance**	**Database Name**
Client _1	Server1	Instance_1	D1
Client_2	Server1	Instance_2	D1
Client_3	Server2	Instance_3	D1
Client_4	Server2	Instance_4	D2

Table 2: Software Allocation Table

UID	Server	Software Allocated
Client1	Server1	Rationale Rose_1
Client2	Server1	Rationale Rose_2
Client3	Server1	Rationale Rose_3
Client4	Server2	Rationale Rose_1

Table 3: Comparison Of Fault Tolerance Models

Parameter	Contributors to Fault Tolerance at Software as service						
	Sushmitha Model	Saurabh Dave	Ifeanyi	Anju Bala	L. Arockiam	Ravi Jhawar	Maitrayi Sabaratnam
Mirroring	√	**X**	**X**	**X**	**X**	**X**	**X**
Buffering	√	**X**	**X**	**X**	**X**	**X**	**X**
File sharing	√	**X**	**X**	**X**	**X**	**X**	**X**
Check pointing	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Multiple Process Communication	√	√	**X**	**X**	**X**	**X**	**X**
Process level redundancy	√	**X**	√	**X**	**X**	**X**	**X**
Virtualisation	√	**X**	**X**	√	**X**	**X**	**X**
Middleware service	√	**X**	**X**	**X**	√	**X**	**X**
Fault tolerance Manager	**X**	**X**	**X**	**X**	**X**	√	**X**
System level fault	**X**	**X**	**X**	**X**	**X**	**X**	√