Research Journal of Applied Sciences, Engineering and Technology 11(12): 1365-1377, 2015
DOI: 10.19026/rjaset.11. 2244
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: June 8, 2015 Accepted: September 5, 2015 Published: December 25, 2015

Research Article

A Survey of Fault-tolerance in Cloud Computing: Concepts and Practice

Ameen Alkasem and Hongwei Liu
School of Computer Science and Technology, Harbin Institute of Technology, China

Abstract: A fault tolerance is an important property in order to achieve performance levels for important attributes for a system's dependability, reliability, availability and Quality of Service (QoS). In this survey a comprehensive review of representative works on fault tolerance in cloud computing is presented, in which general readers will be provided an overview of the concepts and practices of a fault-tolerance computing. Cloud computing service providers will rise and fall based on their ability to execute and deliver a satisfactory QoS in primary areas such as dependability. Many enterprise users are wary of the public clouds' dependability limitations, but also curious about the possibility of adopting the technologies, designs and best practices of clouds for their own data centers such as private clouds. The situation is evolving rapidly with public, private and hybrid clouds, as vendors and users are struggling to keep up with new developments.

Keywords: Availability, cloud computing, fault tolerance, MTBF, redundancy

INTRODUCTION

Cloud computing is one of today's most exciting technologies because of its capacity to lessen costs associated with computing while increasing flexibility and scalability for computer processes. During the past few years, cloud computing has grown from being a promising business concept to one of the fastest growing sectors of the IT industry. On the other hand, IT organizations have expressed concerns about critical issues such as dependability that accompany the widespread implementation of cloud computing. Dependability in particular is one of the most debated issues in the field of cloud computing and several enterprises look warily at cloud computing due to projected dependability risks. Moreover also, there are three important attributes of reliability, availability and QoS of the cloud, as show in Fig. 1. Although each of those issues are associated with usage of the cloud, they will have different degrees of importance. The careful examination of the benefits and risks of cloud computing is necessary of the viability of cloud computing (Sabahi, 2011). The US NIST (National Institute of Standards and Technology) defines the concept of Cloud computing as follows (Mell and Grance, 2011):

- Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." This definition can be represented as shown in Fig. 2.

Fault tolerant systems are a popular research area. In recent years for grid computing and disturbed system technologies are widely used in many research and different applications in dependability. Especially for fault tolerance and a monitoring systems. Naixue *et al*. (2009) authors gave a survey on fault tolerant issue in distributed systems. Jin *et al*. (2003) authors only considered concentrate on fault-tolerant strategies in computational grid. Xiong *et al*. (2009) considered comparing all kinds of adaptive fault detection FD schemes in different experimental environments. This study presented a comprehensive survey on fault tolerance in cloud computing, which will provide general readers an overview of concepts and practice of a fault-tolerance computing.

MATARIALS AND METHODOLOGY

Concepts of fault tolerance: Fault-tolerant computing is a generic term describing redundant design techniques with duplicate components or repeated computations enabling uninterrupted (tolerant) operation in response to component failure (faults).

There are many applications in which the reliability of the overall system must be far higher than that of the reliability of its individual components. In such cases, designers devise mechanisms and architectures that allow the system to either completely mask the effects of a component failure or recover from it quickly

Corresponding Author: Ameen Alkasem, School of Computer Science and Technology, Harbin Institute of Technology, China

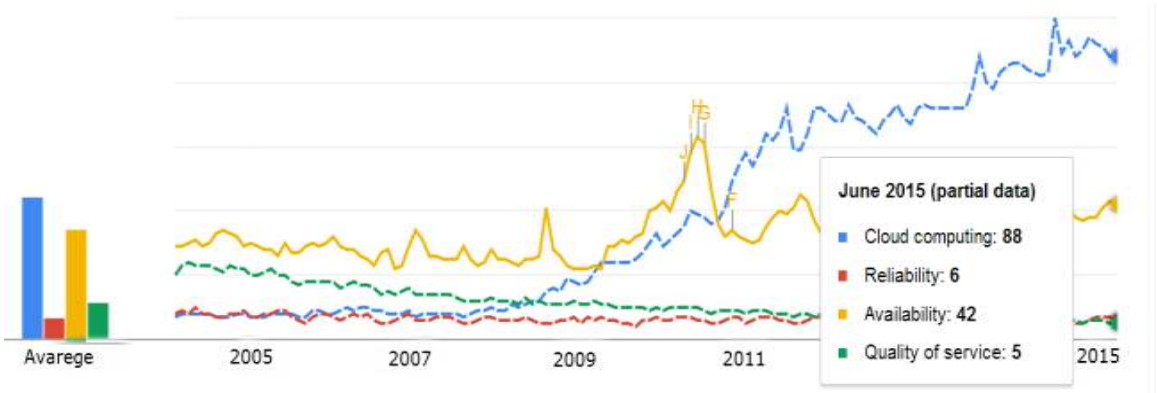

Fig. 1: Cloud computing and dependability attributes (Google trends)

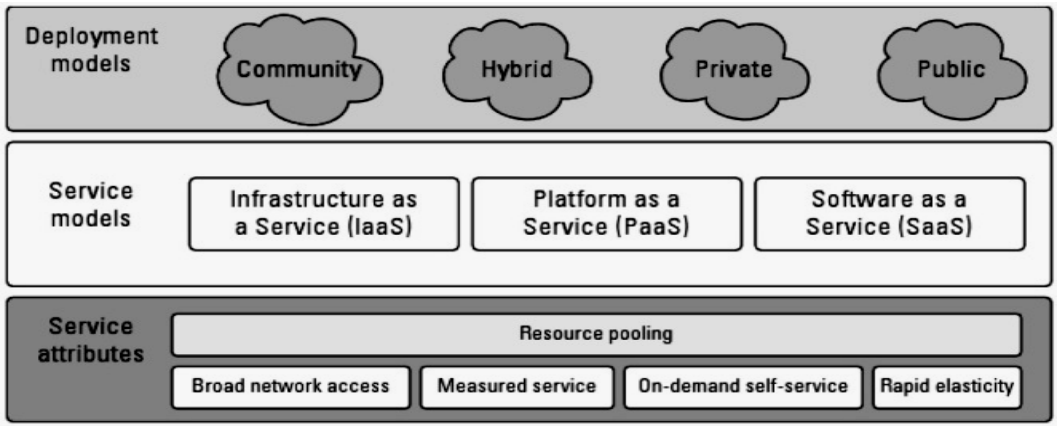

Fig. 2: Cloud computing system

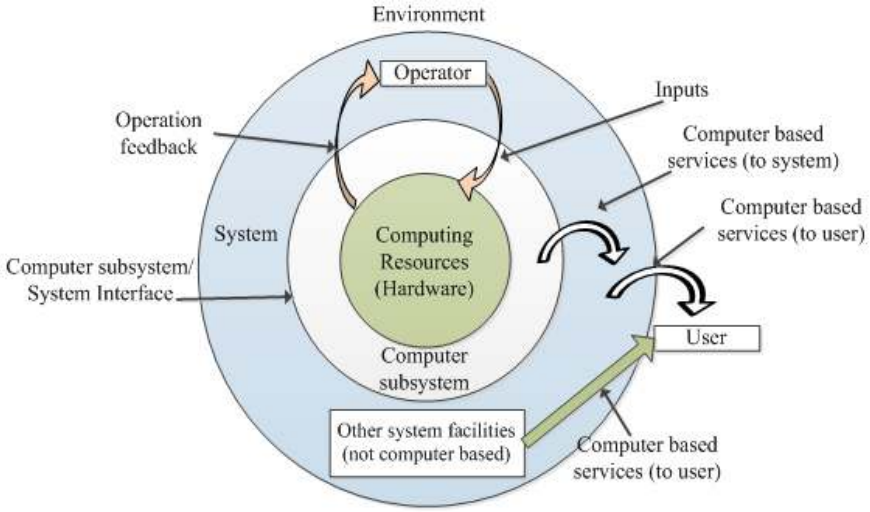

Fig. 3: System relationship

enough so that the application is not seriously affected (Koren and Krishna, 2010).

Dependability of the system: In the field of software engineering, a system is often equated with software, or perhaps with the combination of computer hardware and software. Here, we use the term system in a broader sense. As shown in Fig. 3, a system is the entire set of components, both computer related and non-computer related, that provides a certain service to a user. There are two levels at which fault tolerance can be applied:

Hardware fault tolerance: Measures in hardware fault tolerance include:

- Redundant communications
- Replicated processors
- Additional memory
- Redundant power/energy supplies

Software fault tolerance: Changes in program or data structures due to transients or design errors are examples of software fault tolerance. Mechanisms such as checkpoint/restart, recovery blocks and multiple-version programs are often used at this level (Qureshi *et al*., 2004).

A system is considered dependable if it has a high probability of successfully carrying out its specific functions. This first presumes that the system is available. Furthermore, in order to completely perform a specific function of the system, it is necessary to define all the environmental and operative requirements for the system to provide the desired service. Dependability is therefore a measurement of how much faith there is in the service given by the system (Lazzaroni, 2011).

The design and implementation of "dependable" systems necessitates the appropriate methodology for identifying possible causes of malfunctions, commonly known as "impediments." The technology to eliminate or at least limit the effects of such causes is also necessary. Consequently, in order to deal with the problem of dependability, we need to know what impediments may arise and the technologies to avoid the consequences. Systems that utilize such techniques are called Faults Tolerant (Lazzaroni *et al*., 2011). Impediments to dependability assume three aspects: fault, error and failure. A system is in failure when it does not perform its specific function. A failure is therefore a transition from a state of correct service to a state of incorrect operation service. The periods of time when a system is not performing any sort of service at all are called outage periods. Inversely, the transition from a period of non-service to a state of correct functioning is deemed to be the restoration of service. As shown in Fig. 4. Possible system failures can be subdivided into classes of severity in respect to the possible consequences of system failure and its effect on the external environment. A general classification used in which to separate failures into two categories: benign and catastrophic/malicious (Lazzaroni, 2011).

Constructing a dependable system includes the prevention of failures. To attain this, it is necessary to understand the processes which may lead to a failure, originating from a cause (failure) that may be inside or outside the system. The failure may even remain dormant for a period of time until its activation. The activation of a failure leads to an error that is part of the state of a system that can cause a successive failure.

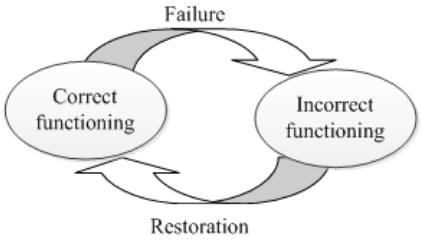

Fig. 4: State of system

Fig. 5: Fault-error-failure chain

The failure is therefore the effect, externally observable, of an error in the system. Errors are said to be in a latent stat until they become observable and/or lead to a state of failure, as shown in Fig. 5.

Similar failures can correspond to many different errors, just as the same error can cause different failures (Birolini, 2007). Systems are collections of interdependent components (elements, entities) which interact among themselves in accordance with predefined specifications. The fault-error-failure chain presented in Fig. 5 can therefore be utilized to describe both the failure of a system and the failure of a single component. One fault can lead to successive faults, just as an error, through its propagation and thus causing further errors. A system failure is often observed at the end of a chain of propagated errors.

Dependability attributes: The attributes of dependability express properties which are expected from a system. Three primary attributes are:

- Reliability
- Availability
- Safety

Other possible attributes include:

- Maintainability
- Testability
- Performability
- Security

Depending on the application, one or more of these attributes are needed to appropriately evaluate the system behavior. For example, in an Automatic Teller Machine (ATM), the duration of time in which system

Table 1: Availability and the corresponding downtime per year

Availability	Downtime
90%	36.5 days/years
99%	3.65 days/years
99.9%	8.76 h/years
99.99%	52 min/years
99.999%	5 min/years
99.9999%	31 sec/years

is able to deliver its intended level of service (system availability) is an important measure. However, for a cardiac patient with a pacemaker, continuous functioning of the device is a matter of life and death. Thus, the ability of the system to deliver its service without interruption (system reliability) is crucial. In a nuclear power plant control system, the ability of the system to perform its functions correctly or to discontinue its function in a safe manner (system safety) is of greater importance (Dubrova, 2013).

Dependability impairment: Dependability impairment is usually defined in terms of faults, errors and failures. A common feature of the three terms is that they give us a message that something has gone wrong. These faults, errors and failures can be differentiated by where they have occurred. In the case of a fault, the problem occurred on the physical level; in the case of an error, the problem occurred on the computational level; and in the case of a failure, the problem occurred on a system level (Pradhan, 1996).

Reliability vs. availability: Reliability R(t) of a system at time t is the probability that the system operates without failure in the interval [0,t] given that the system was performing correctly at time 0. Availability expresses the fraction of time a system is operational. A 0.999999 availability means that the system is at most not operational for one hour over a million hour periods. Availability A (t) of a system at time t is the probability that the system is functioning correctly at the instant of time t.

Steady-state availability: Steady-state availability is often specified in terms of downtime per year. Table 1 shows the values for availability and the corresponding downtime. Availability is typically used as a measurement for systems where short interruptions can be tolerated. Networked systems, such as telephone switching and web servers fall into this category (Dubrova, 2013).

Availability is not equal to reliability: Availability gives information about how time is used, where reliability gives information about the failure-free interval. Both are described in % values. Availability is not equal to reliability except in a theoretical world of no downtime and no failures. Availability, in the simplest form (El-Damcese and Temraz, 2015), is:

A = Uptime/(Uptime+Downtime)
Ai = MTBF/(MTBF+MTTR)

Fault-tolerance vs. high availability: Fault tolerance relies on specialized hardware to detect a hardware fault and instantaneously switch to a redundant hardware component whether the failed component is a processor, memory board, power supply, I/O subsystem, or storage subsystem. The fault tolerant model does not address software failures, which are by far the most common reason for downtime. High availability views availability not as a series of replicated physical components, but rather as a set of system-wide, shared resources that cooperate to guarantee essential services. High availability combines software with industry-standard hardware to minimize downtime by quickly restoring essential services when a system, component, or application fails. While not instantaneous, services are restored rapidly, often in less than a minute. The difference between fault tolerance and high availability is that a fault tolerant environment has no service interruption yes has a significantly higher cost, while a highly available environment has minimal service interruption (Rohit, 2014).

Faults, errors and failures: As shown in Fig. 6, a fault is a physical defect, imperfection, or flaw that occurs in some hardware or software component (Belli and Görke, 2012). Examples are short-circuiting between two adjacent interconnects: Broken pin or a software bug. An error is a deviation from correctness or accuracy in computation, which occurs as a result of a fault. Errors are usually associated with incorrect values in the system state. For example, a circuit or a program computed an incorrect value; or incorrect information was received while transmitting data. A failure is the non-performance of some action which is due or expected. A system is said to have a failure if the service it delivers to the user deviates from compliance with the system specification for a specified period of time. A system may fail either because it does not act in accordance with the specification, or because the specification did not adequately describe its function. Not every fault causes an error and not every error causes a failure. This is particularly evident in the case of software. Some program bugs are very hard to find because they cause failures only in very specific situations. For example, in November 1985, a $32 billion overdraft was experienced by the Bank of New York, leading to a loss of 5$ million in interest. The failure was caused by an unchecked overflow of a 16-bit counter. In 1994, the Intel Pentium I microprocessor was discovered to compute incorrect answers to certain floating-point division calculations.

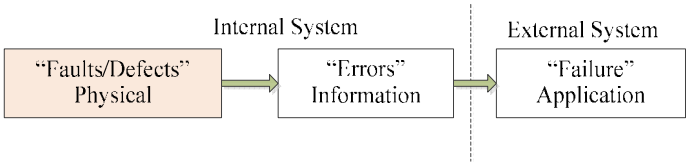

Fig. 6: Faults, errors and failures

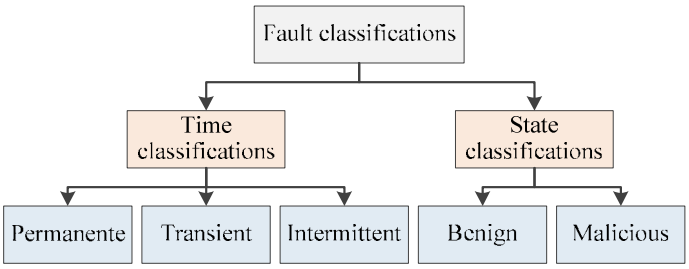

Fig. 7: Fault classifications diagram

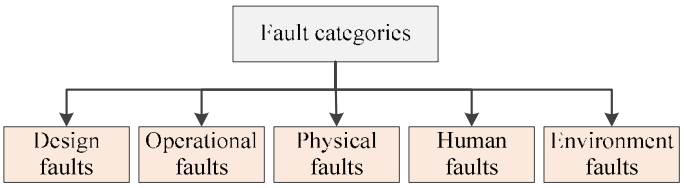

Fig. 8: Fault categories

Practice for fault-tolerance: Fault tolerance is the ability of a system to correctly perform its function even in the presence of internal faults. The purpose of fault tolerance is to increase the dependability of a system. A complementary but separate approach to increasing dependability is fault prevention. This consists of techniques, such as inspection, which has the intent to eliminate the circumstances by which the faults have arised (Saha, 2003).

Fault classifications: Based on duration, faults can be separated into two classifications which are timing and state, as shown in Fig. 7:

- **Permanent fault:** Remains in the system until they are repaired. For example, a broken wire or a software design error.
- **Transient fault:** Starts at a particular time, remains in the system for some period and then disappears, For example, hardware components which have an adverse reaction to radioactivity. Also, many faults in communication systems are transient.
- **Intermittent fault:** Transient faults that occur from time to time. For example, a hardware component that is heat sensitive meaning it works for a time, stops working, cools down and then starts to work again.
- **Benign fault:** A fault that just causes a unit to go dead.
- **Malicious fault:** The component makes a malicious act and sends different valued outputs to different receivers.

A different way to classify faults is by their underlying cause, as shown in Fig. 8.

- **Design faults:** The result of design failures, like coding. While it may appear that in a carefully designed system all such faults should be eliminated through fault prevention, in practice this is usually not realistic. For this reason, many fault-tolerant systems are built with the assumption that design faults are inevitable and theta mechanisms need to be put in place to protect the system against them.
- **Operational faults:** Faults that occur during the lifetime of the system.
- **Physical faults:** Processor failures or disk crashes (McKelvin Jr., 2011).
- **Human faults (Errors):** An inappropriate or undesirable human decision or behavior that reduces, or has the potential for reducing, effectiveness, safety, or system performance.

Finally, based on how a failed component behaves once it has failed, faults can be classified into the following categories, as shown in Fig. 9.

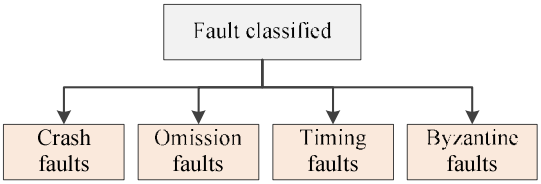

Fig. 9: Fault classified

- **Crash faults:** The component either completely stops operating or never returns to a valid state. For example, a server halts but was working ok until the O.S. failure.
- **Omission faults:** The component completely fails to perform its service. For example, a server not listening or buffer overflow.
- **Timing faults:** The component does not complete its service on time. For example, a server response time is outside its specification and a client may give up. Response: Incorrect response or incorrect processing due to control flow out of synchronization.

Byzantine faults: These are faults of an arbitrary nature. For example, server behaving erratically, like providing arbitrary responses at arbitrary times. Server output is inappropriate but it is not easy to determine this to be incorrect. Duplicated message due to buffering problem is an example. Alternatively, there may be a malicious element involved (UK Essays, 2013).

Fault-tolerant systems:
Definitions:

- Ideally the system is capable of executing their tasks correctly regardless of either hardware failures or software errors.
- A system fails if it behaves in a way which is not consistent with its specification. Such a failure is a result of a fault in a system component.

What is the meaning of correct functionality in the presence of faults?

The answer depends on the particular application (on the specification of the system):

- The system stops and doesn't produce any erroneous (dangerous) result/behavior
- The system stops and restarts after a given time without loss of information
- The system keeps functioning without any interruption and (possibly) with unchanged performance (Latchoumy and Khader, 2011).

Redundancy: Redundancy is at the heart of fault tolerance. Redundancy is the incorporation of extra components in the design of a system so that its function is not impaired in the event of a failure. All fault-tolerant techniques rely on extra elements introduced into the system to detect and recover from fault components and are redundant as they are not required in a perfect system. They are often called protective redundancy.

The aim of redundancy: Minimize redundancy while maximizing reliability, which are subject to the cost and size constraints of the system.

The warning of redundancy: The added components inevitably increase the complexity of the overall system. Thus it can lead to less reliable systems. Therefore, it advisable to separate out the fault-tolerant components from the rest of the system.

Types of redundancy: The types of redundancy are shown in Fig. 10.

Hardware redundancy: Hardware redundancy is a fundamental technique to provide fault-tolerance in safety-critical distributed systems (Gray and Siewiorek, 1991):

- Aerospace applications
- Automotive applications
- Medical equipment
- Some parts of telecommunications equipment
- Nuclear centers

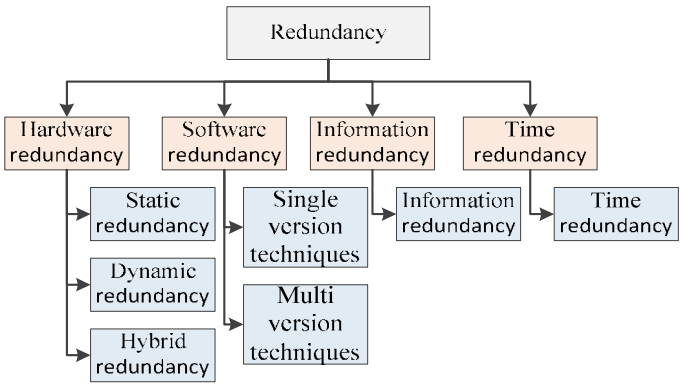

Fig. 10: Types of redundancy

- Military equipment, etc

Static redundancy: redundant components are used inside a system to hide the effects of faults. For example Triple Modular Redundancy TMR-3 identical subcomponents and majority voting circuits. Outputs are compared and if one differs from the other two, that output is masked out and assumes that the fault is not common (such as a design error), rather it transient or due to component deterioration. To mask faults from more than one component requires NMR.

Dynamic redundancy: Redundancy supplied inside a component which indicates that the output is in error. It provides an error detection facility and recovery must be provided by another component.

Hybrid redundancy: A combination of static and dynamic redundancy techniques.

Software redundancy: Software redundancy can be divided into two groups:

Single-version techniques: Single version techniques add a number of functional capabilities to a single software module that are unnecessary in a fault-free environment. Software structure and actions are modified to be able to detect a fault, isolate it and prevent the propagation of its effect throughout the system. Here, we consider how fault detection, fault containment and fault recovery are achieved in a software domain:

- Fault detection techniques
- Fault containment techniques
- Fault recovery techniques
- Exception handling
- Checkpoint and restart

Multi-version techniques: Multi-version techniques use two or more versions of the same software module, which satisfy the design diversity requirements. For example, different teams, different coding languages or different algorithms can be used to maximize the probability that all the versions do not have common faults:

- Recovery blocks
- N-version programming
- N self-checking programming
- Design diversity

Information redundancy: Data are coded in such a way that a certain number of bit errors can be detected and, possibly, corrected (parity coding, checksum codes and cyclic codes).

Time redundancy: The timing of the system is such that if certain tasks have to be rerun and recovery operations have to be performed system requirements are still fulfilled (Koren and Krishna, 2010).

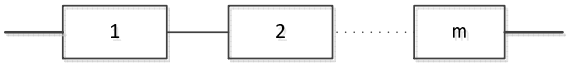

Fig. 11: Block diagram of an m-unit series system

Reliability evaluation of standard configurations: As engineering systems can form various types of configurations in performing reliability analysis, this section presents reliability analysis of some standard networks or configurations (Dhillon, 2007):

Series configuration: A series system is defined as a set of N modules connected together so that the failure of any one module causes the entire system to fail. As shown in Fig. 11, the reliability of the entire series system is the product of the reliabilities of its N modules. Denoted by $R_i(t)$, the reliability of the module is i and $R_s(t)$ the reliability of the whole system:

$$R_s(t) = \prod_{i=1}^{N} R_i(t) \quad (1)$$

where,
R_s : The series system reliability.
N : The total number of units in series.
R_i : The unit *i* reliability; for i = 1, 2, ……, m.

If module i has a constant failure rate, denoted by λ_i, then, according to Eq. (1), $R_i(t) = e^{-\lambda_i t}$ and consequently:

$$R_s(t) = e^{-\int_0^1 \lambda_i dt} = e^{-\lambda_s t} \quad (2)$$

where,
$R_s(t)$: The reliability of unit *i* at time *t*.
$\lambda_s(t)$: Unit *i* hazard rate.
λ_i : Unit *i* constant failure rate.

By substituting Eq. (2) into Eq. (1), we get:

$$R_s(t) = \prod_{i=1}^{m} e^{-\lambda_i t} = e^{-\sum_{i=1}^{m} \lambda_i t} \quad (3)$$

where,
$R_s(t)$ = The series system reliability at time *t*.

Using Eq. (3) in Eq. (4) yields:

$$MTTF_s = \int_0^{\infty} e^{-\sum_{i=1}^{m} \lambda_i t} dt = \frac{1}{\sum_{i=1}^{m} \lambda_i} \quad (4)$$

where,
$MTTF_s$ = The series system mean time to failure.

Parallel configuration: A parallel system is defined as a set of N modules connected together so that it requires the failure of all the modules for the system to fail. The system block diagram is shown in Fig. 12. Each block in the diagram represents a unit.

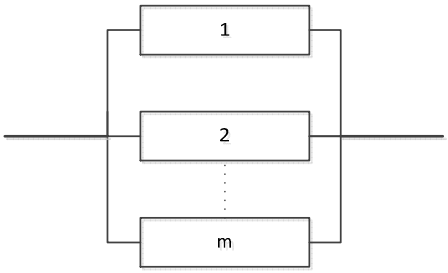

Fig. 12: A parallel systems with m units

The following expression for the reliability of a parallel system is denoted by $R_p(t)$:

$$R_p(t) = 1 - \prod_{i=1}^{N}(1 - R_i(t)) \quad (5)$$

where,
R_p : The parallel system reliability.
N : The total number of units in parallel.
R_i : The unit i reliability; for $i = 1, 2, \ldots\ldots, m$.

If module i has a constant failure rate λ_i, then in Eq. (6), we get:

$$R_p(t) = 1 - \prod_{i=1}^{N}(1 - e^{-\lambda_i t}) \quad (6)$$

As an example, the reliability of a parallel system consisting of two modules with constant failure rates λ_1 and λ_2 is given by:

$$R_p(t) = e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-(\lambda_1 t + \lambda_2)t} \quad (7)$$

where,
$R_p(t)$: The parallel system reliability at time t.

Standby system: In the case of using a standby system, only one unit operates and m units are kept in their standby mode. As soon as the operating unit fails, a switching mechanism detects the failure and turns on one of the standbys. The system contains a total of (m-+1) units and it fails when all the m standby units fail. For a perfect switching mechanism and standby units, independent and identical units, the unit's constant failure rates and the standby system reliability is given by:

$$R_{std}(t) = \sum_{i=0}^{m} \frac{(\lambda t)^i e^{-\lambda t}}{i!} \quad (8)$$

where,
$R_{std}(t)$: The standby system reliability at time t.
m : The total number of standby units.
λ : The unit constant failure rate.

Using Eq. (8) in (9) yields:

$$MTTF_{std} = \int_0^{\infty} \left[\frac{(\lambda t)^i e^{-\lambda t}}{i!}\right] dt = \frac{m-1}{\lambda} \quad (9)$$

where,
$MTTF_{std}$ = The standby system mean time to failure.

Numerical example: A system has two independent and identical units. One of these units is operating and the other is on standby. Calculate the system mean time to failure and reliability for a 200-h mission by using Eq. (8) and (9), if the unit failure rate is 0.0001 failures per hour.

Solution: By substituting the given data values into Eq. (8), we get:

$$R_{std}(200) = \sum_{i=0}^{1} \frac{[(0.0001)(200)]^i e^{-(0.0001)(200)}}{i!} = 0.9998$$

Similarly, substituting the given data values into Eq. (9) yields:

$$MTTF_{std} = \frac{(1+1)}{(0.0001)} = 20{,}000 \text{ hours}$$

Thus, the system reliability and mean time to failure are 0.9998 and 20,000 h, respectively.

M-of-N systems: An M-of-N system is a system that consists of N modules and needs at least M of them for proper operation. Thus, the system fails when fewer than M modules are functional. The best-known example of this type of systems is the triplex, as shown in Fig. 13. It consists of three identical modules whose outputs are voted on. This is a 2-of-3 system: So long as a majority (2 or 3) of the modules produces correct results, the system will be functional (Koren and Krishna, 2010).
The system reliability is therefore given by:

$$R_{M_{of_N}}(t) = \sum_{i=M}^{N} \binom{N}{i} R^i [1 - R(t)]^{N-i} \quad (10)$$

where,

$$\binom{N}{i} = \frac{N!}{(N-i)!i!}$$

The assumption that failures are independent is a key to the high reliability of M-of-N systems. Even a slight extent of positively correlated failures can greatly diminish their reliability. For example, suppose q_{cor} is the probability that the entire system suffers a common failure. The reliability of the system now becomes that of a single module (voter failure rate is considered negligible) to the general case of TMR. This is called N-Modular Redundancy (NMR) and is an M-of-N cluster with N being odd and M = [N/2]:

$$R^{cor}_{M_{of_N}}(t) = (1 - q_{cor}) \sum_{i=M}^{N} \binom{N}{i} R^i [1 - R(t)]^{N-i} \quad (11)$$

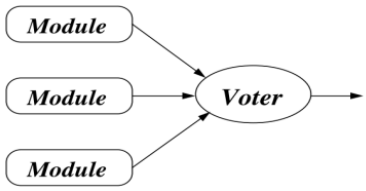

Fig. 13: A Triple Modular Redundant (TMR) structure

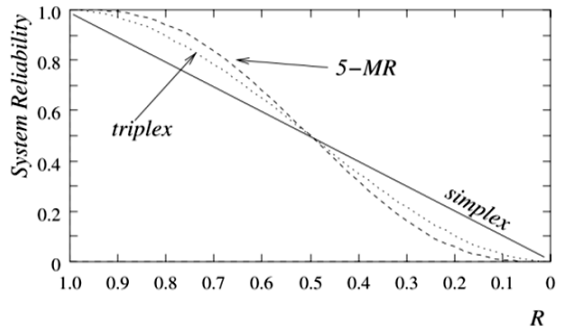

Fig. 14: Comparing NMR reliability (N = 3 and 5)

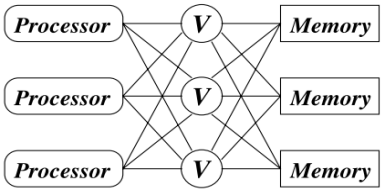

Fig. 15: Triplicated voters in aprocessor/memoery TMR

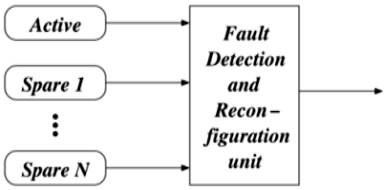

Fig. 16: Dynamic redundancy

A plot of the reliability of a simplex (a single module), a triplex (TMR) and an NMR cluster with N = 5 is shown in Fig. 14. For high values of R(t), the greater the redundancy, the higher the system reliability (Koren and Krishna, 2010). As R(t) decreases, the advantages of redundancy become less marked. When R(t) <0.5, redundancy actually becomes a disadvantage, with the simplex being more reliable than either of the redundant arrangements. This is also reflected in the value of $MTTF_{TMR}$, which (for R voter (t) = 1 and $R(t) = e^{-\lambda t}$) can be calculated by the following equation:

$$MTTF_{TMR} = \int_0^\infty \left(3R^2 - 2R^3(t)\right) dt = \int_0^\infty \left(3e^{-2\lambda t} - 2e^{-2\lambda t}\right) dt = \frac{5}{6\lambda} < \frac{1}{\lambda} = MTTF_{saimple} \quad (12)$$

Voting techniques: a voter receives inputs x_1, x_2, ..., x_N from an M-of-N cluster and generates a representative output. The simplest voter is one that does a bit-by-bit comparison of the outputs and checks if a majority of the N inputs are identical variations on the N-Modular Redundancy Unit-Level Modular Redundancy, as shown in Fig. 15.

Dynamic redundancy: as shown in Fig. 16, the reliability is given by:

$$R_{dynamic}(t) = R_{dru}(t)\lceil(1 - R(t)\rceil^{N+1} \quad (13)$$

where, R(t) is the reliability of each module and $R_{dru}(t)$ is the reliability of the detection and reconfiguration unit. Failures to the active module occur at rate of λ. The probability that a given such failure which cannot be recovered from is 1-c. Hence, the rate at which unrecoverable failures occur is (1-c)λ (Koren and Krishna, 2010). The probability that no unrecoverable failure occurs to the active processor over a duration *t* is therefore given by $e^{-(1-c)\lambda t}$ and the reliability of the reconfiguration unit is given by $R_{dru}(t)$. Therefore the equation is expressed as:

$$R_{dynamic}(t) = R_{dru}(t)e^{-(1-c)\lambda t} \quad (14)$$

Hybrid redundancy: an NMR system is capable of masking permanent and intermittent failures, but as we have seen, its reliability drops below that of a single module for very long mission times if no repair or replacements are conducted. Figure 17 depicts a hybrid system consisting of a core of N processors constituting an NMR and a set of K spares (Koren and Krishna, 2010).

The reliability of a hybrid system with a TMR core and K spares is:

$$R_{hybrid}(t) = R_{voter}(t)R_{rec}(t)(1 - mR(t)\lceil 1 - Rtm \quad (15)$$

where, m = K+3 is the total number of modules, $R_{voter}(t)$ and $R_{rec}(t)$ are the reliability of the voter and the comparison and reconfiguration circuitry, respectively (Koren and Krishna, 2010).

Sift-out modular redundancy: As in a NMR, all N modules in the Sift-out Modular Redundancy scheme are active and the system is operational as long as there are at least two fault-free modules, as shown in Fig. 18.

Duplex systems: A duplex system is the simplest example of module redundancy. Figure 19 shows an example of a duplex system consisting of two processors and a comparator. Both processors execute the same task and if the comparator finds that their outputs are in agreement, the result is assumed to be correct.

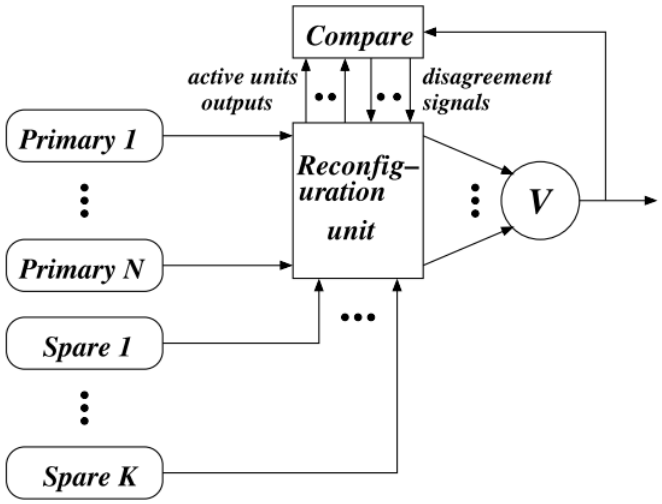

Fig. 17: Hybrid redundancy

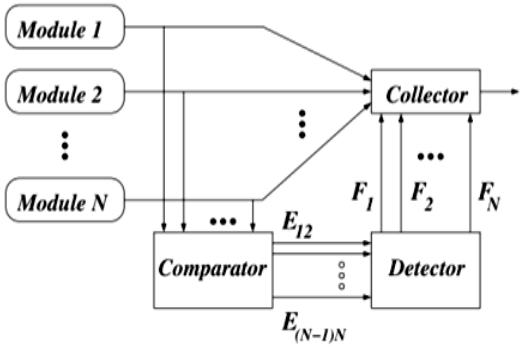

Fig. 18: Sift-out structure

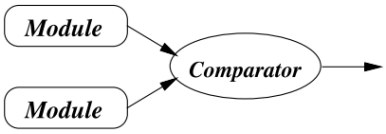

Fig. 19: Duplex system

Assuming that the two processors are identical, each with a reliability R(t), the reliability of the duplex system is:

$$R_{duplex}(t) = R_{comp}(t)\begin{pmatrix} R^2(t) + 2cR(t) \\ [1 - R(t)] \end{pmatrix} \quad (16)$$

where, R_{comp} is the reliability of the comparator. Assuming that there is a fixed failure rate of λ for each processor and an ideal comparator ($R_{comp}(t) = 1$), the MTTF of the duplex system is:

$$MTTF_{duplex} = \frac{1}{2\lambda} + \frac{c}{\lambda} \quad (17)$$

The main difference between a duplex and a TMR system is that in a duplex, the faulty processor must be identified. Next the various ways in which the faulty processor can be identified is discussed (Koren and Krishna, 2010).

Basic measures of fault tolerant: Measures is a mathematical an abstraction that expresses some relevant fact of the performance of its object:

- **Traditional measures:** The system can be in one of two states: Up or down. For examples, light bulb good or burned out and wire: connected or broken.
- **Reliability measures:** formal definitions are as following:
  - **Failure rate:** fraction of unit's failing/unit time, e.g., 1000 units, 3 failed in 2 h, then the failure rate = 3/1000*2 = 1.5*10-3 per hour.
  - **Mean Time to Failure (MTTF):** MTTF an important reliability measure as it is the mean time to failure (MTTF) which is the average time to the first failure. It can be obtained from the mean of the probability density of the time to failure *f*(*t*):

$$MTTF = \int_0^\infty tf(t)dt \quad (18)$$

With a constant hazard rate λ*t* = const:

$$MTTF = \theta = \int_0^\infty \exp(-\lambda t)dt = 1/\lambda \quad (19)$$

Numerical example: The mean time to failure of a component characterized by a constant hazard rate is MTTF = 50000 h. Calculate the probability of the following events:

- The component will survive continuous service for one year.
- The component will fail between the fifth and sixth year.

- The component will fail within a year given that it has survived the end of the fifth year. Compare this probability with the probability that the component will fail within a year given that is has survived the end of the tenth year.

Solution:

- Since MTTF = 50000 h = 5.7 years, the hazard rate of the component is λ = 1/5.7 years. Reliability is determined from R(*t*) = exp(-λ*t*) and the probability of surviving one year is:

$$R(1) = P(T > t) = e^{-1/5.7} \approx 0.84$$

- The probability that the component will fail between the end of the fifth and end of the sixth year can be obtained from the cumulative distribution function of the negative exponential distribution:

$$R(5 < T \leq 6) = F(6) - F(5) = \exp\left(-\frac{5}{5.7}\right) - \exp\left(\frac{6}{5.7}\right) \approx 0.07$$

- Because of the memory less property of the negative exponential distribution, the probability that the component will fail within a year, given that it has survived the end of the fifth year, is equal to the probability that the component will fail within a year after having been put in use:

$$R(5 < T \leq 6) = P(0 < T \leq 1) = 1 - \exp\left(-\frac{1}{5.7}\right) \approx 0.16$$

Similarly, the probability that the component will fail within a year given that it has survived the end of the tenth year is obtained from:

$$R(10 < T \leq 11 > 10) = P(0 < T \leq 1) = 1 - \exp\left(-\frac{1}{5.7}\right) \approx 0.16$$

This probability is equal to the probability from the previous, because of the memory less property of the negative exponential distribution (Todinov, 2005).

Mean Time to Repair (MTTR): MTTR is the expected time until repaired. If we have a system of N identical components and the i_{th} component requires time t_i to repair, then MTTR is given by:

$$MTTR = \frac{1}{N} * \sum_{i=1}^{N} t_i \quad (20)$$

Mean Time Between Failures (MTBF): The mean time between failures can be defined in two ways:

- MTBF is the MTTFs in repairable devices.
- MTBF is the sum of the mean time. of MTTFs of the device plus the MTTR (Mean time to repair/restore):

$$A = \frac{MTTR}{MTBF} = \frac{MTTF}{MTTF+MTTR} \quad (21)$$

A related measure, called point availability, denoted by Ap(t) is the probability that the system is up at the particular time instant t. It is possible for a low-reliability system to have high availability. Consider a system that fails on average every hour but comes back up after only a second, (MTTF). Such a system has an MTBF of just 1 h (60 m*60 s = 3600 s) and, consequently, a low reliability however, its availability is high and is expressed as, A = MTTF/MTBF = 3599/3600 = 0.99972.

MTBF and MTTR: An estimation of system availability from MTBF and MTTR is given by:

$$Availability = \frac{MTBF}{MTBF+MTTR} \quad (22)$$

If the mean MTBF or MTTF is very large as compared to the MTTR, then you will see high availability. This simple equation is easily understood by considering Fig. 20. MTTR is the time to return a system to service and MTBF is the time the system is expected to be up or online before it fails (again). This means that the system will nominally be online and the system is formally defined by [TL9000] as, "A collection of hardware and/or software items located at one or more physical locations where all of the items are required for proper operation. No single item can function by itself." (Bauer and Adams, 2012).

Service availability: Service availability can be quantified by using Eq. (23) (basic availability formula) as service uptime divided by the sum of service uptime and service downtime:

$$\text{Availability} = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}} \quad (23)$$

Equation (24) (practical system availability formula) calculates the availability based on service downtime, as well as the total time the target system (s) was expected to be in service(i.e., the minutes during the measurement period that systems were expected to be online so planned downtime is excluded):

$$\text{Availability} = \frac{\text{TotalInSeviceTime - Downtime}}{\text{TotalInSeviceTime}} \quad (24)$$

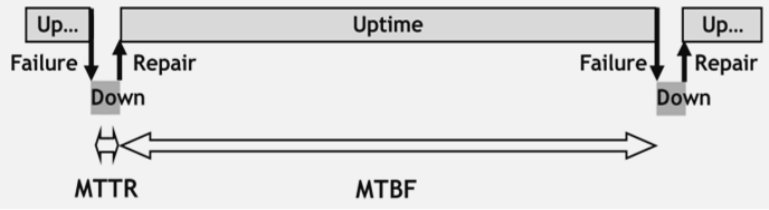

Fig. 20: MTBF and MTTR

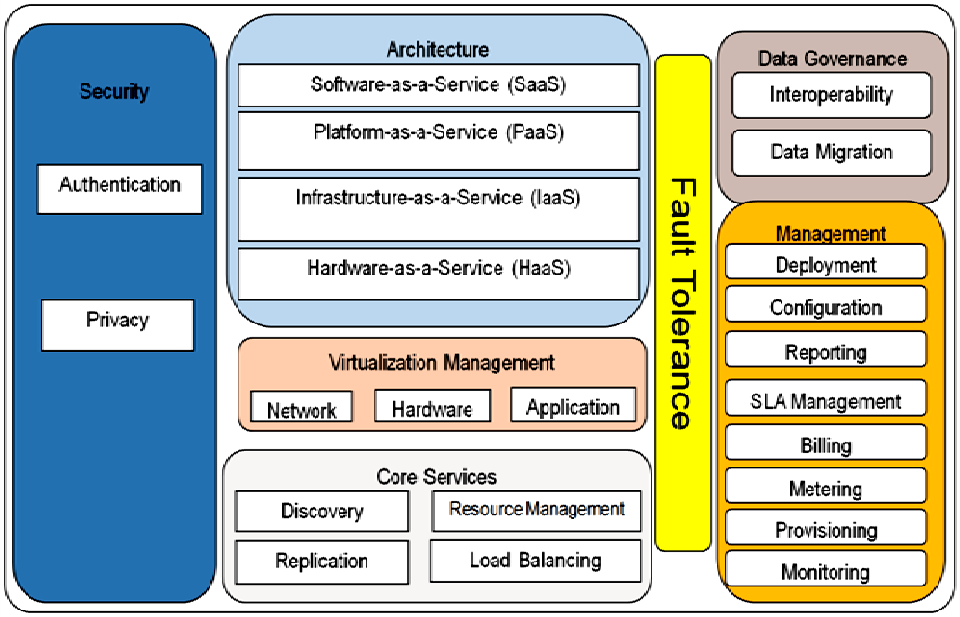

Fig. 21: Fault tolerance in cloud computing platform

where, Total In Service Time is: the sum of minutes per month (or other reporting periods) that the systems in the population were expected to be operational.

Downtime: Is the minutes of service unavailability prorated by the percentage of capacity or functionality impacted during the outage (Bauer and Adams, 2012).

RESULTS AND DISCUSSION

Cloud computing has quickly become the de facto means to deploy large scale systems in a robust and cost effective manner. The combination of elasticity and scale poses a series of challenges to a number of areas, including fault-tolerance. This survey a comprehensive review of representative works on fault tolerance in cloud computing is presented, in which general readers will be provided an overview of the concepts and practices of a fault-tolerance computing.

CONCLUSION

In this study, we surveyed the use of fault tolerance in cloud computing. Cloud computing is positioning itself as a new platform for delivering information infrastructures and a range of computer applications for businesses and individuals as IT services and developing them in the future works as shown in Fig. 21. Cloud customers can then provision and deploy these services in a pay-as-you-go fashion and in a convenient way while saving huge capital investment in their own IT infrastructures. Clouds are evoking a high degree of interest both in developed and emerging markets though challenges such as security, reliability and availability remains to be fully addressed to achieved full fault tolerance services in the cloud platform.

ACKNOWLEDGMENT

Thanks Chinese High-tech R and D(863) program project "Cloud Computer Test and Evaluation System Development (2013AA01A215)" for supporting of this research.

REFERENCES

Bauer, E. and R. Adams, 2012. Reliability and Availability of Cloud Computing. John Wiley and Sons, New York.

Belli, F. and W. Görke, 2012. Fehlertolerierende rechensysteme/fault-tolerant computing systems. Proceedings of the 3rd International GI/ITG/GMA-Fachtagung/Conference Bremerhaven, September 9-11, ISBN: 978-3-540-18294-8.

Birolini, A., 2007. Reliability Engineering: Theory and Practice. Springer, Berlin, Heidelberg.

Dhillon, B.S., 2007. Applied Reliability and Quality: Fundamentals, Methods and Procedures. Springer-Verlag, London Ltd.

Dubrova, E., 2013. Fault-tolerant Design. Springer Science+Business Media, New York, pp: 185.

El-Damcese, M. and N. Temraz, 2015. Analysis of availability and reliability of k–out–of–n: F model with fuzzy rates. Int. J. Comput. Sci. Eng., 10(1-2): 192-201.

Gray, J. and D.P. Siewiorek, 1991. High-availability computer systems. Computer, 24(9): 39-48.

Jin, H., D. Zou, H. Chen, J. Sun and S. Wu, 2003. Fault-tolerant grid architecture and practice. J. Comput. Sci. Technol., 18(4): 423-433.

Koren, I. and C.M. Krishna, 2010. Fault-tolerant Systems. Morgan Kaufmann.

Latchoumy, P. and P.S.A. Khader, 2011. Survey on fault tolerance in grid computing. Int. J. Comput. Sci. Eng. Surv., 2(4): 97-110.

Lazzaroni, M, 2011. Reliability Engineering: Basic Concepts and Applications in ICT. Springer-Verlag, Berlin, Heidelberg.

Lazzaroni, M., L. Cristaldi, L. Peretto, P. Rinaldi and M. Catelani, 2011. Repairable Systems and Availability. In: Reliability Engineering. Springer, Berlin, Heidelberg, pp: 85-92.

McKelvin Jr., M.L., 2011. A methodology and tool support for the design and evaluation of fault tolerant, distributed embedded systems. Ph.D. Thesis, University of California, Berkeley.

Mell, P. and T. Grance, 2011. The NIST Definition of Cloud Computing. Retrieved from: http://csrc.nist.gov/publications.

Naixue, X., Y. Yan, C. Ming, H. Jing and S. Lei, 2009. A survey on fault-tolerance in distributed network systems. Proceeding of the IEEE International Conference on Computational Science and Engineering (CSE, 2009). Vancouver, BC, pp: 1065-1070.

Pradhan, D.K., 1996. Fault-tolerant Computer System Design. Prentice-Hall, Upper Saddle River, NJ.

Qureshi, A.A., M.H. Levine and J.K. Shim, 2004. The International Handbook of Computer Networks. Global Professional Publishi, Barming, Kent, ISBN: 1858820596, pp: 302.

Rohit, S., 2014. Fault Tolerance Vs High Availability. Retrieved from: https://www.ibm.com/developerworks/community/blogs/RohitShetty/entry/fault_tolerance_vs_high_availability?lang=en.

Sabahi, F., 2011. Cloud computing Reliability, Availability and Serviceability (RAS): Issues and challenges. Int. J. Adv. ICT Emerg. Regions, 4(02): 12-23.

Saha, G.K., 2003. Fault management in mobile computing. Ubiquity, 2003(October), Article No. 1 DOI: 10.1145/948943.948944.

Todinov, M., 2005. Reliability and Risk Models: Setting Reliability Requirements. John Wiley & Sons, New York.

UK Essays, 2013. The Fault Tolerance in Distributed Systems Information Technology Essay. Retrieved from: http://www.ukessays.com/essays/information-technology/the-fault-tolerance-in-distributed-systems-information-technology-essay.php?cref=1.

Xiong, N., A.V. Vasilakos, L.T. Yang, L. Song, Y. Pan, R. Kannan and Y. Li, 2009. Comparative analysis of quality of service and memory usage for adaptive failure detectors in healthcare systems. IEEE J. Sel. Area. Comm., 27(4): 495-509.