Increasing System Fault Tolerance with Software Rejuvenation in E-government System

Manish Pokharel[†] and Jong Sou Park[††]

Network Security Lab, Korea Aerospace University, South Korea

Summary
The main goal of E-government system is to provide the services to its user in 24/7/365 philosophy. User expects the entire required services from e-government system at any point of time and location. It is always expected that the e-government system should run without any type of interruptions or faults. A system should have the capability enough to tolerate the faults and operate continuously in spite of faults and interruptions. In this paper, we try to mention the need of fault tolerant system in e-government and also propose architecture for fault tolerant approach in e-government system with the concept of availability and reliability. We use design diversity and software rejuvenation together in this paper to increase the system fault tolerance capability.

Keywords
Availability, Reliability, Fault Tolerant, Design Diversity, E-government

1. Introduction

Electronic government is the method of automating governments' activities to gain maximum efficiency and effectiveness. Efficiency and effectiveness can be evaluated by various ways. A system is said to be effective and efficient only if the service is available to the citizen on time. The services that we expect from the system should be available any time of the year or day. This we call as availability of the services in the system. During the life cycle of the system, it goes into different phases and there is a high probability of malfunctioning or getting attacked by unauthorized users. A good system always promises to provide the services in spite of being attacked or component malfunctioning and such capability of system is known as fault tolerant capability. A system with high fault tolerance capability is always desire in e-government system. Here, we provide the solution to enhance the fault tolerance of a system with a two separate designed systems and fix the fault with software rejuvenation.

Besides, the fault tolerance, reliability and availability are also main concerned in system. System reliability is concerned with the quality of output that it gives, the time taken to produce the output, the capability of system to operate in undesired environment. The more reliability the less chance of system failure. We always desire for maximum reliability in the system. As per the survey based upon the users' perspective, reliability ranks first on the list of customer satisfaction. [11] The cost involved on it and the effort involved in this during developing the system is very high especially in the case of e-government where many efforts from many sectors are required to build it. The country has to put huge amount of budget to develop the system. Especially in the case of developing countries, it is very hard to manage huge budget and use it in a proper way. In an average more than US$30 million is required to develop the e-government system. This is the huge budget for any nation and specially developing nations. In Nepal, total estimated budget for e-government system is US$31.2 million. Unfortunately, even after investing these amounts, it is very difficult to get the reliable system. Country like Nepal cannot bear any fail or loss after putting such a huge amount of budget. As per the Heek's survey, in total, 85% projects are failure, including total and partial failure. [13]

In order to increase the fault tolerance in e-government system, in this paper, we mention the present status in Section 2 and Section 3. We propose architecture and solution for taking care of fault tolerance in Section 4 and Section 5 and conclude it in Section 6 with the conclusion.

2. State of Arts and Related Works

In most of e-government master plan or in project the concept of fault tolerance is not clearly defined or discussed. None of the document mentioned about the recovery model and still there is a huge confusion and big debate on it. There are many cases of system down in e-government system especially in the developing countries. The system does not work for a long time and there is a very less effort to fix the fault. There are some cases in which the system is alright but services are not available. Sometimes, only partial services are available. Even it is fixed; it does not run for a long time. This gives the frustration for citizens and ultimately gives the negative impact on e-government system. Let's take the example, if farmers are using land registration system for certain period of time and if this system goes down

Manuscript received May 5, 2010
Manuscript revised May 20, 2010

because of fault, the farmer would be deprived with such facilities. The time to fix this fault would be very high. By the time, the system gets fixed; farmers would lose their patient and ultimately affects the goal of e-government system. There are many such examples in e-government system.

We try to find out the solution for handling such fault in the master plan of e-government system of developing countries, especially in Nepal but unfortunately we did not find any mechanism for fault management. There is not a single chapter in these issues. Few terminologies like hot standby and high availability is mentioned but the significance and use of these terminologies are not clearly discussed. The country like Afghanistan has tried very good effort in implementing e-government system but they also miss such approach like fault tolerant approach in their ICT strategy. Discussing on two developing countries does not mean that it applies on developing countries alone. It is not only for developing countries, even developed countries require to give more emphasize in these areas. After reviewing the case of Nepal and Afghanistan, we have proposed the architecture for solving the failure system.

There are many works done related in availability, reliability and fault tolerant system but very less work done in e-government system. Rene Meier and Paddy Nixon have worked in managing fault tolerant transparently using CORBA services. They have suggested architecture for banking system. Aaron B. Brown and David A. Patterson also have contributed in the area of availability. Service Oriented Architecture (SOA) is used in some cases to address these challenges but it is in very immature stage. Professor Kishor Trivedi from Duke University, USA has contributed a lot in these domains. Professor Trivedi has demonstrated the modeling concept of high availability systems using Markov chain and evaluated using SHARPE software packages. [5]

3. Proposed Solution

In this section, we propose the architecture and the technologies that we apply on it to provide the best fault tolerance capability in e-government system.

3.1 Proposed Architecture

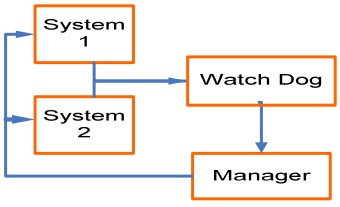

Fig 1: Architecture for Reliable E-Government System

In the architecture in Figure 1, there are two systems running together for the same purpose. System 1 is designed separately than system 2. There are two more components including system 1 and system 2. A component known as watch dog is a software component that monitors continuously to both the systems. If a watch dog identifies the problem or if it observes some abnormalities in any one of the two systems then it immediately informs manager about such abnormalities. A manager examines the defective system and repairs it with the approach of software rejuvenation. We propose the entire e-government system to be designed with separate design and provide the effective services to the citizen. In most of the cases there are many systems which are separately designed in design diversity but in our case, to be more cost effective and to make economically feasible we propose only two separate systems along with a watch dog and manager to enhance the fault tolerance capability and increase the system's properties. A manager is assumed to be very smart to rejuvenate the design defect in the system and switching to new design.

In this paper, the failures mean software failures. The main reason of focusing on it is system failures because of software fault. There are many approaches in addressing software faults like backward recovery, forward recovery etc. After going through all these techniques we have proposed a very popular approach like "Design Diversity" for e-government system.

3.2 Software Rejuvenation

Software Rejuvenation is an act of preventing unexpected error termination by terminating the program before it suffers an error. It restarts the system with clean state. Software rejuvenation refers to the rejuvenation of the environment in which the software is executing. A software system is known to suffer from outages due to transient errors and the state of software system degrades as time goes. [12]So there is a need of proactive approach for managing such faults. In our architecture, a manager takes the responsibility for doing task of software rejuvenation. The system 1 and system 2 in Figure 1 are the software components with a set of software applications. As the nature of the software, after some span of time, the process corresponding to the software slowly degrades with respect to the effective use of their system resources. Process aging in both systems is because of memory leaking, file and data corruption etc. Process aging affects the performance of the system and finally leads to fail. [14] We need software rejuvenation approach to manage the process aging of the both systems.

3.3 Design Diversity

It is the method of providing same services through separate design and implementation. In Design Diversity, different teams make a different system for same purpose. These teams develop the system independently. This is known as Multi-version system or N-version system. The number and the type of versions depend upon the discretion of the developer. Multiple applications are executed in parallel; the best result is used in the system. This decision is made by an independent entity called watchdog.

In e-government system, we list out the number of services or functions that we expect from the system. As per the nature of services, we make more than one design. Each design has the responsibility of providing the services. If fault occurs because of one design then the system will switch to other design immediately without disturbing the functionality of the system.

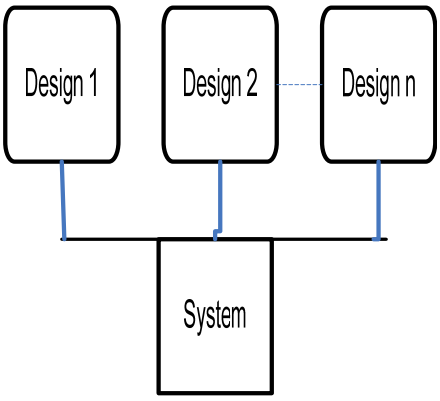

Fig 2: Design Diversity

In Figure 2, there are n numbers of design for some specified set of services provided by the system. One design module operates at one time. If this fails then it will switch to the other design module. The other module has the design diversity for the same services. We can put n numbers of such design module. We have to be very careful in designing the number of modules.

3.3.1Why Design Diversity?

The presence of software is getting increased day by day in e-government system. It is not possible to think e-government system without considering the involvements of software. At one side the use of software is increasing and other side the chances of failure are also increasing. The more use of software is more prone to the failure. A special effort is required to overcome the failure with the maximum use of software. There are some hardware components in e-government system but the probability of failure in hardware is very less as compare to software. The main sources of hardware faults are component aging and environment effects. This can be reduced with a single redundant hardware component which is designed in such a way that it can tolerate the hardware faults.

In the case of software, it is different. The source of software faults is right from requirement gathering, specification, design and till implementation. It is very challenging to tolerate such faults. In e-government, the percentage of software faults is very high. The application which is built to support the e-government system is not made properly i.e. very less attention is given during requirement gathering, and system development. This is one of the reasons that why do we need multiple version of same application in the system. This is the philosophy of design diversity.

Redundancy and dependability are two key words in software applications. Redundancy is the approach of providing extra energy or capabilities and resources that are required to find out the faults and mechanism to tolerate it. Here we give more focus on software redundancy which contains nothing but additional programs, modules, components, architecture etc. Dependability is the intensity of trusting the software. If there is high level dependability then we can obtain high level of trustworthy. The need of dependable software is mandatory and there are many researches going on these topics. The use of redundancy to improve the system dependability is based on the assumption that if one version fails, the remaining versions will take care of this. [2]

The MLDD(Multi-Layered Design diversity) architecture is based on three tier architectures, in which design diversity is applied to every layers of application programs.[3] Service Oriented Architecture (SOA), which is based upon the three layers architectures as: Client , Service Provider, and Repository, the MLDD can be used. Service Oriented Architecture is considered as one of the solutions for e-government systems. Fault tolerance through design diversity has been suggested both for achieving higher reliability. [4]Most of the time software bug makes the problem in system development because of its nature of spreading. The bug can move to another domain if same logic, same code or same design is used in another module. In the case of design diversity, we use different design, different code and different logic for same problem in different modules. Design diversity can take care of this. Hence Design diversity is the solution for reliability, availability and ultimately for developing fault tolerant system.

4. Analysis:

In order to perform the analysis, we consider the two states diagram. One system is with manager, which is assumed to be very smart enough to do task of software rejuvenation, switching from one design to another and another without manager. A system with manager consists of design diversity where as a system without manager does not contain the design diversity.

4.1 With Manager (Design Diversity):

We develop following state diagram based upon the proposed architecture in Figure 1.

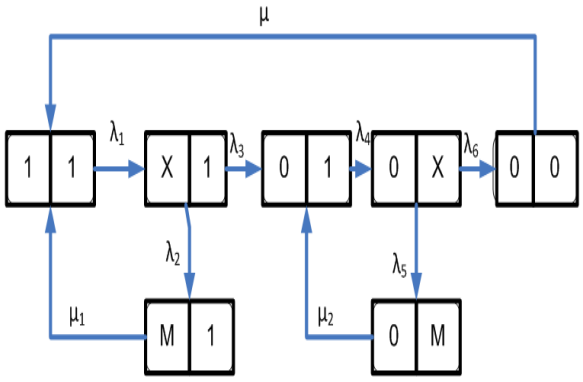

Fig 3: State Diagram

[1, 1]: Both designs are in upstate i.e., both designs are active.
[X, 1]: Design I is in problem where as design II is in upstate or active.
[M, 1]: Design I is getting recovered i.e. in recovery state. Design II is in upstate.
[0, 1]: Design I is down, it does not provide services. Design II is in upstate
[0, X]: Design I is down. Design II gets problem i.e. it is in problematic state.
[0, M]: Design I is down. Design II is getting recovered.
[0, 0]: Both designs are down. This is the failure state.

The above state transition diagram in Figure 3, has seven states in total. These states are identified as per the behavior of the proposed system. Each state has two blocks. Each block represents the status of one design.
Initially both designs are in upstate. [1, 1] One of them provides the service where as another is ready to provide services if something wrong happens to first design. With the fault rate λ_1 it changes its state to [X, 1] where first design encounters the problem and second one is in upstate. With the recovery rate λ_2 it moves to the state [M, 1]. If it does not move to recovery state then it moves to the state in which first design is fail and another takes care with the switchover rate λ_3. Here, the techniques of software rejuvenation are applied to recover the fault. If the design I gets recovered then it moves to the healthy state with repair rate μ_1.
After some time, this also may get problem with the second fault rate λ_4. Here also approach of software rejuvenation is applied to fix the design problems so it moves to recovery state with rate λ_5 and if not then it moves to the total failure state with rate λ_6. If it gets fixed, it will be in switchover state with design II up It is assumed that the total failure can be addressed with some manual repairing activities. It can go to the healthy state with given repairing rate.
We use CTMC (Continuous Time Markov Model) to model the architecture given in Figure 1. We calculate the state probability of each state and use the balance equation to get the final equation for Good State. We emphasize on the probability of a system in which both design works i.e. Good state. In our state diagram in Figure 3, state 1, 1, is a good state. We get the following equation for a system to be in good state.
Good
State=

$$\left(1+\frac{\lambda_1}{\lambda_2+\lambda_3}+\frac{\lambda_1\lambda_2}{\mu_1(\lambda_2+\lambda_3)}+\frac{(\lambda_2+\lambda_3)(\lambda_5+\lambda_6)\lambda_1-\lambda_1\lambda_2(\lambda_5+\lambda_6)}{(\lambda_2+\lambda_3)\lambda_6\lambda_4}+\frac{(\lambda_2+\lambda_3)\lambda_1-\lambda_1\lambda_2}{(\lambda_2+\lambda_3)\lambda_6}+\frac{(\lambda_2+\lambda_3)\lambda_1\lambda_5-\lambda_1\lambda_2\lambda_5}{(\lambda_2+\lambda_3)\lambda_6\mu_2}+\frac{(\lambda_2+\lambda_3)\lambda_1-\lambda_1\lambda_2}{(\lambda_2+\lambda_3)\mu}\right)^{-1}$$

As per the nature of proposed architecture, we assume the following parameters, frequencies and their respective values given in Table 1.

Table 1: Parameters and its Values

S.N.	Parameters	Frequency
1	λ_1	1 time in a six months
2.	λ_2	1 time in a day
3.	μ_1	1 time in an hour
4.	λ_4	1 time in a six months
4.	λ_3	1 time in a week
5.	λ_5	1 time in a day
6.	λ_6	1 time in a month
7.	μ_2	1 time in an hour
8.	μ	1 time in a day

4.1.1 Availability:
It is a probability of a system which provides the services in a given instant of time. In e-government system, it is the probability of a system that makes service available to the citizen. We calculate the steady state availability with the following given equation.
Availability= 1- Unavailability (1)
A system does not provide the service when it is in state [0, M] and [0, 0]. Hence the availability is given by,
Availability = 1- {[0, M][0,0]}

4.1.2 Survivability:
It is the probability of a system in which the system survives or continuous to provide the service in spite of attack or failure.
Survivability = Availability- Service Unavailable (2)
Survivability = Availability – [0, M]

4.1.3 Downtime:
It is a duration that the system goes down. We consider the whole one year calculating it. Downtime can be calculated with following equation.
Downtime= Service unavailable x L (3)
Where,
L is a one year.
We calculate the properties that make the system to counterattack the faults. The properties such as availability, downtime, survivability and reliability are calculated and verifies with SHARPE.

4.1.4 Reliability:
It is the probability of a system in which the system provides the service in a given range of time. We can calculate the reliability from following equation.
Reliability= 1-Unreliability (4)
A system is unreliable if it is in states of service unavailability i.e. states [0.M] and [0, 0]

4.2 Without Manager (No Design Diversity):

We also consider the case in which there is no design diversity. It is the system in which there is no alternative design.

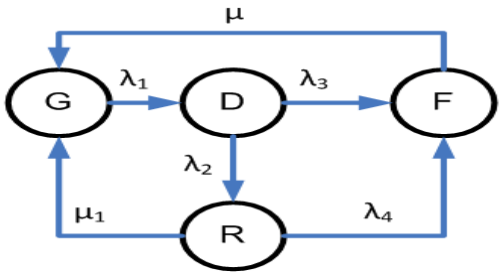

Fig 4: Without Design Diversity

The Figure 4 is the state diagram of a system without design diversity. There are four states in Figure 4. The states, G, D, R, and F are equivalent to states [1, 1], [X, 1], [M, 1] and [0, 0] in Figure 3. Here, also we use CTMC (Continuous Time Markov Chain) and try to find the properties such as availability, survivability, downtime and reliability as we did before with same operation parameters. We also plot the graph in Figure 5 that shows the properties values with the repair rate.

5. Results:

We use above CTMC and corresponding equations to find out the properties of the system. The Table 2 shows the values of obtained properties.

Table 2: With Manager (Design Diversity)

S. N	*Manager Rate(λ2 and λ5)*	*Availability*	*Downtime*	*Survivability*	*Reliability*
1	1 Time in a day	0.99965	181 hours	0.99930	0.99965
2	2 Times in a day	0.99981	97.2 hours	0.99963	0.99981
3.	3 Times in a day	0.99987	66.3 hours	0.99974	0.99987
4.	4 Times in a day	0.99990	50.5 hours	0.99980	0.99990
5.	6 Times in a day	0.99993	33.9 hours	0.99987	0.99993
6.	8 Times in a day	0.99995	25.8 hours	0.99990	0.99995
7.	12 Times in a day	0.99996	17.1 hours	0.99993	0.99996

The values in Table 2 are obtained with the increase of manager rate from one time in a day to 12 times in a day. The equivalent graph is plotted in Section 5.1.

5.1 Graph:

We plot the graph of availability verses rate of rejuvenation on design.

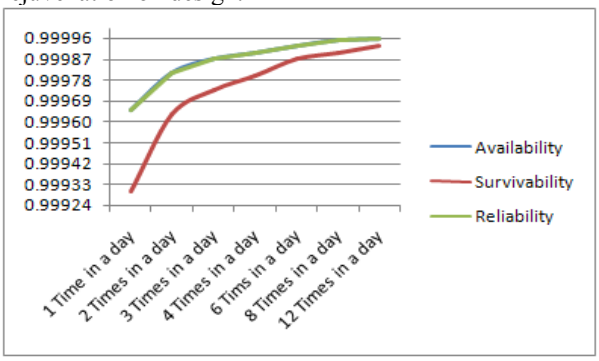

Fig 5: Availability, Survivability and Reliability Verses Rejuvenation

The graph in Figure 5 shows the increase in availability, survivability and reliability as per the increase of times of rejuvenation. We start it from 1 time in a day to 12 times in a day and find the increase trend and it can be achieved almost five 9's.

Table 3: Without Manager (Design Diversity)

S. N	*Repair Rate (λ2)*	*Availability*	*Downtime*	*Survivability*	*Reliability*
1	1 Time in a day	0.99736	4250 hours	0.99473	0.99737
2	2 Times in a day	0.99740	2850	0.99481	0.99740
3.	3 Times in a day	0.99741	2380	0.99482	0.99741
4.	4 Times in a day	0.99742	2140	0.99484	0.99742
5.	6 Times in a day	0.99742	1890	0.99485	0.99742
6.	8 Times in a day	0.99743	1780	0.99486	0.99743
7.	12 Times in a day	0.99743	1650	0.99487	0.99743

The Table 3 and Figure 6 show the clear picture of obtained data and their differences with and without manager. The properties such as availability, survivability, and reliability are better in with manager rather than without manager. It means maximum properties can be obtained with design diversity as compare to without design diversity.

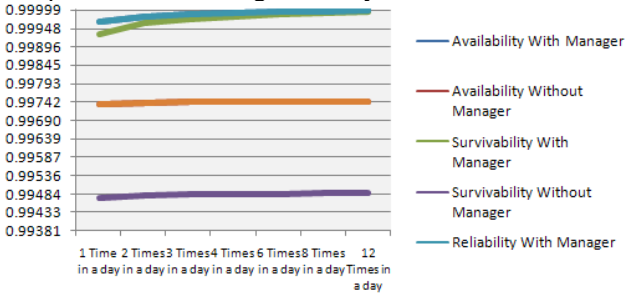

Fig 6: Comparison between with and without manager (Design Diversity).

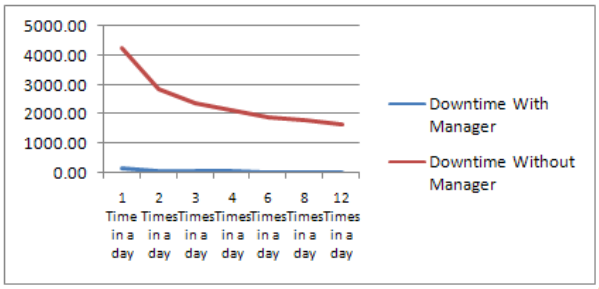

Fig 7: Down Time: With and without manager (Design Diversity)

The above Figure shows the differences in downtime with and without manager. Here, also it shows the downtime is very less with manager and whereas downtime is very high without manager.

6. Conclusion

E-government system has become the integral part of human civilization. People around the world are depending more and more on it. People have a lot of expectations from e-government system. These expectations can be fulfilled in some extent with the consideration of availability, reliability and fault tolerant. We proposed a architecture that consists of design diversity and software rejuvenation to enhance the fault tolerance capability of a system. We identified the differences on availability, survivability; downtime and reliability between with and without manager and these properties can be increased or decreased as per the switching rate of the software rejuvenation. The obtained results clearly showed that features such as availability, survivability and reliability can be achieved maximum with design diversity. The given approach is very suitable to run and sustain the e-government system.

References

[1] Aaron B. Brown and David A. Patterson, Embracing Failure: A Case for Recovery-Oriented Computing (ROC), High Performance Transaction Systems Workshop, University of California at Berkeley 2001

[2] Hawthrone Matthew J. and Perry Dewayne E. Applying Design Diversity to Aspects of system Architectures and Deployment configuration to Enhance System Dependability, The University of Texas at Austin.

[3] Watanabe Aki, Takada Hiroaki, Sakamura Ken. The Multi-Layered Design Diversity Architecture: Application of The Design Diversity Approach to Multiple System Layers. IEEE 1992. [4] Littlewood Bev, Popov Peter, Strigini Lorenzo. Modeling Software Design Diversity – A Review.ACM Computing Surveys, Vol. 33, No. 2 June 2001, 177-208.

[4] Trivedi S. Kishor, Vasirreddy Ranjith;, Modeling High Availiability systems, Duke University, Durham, NC, USA

[5] Trivedi S. Kishor. Probability and Statistics with Relaibility, Queuing and Computer Science Application, Second Edition, Wiley, 2001

[6] KIPA, Government of Nepal, e-Government Master Plan Consulting Report, 2006

[7] Heeks, R. Implementing and Managing eGovernment, Vistar Publication, 2006.

[8] Asian Development Bank, Proposed Asian Development Fund Grant Nepal: Information and Communication Technology Development Project.(Dec. 2007)

[9] Pullum Laura L. Software Fault Tolerance, Techniques and Implementation, Artech House, 2001.

[10] Zhang Miaomiao, Liu Zhiming, Ravn Anders P. Design and Verification of a Fault-Tolerant System, UNU-IIST Report No. 387, November 2007.
[11] Trivedi Kishor S., Vaidyanathan Kalyanaraman, Goseva-Popstojanova Katerina., Duke University Durham, NC USA
[12] Heeks Richard, Implementing and Managing eGovernment., Vistar Publication 2006
[13] Huang Yennum, Kintalla Chandra, Koletiss Nick, and Fulton N. Dubley., Software Rejuvenation: Analysis, Module and Application, IEEE-1995

Manish Pokharel obtained his Bachelor of Engineering in Computer Science from Karnataka University, India. He received his Master of Engineering in Software System from Birla Institute of Technology and Science, Pilani, India. He is currently a Ph.D. student in Korea Aerospace University since September 2007. His research interests are in E-government, Enterprise Architecture, Software Technology, Fault Tolerance and Cloud Computing.

Jong Sou Park received the M.S. degree in Electrical and Computer Engineering from North Carolina State University in 1986. And he received his Ph.D in Computer Engineering from The Pennsylvania State University in 1994. From 1994 - 1996, he worked as an assistant Professor at The Pennsylvania State University in Computer Engineering Department and he was president of the KSEA Central PA, Chapter. He is currently a full professor in Computer Engineering Department, Korea Aerospace University. His main research interests are information security, embedded system and hardware design. He is a member of IEEE and IEICE and he is an executive board member of the Korea Institute of Information Security and Cryptology and Korea Information Assurance Society.