Augmented Architectural Reliability of Split-protocol

Bharat S. Rawal, Oothongsap Phoemphun, Harold Ramcharan, Lloyd Williams
Department of Computer and Information Sciences
Shaw University
North Carolina, USA
{brawal, poothongsap, haroldrm, llwilliams}@shawu.edu

ABSTRACT: *HTTP request splitting is a novel concept where, the TCP connection and data-transfer phases are dynamically split between servers without the use of a central dispatcher or load balancer. Previous work has studied through performance and established a throughput improvement range between 6% and 25 %, and demonstrated 84% reduction in transmission time compare to conventional technique. We believe this higher performance results from inbuilt reliability in the architecture of split protocol. There are plenty of researches, and literatures focusing on component-based reliability. Software Engineering researchers assume each component has a unique service. To distinguish from underlying TCP /IP protocol reliability, we propose modeling approaches that consider the architecture of the software and estimate the reliability of interactions between the components, and interfaces with other components. This paper details the components of the Split –architecture, and devises reliability assessment based on individual component of software and describes how it can be used to examine software behavior. It highlights on an inbuilt reliability in a split-protocol due to the dual and interchangeable role of Connection Server (CS) and Data Server (DS). This innovative approach introduces a more effective way of offering reliability in cluster computing and in general. To assess the reliability of the system, this paper proposes a simple mathematical model which can capture the reliability of the system. This model compares the reliability of the split system against dispatcher system. The results show that split system reliability is far better than dispatcher system reliability. Moreover, the system hazard rates and Survival functions of the parallel system are provided.*

Keywords: Splitting HTTP Requests, Reliability, Cluster Computing, Software, Bare Machine Computing, Parallel component, Series component

Received: 6 November 2012, Revised 10 December 2012, Accepted 14 December 2012

1. Introduction

Reliability is defined as a measure of time between failures (as a numeric value such as meantime between failures), where failure is defined as a departure from acceptable service for an application, a computer system, or the network system. It is a measure of how consistently a machine or set of machines perform the tasks that are their purpose [1]. The reliability of a software system expresses the probability that the system will be able to perform as expected when requested. This probability is interpreted in the context of an expected pattern of use and a specified time frame. [2] The reliability of a software product is usually defined to be "*the probability of execution without failure for some specified interval of natural units or time*" [3].

There are multiple approaches to software reliability. Some people view reliability as binary so that a faulty program would have zero reliability while a perfect one would have a reliability value of one. This view parallels that of program proving whereby the program is either correct or incorrect [31]. Others, however, believe that software reliability should be defined as the relative frequency of the times that the program works as intended by the user. This view is similar to that taken in testing where a percentage of the successful cases are used as a measure of program quality. According to the alternative viewpoint, software reliability is a probabilistic measure and can be defined as the probability that software faults do not cause a failure during a specified exposure period in a specified use environment. The probabilistic nature of this measure is due to the uncertainty in the usage of the various software functions. To interpret this view of software reliability, suppose that a user executes a software product several times according to its usage profile and finds that the results are acceptable 95 % of the time. Then the software is said to be 95 % reliable for that user. A more precise definition of software reliability is as follows [29]. Let F be a class of faults, defined arbitrarily, and T be a measure of the relevant time, the units of which are dictated by the application at hand. Then reliability of the software package with respect to the class of faults F and with respect to the metric T is the probability that no fault of the class occurs during the execution of the program for a pre-determined period time [32]. If software reliability can somehow be measured, a logical question is what purpose does it serve? Software reliability is a useful measure in planning and controlling resources during the development process so that high quality software can be developed. It is also a useful measure for giving the user confidence about software correctness. As more and more faults are exposed by testing and verification process, the additional cost of exposing the remaining faults usually rises very quickly. Thus, there is a point beyond which continuation of testing to improve the quality of software can be justified only if such improvement is cost effective. An objective measure like software reliability can be used to study such a tradeoff [30].

Load balancing is frequently used to enable Web servers to dynamically share the workload and improve system availability and reliability. For load balancing, a wide variety of clustering server techniques [22]-[23] are employed. Most load balancing systems used in practice require a central control system such as a load balancing switch or dispatcher [22]. Load balancing can be implemented at various layers in the protocol stack [21]. This paper considers a recently developed a new approach to load balancing that involves splitting HTTP requests among a set of two to four servers, where one or more connection servers (CSs) handle TCP connections and may delegate a fraction (or all) requests to one or more data servers (DSs) that serve the data [25]. For example, the data transfer of a large file could be assigned to DSs, and the data transfer of a small file could be handled by the CS itself. One advantage of splitting is that splitting systems are completely autonomous and do not require a central control system such as dispatcher or load balancer.

We will consider split-protocol as a system and each individual protocol transaction as a component of a system. Such as SYN, SYN-ACK, ACK, GET, GET-ACK, ISP (Inter Server Packet), DATA, DATA-ACK, FIN-ACK, and FIN-ACK-ACK are functional component of protocol (system). Figure 1 shows that the CS is composed of 10 components while DS is composed of three components. The CS component types may be different from the DS component types. The numbers of CS and DS components may be different. As subsystems, CS and DS possess different reliability. In our experiment, the make and model of all servers are the same and their hardware reliability is perfect and also the same (for simplicity, the hardware reliability is equal to one). All protocol components have the same software reliability. To assess the reliability of split protocol components we do not include reliability characteristics of TCP/IP protocol, we presume that the TCP/IP offers reliability on top of the component reliability.

This paper proposes a simple mathematical model which can capture the reliability of the system. This model is also used to compare the reliability of the split system against dispatcher system. The results show that split system reliability is far better than dispatcher system reliability.

The remainder of the paper is organized as follows. Section II presents related work, and Section III describes the split configurations used in this study. Experimental setup is presented in section IV. Section V and VI present reliability analysis and comparison study, respectively. Result discussion is presented in section VII. Section VIII contains the conclusion.

2. Related Work

Component-based reliability analysis has been done, from the mid to late 1990s, to until this date many component-based software models have been proposed [4], [5], [6], [7], [8], and [9]. Most of the reliability models based on architecture take the reliability of component as an invariable property of the component itself, disregard the fact that the reliability changes when the component interacts [3].

We implemented HTTP request splitting on a bare PC with no kernel or OS running in the machine. Bare PC applications use the Bare Machine Computing (BMC) or dispersed OS computing paradigm [14], wherein self-supporting applications run on a bare PC. That is; there is no operating system (OS) or centralized kernel running in the machine. Instead, the application is written in

C++ and runs as an application object (AO) [15] by using its own interfaces to the hardware [16] and device drivers. While the BMC concept resembles approaches that reduce OS overhead and/or use lean kernels such as Exokernel [11]. There are significant differences such as the lack of a centralized code that manages system resources and the absence of a standard TCP/IP protocol stack. In essence, the AO itself manages the CPU and memory and contains lean versions of the necessary protocols.

Splitting an HTTP request by splitting the underlying TCP connection is different from related techniques that migrate or splice TCP connections. In migratory TCP (M-TCP), client involvement is needed [18]. Agents migrate between nodes [17], or virtualization is used in a mobile environment [10], two separate TCP connections are established for each request. Splitting is similar to masking failures in TCP [19].

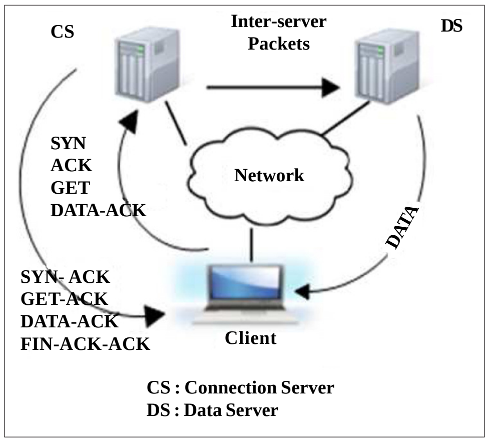

Figure 1. Split architecture (Configuration 1)

3. Split Configurations

The intertwined HTTP and TCP protocols for request splitting [25] are shown in Figure 1. Connection establishment and termination are performed by one or more connection servers (CSs), and data transfer is done by one more data servers (DSs). When a http request is split, the client sends the request to a CS, and the CS sends an inter-server packet to a DS, and the DS sends the data packets to the client. Inter- server packets may also be sent during the data transfer phase to update the DS if retransmissions are needed. With partial delegation, the CS delegates a fraction of its requests to DSs. With full delegation, the CS delegates all its requests to DSs. Protocol intertwining is a form of cross-layer design. The design and implementation detail of protocol splitting is provided in [25]. Further details on bare PC applications and bare machine computing (BMC) can be found in [12] [13]. We consider mini Web server clusters consisting of two or more servers with protocol splitting, and study the performance of four different server configurations with a varying number of CSs and DSs: (i) one CS with one DS, (ii) one CS with multiple DSs (CS does not participate in data transmission), (iii) multiple CSs and single DS (CSs do not participate in data transmission), and (iv) one CS with multiple DSs (CS participates in data transmission).

Configuration 1 in Figure 1 shows a full delegation with one CS, one DS, and a set of clients sending requests to the CS. The DS and CS have different IP addresses, but the DS sends data to a client using the IP address of the CS.

Configuration 2 in Figure 2 shows a single CS with two or more DSs in the system with full delegation (CS does not participate in data transmission). In partial delegation mode, clients designated as non-split request clients send requests to the CS, and these requests are processed entirely by the CS as usual. With the full delegation, clients designated as split-request clients make requests to the CS, and these requests are delegated to DSs [26]. When requests are delegated to DSs, we assume that they are equally distributed among DS1, DS2 and DS3 in round-robin fashion. It is also possible to employ other distribution strategies.

Configuration 3 in Figure 2 shows two CSs and one DS with clients. For this configuration, we used small file sizes to avoid overloading the single DS. Although we have not done so, multiple DSs could be added as in configuration 2, and DS can also participate in connection closing [27].

Configuration 4 in Figure 3 shows one CS with multiple DSs. The CS participates in data transmission by sending the initial block of data. For this configuration, we used large file sizes.

Configuration 5 and 6 in Figure 4 shows the multiple clients added to system to avoid the bottleneck at client's sides. This architecture ensures reliability at clients end. In configuration 5 each DSs transmit blocks of data to corresponding clients. Clients do not communicate with any of DS, only Client1 communicate with CS. However in configuration 6 DSs transmit blocks of data to corresponding clients. And each client communicates with respected DSs from whom they receive data. However only Client 1 initiates connection closing with CS, after receiving data completion massage from all other clients.

A resource file is distributed in many different parts and stored on multiple DSs. All DSs carries different segments of the resource file, without any duplication. An advantage of distributing resource file on different DSs is that, in case some intruder is able to access data servers, he/she may not get information in its entirety.

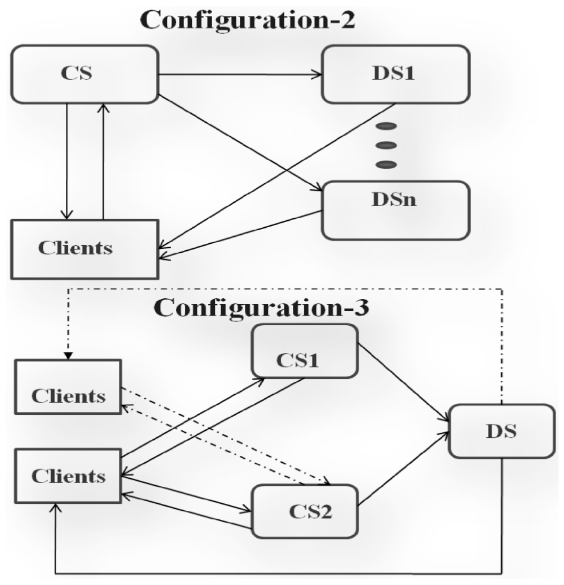

Figure 2. Split architecture (Configurations 2 and 3)

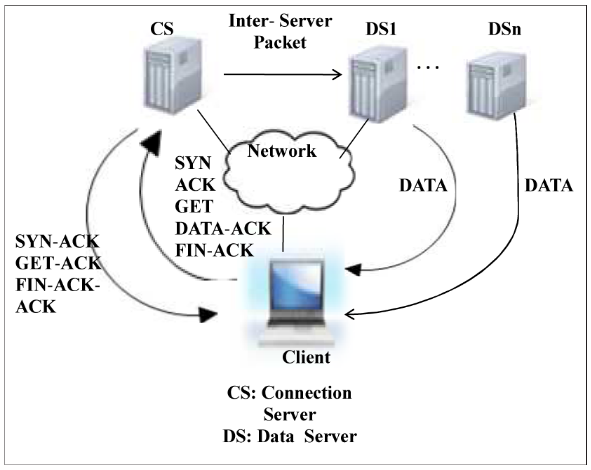

Figure 3. Split architecture (Configurations 4)

In practice, information is encrypted, and encryption keys are safely protected somewhere on other data servers. Our mechanism allows encryption keys to be stored on different DSs. In this way, one can enforce better physical information security management. In case of packet loss /server failure, we have implemented data recovery and error correction algorithm similar to RAID 2. However, one can implement any other reliability algorithm.

3.1 Experimental Setup

The experimental setup involved a prototype server cluster consisting of Dell Optiplex GX260 PCs with Intel Pentium 4, 2.8GHz Processors, 1GB RAM, and an Intel 1G NIC on the motherboard.

All systems were connected to a Linksys 16 port 1 Gbps Ethernet switch. Linux clients were used to run the http_load stress tool [20], and in addition, bare PC Web clients capable of generating 5700 requests/s was used to increase the workload beyond the http_load limit of 1000 concurrent HTTP requests/s per client. The split server cluster was also tested with Internet Explorer browsers running on Windows and Firefox browsers running on Linux.

4. Reliability Analysis

Protocol interactions are separated into two parts: connection setup and data transmission.

4.1 Analysis of Connection setup

From Figure 1, let φ denotes the probability of failure of each transaction.

$\pi = 1 - \varphi$ denotes the probability of success of each transaction.

For each transaction (component or packet), π is equivalent to probability of successful transmission (which is a probability of no bit error in the packet). Thus,

$$\pi = \binom{n}{0} p^0 (1-p)^n \qquad (1)$$

Where p denotes probability of a bit error and n denotes the number of bits in one packet.

Thus, let R denotes the probability of successful connection setup between client and server.

Then

$$R = \pi^8 \qquad (2)$$

because there are eight packets for connection setup.

4.2 Analysis of Multi-Server Configuration

This configuration represents the ordinary client-server packet transmission. A client sends a request for packet transmission to the server. During this connection setup process, if the client doesn't know IP address of a server, it will send packets to ask Domain Name Server (DNS).

4.2.1 Single Server

Assume that the client knows IP address of the server, it will initiate the connection setup directly with the server. In this case, we consider there is one data packet. Thus, the reliability of the connection setup and data transmission between client and CS is equal to

$$R = \pi^9 \qquad (3)$$

4.2.2 Multiple Servers

Assume that DNS/dispatcher (in the cluster system, DNS also works as a dispatcher) does round robin. The system will be in series with parallel system of servers as shown in Figure 5 configuration -2b and reliability of the system will be

$$R = \pi^3 (1-(1-\pi^9)^i) \qquad (4)$$

where i denoted number of servers.

4.2.3 Multi-CS and Multi-DS Configuration

Split Architecture in Figure 2 and 3 allows multiple CSs and multiple DSs in the system. During data transmission CSs and DSs work independently; however, during connection setup, DSs will depend on the successful connection setup between CSs and clients. In this analysis, the relationship during connection setup (i) CSs and DSs are viewed as a series component in the system, (ii) in the system composed of multiple CSs, all CSs are considered parallely connected components, and (iii) in the system composed of multiple DSs, all DSs are considered parallely connected components.

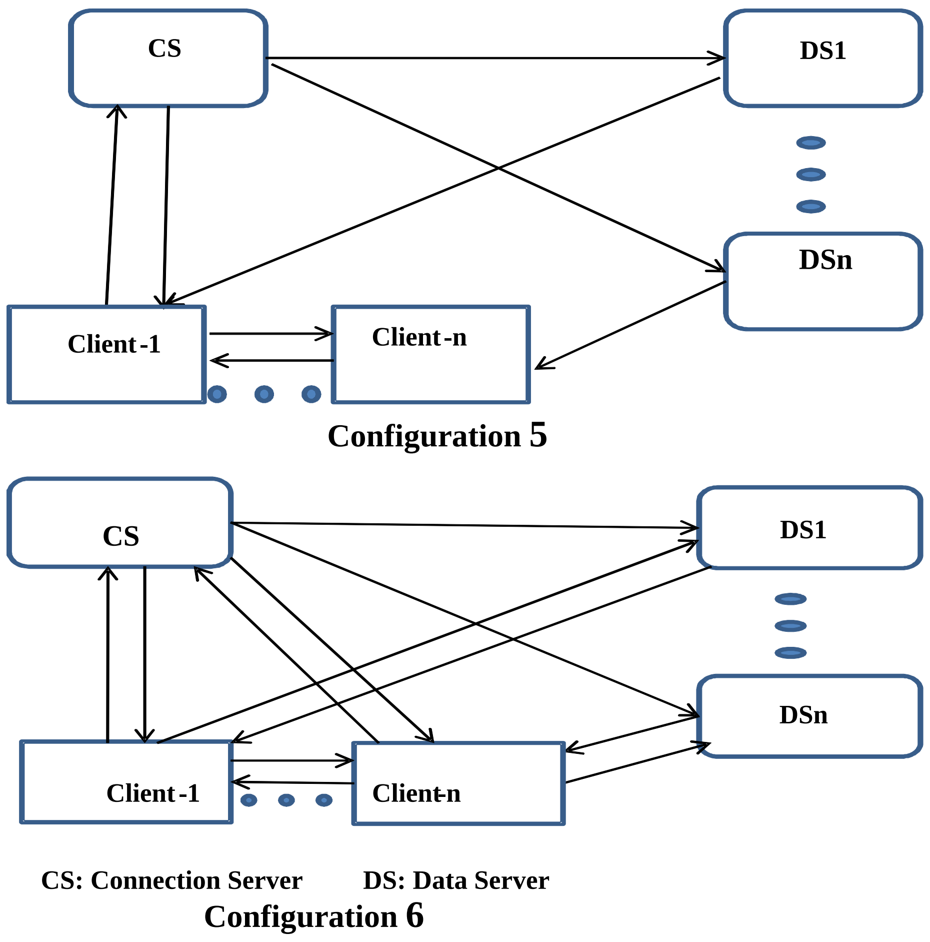

Figure 4. Split architecture (Configurations 5and 6)

Thus, there are totally eight packets for connection setup between a client and CS and two inter server packets between a CS and DS per connection.

Let R_{CS} denotes the probability that CSs work successfully. R_{DS} denotes the probability that CSs and DSs have a successful inter server packet transmission.

R denotes the probability of successful system setup or system reliability

$$R = R_{CS} R_{DS} \tag{5}$$

R_{CS} = Probability that one of CSs works

= 1 – Probability that all CSs fail

= $1 - (1 - \textit{Probability of the successful setup})^{i}$

$$R_{CS} = 1 - (1 - \pi^{10})^{i} \tag{6}$$

where i denotes number of CSs

R_{DS} = Probability that one of DSs works

= 1 – Probability that all DSs fail

= $1 - (1 - \textit{Probability each DSs work properly})^{j}$

$$R_{DS} = 1 - (1 - \pi^3)^j \quad (7)$$

where *j* denotes number of DSs

Thus

$$R = (1 - (1 - \pi^{10})^i)(1 - (1 - \pi^3)^j) \quad (8)$$

4.2.4 Analysis of varies configurations

4.2.4.1 Experimental setup with 1 CS, 1 DS with full delegation

The configuration of this case is shown in Figure 4 configuration -1a. We are using the same type of hardware configuration for CS, DS or Single Server (SS) and only differ in software architecture. All machines have the same hardware reliability and different software reliability according to their role as CS, DS or SS. Reliability of this setup will be

$$R = (1 - (1 - \pi^{10}))(1 - (1 - \pi^3)) \quad (9)$$

For simplicity, assume a http request is for a file of only one packet. Thus, DS is sending only one packet, and it is involved in only three transactions which are two inter server packets, and one data packet. The reliability of the system will be

$$R = (1 - (1 - \pi^{10}))(1 - (1 - \pi^3))$$

$$R = \pi^{10}(\pi^3) = \pi^{13} \quad (10)$$

4.2.4.2 Experimental setup with 1 CS, 2 DS with full delegation

The configuration of this case is shown in Figure 5 configuration-2a. A system will never break entirely, unless all CS and DSs fails at the same time, so configuration is the parallel configuration. Reliability of the whole system (CS and DSs) will be

$$R = (1 - (1 - \pi^{10}))(1 - (1 - \pi^3)^n)$$

$$R = (\pi^{10})(1 - (1 - \pi^3)^n) \quad (11)$$

Where *n* is the number of DSs.

4.2.4.3 Experimental setup with n CS, 1 DS with full delegation

The configuration of this case is shown in Figure 2 configuration 3. The reliability of the system will be

$$R = (1 - (1 - \pi^{10})^n)(1 - (1 - \pi^3))$$

$$R = (1 - (1 - \pi^{10})^n)(\pi^3) \quad (12)$$

4.2.4.4 Special case with n CSs and m DSs

For the special case, if the experimental setup with *n* CS, *m* DS with full delegation and CS takes part in data transmission. CS sends *x* percent of the packets, and DS send the remaining.

The reliability of the system will be

$$R = (1 - (1 - \pi^{10+x})^n)(1 - (1 - \pi^{3+(z-x)})^m) \quad (13)$$

Assume that the total number of packets in that particular connection is *z* and CS sends *x* packets.

4.3 Comparison of DNS system and split-protocol system

4.3.1 Normal Transaction

Assume the server without split protocol, as shown in Figure 5 configuration-1b, does minimum 10 transactions, a single server system reliability is

$$R = \pi^{10}$$

For the system, of two Non-Split severs requires some kind central control (scheduler/dispatcher) mechanism to forward requests between two different servers as shown in Figure 6 configuration 3-b. Let us assume for simplicity it follows the round-

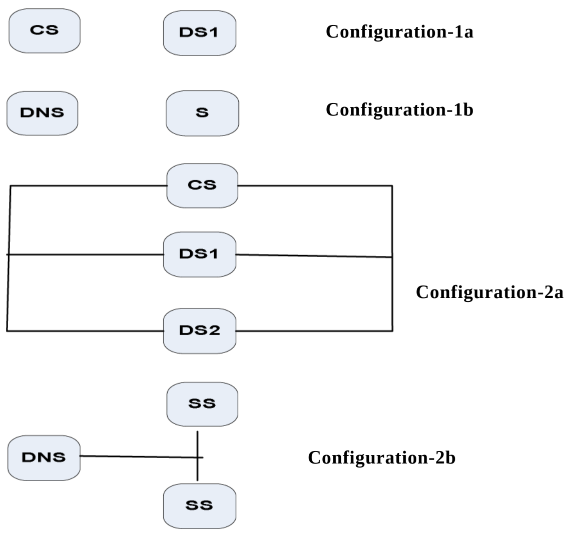

Figure 5. Architecture for two and three system components (Configuration 1and 2)

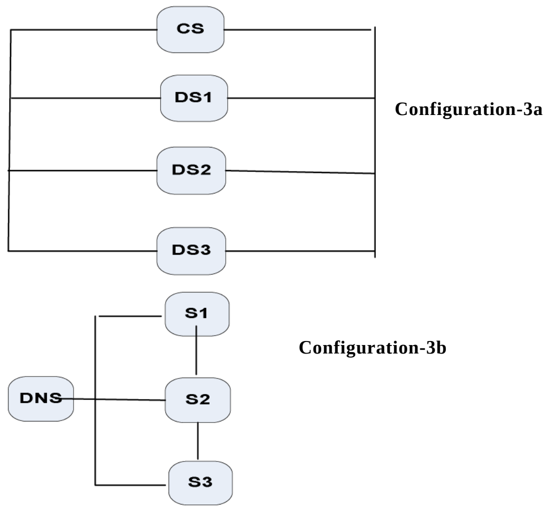

Figure 6. Architecture for four system components

robin mechanism to share requests, and it does most straightforward transactions it receives SYN and forward SYN to one of the servers, and receives ACK from SS then SS handles the rest of connection and data transfer. The reliability of the system will be the reliability of this DNS time the reliability of server.

$$R = \pi^3(\pi^{10}) = \pi^{13}$$

4.3.2 One DNS with n servers

The configuration is similar to Figure 6 configuration 3-b except the number of servers is variable. The reliability of the system is

$$R = \pi^3(1-\pi^{10})^n) \tag{14}$$

5. Reliability Comparision

The reliability comparison between DNS/dispatcher cluster and split cluster is as follows. In this scenario, only data packet is considered.

Let $R_{DNS=1, SS=i}$ denotes the reliability of one DNS and number of SSs is *i*.

Let $R_{CS=i, DS=j}$ denotes the reliability of the number of CS is *i* and as the number of DS is *j*.

We have compared two different scenarios.

Case 1: one DNS/one SS vs one CS/one DS

$$R_{DNS=1,SS=1} = (\pi^3)(\pi^{10}) = \pi^{13}$$

$$R_{CS=1,DS=1} = (\pi^3)(\pi^8) = \pi^{11}$$

$$\frac{R_{CS=1,DS=1}}{R_{DNS=1,SS=1}} = \frac{\pi^{11}}{\pi^{13}} = \pi^{-2} \quad (15)$$

Case 2: one DNS/multiple SSs vs one CS/multiple DSs

$$R_{DNS=1,SS=i} = (\pi^3)(1-(1-\pi^{10})^i)$$

$$R_{CS=i,DS=j} = (1-(1-\pi^8)^3)(1-(1-\pi^3)^j)$$

$$\frac{R_{CS=i,DS=j}}{R_{DNS=1,SS=i}} = \frac{(1-(1-\pi^8)^i)(1-(1-\pi^3)^j)}{(\pi^3)(1-(1-\pi^{10})^i)} \quad (16)$$

For instance, one CS/two DS vs one DNS/two SSs

$$\frac{R_{CS=1,DS=2}}{R_{DNS=1,SS=2}} = \frac{(1-(1-\pi^8))(1-(1-\pi^3)^2)}{(\pi^3)(1-(1-\pi^{10})^2)}$$

$$\frac{R_{CS=1,DS=2}}{R_{DNS=1,SS=2}} = \frac{(\pi^8)(1-(1-\pi^3)^2)}{(\pi^3)(1-(1-\pi^{10})^2)}$$

Reliability ratio will be

$$R_{CS=1,DS=2} = \frac{(\pi^8)(1-(1-\pi^3)^2)}{(\pi^3)(1-(1-\pi^{10})^2)} R_{DNS=1,SS=2} \quad (17)$$

π	# of DSs or SSs	Reliability ratio
0.1	2	99.94999
0.5	2	3.751832
0.9	2	0.95023

Table 1. The reliability ratio between split and DNS dispatcher system

In split system, CS can take the role of DS after the connection setup and vice versa. Thus, the reliability ratio for this case is:

$$R_{CS=1,DS=2} = \frac{\pi^8 + (1-(1-\pi^3)^{2+1})(1-(\pi^8)*(1-\pi^3)^{2+1})}{(\pi^3)(1-(1-\pi^{10})^2)} R_{DNS=1,SS=2} \quad (18)$$

π	# of DSs or SSs	Reliability ratio
0.1	2	14985053610
0.5	2	1363.384465
0.9	2	2.355400852

Table 2. The reliability ratio between split and DNS dispatcher system when the role of CS and DS are interchangeable

When CS and DS are capable of interchanging their roles, all components within the system become parallel. The parallel nature of this type of system means that all components must fail in order to failure of the system. From Table, I and II, it is quite apparent that the reliability of the split system is much higher than that seen in the DNS/dispatcher system.

If there are *n* data packets and *m* servers, the reliability ratio will be (this is a generic case)

$$R_{CS=1,\,DS=m} = \frac{\pi^8 + (1-(1-\pi^{2+n})^{m+1})\,(1-(\pi^8)*(1-\pi^{2+n})^{m+1})}{(\pi^3)\,(1-(1-\pi^{9+n})^m)}\,R_{DNS=1,\,SS=m} \tag{19}$$

6. Discussion

The role change over and migratory ability of split server [28] (CS can become DS vice versa) make split cluster configuration as the system of completely parallel components. Figure 6 shows that a system of one CS /Dispatcher and multiple DSs /SSs for a given component reliability $\pi = 0.1$ with one data packet, the reliability ratio of Split cluster and dispatcher based cluster $\frac{R(CS=1, DS=3)}{R\,(DNS=1, SS=3)} >> 1$. It indicates the reliability of the split system is comparatively very high. The reliability starts to drop linearly with increase's numbers of DSs/SSs. However, the reliability of split system is still a lot higher than the one of dispatcher system. With the addition of each DS/SS, system reliability ratio drops by 0.05%. And the reliability ratio approaches one for system of 100,000 DSs/SSs.

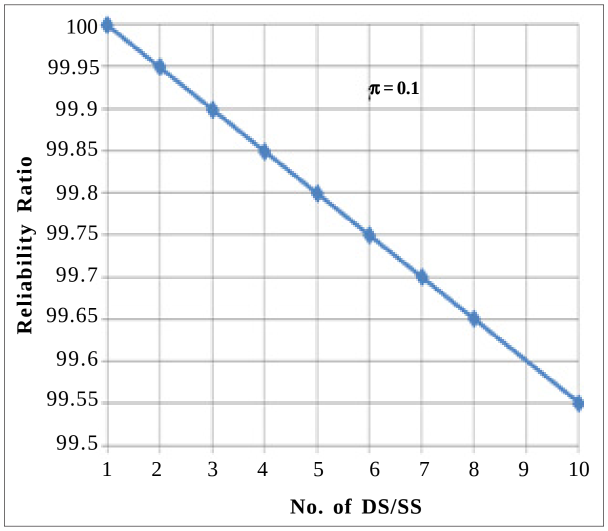

Figure 7. Reliability ratio for system of multiple DS/SSs for $\pi = 0.1$

From Figure 8, we can notice that for a system of one CS /Dispatcher and multiple DSs /SSs for given component reliability π = 0.9 with one data packet, the reliability ratio of Split cluster and dispatcher based cluster $\frac{R(CS=1, DS=3)}{R\,(DNS=1, SS=3)} >> 1$. It indicates reliability of the split system is comparatively high. As the reliability starts dropping exponentially with increase's numbers of DSs/SSs, the reliability ratio approaches less than one for the system of two DSs/SSs. For higher individual component reliability (π), split system for one data packet does not offer any advantage over the regular dispatcher cluster system.

However, if we increase the number of data packets from 7 to 297, the reliability ratio becomes greater than 1 for numbers of DS/SS less than 4. When the number of data packets greater than 297, reliability ratio starts dropping less than 1 and there is no effect on the increase's number of DSs/SSs in the system.

From Figure 9, we can notice that for a system of one CS /Dispatcher and three DSs /SSs for given component reliability π = 0.9 with multiple data packets, the reliability ratio becomes greater than one when the number of data packets are greater than 7 and continue to increase until 300 data packets. The reliability ratio approaches to the maximum value 1.24 and stays higher than one for large numbers of packets.

Figure 10 shows the reliability ratio of completely parallel split system and combination of the series and parallel component of DNS/dispatcher based system. We can notice that reliability ratio is always greater than one. This ratio is >> 1 for reliability, π of individual component < 0.4 and approaches 1 when individual component reliability becomes 1.

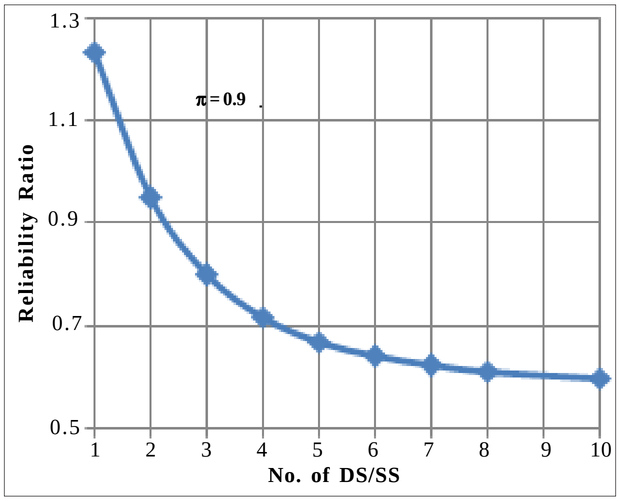

Figure 8. Reliability ratio for system of multiple DS/SSs for $\pi = 0.9$

Figure 11 shows the reliability ratio with multiple variables like numbers of packets (n), reliability of individual components (π) and numbers of DSs/SSs (j). We have noticed that the reliability ratio decreases if all other components n, π and j increases simultaneously. In real-world applications, TCP/IP protocol provides reliability by offering retransmission and various control mechanism. In our discussion, we assume protocol reliability as additional reliability offered in the system.

5.1 Failure Rate and Survival Function

From the previous section, the distribution of each protocol component is the binomial distribution. With the number protocol of component n approach infinity, distribution of each component is poison distribution [33] with arrival rate, λ which is equal to the receiving rate of each component. Thus, service time or failure rate (FR) of each component is an exponential distribution [34].

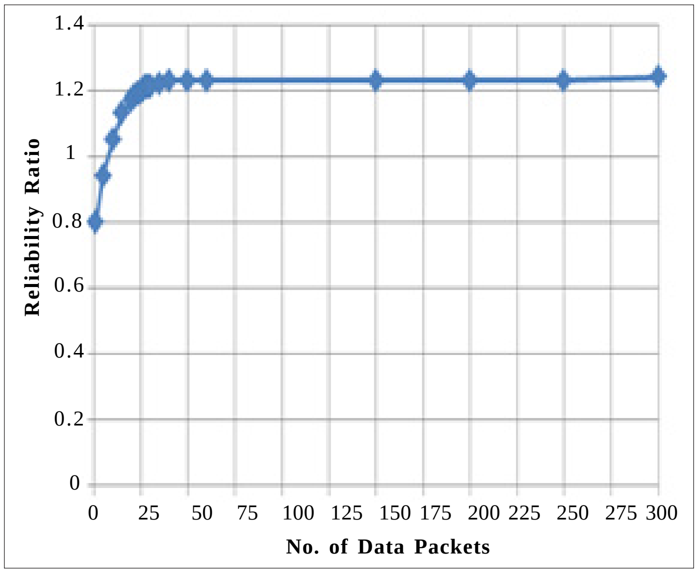

Figure 9. Reliability ratio for system of 3DS/SSs for $\pi = 0.9$

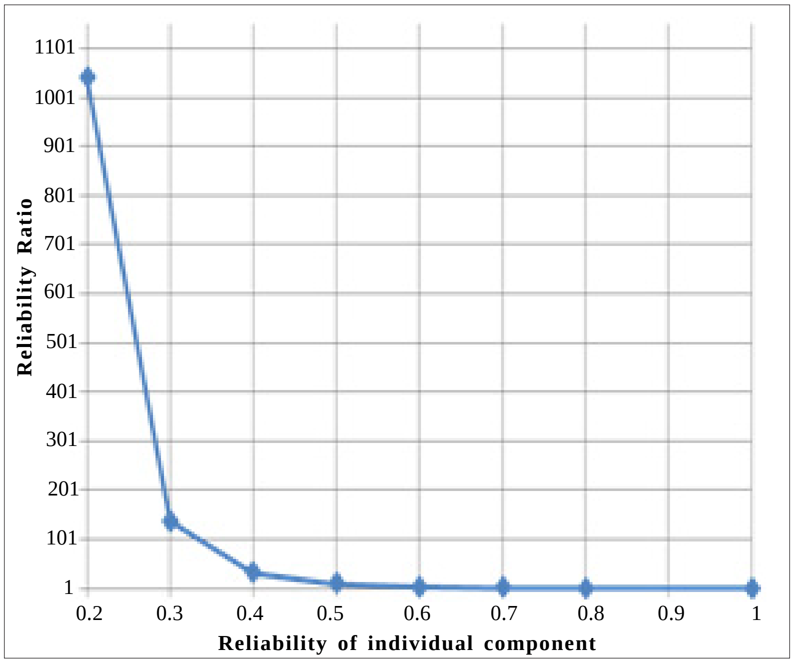

Figure 10. Reliability ratio for system of multiple DS/SSs for variable π

The system is modeled as following [33, 34]

1. All the n components service time X are Exponentially distributed:

$$F(T) = P\{X \le T\} = 1 - e - \lambda T\,;\; f(T) = \lambda e - \lambda T$$

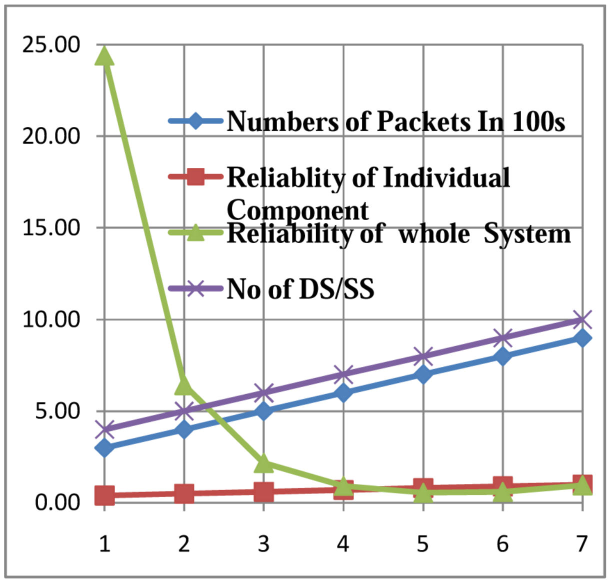

Figure 11. Reliability ratio for system variable DS/SSs, n packet and variable π

1. Each i^{th} component $1 \le i \le n$, *Failure Rate* (*FR*) *is constant.* $(,\lambda_i(t) = \lambda_i)$

2. All n components are identical. Then *FR* of each component is equal to $\lambda\,(\lambda_i = \lambda;\ 1 \le i \le n)$

3. All n components are independent. Then

$P\{X_1 \wedge X_2 \wedge ... X_n > T\} = P\{X_1 > T\}\, P\{X_2 > T\} \dots P\{X_n > T\}$

4. The reliability of each component, $R_i(T)$ is

$R_i(T) = P\{X_i > T\} = e^{-\lambda T}$

$\lambda = -\ln(R_i(T)) / T$

T denoted system mission time.

5. System failure rate is $1 - R(T)$ where $R(T)$ denoted the reliability of the whole system.

There are supposed "*n*" protocol components and probability of non-failure (of each component (*x*1, *x*2, *x*3, *x*4...) are exponentially distributed: For simplicity we will assume, every i^{th} component $1 \leq i \leq n$ probability of failure is equal for all component, i.e. Failure Rate (*FR*) for each component is same and ($\theta i\,(\tau) = \theta i$). For given operational time and all system components are identical and their failure time is independent. Therefore, the reliability of, any ith component ($1 < i < n$) reliability "$\Pi_i(\tau)$": "$\Pi_i(\tau)$" $= P\,(Xi > \tau)$ $= e^{\theta i \tau} \Rightarrow \theta_i = -\ln(\Pi_i(\tau)) / \tau$.

First, we have assumed identical components, that are identical DS in a cluster System, and they have same FR. Also, they are independent components are those whose failure does not affect the performance of any other system component.

First, we have assumed identical components, that are identical DS in a cluster system, and they have same FR. Also they are independent components are those whose failure does not affect the performance of any other system component. And Reliability of such system is the probability of a component (or system) of surviving for its mission time.

6.2 Reliability of series of identical component:

$\Pi(\tau) = P$ (System operate without failure) $= P$ (Comp1and Comp 2…and Comp n operate without failure)

If $\Pi\, 1(\tau) = \Pi\, 2\,(\tau) = \Pi\, \mathrm{n}\,(\tau)$

$= \Pi\, 1(\tau)\,.\,\Pi\, 2\,(\tau)\,.\,\Pi\, 3\,(\tau) \dots \Pi\, \mathrm{n}\,(\tau) = e^{-\theta\tau} e^{-\theta\tau} e^{-\theta\tau} \dots e^{-\theta\tau} = (e^{-\theta\tau}) \wedge n.$

$$\Pi(\tau) = [\Pi\, i\,(1)]^{n\tau} \qquad (20)$$

Reliability of series of non-identical component: That means each component has different reliability and failure rate.

$\Pi(\tau) = \Pi\, 1(\tau)\,.\,\Pi\, 2\,(\tau)\,.\,\Pi\, 3\,(\tau) \dots \Pi\, n\,(\tau)$

$= e^{-\theta 1\tau} e^{-\theta 2\tau} e^{-\theta 3\tau} \dots e^{-\theta n\tau}$

$= e^{-\tau}\, \Sigma i\, \theta i$

$= e^{-\theta \mathrm{s}\,(\tau)}$

$$\theta s = \sum_1^n \theta i \qquad (21)$$

Reliability of Parallel identical components:

$\Pi_S = 1 - (1 - \Pi_1) \times (1 - \Pi_2) \times \dots (1 - \Pi_n)$; if the component reliabilities differ

$\Pi_S = 1 - (1 - \Pi) \times (1 - \Pi) \times \dots (1 - \Pi)$; if the component with similar reliabilities

$1 - [1 - \Pi]^n$

Let us calculate system of two parallel components

$\Pi\, \tau = 1 - \{(1 - \Pi\, 1(\tau))\,(1 - \Pi\, 2(\tau))\}$

$= 1 - \{(1 - e^{-\theta 1 \tau})\,(1 - e^{-\theta 2 \tau})\}$

$= e^{-\theta 1 \tau} + e^{-\theta 2 \tau} - e^{-(\theta 1 + \theta 2)\tau}$

And MTTF $= \mu = \int_0^\infty \pi(T)\, d\tau = \int_0^\infty (e^{-\theta 1 \tau} + e^{-\theta 2 \tau} - e^{-(\theta 1 + \theta 2)\tau})\, d\tau$

$= \dfrac{1}{\theta 1} + \dfrac{1}{\theta 2} - \dfrac{1}{\theta 1 + \theta 2}$

And FR $= \theta s$ = Density Function / Survival Function $= -\dfrac{d}{dt}\, \pi(\tau) / \Pi(\tau)$

$$= (\theta 1 e^{-\theta 1\tau} + \theta 2 e^{-\theta 2\tau} - (\theta 1 + \theta 2)\, e^{-(\theta 1+\theta 2)\tau}) / (e^{-\theta 1\tau} + e^{-\theta 2\tau} - e^{-(\theta 1+\theta 2)\tau}) \qquad (22)$$

This system hazard rate $\theta s(\tau)$ can be calculated as a function of any mission time τ [34] .

9. Conclusion

In this paper, describe a simple modeling approach that consider the architecture of the software and estimate the reliability of interactions between the components, and interfaces with other components. This paper details the components of the Split – architecture, and devises reliability assessment based on individual component of software and describes how it can be used to examine software behavior. It highlights on an inbuilt reliability in a split-protocol due to the dual and interchangeable role of Connection Server (CS) and Data Server (DS). We discussed the split protocol reliability concept citing different application configuration instances where spilt is ideal. Various configurations discussed for Mini-cluster and migratory models. Model demonstrates that the split system offers very high reliability in comparison to the traditional cluster system. Split architectures offer inherent redundancy, high performance, and extremely reliable at the lower cost. The ideas presented here could also be leveraged to construct scalable migratory server clusters based bare PC and the split protocol concept.

References

[1] Palmer, Michael. (2010). MCITP Guide to Microsoft Windows Server 2008, Server Administration: Exam# 70-646. Course Technology Ptr.

[2] Gayen, Tirthankar, Misra, R. B. (2008). Reliability bounds prediction of COTS component based software application, *International Journal of Computer Science and Network Security*, 8 (12) 219-228.

[3] Fan Zhang, Xingshe Zhou, Junwen Chen, Yunwei Dong. (2008). A Novel Model for Component-based Software Reliability Analysis, *School of Computer Science and Engineering, Northwestern Polytechinical University*,11th IEEE High Assurance Systems Engineering Symposium.

[4] Gokhale, S., Trivedi, K. S. (1997). Structure-Based SoftwareReliability Prediction, *In*: Proc. Fifth Int'l Conf. Advanced Computing (ADCOMP'97), p. 447-452, Dec.

[5] Gokhale, S., Lyu, M. R., Trivedi, K. S. (1998). Reliability Simulation of Component-Based Software Systems, *In*: Proc.Ninth Int'l Symp. Software Reliability Eng. (ISSRE '98), p.192-201, Nov.

[6] Gokhale, S., Wong, W. E., Trivedi, K. S., Horgan, J. R. (2004). An Analytic Approach to Architecture-Based Software Performance and Reliability Prediction, *Performance Evaluation*, 58 (4) 391-412, Dec.

[7] Krishnamurthy, S., Mathur, A. P. (1997). On the Estimation of Reliability of a Software System Using Reliabilities of Its Components, *In*: Proc. Eighth Int'l Symp. Software ReliabilityEng, p. 146-155, Nov.

[8] Goseva-Popstojanova, K., Mathur, A. P., Trivedi, K. S. (2001). Comparison of Architecture-Based Software Reliability Models, *In*: Proc. Int'l Symp. Software Reliability Eng., p. 22-31.

[9] Goseva-Popstojanova, K., Kamavaram, S. (2004). SoftwareReliability Estimation under Uncertainty: Generalization of the Method of Moments, *In*: Proc. Eighth IEEE Int'l Symp. High Assurance Systems Eng., p. 209-218.

[10] Ammons, G., Appayoo, J., Butrico, M., Silva, D., Grove, D., Kawachiva, K., Krieger, O., Rosenburg, B., Hensbergen, E., Isniewski, R.W. (2007). Libra: A Library Operating System for a JVM in a Virtualized Execution Environment, *In*: VEE '07, Proceedings of the 3rd International Conference on Virtual Execution Environments, June.

[11] Canfora, G., Di Santo, G., Venturi, G., Zimeo, E., Zito, M. V. (2005). Migrating web application sessions in mobile computing, *In*: Proceedings of the 14th International Conference on the World Wide Web, p. 1166-1167.

[12] He, L., Karne, R. K., Wijesinha, A. L. (2008). The Design and Performance of a Bare PC Web Server, *International Journal of Computers and Their Applications, IJCA*, 15 (2) 100-112, June.

[13] He, L., Karne, R. K., Wijesinha, A. L., Emdadi, A. (2009). A Study of Bare PC Web Server Performance for Workloads with Dynamic and Static Content, The 11th IEEE International Conference on High Performance Computing and Communications (HPCC-09), Seoul, Korea, June, p. 494-499.

[14] Karne, R. K., Jaganathan, K. V., Ahmed, T. (2005). DOSC: Dispersed Operating System Computing, OOPSLA '05, 20th Annual ACM Conference on Object Oriented Programming, Systems, Languages, and Applications, Onward Track, ACM, San Diego, CA, October, p. 55-61.

[15] Karne, R. K. (2002). Application-oriented Object Architecture: A Revolutionary Approach, 6th International Conference, HPC Asia 2002 (Poster), Centre for Development of Advanced Computing, Bangalore, Karnataka, India, December.

[16] Milojicic, D. S., Douglis, F., Paindaveine, Y., Wheeler, R., Zhou, S. (2000). Process Migration, ACM Computing Surveys, 32 (3), September, p. 241-299.

[17] Zagorodnov, D., Marzullo, K., Alvisi, L., Bressourd, T. C. (2009). Practical and low overhead masking of failures of TCP-based servers, *ACM Transactions on Computer Systems*, 27 (2), Article 4, May.

[18] http://www.acme.com/software/http_load.

[19] Jiao, Y., Wang, W. (2010). Design and implementation of load balancing of a distributed-system-based Web server, 3rd International Symposium on Electronic Commerce and Security (ISECS), p. 337-342, July.

[20] Ciardo, G., Riska, A., Smirni, E. (2001). EquiLoad, A Load Balancing Policy for Clustered Web Servers. *Performance Evaluation*, 46 (2-3) 101-124.

[21] Sujit Vaidya, Kenneth J.Chritensen. A Single System Image Server Cluster using Duplicated MAC and IP Addresses, *In*: Proceedings of the 26th Annual IEEE conference on Local Computer Network (LCN'01).

[22] Rawal, B., Karne, R., Wijesinha, A. L. (2011). Splitting HTTP Requests on Two Servers, The Third International Conference on Communication Systems and Networks: COMPSNETS 2011, January, Bangalor, India.

[23] Rawal, B., Karne, R., Wijesinha, A. L. (2011). Mini Web Server Clusters for HTTP Request Split, 13th International Conference on High performance Computing and Communication, HPCC-2011, Banff, Canada, I Sept., p. 2-4, .

[24] Rawal, B., Karne, R., Wijesinha, A. L. (2012). Split Protocol Client/Server Architecture, The 17th IEEE Symposium on Computers and Communications - ISCC 2012, p. 1 - 4 July, Cappadocia, Turkey .

[25] Rawal, B., Karne, R., Wijesinha, A. L. (2012). A Split Protocol Technique for Web Server Migration, The 2012 International workshop on Core Network Architecture and protocols for Internet (ICNA-2012) October, p. 8-11, Las

[26] Sultan, K., Srinivasan, D., Iyer, D., Lftod, L. (2002). Migratory TCP: Highly Available Internet Services using Connection Migration, *In*: Proceedings of the 22nd International Conference on Distributed Computing Systems, July.

[27] Cohen, A., Rangarajan, S., Slye, H. (1999). On the performance of TCP splicing for URL-Aware redirection, *In*: Proceedings of USITS'99, The 2nd USENIX Symposium on Internet Technologies & Systems, October.

[28] Zagorodnov, D., Marzullo, K., Alvisi, L., Bressourd, T. C. (2009). Practical and low overhead masking of failures of TCP-based servers, ACM Transactions on Computer Systems, 27 (2), Article 4, May.

[29] Cizulius, W. (1982). *Private communication*.

[30] Goel, A. L. (1985). Software reliability models: assumptions, limitations, and applicability, *IEEE Trans. Software Eng*. 11 (12) 1411–1423.

[31] http://ftp.math.utah.edu/pub/tex/bib/sigmetrics.html

[32] http://www.ijcst.com/vol34/3/khasim.pdf

[33] Sheldon M. Ross, A First Course In Probability, Eighth Edition

[34] START Understanding Series and Parallel Systems Reliability, Selected Topics in Assurance Related Technologies, 11 (5).