Availability Measurement and Modeling for An Application Server

Dong Tang, Dileep Kumar, Sreeram Duvur, Oystein Torbjornsen
Sun Microsystems, Inc.
4150 Network Circle, Santa Clara, CA 95054
Email: dong.tang{dileep.kumar, sreeram.duvur, oystein.torbjornsen}@sun.com

Abstract

Application Server is a standard middleware platform for deploying web-based business applications which typically require the underlying platform to deliver high system availability and to minimize loss of transactions. This paper presents a measurement-based availability modeling and analysis for a fault tolerant Application Server system – Sun Java System Application Server, Enterprise Edition 7. The study applies hierarchical Markov reward modeling techniques on the target software system. The model parameters are conservatively estimated from lab or field measurements. The uncertainty analysis method is used on the model to obtain average system availability and confidence intervals by randomly sampling from possible ranges of parameters that cannot be accurately measured in limited time frames or may vary widely in customer sites. As demonstrated in this paper, the combined use of lab measurement, analytical modeling, and uncertainty analysis is a useful evaluation approach which can provide a conservative availability assessment at stated confidence levels for a new software product.

1. Introduction

Application Server has become an important category of general-purpose middleware for deploying web-based business applications such as on-line banking, stock trading, merchandise purchasing and auction services. The system availability supported by Application Server is a critical metric for evaluating this category of software. Although recent research effort has been made in benchmarking dependability for OLTP systems [20] and evaluating user-perceived availability for specific web-based applications [6], there has been no published study to demonstrate how to evaluate availability for the Application Server middleware using current modeling and analysis techniques, combined with measurement.

In this paper, we present an availability evaluation study, which combines measurement, modeling, and statistical analysis techniques, for a particular Application Server – Sun Java System Application Server, Enterprise Edition 7 (JSAS EE7). The model parameters are conservatively estimated based on data collected from lab or field measurements. For parameters that cannot be accurately measured in limited time frames or may vary widely in customer sites, we apply the uncertainty analysis method on the model by randomly sampling these parameters in wide ranges to obtain average system availability and confidence intervals.

Previous studies [7, 10] have shown that in many cases, it is possible to use a combination of measurement and mathematical models to derive a quantitative dependability assessment for the target software system. Hsueh [4] was the first study to use Markov reward models, combined with operational failure data, to model an operating system (IBM/MVS). Later, similar techniques were applied to the Tandem Guardian and VAX/VMS operating systems [9]. The methodology was further extended from using operational data to using test data in evaluating availability for air traffic control software [16]. All of these studies rely on failure data in estimating parameters plugged into the models which can be solved to generate system availability or performability measures. Methods to estimate parameters and associated confidence levels, including situations in which failure was not observed during the measurement period, have been addressed in [8].

Markov reward model is one of the most powerful and widely accepted mathematical models in availability and reliability analysis [3, 19]. The state space based model structure and the reward rate associated with each state in Markov reward models provide capabilities to evaluate availability, performability, service cost, and various metrics of interest for the modeled system. However, in modeling real systems, the number of states in the model often exceeds the range that can be handled manually and introduces the state explosion problem. To simplify model specification, stochastic Petri nets have been used to construct models and solutions are supported in well-known tools [2, 14]. To reduce model complexity, the hierarchical modeling approach has been proposed to decompose a complex model into multiple submodels and implemented in modern modeling tools [13, 18].

The software tool used in this analysis is RAScad [17, 18], a Sun internal web-based Reliability, Availability, and Serviceability (RAS) architecture modeling tool for use in system design and development phases. RAScad has been heavily used in designing new Sun hardware products. It has also been used to develop availability models for Sun Cluster software systems [12]. The model and analysis presented in this paper were developed using the RAScad hierarchical Markov modeling and uncertainty analysis capabilities. The system metrics of interest are availability, the associated yearly downtime, and mean time between system failures.

The rest of the paper is organized as follow. Section 2 briefly introduces architecture for the target system. Section 3 describes test environment and measurements. Sections 4 and 5 discuss assumptions and parameters used in the model. Section 6 presents the model. Section 7 analyzes results obtained from the model and conducts uncertainty analysis. Section 8 concludes this study.

2. System Architecture

Figure 1 shows a general configuration for JSAS EE7 [15]. A Java 2 Enterprise Edition (J2EE) technology based web services deployment model typically consists of three tiers: web server, application server, and database. The web server tier, including load balancers, communicates with the Internet and distributes incoming requests to application server instances, performing a reverse proxy function. The web server tier may also be used to serve static content such as images and dynamic content that need not be secure. The web server tier does not retain memory of prior requests and is thus stateless. An application server specific Load Balancer Plugin (LBP) is installed on the web server. This plugin is aware of the state of application servers and performs user application session load balancing and proxy functions. Load balancing decisions are recorded as HTTP cookies, keeping the web server stateless.

The application server tier, including its data persistence support, processes user's requests in a stateful context and potentially carry out single- or multi-step business transactions. A transaction is typically a series of dependent business logic functions that communicate with the database tier. The database tier contains various business data and users' information and is not necessarily restricted to one database instance, as user application transactions may span multiple databases. The database tier is not necessarily always a relational database. It can be any transactional data repository or even a message broker that can store or transmit application data reliably. To focus on the application server tier, we have omitted showing the database tier, even though it is required and used by our test applications.

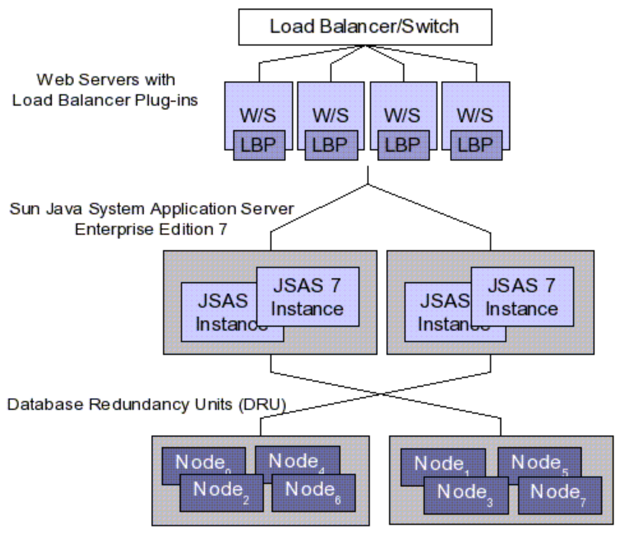
Load Balancer/Switch
Web Servers with Load Balancer Plug-ins
W/S
LBP
Sun Java System Application Server Enterprise Edition 7
JSAS Instance
JSAS 7 Instance
Database Redundancy Units (DRU)
Node0
Node4
Node2
Node6
Node1
Node5
Node3
Node7

Figure 1. General Configuration of JSAS EE7

Target System Modeled

As shown in the figure, multiple Application Server (AS) instances are organized as a cluster to serve user requests from load balancers. The session conversational state of AS instances is written to a highly available database (HADB), developed based on the "Always-On" availability technolog [11]. Multiple pairs of HADB nodes are organized as Database Redundancy Units (DRU) to provide redundancy. When an AS instance fails, the LBP plugin in web servers can detect the failure and forward subsequent requests to another AS instance which can then restore the last saved conversational state from HADB. The AS instances and HADB pairs constitute the target system to be analyzed in this paper.

The target system provides automatic recovery and self-repair features, including HTTP session failover, which is necessary for tolerating faults in a stateful environment. For example, an insurance quote or loan application typically requires multiple screens of user input. A failure at any step of process results in loss of state information, and a complete transaction loss which requires the transaction to restart from the beginning. JSAS EE7 supports persistence of HTTP session data in its bundled HADB which is available to all instances in a cluster. In the event of an instance failure, session data can be recovered by other instances in the cluster. In addition to unplanned events, this capability, along with the HADB self-repair capability described below, also provides for minimal impact on planned maintenance.

HADB is a highly scalable and available distributed database. It consists of two mirrored DRUs which are logical groupings of nodes. Each DRU contains the complete set of session data evenly distributed over multiple nodes, ensuring optimal throughput and response time. A node is a collection of processes, a dedicated area of main memory, and some physical disk space. Two different nodes normally do not share a physical host. A node may be active, providing data access and update, or spare, ready to take over for the failure of an active node. Nodes are grouped and provide a mutual watchdog service in the group.

For each node in a DRU, there is a mirrored node in the other DRU. Should a node fail, the failed node attempts to restart itself and and to recover data from the companion node. If the restart procedure is successful, the recovered node will return the system back to its pre-failure configuration. If the restart attemp is not successful, the companion node initiates a repair procedure on a spare node by reconstructing data on the spare. Completion of this repair procedure will convert the spare node to an active node. The failed node will become a spare node after a physical repair action is fulfilled.

3. Measurements

To assess the stability of the target software system, multiple longevity tests were performed by running representative application benchmarks on the following configuration: Two Application Server instances running on two 4-CPU Sun E450 systems and two pairs of HADB nodes running on four Sun Ultra 80 systems. Table 1 shows the test configuration and their logical relationship between layers. System load was generated through a commercial workload generator running on a Microsoft Windows machine. A load balancer was included in the test configuration to perform sticky round-robin load balancing between multiple server instances. In these tests, the systems were utilized at a load factor of 60-70% and multiple 7-day duration runs were performed. Roughly seven million requests were processed by the system in each run.

Load Balancers	
AS Instance 1 J2EE Web App/Nile Bookstore	AS Instance 2 J2EE Web App/Nile Bookstore
HADB Pair 1 (2 Nodes)	HADB Pair 2 (2 Nodes)
Oracle Database & Sun Java System Directory Server	
Solaris™ 9 OS running on Sun Enterprise 450 server	

Table 1. Test Environment

The stability test was conducted on two large applications. The first application is a sophisticated, real-world J2EE Web Application benchmark for running digital marketplaces. It includes Catalog, Auction, Pricing, and Order Management modules and is used in many customer deployments. The application uses pooled JDBC technology to access an Oracle database and the Java LDAP Software Developer Kit (SDK) to access the Sun Java System Directory Server. The average session size is 50KB, which is larger than the typical size of HTTP sessions.

The second test application is the Nile Bookstore benchmark, which uses the JSAS EE7 connection pooling capabilities to access an Oracle database. The average session size is roughly 30KB. This application is a complete, end-to-end, e-commerce application server benchmark that has been widely used by independent testing laboratories. Both applications were stable during the course of the test. Application redeployment or server restart was not necessary. One of the tests – which was continued for 24 days for sanity checking and availability demonstration purposes – survived a system reboot due to a recoverable hardware problem.

To assess fault tolerance, a number of manual fault injection tests were performed to ensure that system can tolerate single faults, and behaves as expected. Some of the faults injected are listed below:

- HADB node is brought down by killing all related processes
- HADB node communication is disrupted by unplugging network cable
- HADB node hardware power is unplugged
- Application Server node is brought down by killing processes
- Application Server node host network cable is unplugged
- Application Server node host power is unplugged

For all the fault injection tests listed above, the system continued functioning without any major departure from the expected performance. In addition to these manual fault injection tests, automated fault injections were conducted extensively on the HADB system. Some of the faults injected are listed below:

- Simultaneously kill all processes in a node to simulate a full node failure
- Randomly kill one of the processes to simulate software bugs
- Ask processes to terminate immediately to simulate fast fail scenarios

Both single-node and multi-node (not in a pair) failures were induced in the fault injections. The workloads were fluctuated from idle to fully loaded states, in combination with rare conditions such as repair and data reorganization modes, during the fault injection process. For over 3,000 fault injections covering a variety of failure scenarios, all recoveries were successful.

During these fault injection tests, measurements were made to determine AS/HADB node recovery/restart times under different failure scenarios. These fault injection tests, along with the stability tests described previously, provided data for estimating various temporal and the imperfect recovery parameters used in the model. Details are discussed later in Section 5.

4. Model Assumptions

Although the test environment is a fixed configuration which consists of only 2 AS instances and 2 HADB node pairs, the modeled configurations are not limited to the test environment. Specifically, two common configurations are repeatedly used in this analysis:

- **Config 1**: 2 AS instances, 2 HADB node pairs and 2 HADB spare nodes
- **Config 2**: 4 AS instances, 4 HADB node pairs and 2 HADB spare nodes

The modeled system is considered *available*, if at least one AS instance is up and able to service requests and the system is able to persist session state. Translating this definition to the model specification, we require at least one AS instance and at least one node in each HADB node pair to be in the working state. The requirement for the HADB node pairs is based on the fact that each AS instance is potentially using all the HADB node pairs, as the data table is fragmented across all node pairs due to data partitioning.

Both hardware permanent faults and hardware/ software transient faults occurring on all computer systems (nodes) in the configuration are modeled. When a permanent HW failure occurs, that node has to be shut down for repair. The result of a HW failure on an AS node is that the affected server instance is put out of use until repairs are completed. For HW failures on an HADB node, the surviving companion node initiates repair by invoking a spare node. When a transient fault occurs due to hardware or software problems on a node, there are two possible results: (1) Restart of the applications without a system reboot or (2) reboot of operating system and cold restart of all processes. The first event is an *HADB failure* if it occurs on an HADB node or *AS failure* if it occurs on an AS node. The second event is an *OS failure*.

It is assumed that failure/repair processes on different computer systems are independent. The implication of this assumption for the Application Server nodes is that the failure and restart of an AS instance will not introduce a failure on another AS instance. This is because the interaction between the surviving AS instances and the recovering AS instance is minimal and imperfect recovery will have no effect on the surviving node except that recovery time may be increased. However, when an HADB node pair is in the recovery mode, the surviving companion node is actively participating the recovery process by transferring data updated during the outage to the recovering node, and thus the possible common mode failure due to imperfect recovery should be modeled.

It is also assumed that the failure rate is constant (exponential distribution) for the purpose of steady-state analysis. In the uncertainty analysis discussed later, all the failure rates in given ranges are changed to quantify the impact of different failure rates on availability. In addition, the failure rate/workload dependency is modeled in the following way. After a failure of HADB node or AS instance, the failure rate on the remaining HADB node or AS instances is doubled to reflect the increased failure risk due to increased load. Let La_0 denote the base failure rate for an AS instance. The AS failure rate after the *i*th instance has failed, $La\_i = La\_0 \times 2^i$. This is based on the observation that the risk of software failure increases exponentially with increasing workload [5].

There are several other aspects of a physical deployment that are beyond the scope of the current model. To simplify our study, failures of the following elements are not included in the model: The web server tier, the database tier, network communications, electrical power and air conditioning. It should be emphasized that human error, which is not considered in the model, could be critical to system availability. Historical data from many sources suggest that human error accounts for roughly 50% of all outages in production server environments during the past three decades [1]. Although redundancy provisions in the target system could tolerate a human error induced failure on an AS instance or an HADB node, the product is not designed to prevent catastrophic failure from human error introduced during on-line maintenance when redundancy may become temporarily unavailable.

Online upgrades to the applications, Application Server or the underlying operating system and hardware, can be orchestrated by the administrator, using single or dual cluster deployments. This model is restrict to simple one cluster deployments. As such, only four scheduled maintenance events are modeled for the HADB nodes. It is possible to extend this hierarchical model to include more events and subsystems, but the focus of this effort is the Application Server and associated HADB.

5. Model Parameters

The following parameters are used in the model. Lambda (or La) represents a mean failure rate, and T represents a mean recovery time. These parameters are estimated conservatively from test or field data. For example, many recovery times are longer than those measured and OS and HW failure rates are higher than those observed in the field.

HADB Node Parameters

- Node failure rate (all failures) = 4/year
  - HADB (restartable) failure rate: La_hadb = 2/year
  - OS failure rate: La_os = 1/year
  - HW failure rate: La_hw = 1/year
- Restart Time
  - Restart time for HADB failure: Tstart_short = 1 min.
  - Restart time for OS failure: Tstart_long = 15 min.
  - Repair time for HW failure: Trepair = 30 min.
- Fraction of Imperfect Recovery: FIR = 0.1%
- Maintenance event
  - Maintenance rate: La_mnt = 4/year
  - Maintenance switchover time: Tmnt = 1 min.
- Restore time: Trestore = 1 hour

AS Instance Parameters

- AS instance failure rate: La = 52/year
  - AS (restartable) failure rate: La_as = 50/year
  - OS failure rate: La_os = 1/year
  - HW failure rate: La_hw = 1/year
- Recovery time: Trecovery = 5 sec.
- Restart time
  - Restart time for AS failure: Tstart_short = 90 sec.
  - Restart time for OS/HW failure: Tstart_long = 1 hour
    - Outage time for HW failure = 100 min.
    - Outage time for OS failure = 15 min.
- Restore time: Tstart_all = 30 minutes

HADB Restart/Repair Time. When an HADB node suffers a failure, the node automatically tries to restart. The restart may be successful (for HADB and OS failures) or not successful (for HW failure). In the first case, the time from the detection of the failure to completion of the restart is the *restart time*. Although the measured restart time under an HADB failure is only around 40 seconds, a more conservative value of 1 minute is used in the model. The restart time for the OS failure is assumed to be 15 minutes. In the second case, an automatic repair process is initiated. All the session data in the companion node are copied to a spare node to make it the new mirroring node. The time taken to complete this procedure is called *repair time* which depends on the amount of data stored on the node. It is estimated that the size of data on an HADB node is within 1GB which can support up to 10,000 concurrent sessions with the average size of 50KB on each AS instance. Measurements show it takes about 12 minutes to copy 1GB data from one node to another. The model sets this parameter to 30 minutes to accommodate possible variance on difference configurations.

Fraction of Imperfect Recovery. The HADB automatic recovery (restart or repair) process may not be perfect, due to various unexpected situations such as defects in the fault handling software running on the companion node and latent faults on the disk activated during the data reconstruction. Thus, there is a small chance that the companion node could also fail during the recovery process, resulting in system failure. Although the occurrence probability of such an event is very low, we use a parameter, Fraction of Imperfect Recovery (FIR), to model the event. Based on the fact that imperfect recovery was not observed for over 3287 fault injections covering various failure scenarios, FIR is estimated to be below 0.1% (default value in the model) at the 95% confidence level and below 0.2% (upper bound in the uncertainty analysis) at the 99.5% confidence level, using the following statistical function for estimating lower bounds for the coverage parameter, C = 1–FIR, a binomial random variable [8]:

$$C_{low} = \frac{1}{1 + \frac{n-s+1}{s} F_{1-\alpha;2(n-s)+2;2s}} \qquad (1)$$

where n is the total number of trails, s the number of successful trials, F the F distribution function, and α the significance level (1 – confidence level).

HADB Restore Time. When both nodes in an HADB node pair are down (a rare event), the session data supported by the pair is irretrievably lost, resulting in a catastrophic failure. A human intervention has to be invoked to recreate a functional HADB node pair and system. The time to complete this procedure is *restore time* which includes "time to notice failure" plus the HADB recreation time. For 7x24 on-site maintenance, the restore time is estimated to be less than one hour. During this time, the system is not available.

Session Recovery Time. When an AS instance experiences a failure, the user requests originally serviced by this instance are forwarded to other working instances. Average response time for a user request after this failover is increased due to the time spent reestablishing the session on another instance. This time increment is defined as *session recovery time*. Although the session recovery time was measured at the sub-second levels, to be conservative, it is conservatively set to 5 seconds in the model.

AS Restart Time. When an AS instance suffers a failure, it is automatically restarted on the same computer if no system reboot is required for the recovery (result of an AS failure). The time to complete this procedure is *short restart time*. The measured short restart time is less than 25 seconds. Taking into account the time for a load balancer to notice the recovery of the failed AS node (time interval between two health condition checks is 1 minute), the parameter is set to 90 seconds. In an OS failure case, it requires a system reboot (15 minutes). In a HW failure case, it requires a physical repair action (100 minutes, based on field data). So the average restart time for the HW/OS failures (once a year for each) is approximately one hour, which is called *long restart time*.

AS Restore Time. When all AS instances are down (a rare event), human intervention is required to restart them. The time to complete this procedure is *restore time* which includes "time to notice failure" plus actual time to restart all AS instances. For the 7x24 on-site maintenance, the average AS restart time is estimated to be 30 minutes. During this time, the system is not available.

AS Failure Rate. The failure rate on an AS node is conservatively set to once a week (including HW and OS failures), based on the fact that the duration for most test runs is one week. This rate is higher than upper bounds estimated based on the longest duration test. Given that no failure was observed during a 24-day test for two AS instances, a failure rate upper bound is estimated to be 1/16 days at the 95% confidence level and to be 1/9 days at the 99.5% confidence level, using the following statistical function [8]:

$$\lambda_{upp} = \frac{X^2_{1-\alpha;2n+2}}{2T} \qquad (2)$$

where T is the total execution time, n the number of failures observed, X^2 the Chi-square distribution function, and α the significance level (1 – confidence level).

6. Model Structure

The target software system is modeled by a hierarchy of three Markov diagrams. The first diagram (Figure 2) models the overall system, as a 3-state Markov model. Each state has a number associated with it. This is number is called *reward rate*. A reward rate of 1 means the state is a working state. A reward rate of 0 means the state is a failure state. The notation of these states is listed below:

- **Ok**: At least one node in each HADB node pairs is functioning properly and at least one AS instance is functioning properly. Working state.
- **AS_Fail**: All AS instances have failed. Failure state.
- **HADB_Fail**: At least one pair of HADB nodes have a double node failure. Failure state.

The system is normally working in the Ok state. It goes into state AS_Fail at the rate La_appl and comes back at the rate Mu_appl, where La_appl and Mu_appl are the failure rate and recovery rate evaluated from the subsystem model "Appl Server". The system also goes into state HADB_Fail at the rate N_pair*La_hadb and comes back at the rate Mu_hadb, where La_hadb and Mu_hadb are the failure rate and recovery rate evaluated from the subsystem model "HADB Node Pair" and N_pair is the number of HADB node pairs in the system.

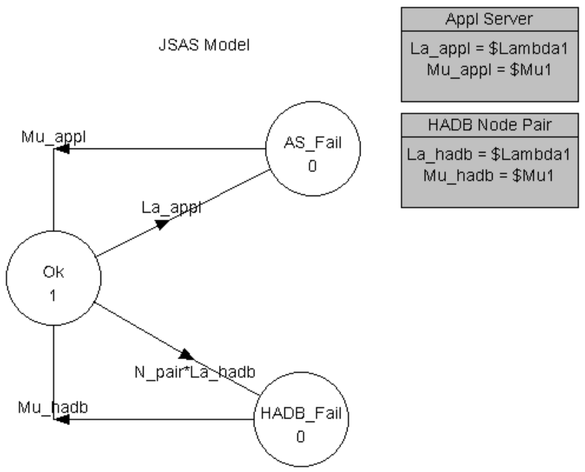

Figure 2. JSAS System Model

Figure 3 shows the HADB Node Pair subdiagram. The state notation used in this diagram is:

- **Ok**: Both nodes are functioning. Working state.
- **RestartShort**: A node is being restarted from an HADB failure. Working state.
- **RestartLong**: A node is being restarted from an OS failure. Working state.
- **Repair**: A spare node is being rebuilt to replace a node with HW failure. Working state.
- **Maintenance**: A node to be serviced is switching over to a standby node. Working state.
- **2_Down**: Both nodes are down. Failure state.

The transitions and associated rates at which the system moves between states are clearly shown in the diagram. When a failure occurs, with a probability of 1-FIR (successful recovery), the system goes into a recovery state (RestartShort, RestartLong, or Repair), depending on the type of failure. The system also goes into the failure state with probability FIR (unsuccessful recovery) which is a coefficient of the overall failure rate La. When the system is in a recovery or the Maintenance state, only one node is functioning. As stated previously, the failure rate on the remaining node is assumed to be accelerated by a factor of 2. A second failure on this node results in data loss and system failure. Should a system failure occur, it would take one hour (at the rate of 1/Trestore) for the system to restore to its working state.

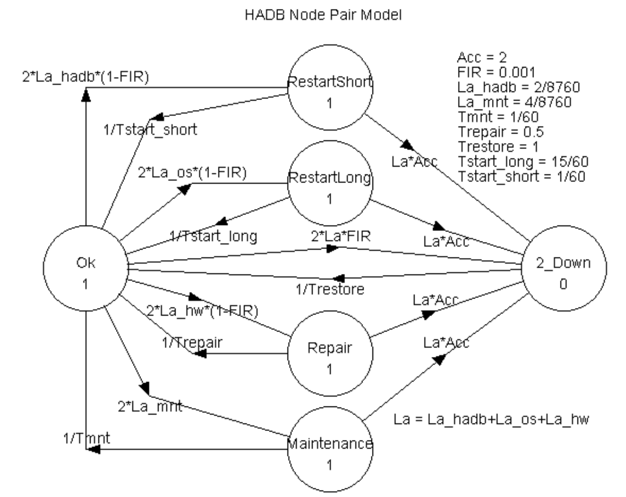

Figure 3. HADB Node Pair Model

Figure 4 shows the Appl Server Model (2 instances) subdiagram. The notation used for the states in the diagram is listed below:

- **All_Work**: All instances are functioning. Working state.
- **Recovery**: After an AS failure, sessions originally running on the failed AS are being reestablished on the remaining AS instance. Working state. It could be a degraded state in performability modeling.
- **1DownShort**: An instance is being restarted from an AS failure. Working state.
- **1DownLong**: An instance is being restarted from an HW or OS failure. Working state.
- **2_Down**: Both instances are down. Failure state.

The transitions and associated rates at which the system moves between states are clearly shown in the diagram. The overall failure rate (La) for an AS instance is the sum of AS failure rate (La_as), HW failure rate (La_hw), and OS failure rate (La_os). When an instance fails, the system goes into the Recovery state and stays there for a short time interval Trecovery. Then with the probability of FSS (fraction of short start = La_as/La), it will go into state 1DownShort and stay there for a period of Tstart_short, or with the probability of 1-FSS, it will go into state 1DownLong and stay there for a period of Tstart_long, before going back to the normal state.

When the system is in the states Recovery, 1DownShort, and 1DownLong, should the second failure (with an accelerated rate) occur on the remaining instance, the system goes into the failure state 2_Down and stays there for Tstart_all (0.5 hour) to go back to the normal state. But this is not the case for the configuration of 4 AS instances (Config 2), because such a configuration is able to tolerate up to three instance failures. The 4-instance Application Server model is more complex and not discussed in detail in this paper.

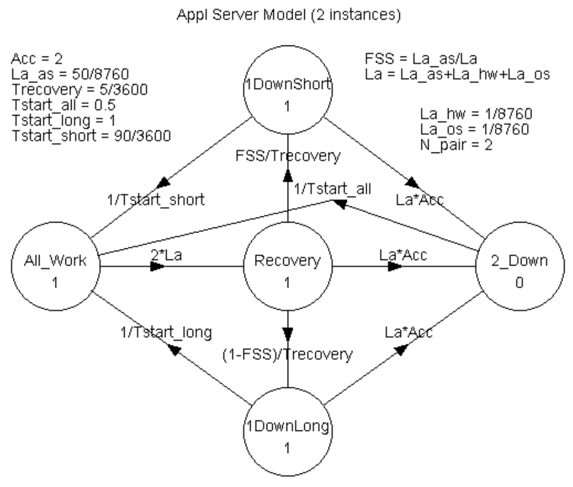

Figure 4. Application Server (2 instances) Model

7. Analysis of Results

Given the models and parameters presented in previous sections, system results for the two modeled configurations generated by RAScad are shown in Table 2. For both Config 1 and Config 2, the system availability is above five 9's. In Config 1, yearly downtime is dominated by the AS submodel (67%) while in Config 2, yearly downtime is dominated by the HADB submodel. When the number of AS instances is 4 or above, the AS submodel's yearly downtime is at the millisecond level and can be ignored in availability analysis.

Configura-tion	*Availability*	*Yearly Downtime (YD)*	*YD due to AS Submodel*	*YD due to HADB Submodel*
Config 1	99.99933%	3.5 min.	2.35 min. (67%)	1.15 min. (33%)
Config 2	99.99956%	2.3 min.	0.01 sec. (<0.01%)	2.3 min. (99.99%)

Table 2. System Results

The model parameters can be classified into three categories: failure rate, recovery rate, and coverage (1-FIR) parameters. Failure rates cannot be accurately measured in limited time frames through testing and may vary on different customer sites, depending on the configuration, workloads, and environmental factors. All the failure rates used in the model are varied in the uncertainty analysis discussed later. Fraction of Imperfect Recovery (FIR) has been estimated to be below 0.1% at the 95% confidence level, based on the fault injection data. In the uncertainty analysis, it is allowed to go up to 0.2% which is above the 99.5% confidence level upper bound.

Most recovery times (automatic restart time, repair time, etc.) are deterministic and are measured in the lab testing. An exception is the recovery time for hardware/OS failure on an AS node. This parameter is, to a large extent, controllable by customers. A customer can minimize the hardware/OS failure recovery time by implementing high quality maintenance procedures and deploying a standby AS node. The RAScad parametric analysis capability is used to investigate how this parameter can impact availability. Figures 5 and 6 show the analysis results on the AS node HW/OS failure recovery time (Tstart_long varies from 0.5 to 3 hours). When the HW/OS failure recovery time increases to 2.5 hours, the five 9's availability is no longer retained for Config 1. However, even if the parameter increases to 3 hours, the 99.9995% availability is still retained for Config 2.

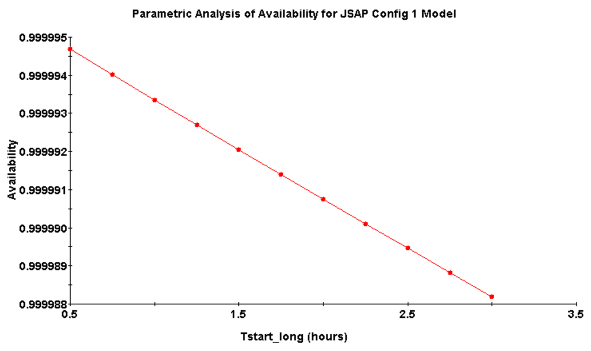

Figure 5. Sensitivity of Availability to HW/OS Failure Recovery Time on AS node for Config 1

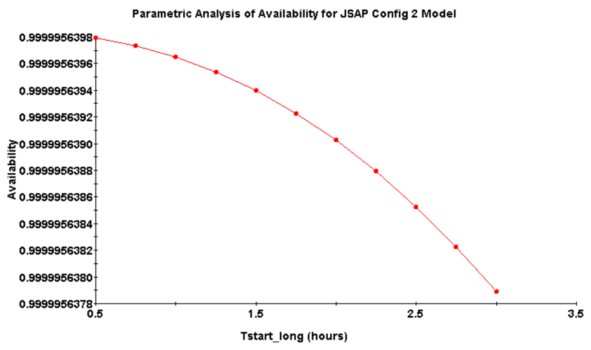

Figure 6. Sensitivity of Availability to HW/OS Failure Recovery Time on AS node for Config 2

One of the advanced analysis capabilities of RAScad is the uncertainty analysis which performs random sampling from parameter ranges defined by the user. This analysis method can address questions such as: Assume we have N systems with each system's parameters selected by randomly sampling from possible ranges in customer sites, what is the average system availability and confidence intervals? In the uncertainty analysis for this study, we select six parameters which either cannot be accurately measured in limited time frames through testing, or may vary on different customer sites. The selected parameters and their varying ranges are:

- AS failure rate La_as: 10/year – 50/year
- HADB failure rate La_hadb: 1/year – 4/year
- OS failure rate La_os: 0.5/year – 2/year
- HW failure rate La_hw: 0.5/year – 2/year
- AS HW/OS failure recovery time Tstart_long: 0.5 – 3 hours
- Fraction of imperfect recovery FIR: 0 – 0.2%

The uncertainty analysis results for Config 1 and Config 2, generated for a sample size of 1,000, are shown in Figures 7 and 8, respectively. For Config 1, the average yearly downtime for 1,000 systems is 3.8 minutes and the 80% confidence interval is (1.9 min., 6.0 min.). Over 80% of sampled systems have yearly downtime less than 5.25 minutes, or above the 99.999% availability level. For Config 2, the average yearly downtime is 3 minutes and the 80% confidence interval is (1.0 min., 5.2 min.). Over 90% of the sampled systems have yearly downtime less than 5.25 minutes, or above the 99.999% availability level.

Finally, we compare system availability for different configurations listed in Table 3. In the table, one instance means there is no failover mechanisms and the server can be restarted in 90 seconds for AS failures and in 1 hour for HW/OS failures. The following observations are obtained from the results:

# of Instances	*# of HADB Pair*	*Availability*	*Yearly Downtime*	*MTBF (hr.)*
1	N/A	99.9629%	195 min.	168
2	2	99.99933%	3.49 min.	89,980
4	4	99.99956%	2.29 min.	229,326
6	6	99.99934%	3.44 min.	152,889
8	8	99.99912%	4.58 min.	114,669
10	10	99.99891%	5.73 min.	91,736

Table 3. Comparison of Configurations

- Availability is significantly improved from a 1-instance configuration to a 2-instance configuration. That is, the redundancy and failover provisions in JSAS EE7 are able to enhance system availability by two 9's.

- When the number of AS instances increases to 4, availability is dominated by HADB. While increasing the number of HADB node pairs provides better scalability, it also introduces more chances to lose fragments of data, resulting in system failures. The 99.999% availability level can no longer hold when the number of HADB node pairs reaches 10.
- The configuration with 4 AS instances and 4 HADB node pairs is the optimal configuration in terms of availability.

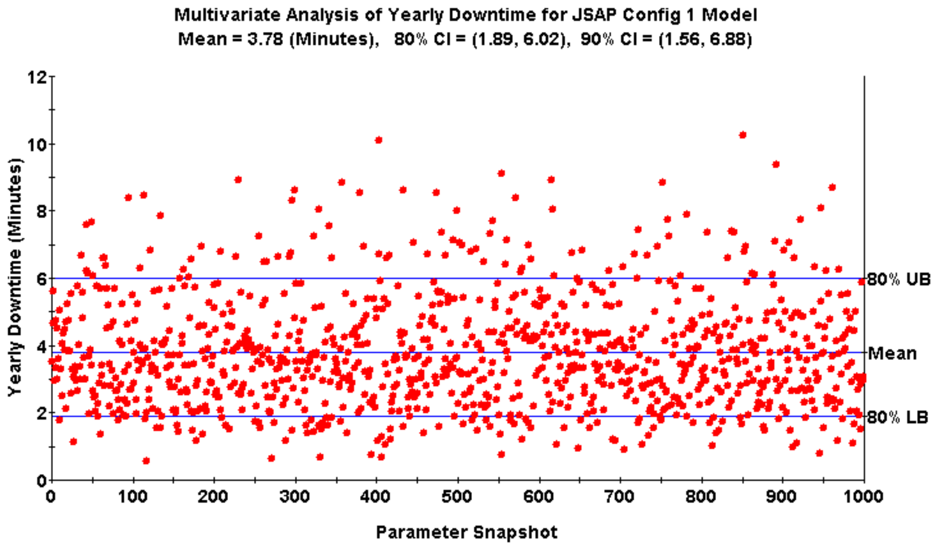

Figure 7. Uncertainty Analysis Results for Config 1

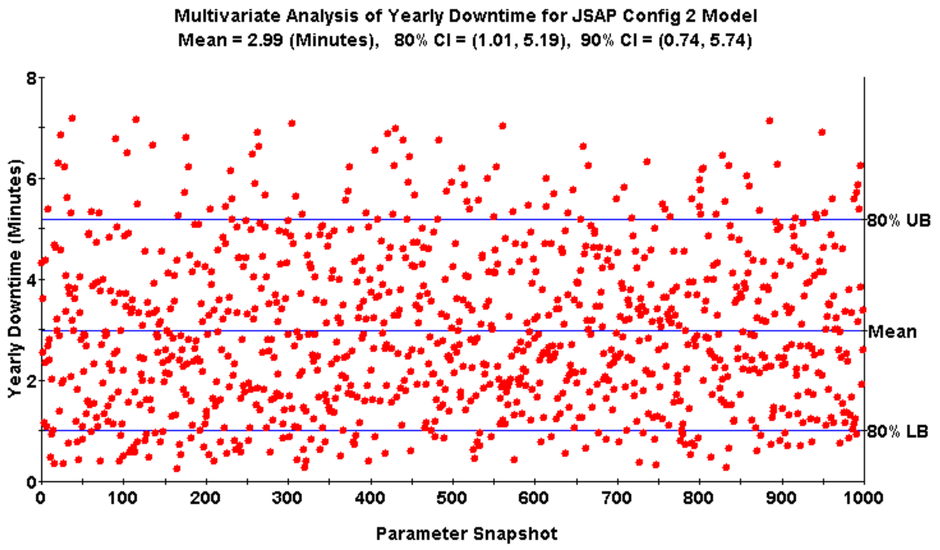

Figure 8. Uncertainty Analysis Results for Config 2

8. Conclusions

In this paper, we demonstrated a measurement-based availability evaluation approach for a new middleware platform – Sun Java System Application Server, Enterprise Edition 7. Both hardware permanent faults and software transient faults, as well as workload dependent failure rates, were considered in the model. Extensive lab measurements were conducted to obtain data for estimating model parameters. The model was developed based on the widely accepted methodology using RAScad. Under conservative assumptions used in building the model and estimating model parameters from test and field data, the average system availability was evaluated to be at the 99.999% level in the Solaris and Sun server environment. Uncertainty analysis on wide ranges of input parameters not accurately measurable in limited time frames or likely variable in customer sites were applied on the model to obtain availability confidence intervals. The analysis also showed that the configuration of four AS instances and four pairs of HADB nodes is the optimal configuration in terms of availability. These results could be useful in planning data centers and web services deployments.

Acknowledgments

The authors would like to thank William Shannon, Richard Sharples, David Van Couvering, Lawrence White, and Masood Mortazavi for their valuable comments on this paper. Thanks are also due to Kenneth Chan and Suveen Nadipalli for participating the early model development and providing the JSAS EE7 test data. Special thanks go to William Franklin, Timothy Cramer, Madhu Konda and Roy Andrada for their support for this work and constructive input to the model.

References

[1] A. Brown and D. A. Patterson, "To Err is Human," *Proceedings of the First Workshop on Evaluating and Architecting System dependability* (EASY 01), Göteborg, Sweden, July 2001.

[2] G. Ciardo, J. Muppala, and K. S. Trivedi, "SPNP: Stochastic Petri Net Package," *International Conference on Petri Nets and Performance Models*, 1989.

[3] A. Goyal, S. S. Lavenberg and K. S. Trivedi, "Probabilistic Modeling of Computer System Availability," *Annals of Operations Research*, No. 8, March 1987, pp. 285-306.

[4] M. C. Hsueh and R. K. Iyer, "Performability Modeling Based on Real Data: A Case Study," *IEEE Transactions on Computers*, April 1988, pp. 478-484.

[5] R. K. Iyer and D.J. Rossetti, "Effect of System Workload on Operating System Reliability: A Study on IBM 3081," *IEEE Transactions on Software Engineering*, Dec. 1985, pp. 1438-1448.

[6] M. Kaaniche, K. Kanoun, and M. Martinello, "A User-Perceived Availability Evaluation of a Web Based Travel Agency," *Proceedings of the International Conference on Dependable Systems and Networks* (DSN-2003), San Francisco, June 2003.

[7] K. Kanoun, M. Kaaniche and J. C. Laprie, "Qualitative and Quantitative Reliability Assessment," *IEEE Software*, March/April 1997, pp. 77-87.

[8] D. Kececioglu, *Reliability and Life Testing Handbook*, Vol. 1 & 2, PTR Prentice Hall, Englewood Cliffs, NJ, 1993.

[9] I. Lee, D. Tang, R. K. Iyer and M. C. Hsueh, "Measurement-Based Evaluation of Operating System Fault Tolerance," *IEEE Transactions on Reliability*, June 1993, pp. 238-249.

[10] M. R. Lyu, Editor, *Handbook of Software Reliability Engineering*, McGraw-Hill, New York, 1996.

[11] Oystein Torbjornsen. *Multi-Site Declustering Strategies for Very High Database Service Availability,* Dr. Ing. Thesis, Division of Computer Science and Telematics, The Norwegian Institute of Technology, University of Trondheim, Norway, 1995.

[12] I. Pramanick, "Modeling Sun Cluster Availability," *Sun Users Performance Group Conference, SUPerG-2002*, San Francisco, 2002.

[13] R. A. Sahner and K. S. Trivedi, "Reliability Modeling Using SHARPE," *IEEE Transactions on Reliability*, Feb. 1987, pp. 186-193.

[14] W. H. Sanders, W. D. Obal II, M. A. Qureshi and F. K. Widjanarko, "The UltraSAN Modeling Environment," *Performance Evaluation*, Oct./Nov. 1995, pp. 89-115.

[15] Sun Microsystems, *Sun Java System Application Server, Enterprise Edition 7*, White Paper, August 2003, available at http://wwws.sun.com/software/products/appsrvr_ee/home_appsrvr_ee.html

[16] D. Tang and M. Hecht, "Evaluation of Software Dependability Based on Stability Test Data," *Proceedings of the 25th International Symposium on Fault-Tolerant Computing* (FTCS-25), June 1995, pp. 434-443.

[17] D. Tang, J. Zhu, and R. Andrada, "Automatic Generation of Availability Models in RAScad," *Proceedings of the International Conference on Dependable Systems and Networks* (DSN-2002), June 2002, pp. 488-492.

[18] D. Tang and K. S. Trivedi, "Hierarchical Evaluation of Interval Availability in RAScad," *Proceedings of the International Conference on Dependable Systems and Networks* (DSN-2004), Florence, Italy, June 2004.

[19] K. S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, Second Edition, John Wiley & Sons, Inc., New York, 2002.

[20] M. Vieira and H. Madeira, "Benchmarking the Dependability of Different OLTP Systems," *Proceedings of the International Conference on Dependable Systems and Networks* (DSN-2003), San Francisco, June 2003.