Proceedings of the 25th Annual International Symposium on Fault-Tolerant Computing
Doubletree Hotel, Pasadena, California, USA, June 27-30, 1995.

Evaluation of Software Dependability Based on Stability Test Data

Dong Tang and Myron Hecht
SoHaR Incorporated
Beverly Hills, CA 90211-3204

Abstract

This paper discusses a measurement-based approach to dependability evaluation of fault-tolerant, real-time software systems based on failure data collected from stability tests of an air traffic control system under development. Several dependability analysis techniques are illustrated with the data: parameter estimation, availability modeling of software from the task level, applications of the parameter estimation and model evaluation in assessing availability, identifying key problem areas, and predicting required test duration for achieving desired levels of availability, and quantification of relationships between software size, the number of faults, and failure rate for a software unit. Although most discussion is focused on a typical subsystem, Sector Suite, the discussed methodology is applicable to other subsystems and the system. The study demonstrates a promising approach to measuring and assessing software availability during the development phase, which has been increasingly demanded by the project management of developing large, critical systems.

1. Introduction

As hardware dependability has been greatly improved in fault-tolerant systems, software is becoming a major concern of system dependability [11]. In the context of software dependability, two issues must be addressed: achieving high dependability and assessing the achieved dependability. It is generally believed that with a reasonable process, we can develop highly dependable software that fails very infrequently. Several systematic approaches have been used to achieve high dependability, such as formal methods, fault tolerance techniques, testing and debugging, and quality management. However, similar systematic approaches to software dependability assessment do not exist. There are currently no well-developed techniques to enable this assessment to be carried out in a scientific fashion [9], although a great deal of research has addressed the area.

In the area of software dependability evaluation, research has been active in several aspects, including software reliability growth models, software dependability models, and dependability analysis of operational software. Software reliability growth models have been studied for nearly 20 years [21]. Many different models were proposed (reviewed in [12] and [22]) to characterize reliability growth for the software under development. Some of the recommended models are the Schneidewind model, the generalized exponential model, the Musa/Okumoto Logarithmic Poisson model, and the Littlewood/Verrall model [1]. Dependability modeling of operational software systems has also been addressed in the literature. In [17], an approximate model was derived to account for failures due to software faults. The study discussed in [2] proposed approaches to evaluating fault tolerant systems using recovery blocks [23] or N-version programming [4].

Measurement-based dependability analysis of operational software has a history of over 10 years [16]. Early studies investigated relationship between workload and software reliability [7] [15] and hardware-related software errors [14]. Recent studies of manufacturer-collected software fault/failure data (from multiple installations) for an AT&T switching system [20], an IBM operating system [25], and a Tandem operating system [19] investigated a variety of dependability issues such as defect/failure categorization, defect detection and removal, fault symptoms, the impact of faults on availability, and the effect of fault tolerance embedded in the system. A study that is most related to the work discussed in this paper is the Markov reward modeling of three major operating systems (IBM/MVS, VAX/VMS, and Tandem/GUARDIAN) based on field data to evaluate operational performability [18]. Similar approaches have not been used in the software development phase in which realistic availability evaluation is increasingly demanded by the project management of developing large, critical systems.

In this paper, we demonstrate a measurement-based approach to dependability evaluation of fault-tolerant, real-time software systems based on failure data collected from stability tests of an Air Traffic Control (ATC) system. A stability test is to exercise the target system with representative input data on a large configuration continuously for a number of hours (typically 72 to 120 hours). Although the

duration of the test is not very long, due to the large configuration in which multiple copies of a software task are processing different input data, the accumulative execution time for the task can be as long as thousands of hours. The objectives of a stability test are to determine effects of long term operations on the system, to collect metrics for characterizing the stability of the system, and to identify risk areas. In the development phase of such systems, the performance of the system in stability tests is the best approximation of system behaviors in the operational phase. Thus, analysis of data collected from the tests can provide the best assessments of system dependability at this phase. The authors would like to emphasize that the data presented here do not indicate the operational reliability and availability of the discussed system which are expected to be much higher than the results given in this paper.

The remainder of the paper is organized as follows: Section 2 introduces the tested system and data. Section 3 estimates model parameters from data. Section 4 develops software availability models from the task level. Section 5 performs availability evaluation for several purposes and validates the evaluated results against data. Section 6 identifies two other dependability characteristics from the data: TBF distribution and correlations between code size, number of faults, and failure rate for a software unit. Section 7 summarizes this study.

2. The tested system and data

The tested ATC system is a segment of the Advanced Automation System (AAS) being developed by the Federal Aviation Administration (FAA) [13]. The ATC system consists of a distributed network of workstations and advanced common consoles. The system communicates with the outside data sources, the Host Computer System (HCS) and the Enhanced Direct Access Radar Channel (EDARC), to gather ATC data and to display the data on the common consoles. Most of the workstations are used to support common consoles, and such a workstation is considered to be a part of a common console. Other workstations play various roles such as network monitoring and control, data reduction and analysis, display recording and playback, and interface with external data sources (HCS and EDARC). The air traffic controller in charge of a sector of airspace uses a set of common consoles to manage the air traffic in the sector. The set of common consoles is called a *sector suite*. A complete description of the AAS architecture can be found in [8].

The ATC system can be classified into the application software and the system support software (a layer between the operating system and applications). The basic unit of the application software is called *operational unit* (OU). The basic unit of the system support software is called *Functional Group* (FG). Each OU or FG consists of a number of identical processes (typically 2-4), called *Address Spaces* (AS's), running on separate processors of a group (e.g., a sector suite). An address space is an independently dispatchable software element. Two or more AS's are grouped as an OU or FG to perform a task.

In an OU, one AS is the Primary Address Space (PAS), and the other AS's are standby address spaces (SAS's). The PAS performs primary processing for the OU and periodically sends state data changes to the SAS's. The SASs keep track of the PAS state and remain ready to become the PAS when necessary. In case the PAS or its resident processor fails, one of the SAS's will take over, based on the most recently saved state data, to continue the processing. This fault tolerant scheme is similar to the "process pairs" introduced by Tandem [5]. In an FG, all AS's are actively running and providing services for other applications (OUs) on the same processor. If an FG fails on a processor, all applications on that processor have to switch to another processor in the same group. The primary distinction of an OU from an FG is the need to preserve state data and to maintain consistency among multiple AS copies in the OU.

The data were collected from two stability tests of the ATC system, referred to as *checkpoint 3* and *checkpoint*

Table 1 A sample of extracted event log for Checkpoint 3 test

Time	OU/FG	Processor	Group	Rank	PTR	Comments
00:53:00.000	DRP	64	125	RNK1	132530	PAS failure, covered.
03:50:41.282	FDM	31	5	RNK1	133351	OU failure; AS copy 1 failed.
03:50:41.351	FDM	32	5	RNK2	133351	OU failure; AS copy 2 failed.
03:50:41.351	FDM	33	5	RNK3	133351	OU failure; AS copy 3 failed.
03:50:41.352	FDM	34	5	RNK4	133351	OU failure; AS copy 4 failed.
04:04:17.257	CIP	31	5	RNK1	133362	PAS failure, covered.
04:04:53.261	BS	31	5	RNK1	133362	PAS failure, covered.
04:07:35.260	HFC	32	5	RNK3	133362	SAS failure.
04:07:47.333	CCP	31	5	FAIL	128828	FG failure, processor down.
04:14:49.340	---	31	5	INIT	------	Processor initialized.

4. The two tests were separated by about five months (September 1993 and February 1994). The checkpoint 3 test had a configuration of 69 processors and a test duration of 33.6 hours. During the test, 92 AS/OU failures occurred. The checkpoint 4 test had a configuration of 87 processors and a test duration of 78.3 hours. during the test, 25 AS/OU failures occurred. Both failure event logs and Problem Trouble Reports (PTRs) were available for this analysis. The event log provide failure information on the OU/FG level. Table 1 shows the first 10 reduced entries in the Checkpoint 3 test event log plus our comments.

In the following three sections, the discussion will be focused on the software units running in a sector suite. Table 2 lists these software units and their functionality. These OUs and FGs coordinate to provide services to the air traffic control functions required by a sector suite. Further detailed discussing of these software units is out of the range of this paper. Some of the listed software units are also used in other types of group. The discussed methodology is applicable to all other software units and subsystems [26].

Table 2. Software units running in a sector suite

Name	Type	Nature	Functionality
BS	FG	System Support	Basic Service
CCP/CCS	FG	System Support	Common Console Processor/System Software
CIP	FG	System Support	Controller Interface Processing
EFC	OU	Application	EDARC Format Conversion
FDM	OU	Application	Flight Data Management
HFC	OU	Application	HCS Format Conversion

3. Estimation of parameters

The data provide information for estimating parameters (failure rate, recovery rate, and coverage) which are required in the software availability models developed on the task level, to be discussed in the next section. This section discusses methods used in the parameter estimation. For some parameters which cannot be estimated from the data, an explicit assumption is made based on the design requirements or on experience.

3.1. Failure rate

There are two possible situations for a software unit that has gone through a test: 1) at least one failure has occurred during the test, and 2) no failure has occurred during the test. For either situation, an upper bound of failure rate for the software unit can be estimated with a certain level of confidence, which is often desired in modeling critical systems. In the following discussion, we assume that the random variable TBF (Time Between Failures) for a software unit under the test is exponentially distributed with a mean of $1/\lambda$ (λ is the failure rate of the software unit). This assumption is partially justified by a result discussed in Section 6 that the aggregate TBF of all the ATC software units under the test fits a hyperexponential function. This is because a hyperexponential function can be decomposed into a number of parallel exponential functions which may represent the TBF distributions of individual software units [27].

If n failures have been observed during the accumulative test period T (operating time), then the sample mean

$$\overline{X} = \frac{T}{n} \tag{1}$$

is an estimate of $1/\lambda$. It can be shown that $2\lambda T$ has approximately a chi-square distribution with $2n$ degrees of freedom [27]. Thus, we can find a number, $\chi^2_{1-\alpha;2n}$, from the chi-square distribution table such that

$$P(2\lambda T < \chi^2_{1-\alpha;2n}) = 1 - \alpha \ . \tag{2}$$

That is, with a 100(1-α)% confidence, we can say that

$$\lambda_0 = \frac{\chi^2_{1-\alpha;2n}}{2T} \tag{3}$$

is an upper bound of λ.

In the situation in which no failure has been observed in T, we make hypotheses:

H_0: $\lambda > \lambda_0$
H_1: $\lambda \le \lambda_0$

Where λ_0 is an upper bound of λ. If H_0 holds, the probability that TBF is less than or equal to T should satisfy

$$P(TBF \le T) = 1 - e^{-\lambda T} > 1 - e^{-\lambda_0 T} \ , \tag{4}$$

making the right part of Equation (4) equal to $1 - \alpha$,

$$1 - e^{-\lambda_0 T} = 1 - \alpha \ . \tag{5}$$

Since we have not seen a single TBF instance within T, we reject H_0 at the significance level of α. Solving the above equation for λ_0, we obtain

$$\lambda_0 = \frac{-ln(\alpha)}{T} \ , \tag{6}$$

which is a conservative upper bound of λ at the $100(1-\alpha)$% confidence level.

Equations (3) and (6) can be used to estimate failure rate upper bounds for software units based on failure data collected from either field operations or duration tests

during which workloads running on the system are representative. Table 3 shows the estimation process and results for the OUs/FGs running in a sector suite based on the Checkpoint 3 data. In the table, the column of "OU/FG" is the name of the tested software unit, the column of "No. of Copies" is the number of software copies who were actively running in the test (the standby AS's were not actively running so they are not counted), and the "Operating Time" is the product of the number of copies and the test duration.

Table 3. Failure rate estimation for selected OUs/FGs from Checkpoint 3 data

OU/FG	No. of Copies	Operating Time (hr.)	No. of Failures	Failure Rate Upper Bound
BS	43	1444.8	10	8.67×10^{-3}
CCP/CCS	43	1444.8	22	1.79×10^{-2}
CIP	43	1444.8	15	1.25×10^{-2}
EFC	12	403.2	4	1.36×10^{-2}
FDM	12	403.2	11	3.39×10^{-2}
HFC	12	403.2	5	1.67×10^{-2}

3.2. Coverage

For each OU/FG, there are two kinds of failures: covered failure and non-covered failure. A covered failure is a single AS failure and the switchover to a standby AS in the same OU/FG, following the failure of the primary AS, is successful. This is the case for most failures. A non-covered failure is a failure of the entire OU/FG including both primary and standby AS's. In such cases, either all AS's in the same OU/FG fail simultaneously, or after the failure of the primary AS, all standby AS's also fail in a small time window. If the data showed both covered and non-covered failures for an OU/FG, the percentage of the covered failures was used to estimate the mean failure recovery coverage (or simply *coverage*) for the OU/FG.

For many OUs/FGs, there were no non-covered failures. In order to estimate coverage for these software units, we first give a lower bound for the coverage. Let E denote the event that n failures have been observed in the test and none of them is a non-covered failure. Assume that E is likely to fall into the probability space $[\alpha, 1]$, i.e.,

$$\alpha \le P(E) \le 1 , \tag{7}$$

where α is the significance level. Let C be the coverage for the software unit. Then,

$$P(E) = C^n . \tag{8}$$

The minimum C, denoted by C_0, that satisfies Equation (7) can be found by solving

$$C_0{}^n = \alpha . \tag{9}$$

The result is

$$C_0 = \alpha^{\frac{1}{n}} , \tag{10}$$

which is a lower bound of C at the significance level α.

When n is not large enough (below 30), the estimated C_0 is less than 0.95 which is usually the default value used in dependability modeling at the design phase. In this case, the estimated coverage low bound may be too conservative, because it is far from the observed mean value — 1.0. Previous research has shown that system dependability is very sensitive to the coverage parameter [3] [10]. To avoid that the evaluated availability is over biased by this conservative estimate, if C_0 is greater than 0.95, C_0 is used; otherwise, 0.95 is used. (Although in the early test phase, the estimated low bound is normally less than 0.95, as the number of covered failures accumulates, it is possible for the lower bound to reach a high value.)

Table 4 shows coverage parameters for the OUs/FGs running in a sector suite, estimated from the Checkpoint 3 data. All assumed coverages are higher than the lower bounds at the 80% confidence level for this test. The overall coverage is 0.9, which indicates that most software errors in the tested system are state and timing related and can be covered by activating standby copies. This observation is consistent with a recent result reported in [19] that Tandem systems (using a similar fault tolerance approach) tolerate nearly 80% of all recorded software faults.

Table 4. Coverage estimates for selected OUs/FGs from Checkpoint 3 data

OU/FG	Total No. of Failures	No. of Non-covered Failures	80% Confidence Lower Bound	Coverage Estimate
BS	10	0	0.85	0.95
CCP/CCS	22	2		0.91
CIP	15	0	0.90	0.95
EFC	4	0	0.67	0.95
FDM	11	4		0.64
HFC	5	1		0.80
Total	67	7		0.90

3.3. Recovery time

The Checkpoint 3 data contained information on the time when a specific processor fails and when it reinitializes. All of these processor failures were caused by CCS/CCP failures. Thus, the interval between a processor

failure and its re-initialization can be used to estimate the recovery time for CCS/CCP. The average was measured to be about 7.5 minutes. The mean recovery times for other OUs/FGs were much faster and were not measurable from the data. They were assumed to be 10 seconds — the design requirement for the software recovery time.

4. Availability modeling

In this section, availability models are developed for the ATC software running in a sector suite (group). In a sector suite, there are two types of software units: the application software (OUs) and the system support software (FGs). To evaluate availability for each type of software and to reduce complexity, the overall sector suite model is decomposed into two submodels: the application model and the system support model, each of which is a Markov chain. The sector suite availability is approximately the product of the results evaluated from the two submodels.

Figure 1 shows the application model. The modeled software units are three application OUs: EFC, FDM, and HFC. All of these OUs are considered necessary for the modeled group to be functioning properly. Each OU is assumed to consist of a PAS and a single SAS. The notation used in the model is explained as follows:

N — normal state in which all OUs are functioning properly

F — failure state in which the group is unable to provide required service

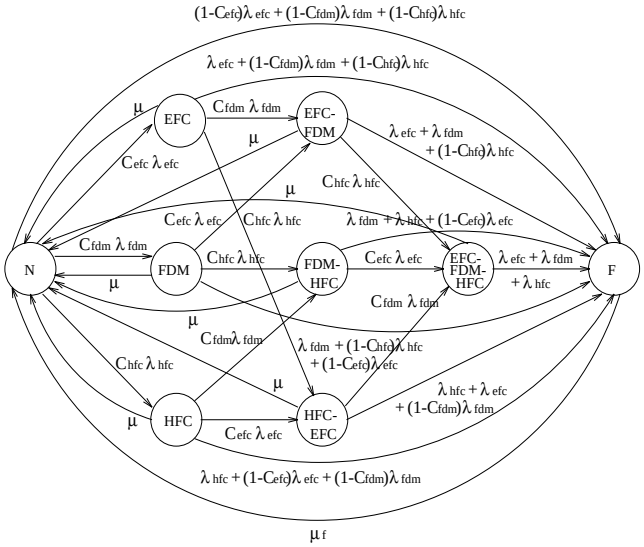

Fig. 1. The sector suite application model

x — state in which x is recovering from a PAS failure

x_1-x_2 — state in which x_1 and x_2 are recovering from PAS failures

x_1-x_2-x_3 — state in which x_1, x_2, and x_3 are recovering from PAS failures

λ_x — failure rate of x's PAS

C_x — coverage for x

μ — recovery rate for a PAS failure

μ_f — recovery rate for an OU failure

where x (and x_1 to x_3) can be EFC, FDM, or HFC.

As shown in the figure, failure dependency between AS's inside an OU is modeled by coverage C. For each OU, if the PAS fails, the SAS will take over successfully with a probability of C. If the SAS cannot take over successfully, the OU will fail, which leads to a failure of the modeled group. In the successful case, if the new PAS fails during the recovery from the failure of the previous PAS, then the OU is considered to be failed and so is the group. The model assumes failure independence between OUs. This is basically supported by the data. No simultaneous OU failures were found from the data. In the Checkpoint 3 data, there were cases in which two types of AS's failed very closely in time (in the order of seconds). But they have no impact on results because these individual failures do not contribute to unavailability.

Figure 2 shows the system support model. The modeled software units are three system support FGs: BS, CCP/CCS, and CIP. All of these FGs are considered necessary for the modeled group to be functioning properly. The modeled group is assumed to consist of three processors, each running the same set of FGs. For simplicity, the

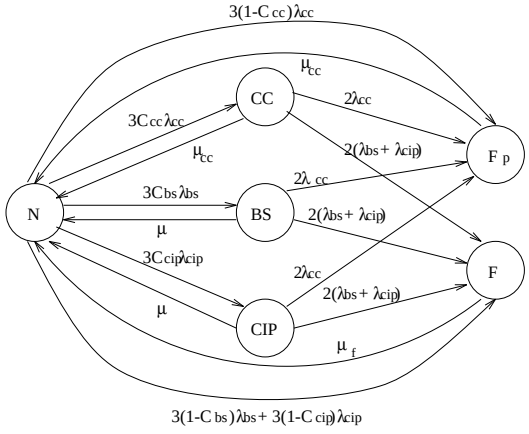

Fig. 2. The sector suite system support model

model assumes a loss of two processors is a group failure. This is supported by the data: Whenever the second processor failed, the third processor always failed. The model construction is similar to that of the application model. Differences are: 1) For each FG, since three AS's are actively running, the failure rate out of the normal state (N) is $3\lambda_x$ where x is the name of the FG. 2) The recovery process of CCP/CCS involves a processor reboot which is much longer than the recovery process of other FGs (minutes vs. seconds). Thus, a separate recovery rate for this FG, μ_{cc}, is used to distinguish it from others, and a separate group failure state, F_p, is added to characterize the processor failure.

The above models are developed for the software running in a sector suite. Similar models have also been developed for other software subsystems in the ATC system. Because failures in different subsystems are relatively independent, the overall ATC software availability can be estimated from the subsystem-level availabilities using a combinatorial model.

5. Model evaluation and validation

The models developed in Section 4 and the parameters estimated in Section 3 allow us to evaluate availability (and other dependability measures) for the software in a sector suite. In this section, availability is evaluated from the models with different parameters to serve several purposes. The model validation issue is also addressed.

5.1. Availability evaluation

Availability evaluation is performed for three purposes: 1) estimation of current availability, 2) quantification of possible availability improvement if key problems reported in a few PTRs are resolved, and 3) prediction of the possible range of availability lower bounds that can be reached through stability tests with reasonable test durations. The solution tool used in the evaluation is SHARPE [24].

Current availability

Table 5 lists software unavailability upper bounds evaluated based on the 80% confidence level failure rate upper bounds (Section 3), for Checkpoints 3 and 4. In Checkpoint 3, the system support software dominates the group availability. In Checkpoint 4, both submodels contribute to the group unavailability equally, after the availability of the system support software has been greatly improved. The overall sector suite availability is improved by an order of magnitude from Checkpoint 3 to Checkpoint 4. To quantify the improvement, we define the *improvement factor* from test i to test j as

$$f(i, j) = \frac{U_i}{U_j} \tag{11}$$

where U_i is the unavailability evaluated from test i, and U_j is the unavailability evaluated from test j. An f value of 1 means no improvement. An f value of 10 means an improvement of one order of magnitude. By this measure, the improvement factor of the sector suite software availability is 17.7 from Checkpoint 3 to Checkpoint 4, as shown in the table.

Table 5. Unavailability evaluated for Checkpoints 3 & 4

Test	Application Submodel	System Support Submodel	Sector Suite Model
Checkpoint 3	4.50×10^{-5}	6.36×10^{-4}	6.81×10^{-4}
Checkpoint 4	1.90×10^{-5}	1.95×10^{-5}	3.84×10^{-5}
Improv. Factor	2.4	32.6	17.7

Sensitivity analysis

Given data from the recent test, the approach discussed so far can give an estimate of the current availability. An important question is: What are the bottlenecks of the current availability and how much can be improved if the bottlenecks are removed? One way to address the question is to analyze model sensitivity to different PTRs. Now we use the Checkpoint 4 data to illustrate the analysis.

By observing the parameters estimated from the Checkpoint 4 data, it can be found that there are two lower coverage parameters: 0.5 for EFC and 0.75 for CIP. The failure instances responsible for these lower coverages are attributed to problems reported in two particular PTRs. Significant improvement could be made by resolving the problems. To quantify the possible improvement, we took off all failure instances related to the two PTRs from the Checkpoint 4 data. The reduced data set is referred to as the *Checkpoint 4'* data. The results evaluated from the Checkpoint 4' data are shown in Table 6. For comparison, the Checkpoint 4 results are also listed in the table. The application software could be improved by a factor of 26.6 from Checkpoint 4 to Checkpoint 4'. For the overall sector suite software availability, the improvement factor is 2.2, which is again dominated by the system support software.

Table 6. Unavailability predicated for Checkpoint 4'

Test	Application Submodel	System Support Submodel	Sector Suite Model
Checkpoint 4	1.90×10^{-5}	1.95×10^{-5}	3.84×10^{-5}
Checkpoint 4'	7.13×10^{-7}	1.67×10^{-5}	1.74×10^{-5}
Improv. Factor	26.6	1.2	2.2

Achievable availability by test

The results evaluated for Checkpoints 3 and 4 are based on the stability tests of 33 and 78 hours, respectively. The number of failures observed was significantly reduced from Checkpoint 3 to Checkpoint 4 (from 92 to 25). As the software quality is continuously improved, the observable failures in a limited test duration can disappear. An interesting question is: How many no-failure test hours are necessary for achieving a given availability estimate at a desired level of confidence? To answer the question, we applied the methodology discussed so far (parameter estimation and modeling approach) to plot the curves shown in Figure 3. In evaluating the points on the curves, the failure rates were estimated assuming no failure occurred during the test, at the 80% confidence level, and the system configuration and the failure recovery rates used were the same as those for checkpoint 4.

The figure shows that a no-failure test duration of 100 hours can demonstrate an availability of five 9's (0.999992) with an 80% confidence if the coverage is assumed to be 0.95. If the coverage is reduced to 0.90, the required test hours for the same availability at the same confidence level are increased to 200 hours. The availability grows fast when the test duration increases from 100 to 200 hours. As the test duration further increases, the availability growth slows down. For the coverage of 0.95, it takes approximately 800 hours for availability to reach six 9's (0.99999905). For the coverage of 0.90, 1000 hours of test are not sufficient for reaching this level of availability.

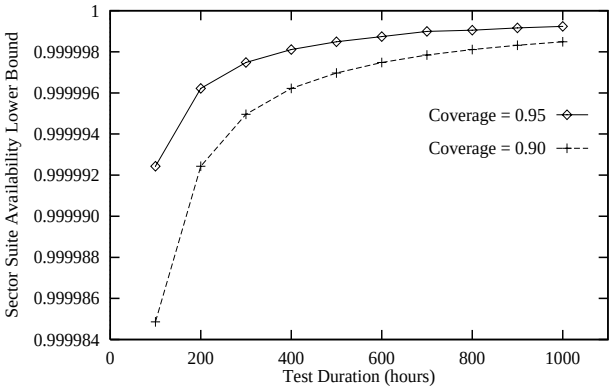

Fig. 3. Availability achieved under various test durations

5.2. Validation of results

Given a set of failure data collected from a system, there are two possible situations in terms of system-level failures: 1) the data contain one or more system failure events; 2) the data contain no system failure event. In situation 1, availability can be measured or estimated directly from the data. In either situations, availability can be evaluated from the component-level failure information in the data based on some assumptions. Both Checkpoints 3 and 4 data belong to situation 1, because sector suite level failures occurred in both tests. Thus, availability can be estimated directly from the data and then compared with the results evaluated based on the component-level parameters estimated from the same data, as discussed in previous sections, to validate our approach.

Estimation of actual unavailability for the tested sector suites is straightforward. It is simply the total down time divided by the total operating time. The total down time is determined by the sum of all recover times for individual OU/FG failures (non-covered failures). The total operating time is determined by the test duration multiplied by the number of sector suites participating the test. To distinguish with the evaluated unavailability listed in Table 5, the unavailability estimated directly from the data is referred to as *measured* unavailability. Table 7 compares the measured unavailability and the evaluated unavailability for Checkpoints 3 and 4.

Table 7. Comparison of measured and evaluated unavailability for a sector suite

Test	Measured	Evaluated	Evaluated/Measured
Checkpoint 3	6.54×10^{-4}	6.81×10^{-4}	1.04
Checkpoint 4	1.77×10^{-5}	3.84×10^{-5}	2.17

Note that the evaluated results are unavailability upper bounds at a certain confidence level, so they should be greater than the measured unavailabilities. Table 7 does show this fact. In the Checkpoint 3 results, the evaluated upper bound is only a little higher than the measured value. This is possibly due to two reasons:

(1) The number of failures for each modeled component was not zero in the Checkpoint 3 test. Thus, the estimated component-level failure rate upper bounds are not much far from the the actual failure rates (sample means). That is, the parameter values used in the model is not so conservative.

(2) In the system support software model, three, instead of four processors were assumed. The actual test configuration is that some sector suites consisted of three processors while others consisted of four. In a sector suite consisting of four processors, the failure rates for the modeled software units should actually be 4λ, instead of 3λ as used in the model.

In the Checkpoint 4 results, the evaluated upper bound is much higher than the measured value. This is possibly due

to:

(1) Conservative failure rate estimates were used for 1/3 modeled software units in which no failure occurred during the test. The actual failure rates in the test should be zero for these units. The estimator we used essentially assumes that one failure has been encountered in the test.

(2) Conservative coverage estimates were used for 2/3 modeled software units. The actual coverage in the test should be 1.0 for these units. In our model, 0.95 was assumed.

To summarize, the evaluated results are valid in terms of upper bounds. As the number of failures encountered in the test is reduced, the results tend to be more conservative. For the Checkpoint 4 data, the evaluated unavailability upper bound is 2.2 times of the actually measured value. By our opinion, a conservative availability estimate within an order of magnitude of the actual unavailability is valuable for the project management.

6. Other dependability characteristics

This section presents two additional dependability characteristics identified from the data: probability distribution of TBF and correlations between software size, number of faults, and failure rate. The obtained results provide insight into software testing, reliability growth modeling, and availability evaluation.

6.1. TBF distribution

Figure 4 shows the empirical probability density function (p.d.f.) for the TBF constructed from the Checkpoint 3 data. The Checkpoint 3 data was used to study the distribution issue because it had timing information and reasonable sample size (92). The way to construct the distribution is similar to that described in [18]: Failures occurring in any of the tested software units were recorded sequentially (sample size on any single unit was not statistically significant for constructing a distribution).

Due to the immaturity of the tested software and the large configuration involved in the test, AS/OU failures occurred frequently, on the order of minutes, even seconds. The first bucket from the left of the figure indicates that about 50% of failures occurred in bursts with inter-arrival time less than a minute. Analysis of PTRs showed that most of these burst failures were due to two problems:

(1) A failure of one address space indirectly causes failures of other address spaces in the same processor. When the failed address space is recovering, it puts a tremendous load on the processor in terms of paging and CPU spike, making other address spaces in the processor miss their steady-state heartbeats.

(2) Errors in accessing a common database bring down multiple address spaces in a short period of time. When a critical database is in an abnormal state, all applications accessing the database receive error messages and may be aborted by the fault tolerance management software.

After these problems were resolved, failure bursts disappeared in the Checkpoint 4 test. Software failure bursts have been reported in many studies, including laboratory experiments [6] and field data analyses [18].

The empirical TBF distribution was found to fit the following 2-phase hyperexponential function [27]:

$$f(t) = a_1\lambda_1 e^{-\lambda_1 t} + a_2\lambda_2 e^{-\lambda_2 t}, \quad a_1 + a_2 = 1 \qquad (12)$$

where a_1=0.28, a_2=0.72, λ_1=0.013, and λ_2 =0.992. The goodness-of-fit passed the Kolmogorov-Smirnov test at the significant level of 0.1. Figure 5 plots both empirical and fitted cumulative distribution functions (c.d.f.). It can be seen that the analytical curve fits the data points very well.

The hyperexponential function has been found to fit TBF distributions for several distributed operating systems on multicomputers [18]. These distributions can be

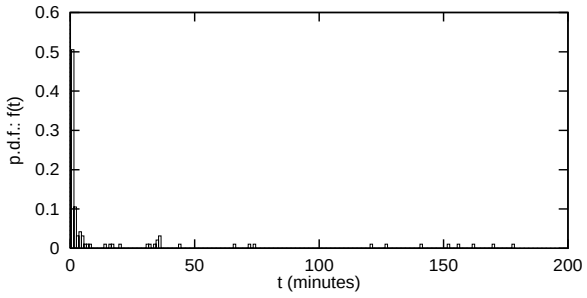

Fig. 4. Empirical p.d.f. for TBF in Checkpoint 3

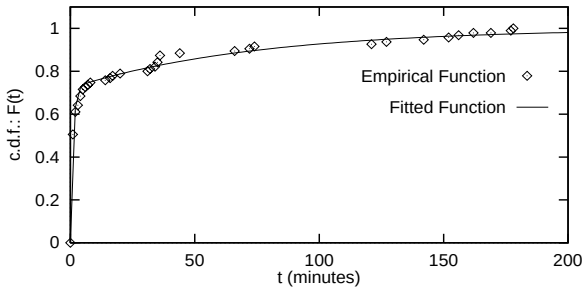

Fig. 5. Fitted c.d.f. for TBF in Checkpoint 3

modeled as a probabilistic combination of two exponential random variables representing two failure modes: a higher failure rate mode and a lower failure rate mode. The higher failure rate, λ_2, with occurrence probability a_2, characterizes failure bursts. The lower failure rate, λ_1, with occurrence probability a_1, characterizes lone failures and inter-burst failure arrivals.

6.2. Correlations between size, faults, and failures

For each OU/FG in which failures have occurred in the test, its MTBF can be easily estimated from the data. In our analysis, two other measures that are of concern of software testing, software size and number of open PTRs (i.e., number of known faults), were also available for some OUs/FGs. Thus, for each of these OUs/FGs, there are three associated parameters: Size, PTRs, and MTBF. These parameters can be viewed as three variables. Figure 6 shows relationships between the three variables for available OUs/FGs. The X axis is the software size in Kilo Source Lines Of Code (KSLOC). The number of PTRs and MTBF are normalized to the range [0,1] for the purpose of comparison.

There is an approximately linear relationship between Size and PTRs (all "+" symbols are around a line), and the similar relationship does not exist between Size and MTBF, or between PTRs and MTBF. These relationships are further quantified by the correlation coefficients listed in Table 8. The correlation coefficient between Size and PTRs is extremely high (0.99). This result reflects a small variation in the density of discovered faults across the studied OUs/FGs, i.e., the number of discovered faults (PTRs) per KSLOC is approximately constant for these OUs/FGs. The low correlation coefficient between PTRs and MTBF indicates that the number of failures occurring in an OU/FG is not proportional to the number of faults discovered in the unit.

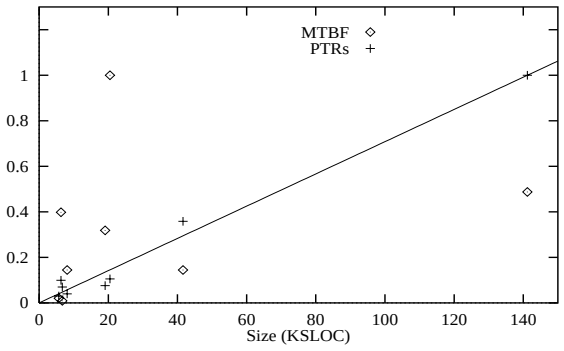

Fig. 6. Relationships between Size, PTRs, and MTBF for available OUs/FGs

Table 8. Correlation coefficients between Size, PTRs, and MTBF for selected OUs/FGs

Variable 1	Variable 2	Correlation Coefficient
Size	PTRs	0.99
Size	MTBF	0.25
PTRs	MTBF	0.20

The above result may be argued by the fact that some of the failures in a software unit are caused by faults in other software units. To further investigate this issue, We identified failures/per-processor-hour attributed to some individual faults (PTRs) from the Checkpoints 3 and 4 data. The results are listed in Table 9. The maximum value and the minimum value differ by two orders of magnitude, which also suggests a non-linear relationship between faults and failures. That is, in a certain period before a fault is removed, the fault may cause a single failure or multiple failures, depending on various factors such as the functionality affected by the fault, the workload running on the system, and the operational environment. This result is consistent with the failure recurrence phenomena addressed in [19] and [20]. The significance of this result is twofold:

1. A class of reliability growth models assumes that the failure hazard rate of a program is proportional to the number of faults remaining in the program and that a fault will be removed after it causes a failure (and before it can cause additional failures). This is often not the case in the software development phase (as shown above) and is rarely the case in the operational phase [19] [20]. Some faults reoccur (perhaps many times) before they can be removed.

2. Fault rates derived from PTRs, which have been used to model availability or reliability growth in some projects, are not appropriate for use in the evaluation of operational availability which requires actual failure rates.

Table 9. Failure rates attributed to different faults

Fault id.	A	B	C	D	E	F
Failures/hr.	2.8×10^{-2}	1.5×10^{-2}	5.5×10^{-3}	2.8×10^{-3}	5.7×10^{-4}	2.8×10^{-4}

7. Conclusion

In this paper, we discussed a measurement-based approach to dependability evaluation of fault-tolerant, real-time software systems based on stability test data. Several dependability analysis techniques were illustrated with the data collected from two stability tests of an air traffic control system, including parameter estimation, availability modeling of software from the task level, applications of

the parameter estimation and model evaluation in assessing availability, identifying key problem areas, and predicting required test hours, and quantification of relationships between software size, the number of faults, and failure rate for a software unit. The discussion of parameter estimation addressed how to estimate failure rate upper bound in the case of no failure occurrence, which was then used to roughly predict required test duration for achieving desired levels of availability. The discussed approach was validated against the data. The non-linear relationship between fault rate and failure rate indicated that failure data, not fault data, should be used in availability evaluation. Although most discussion was focused on a typical subsystem, Sector Suite, the discussed methodology is applicable to other subsystems and the system. The study demonstrated a promising approach to measuring and assessing software availability during the development phase, which is valuable for the project management of developing large, critical systems.

Acknowledgment

This research was supported by the Federal Aviation Administration under Contract DTAF01-89-C-00215.

References

[1] American National Standards Institute, *Recommended Practice for Software Reliability*, ANSI/AIAA R-013-1992.

[2] L. Arlat, K. Kanoun, and J.C. Laprie, "Dependability Modeling and Evaluation of Software Fault-Tolerant Systems," *IEEE Trans. Computers*, Vol. 39, No. 4, pp. 504-513, April 1990.

[3] T.F. Arnold, "The Concept of Coverage and Its Effect on the Reliability Model of a Repairable System," *IEEE Trans. Computers*, Vol. 22, No. 3, pp. 251-254, March 1973.

[4] A. Avizienis and J.P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments," *IEEE Computer*, pp. 67-80, Aug. 1984.

[5] J.F. Bartlett, "A NonStop Kernel," *ACM 8th Symp. Operating Systems Principles (SIGOPS)*, Pacific Grove, CA, Vol. 15, No. 5, pp. 22-29, Dec. 1981.

[6] P.G. Bishop and F.D. Pullen, "PODS Revisited — A Study of Software Failure Behavior," *Proc. 18th Int. Symp. Fault-Tolerant Computing*, pp. 2-8, 1988.

[7] X. Castillo and D.P. Siewiorek, "A Workload Dependent Software Reliability Prediction Model," *Proc. 12th Int. Symp. Fault-Tolerant Computing*, pp. 279-286, June 1982.

[8] F. Cristian, B. Dancey, and J. Dehn, "Fault-Tolerance in the Advanced Automation System," *Proc. 20th Int. Symp. Fault-Tolerant Computing*, pp. 6-17, June 1990.

[9] C. Dale, "Software reliability Issues," *Software Reliability Handbook*, P. Rook Ed., Elsevier Science Publishing Co., Inc., New York, pp. 1-20, 1990.

[10] J.B. Dugan and K.S. Trivedi, "Coverage Modeling for Dependability Analysis of Fault-Tolerant Systems," *IEEE Trans. Computers*, Vol. 38, No. 6, pp. 775-787, June 1989.

[11] J. Gray, "A Census of Tandem System Availability Between 1985 and 1990," *IEEE Trans. Reliability*, Vol. 39, No. 4, pp. 409-418, Oct. 1990.

[12] A.L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Trans. Software Engineering*, Vol 11, No. 12, pp. 1411-1423, Dec. 1985.

[13] IBM Federal System Company, *AAS Fault-Tolerant Design Overview*, FAA Advanced Automation System Acquisition Phase System Safety Report, Vol. II, Book 1, Rockville, Maryland, Jan. 20, 1993.

[14] R.K. Iyer and P. Velardi, "Hardware-Related Software Errors: Measurement and Analysis," *IEEE Trans. Software Engineering*, Vol. 11, No. 2, pp. 223-231, Feb. 1985.

[15] R.K. Iyer and D.J. Rossetti, "Effect of System Workload on Operating System Reliability: A Study on IBM 3081," *IEEE Trans. Software Engineering*, Vol. 11, No. 12, pp. 1438-1448, Dec. 1985.

[16] R.K. Iyer and D. Tang, "Measurement-Based Dependability Evaluation of Operational Computer Systems," *Foundations of Dependable Computing: Models and Frameworks for Dependable Systems*, G.M. Koob and C.G. Lau Ed., Kluwer Academic Publishers, Boston, pp. 195-234, 1994.

[17] J.C. laprie, "Dependability Evaluation of Software Systems in Operation," *IEEE Trans. Software Engineer*, Vol. 10, No. 6, pp. 701-714, Nov. 1984.

[18] I. Lee, D. Tang, R.K. Iyer, and M.C. Hsueh, "Measurement-Based Evaluation of Operating System Fault Tolerance," *IEEE Transactions on Reliability*, Vol. 42, No. 2, pp. 238--249, June 1993.

[19] I. Lee and R.K. Iyer, "Faults, Symptoms, and Software Fault Tolerance in the Tandem GUARDIAN90 Operating System," *Proc. 23rd Int. Symp. Fault-Tolerant Computing*, pp. 20--29, June 1993.

[20] Y. Levendel, "Reliability Analysis of Large Software Systems: Defect Data Modeling," *IEEE Trans. Software Engineering*, Vol. 16, No. 2, pp. 141-152, Feb. 1990.

[21] B. Littlewood, "Modeling Growth in Software Reliability," *Software Reliability Handbook*, P. Rook Ed., Elsevier Science Publishing Co., Inc., New York, pp. 137-154, 1990.

[22] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill Book Company, 1987.

[23] Randell, "System Structure for Software Fault Tolerance," *IEEE Trans. Software Engineering*, Vol. 1, No. 2, pp. 220-232, June 1975.

[24] R.A. Sahner and K.S. Trivedi, "A Software Tool for Learning About Stochastic Models," *IEEE Trans. Education*, Vol. 36, No. 1, pp. 56-61, Feb. 1993.

[25] M.S. Sullivan and R. Chillarege, "Software Defects and Their Impact on System Availability — A Study of Field Failures in Operating Systems," *Proc. 21st Int. Symp. Fault-Tolerant Computing*, pp. 2-9, June 1991.

[26] D. Tang and M. Hecht, *Measurement of ISSS Reliability and Availability from Checkpoint 3E Data*, Technical Report, SoHaR Inc., Dec. 1993.

[27] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1982.