Automated generation of test cases using a performability model

A. Avritzer¹  E. de Souza e Silva²  R.M.M. Leão²  E.J. Weyuker³

¹Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540, USA
²Federal University of Rio de Janeiro, COPPE, P.O. Box 68511, 21941-972, RJ, Brazil
³AT&T Labs - Research, 180 Park Avenue, Florham Park, NJ 07932, USA
E-mail: alberto.avritzer@siemens.com

Abstract: The authors present a new approach for the automated generation of test cases to be used for demonstrating the reliability of large industrial mission-critical systems. In this study they extend earlier work by using a performability model to track resource usage and resource failures. Results from the transient Markov chain analysis are used to estimate the software reliability at a given system execution time.

1 Introduction

We introduce a new model-based test case generation approach to be used to assess the reliability of large industrial mission-critical systems. For such systems, the reliability objective is usually specified in terms of the probability of failure-free operation for a certain period of time under given operational conditions. Therefore to study the reliability evolution of a large industrial system, the reliability model must track failure evolution as a function of the system execution time, \( t \), under realistic operational conditions.

With this goal in mind, we extended in [1] the software reliability model introduced in [2] by adding both failure tracking and system execution time. The work presented in [1] assumed that the Markov chain had a specific structure, such that it could be modelled using the approximation introduced in [2]. In this paper we use performability theory and exact Markov chain solution to generalise the work presented in [1] to systems that can be modelled by Markov chains with a general structure.

Both the reliability modelling and test case generation approaches that we will present in this paper use the function \( p(n, t) \), the transient value of the probability of program \( P \)'s correct execution, for input \( n \) and time \( t \).

In our earlier paper [2] we introduced an automated approach for test case generation and test case execution that was applied to several large industrial telecommunications systems. These systems were modelled by using Markov chains because the arrival processes could be modelled as Poisson processes, and it was reasonable to assume that the service times were exponentially distributed. We further assumed that the types of reliable telecommunications systems we were studying were designed to operate at low-to-medium utilisation rates. Again, this was a realistic assumption for such systems.

Therefore the test case generation approaches we introduced in [2], known as ‘deterministic state testing’ or ‘DST’, were able to achieve the specified model coverage by using search algorithms based on the underlying spanning-tree generated by the depth first search algorithms rooted at the idle state. In that approach, all states with steady-state probability values that were determined to be smaller than an empirically defined \( \epsilon \) could be safely discarded because of the low to medium utilisation assumption.

Taking the DST approach as a starting point, we extended in [1] the test case generation algorithm to incorporate both the resource usage perspective used in [2], and also resource failures.

In this paper we extend the test case generation and reliability assessment approach introduced in [1] by applying performability theory [3, 4]. We present two case studies to illustrate the application of our new approach to large industrial systems.

The outline of the remainder of the paper is as follows. In Section 2 we survey the relevant literature with particular emphasis on Markov modelling approaches that are useful for our automated test case generation and reliability estimation approaches. In Section 3 we describe the approach for test case generation introduced in this paper. Section 4 describes the application of performability theory to the reliability assessment of systems that are tested using the test case generation approach introduced in this paper. In Section 5 we present some empirical results, while Section 6 contains our conclusions and suggestions for future research.

2 Related work

Whittaker and Thomason [5] introduced a statistical testing approach supported by Markov models representing usage
estimating reliability as a function of system execution time. Analysis of the Markovian model. Solution techniques for both the stationary and transient [11] tool transient solution method that provides different representations of failure-free operation. They used the Tangram-II failure part. Pass

defined reliability to be the probability of failure-free operation at time \( t \), it was computed by evaluating the transient Markov chain states’ probability distribution for time \( t \) and then summing over the set of states that represented failure-free operation. They used the Tangram-II [11] tool transient solution method that provides different solution techniques for both the stationary and transient analysis of the Markovian model.

This approach provided a very practical means of estimating reliability as a function of system execution time and expected failure rates, since the Markovian state was defined as the number of failed resources at time \( t \). Therefore the Markovian state explosion problem could be solved by limiting attention to only those states that have a certain number of failed resources. The drawback of this approach is that resource usage is not modelled.

For this reason, the new approach introduced in this paper combines the DST perspective [2] which models resource usage but does not include reliability computation based on system execution time and failures, with the system execution time-based approach described in [10].

Our approach is also related to the ones introduced in [8, 9], which use a Bayesian framework to evaluate the overall system reliability by decomposing the system into modules. In contrast, the algorithm which we introduce in Section 3, decomposes the system Markov chain state into a resource usage part and a failure part. Pass/fail criteria is assessed for each resource usage test case, for all possible failure states.

### 3 Test case generation

We now introduce our test case generation approach that takes into account both resource usage and resource failure events. The intuition is that, when designing test suites for performance testing, we want to select test cases that have a significant impact on the system’s performance.

For example, one might select test cases with frequent modes of failure that each result in a noticeable loss of efficiency, or failures that have a relatively low probability of occurrence but have a major impact on the overall system performance. We determine the likelihood of failure based on the past behaviour.

Typically, these systems use redundant modules to cope with failures. In some architectures, the redundant modules execute identical tasks in parallel, whereas other systems are designed to make use of the full processing power available in the system even if some components are not working correctly. Each design has its advantages and disadvantages.

We use redundancy to reproduce tasks in parallel to assure that if one module is unavailable, others remain likely to be available. Although that means that the potential extra capacity from the redundant modules is not utilised, the system is more likely to have uninterrupted operation. In contrast, when using the excess capacity for computation until a failure occurs, there must be a period of time during which the load is redistributed among the surviving components. In this case the system is said to operate in degradable mode.

The latter approach is frequently used by large database systems, data centres, web services, aircraft control systems, to cite a few examples. Obviously when this approach is used, the system performance is affected by the type of failure experienced by the system. This can occur in a more complex manner than is observed when the system operates at full capacity with some of its components non-operational, until a non-recoverable failure occurs.

In order to model these systems, one must capture the probabilistic nature of user demands for a set of operational system resources. In other words, the system may still perform useful work after one or more faults but perhaps at a different capacity level. The combined modelling of reliability and its impact on performance has been called performability modelling in the literature (e.g. see [4] and references therein).

When doing performability modelling, two Markov chain models are constructed: a resource failure-based Markov chain \( \mathcal{X} = \{ X(t), t \geq 0 \} \) and a resource usage-based Markov chain \( \mathcal{Y} = \{ Y(t), t \geq 0 \} \). In the first model, the types of operational/failed components are captured in the model state and the events are those that impact the resource’s ability to perform useful work (for instance, failure and repair events). The second Markov chain model captures the measure of performance the modeller is interested in, given the particular state of the failure-based model. A common assumption in this approach is that the event rates of the resource usage-based Markov chain are much higher than those for the failure-based chain. This is reasonable, since the elapsed time between failure and/or repair events are usually orders of magnitude higher than the time interval between the arrival of tasks and the amount of time that takes to process them.

By combining the two types of Markov chains, we are able to focus on both types of issues – testing for failures that are most likely to occur and testing for cases that have a significant impact on performance under a given configuration. In both cases these classes are determined based on historical data.

We assume that resources may be allocated and de-allocated both in the course of normal system operation...
and because of failure and repair events. We first generate a failure-based Markov chain. For this Markov chain, a state \( \phi \) represents the resources of each type that are unavailable to process tasks. Let \( N \) be the number of distinct types of resources. The state is an \( N \)-tuple \((f_1, f_2, \ldots, f_N)\) where \( f_i \) is the number of resources of type \( i \) that are unavailable to perform useful work. Clearly, state \((0, 0, \ldots, 0)\) represents a system in which all components are fully operational and available for processing tasks.

Next we construct the resource usage-based Markov chain. This Markov chain is built from the specification of the system demands and the resources available to work on these demands. The model depends not only on the type of resources available, but the nature of the demands and the particular measure we are interested in obtaining. For example, suppose we have \( M \) resources available. An entry \( s_i \) of state \( s = (s_1, s_2, \ldots, s_M) \) may indicate: (i) the number of resources of type \( i \) that are being utilised; or (ii) the number of tasks that are contending for type \( i \) resource.

In the examples we will present in this paper, we assume that failure and repair events are exponentially distributed but, clearly, phase-type distributions can be handled provided that the state space of \( X \) is manageable. Similar comments apply to the resource model events, including the time between resource deallocation and allocation events, interval between task arrivals or task processing.

For very large models, it may be necessary to truncate each of the Markov chains \( X \) and \( Y \) to compute the required measures of interest. A common approach that is employed for failure-based models is to truncate the chain up to a given number of failures. If the measures of interest are calculated for a given time interval \( 0, t \), the error can be bounded by the probability that the chain reaches the set of discarded states by time \( t \). Intuitively, if the probability of reaching the discarded states is very small, they are unlikely to occur in the time frame considered and their effect on the overall performance may be disregarded.

For the resource-based model, if we are able to generate all the states in the model, then we can disregard the less probable states. However, if the state space of \( Y \) is too large to generate, it may be difficult to select the most probable states ahead of time, unless the Markov chain \( Y \) has a special structure. Note that, since we assume that process \( Y \) reaches steady state while process \( X \) is in any of its states, we cannot obtain a bound by simply calculating the probability that \( Y \) reaches the discarded states by the end of the observation interval. Reference [12] discusses these issues.

In this paper, we are interested in determining which are the cases to be tested and if the system satisfies the requirement specifications both in terms of reliability and performance. We assume that we can easily compute the steady-state probabilities from \( Y \). As such, we can test only those cases which are most likely to occur in \( Y \) in order to determine if they satisfy the system requirements. For instance, assume that the resource-based model is a simple queueing model of software tasks contending for a single resource and the system specification requires that the average waiting time to process a task is smaller than a given value. Further assume that, if the system has more than \( L \) tasks queued, the requirements are not satisfied. Clearly, we need only test cases up to \( L \) tasks queued for the available resource, since, if more than \( L \) tasks are queued, the requirements will not be satisfied. It is also very easy to compute the state probabilities for this resource-based model. From the model, we can calculate the probability that more than \( K \) tasks are queued for processing. If the model indicates that this probability is very small, then we do not need to test cases with more than \( K \) tasks queued, since they are unlikely to occur and will have negligible impact on the overall system requirements. Consequently, the only cases to be tested are those up to \( L \) tasks in the system for \( L = \min\{L, K\} \).

We describe an example that will be discussed throughout the paper in order to clarify the above concepts. This is a ‘general’ example, motivated by applications such as cloud computing, data centres etc. but the approach described can be used to analyse many other applications as well. Consider a large distributed processing system composed of several data centres or clusters of computers, located at different sites, as shown in Fig. 1. The machines at each centre are capable of processing any of the requests generated by the users. (Subsets of machines could be specialised to process certain tasks, but we adhere to simple cases to facilitate the model description.)

When a request is submitted to the system, it is sent to a dispatcher module that is in charge of selecting the best site to process the job. Thus, the dispatcher performs tasks such as load balancing among centres, and also knows when any of the centres becomes unavailable. Machines at each data centre can fail because of either a hardware or a software problem.

When a machine fails, the processing capacity of the centre is reduced until the machine is placed back into operation. An entire data centre may become unavailable when, for instance, it runs out of processing resources or it is disconnected from the ‘outside’ because of a network failure. When a centre is disconnected, the load is redistributed among the remaining sites. Assume that the system’s specifications imposes a limit on the expected waiting time to process tasks. That is, assume that the specifications require that the fraction of time the system operates in an overload condition must be smaller than a given threshold value. (Here the overload condition means that expected waiting time to process tasks is above a given value.)

We are interested in developing a test case generation approach to assess whether the software system meets its requirements.

When a data centre becomes unavailable, the load at other centres increases since the dispatcher performs load balancing. The tests to be developed must, in principle, take into account all the combinations of possible loads in order to assess the requirement’s specifications. However, using the Markov models, we can limit the tests by ignoring the loads that are unlikely to occur and those that are identified...
to be above the maximum load above which performance is unacceptable. We emphasise that we may obtain from the models several performance and reliability measures, but we focus on the fraction of time the system is expected to satisfy the requirement’s specifications as the measure of choice.

In summary, this is an example of a dynamic distributed system in which the computational performance varies with time. Both performance and reliability are considered jointly. In the following section we formally introduce the above concepts.

4 Performability modelling and assessment approach

In the preceding section we outlined an approach to construct models of systems in which both reliability and resource usage play an important role for assessing the overall system performance requirement goals. In this section we formalise some of the concepts introduced in Section 3.

As discussed in Section 3, requirements are usually specified in terms of either dependability or performance measures. In the first case, the fraction of time the system is available during an observation period is a common measure of interest and the requirement to be achieved is a given threshold value above which the system must operate. For example, one might require that the system be available to the user for 99% of the time.

In the second case, the requirements are specified in terms of a chosen performance metric. The expected response time and the number of tasks executed per unit time (throughput) are two common metrics. One might, for example, require the expected time to execute a task to be smaller than \( x \) units of time.

However, as we argued in the previous sections, neither requirement individually is adequate for systems in which the available capacity varies with time when failure occurs. Although changes in the system’s ability to perform useful work may vary because of events other than failures, we consider only failure events to illustrate the concepts in this paper. Therefore we first need to choose an appropriate measure that is meaningful to the overall system’s objectives. For systems similar to that described at the end of Section 3, a useful metric is the relative frequency that the tasks are executed below a required execution time threshold over a given observation period \((0, t)\).

Suppose that the system has a set \( C = \{c_1, c_2, \ldots, c_C\} \) of \( C \) different configurations. Each configuration results from different combinations of system failures and each incurs a different level of available capacity to the users. Then, each configuration \( c_i \), \( i = 1, \ldots, C \), must be investigated to determine if and when the requirements are satisfied, given that the system is in \( c_i 100\% \) of the time. Referring to the example of Section 3, the failures may incur the relative frequency that the tasks are executed below a required execution time threshold over a given observation period \((0, t)\). Suppose the system spends \( \delta_i \) units of time in configuration \( c_i \) during the observation period \((0, t)\). Then \( \delta_i * r_i \) is the fraction of time the system is expected to pass the requirements test.

In order to calculate \( \Gamma(t) \) we refer to the results from transient analysis of Markov reward models (e.g. [13]). We first uniformise the failure-based Markov chain \( X \) with rate \( \Lambda \) and obtain \( P_{X,t} \), the one-step state transition probability matrix for the uniformised chain \( X \) (see [12] for details). Roughly, \( P_{X,t} = I + Q^X * \Lambda \), where \( Q^X \) is the generator matrix for \( X \) and rate \( \Lambda \) satisfies \( \Lambda \geq \max(q_i) \), where \( q_i \) is the total rate out of state \( i \) for chain \( X \).

Let \( r = (r_1, \ldots, r_C) \) be the vector of reward rates. Equation (58) of [15] gives an expression for \( \Gamma(t) \)

\[ \Gamma(t) = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{\lambda^n}{n!} \left[ \sum_{j=0}^{\infty} r_j \cdot n \right] \]

Equation (2) can be re-written as

\[ \Gamma(t) = \frac{1}{\lambda} \sum_{n=0}^{\infty} E_{n+1, \Lambda}(t) \times \left[ \sum_{i=1}^{C} \phi_i(n) \left( \sum_{j=1}^{M(i)} \gamma_j(i) \alpha_j(i) \right) \right] \]

where \( E_{n+1, \Lambda}(t) \) is the \( (n+1) \)-stage Erlangian distribution.

Let \( \Gamma \) be the long-term fraction of time the system satisfies the requirements. For \( t \to \infty \) (3) reduces to

\[ \Gamma = \sum_{i=1}^{C} \phi_i(n) \left( \sum_{j=1}^{M(i)} \gamma_j(i) \alpha_j(i) \right) \]
where we recall from Section 3 that $\phi_i$ is the $i$th state of the failure-based Markov chain $X$.

The modelling and assessment process outlined above can be summarised as follows:

1. Define the requirement goals for the system: the fraction of time the system performs above a given performance level during the observation period $\Gamma(t)$.
2. Define the resource failure-based Markov chain $X$ for the system: $X$ produces $C$ different system resource configurations.
3. For each configuration $c_i$, obtain $Y(i)$, the resource usage-based Markov chain for that configuration.
4. From $Y(i)$, determine $c(g)$, that is the pass/no pass conditions from the system’s requirements under configuration $c_i$.
5. Solve Markov chain $Y(i)$ and obtain $\gamma(j) = \gamma_1(j), \ldots, \gamma_M(j)$, the long-term fraction of time configuration $c_i$, spends in each state $s_j$.
6. Uniformise chain $X$ and obtain $\Lambda$ and $P_X$.
7. Use (3) to obtain $\Gamma(t)$.

We briefly illustrate the steps above referring to the example we introduced in Section 3. (This example is detailed in the following section.) The distributed processing system example is formed by dispatcher machines, and data centres, each consisting of many machines capable of processing the user requests. For the sake of simplicity, we assume that all computers in any of the data centres can process any task. When a computer of a data centre fails, the processing capacity of that centre decreases and the centre ceases operating when all its computers fail. In addition, a data centre may be unavailable to the user when its network connection is down. Furthermore, when a data centre is unavailable for processing (because of either the exhaustion of its resources or the failure of its network connection) the load is redistributed among the remaining data centres.

The state variables of resource failure-based Markov chain $X$ are the number of the dispatchers that failed and, for each data centre, the number of machines of that centre that are unavailable to the user and a flag that indicates whether the centre’s network connection to the outside world is down or not.

The state variables chosen for the resource usage-based Markov chain $Y$ depend on how the data centres process the resources, the way they interact among themselves and the performance measure to be obtained. In the example of the following section we use simple queueing models for each centre.

The remaining steps outlined above requires the solution of the equations we present and can be automated. We used the Tangram-II tool [11] both for describing the system at a high abstraction level and subsequently solving the model.

In order to apply the modelling methodology described above to large system configurations one may be faced with large cardinalities of the space for the Markov chains $X$ and $Y$ and, in addition, the large number of software tests needed to assess the requirement’s specifications. In order to cope with these issues we need to investigate which are the irrelevant states of each chain, in the sense that they do not contribute significantly to the final solution.

For Markov chain $X$ we can partition the state space in subsets of states with the same number of failures and generate the states in increasing order of the total number of failures in the system. The non-generated states are aggregated into a single absorbing state. Then, the probability of reaching the absorbing state in the observation interval is a bound to our overall assessment measure. Note that each discarded state in $X$ is a system configuration that does not need to be tested and significant savings are possible if this number is large.

For Markov chain $Y$, it may not be so clear that which software tests can be discarded while still obtaining a meaningful solution, since this depends on the structure of the performance model built. However, it is clear that, if the system does not satisfy the requirements for a load in a given configuration, it will not satisfy the requirements for higher loads, and these loads can be ignored. Furthermore, we can incrementally perform pass/no-pass tests for a subset of states in $Y$ and check if the overall model achieved the desired requirements, assuming ‘no-pass’ for the non-tested cases. This clearly gives a lower bound for the requirements. The example in the next section clarifies these issues.

In summary, the algorithm we suggest to be used for test case generation using the performability approach introduced here is

- Start with a state probability coverage objective for the resource failure-based and resource usage-based models.
- Model the system under study to build a resource failure-based Markov chain $X$. This Markov chain may lead to $C$ possible distinct configurations and $C$ different resource usage-based Markov chains $Y(i)$ for configuration $c_i$.
- Solve for the distinct resource usage-based Markov chains to obtain the long-term fraction of time in each state of the individual chains. The goal here is to obtain $\gamma_j(i)$, the fraction of time during $(0, t)$ that $X$ spends in state $s_j$.
- The resource failure-based Markov chain is solved to obtain the fraction of time during $(0, t)$ that $X$ spends in each state $\phi_i$.
- Sort the set of results for each individual chain in decreasing order.
- For the resource failure-based Markov chain generate as a test suite the list of states from $X$ such that the sum of the total fraction of time in these states is above a given threshold value, in order to cover the most cases during the observation interval.
- Order the states for each resource usage-based Markov chain and test the most probable scenarios for each configuration.

Assuming that the non-tested states did not pass the performance requirements, calculate $\Gamma(t)$ and check if the overall system requirements are met. If not, verify if any extra tests will help the system to reach the requirements by checking the calculated fractions for the neglected states and scenarios. If no further improvements are possible, we may conclude that the system did not pass the requirements. Otherwise, we re-calculate $\Gamma(t)$ including the new test cases performed.

As in many modelling studies, major concerns related to the applicability of the model include the accuracy of the input parameter values as well as the validity of the modelling assumptions. Concerning the assumptions, Markov modelling has been extensively employed with success in reliability/availability, including real complex systems, as a useful guide through the design process. Our basic assumptions are no different than those commonly used. The accuracy of the input parameter values may not be determined ahead of time. In order to evaluate the impact of variations on the parameter values, one should
perform an analysis of the sensitivity of the measures of interest with respect to the parameter values. In other words, the analyst should determine which are the modelling parameters such that perturbations on their values cause a relatively large change in the model output measures. Once these parameters are found (using the model) then the test cases should be devised to cope with possible errors when estimating the parameters.

5 Empirical results

5.1 Distributed data centre

In this section we first illustrate the modelling and assessment process we have introduced in this paper using the example described in Section 3. The system has two dispatchers and three data centres. Each data centre has two machines that can execute any task and we assume that the system load is equally distributed among the three data centres. The system is considered operational if one dispatcher is available and at least one machine from one data centre is operational. Furthermore, we neglect the failures in the data centres’ network connection.

The first step of the approach we described in previous sections is the selection of the requirements for the system. We set $\Gamma(t)$ equal 0.995 for $t = 480$ h (20 days).

The second step is to build the system failure model. For that we use the Tangram-II tool [11]. We choose the mean time between failures for the dispatchers and machines equal to 3 months, and the mean time to repair equal to 1 day. The failure model has 81 states, which means that the system has 81 configurations. For each configuration $c_i$, we construct $\mathcal{Y}(i)$, the resource usage-based model. The resource usage-based model represents the three data centres. Each data centre is modelled by a $M/M/2/K$ queue when both machines are operational, and by a $M/M/1/K$ queue when one machine is down. Therefore the state variables for the resource usage-based model for any configuration is the concatenation of the individual state variables for each individual queue.

Consider, for example, the configuration where there is one machine unavailable at a data centre. In this case, the resource usage-based model comprises of two $M/M/2/K$ and one $M/M/1/K$ queue. The state of the resource usage-based model $s_i(i)$ is equal to $(q_1, q_2, q_3)$, where $q_k$, $k = 1, 2, 3$ represents the size of the queue of the data centre $k$.

We select the system parameters as follows. Each queue has load equal to $\lambda/3$, and the service rate is equal to $\mu$. We assume that the arrival rate of users is equal to 75 users per minute and each user generates 10 requests per second. Then $\lambda/3$ is equal to 4. The service time of each request is equal to 180 ms. In this scenario, the system’s utilisation is 51%. We also consider the case with 83% utilisation (the arrival rate is equal to 150 users per minute in this last case).

The next step is to determine $\alpha_s(i)$, that is the pass/no pass conditions from the system’s requirements under configuration $c_i$. We assume that the system’s specifications impose a limit on the expected waiting time of each user’s request. Then, $\alpha_s(i)$ is equal to 1 if all queue sizes are equal or below a certain threshold using the resource usage-based model for configuration $c_i$. The selected threshold in this example is equal to 30 for the $M/M/2/K$ queues and 15 for the $M/M/1/K$ queues.

The fraction of time configuration $c_i$ spends in each state $s_i(i)$ is obtained from the $M/M/2/K$ and $M/M/1/K$ steady-state solutions. For this particular example, since the data centres behave independent of each other, each queue can be solved in isolation. The fraction of time in state $s_i(i)$ is simply the product of the individual solutions:

\[
\pi_{s(i)} = \pi^{(1)}_{s(i)} \pi^{(2)}_{s(i)} \pi^{(3)}_k,
\]

where $\pi^{(l)}_k$, $l = 1, 2, 3$, is the fraction of time the queue which represents data centre $l$, spends in state $k$. The Tangram-II tool is also used to obtain the queues’ steady-state solution.

Fig. 2 shows the fraction of time the system satisfies the requirements specifications during 1000 h. We note that, for the case with 0.83 utilisation, $\Gamma(t)$ quickly drops from 0.9998 ($t = 1$ h) to 0.9605 ($t = 50$ h). When the utilisation is equal to 0.51, $\Gamma(t)$ is 0.9999 for $t = 1$ h and equal to 0.9950 for $t = 480$ h (20 days). For these scenarios, the requirements goals are only achieved when the system utilisation is equal to 0.51.

In order to select the configurations for the test case generation, we calculate the fraction of time the system is in each one of them during the observation interval $(0, t)$. For the parameters used, the fraction of time the system has three or more failures is less than $10^{-3}$ and so we need to test only four configurations and disregard the others, since they have negligible impact in the overall goals. The tested configurations are: (i) all system machines are operational; (ii) there is a single unavailable machine in one of the data centres; (iii) one data centre is unavailable because of a failure in each of its two machines; (iv) there are two unavailable machines, each in a distinct data centre.

The performance parameters to be used to test each configuration can be defined from the resource usage-based model. In this example, we assume that the system’s specifications impose a limit on the expected waiting time of each user request. This limit is equal to 30 requests waiting in the data centre queue when the two machines in the centre are operational, and 15 requests when only a single machine is available. Then, we only need to test the system using a set of load values for which the probability of $K$ requests in the data centre queue is, for example, greater than 0.99.

With this approach, we can select the most probable configurations and the corresponding performance parameters to be tested in order to achieve the overall system requirements.

5.2 Application to a high-reliability mission critical system

We describe the application of our approach to a mission critical system that is required to have no single point of
failure. The system architecture uses voting, triple replicated sensors, triple replicated machines and redundant networks, as shown in Fig. 3.

The requirement specification for this system is the fraction of time the system is operational during a given observation period. The system is considered operational if at least one GPS sensor, one GYRO sensor, one processor machine and one of the two networks are available.

For this example we assume the following failure rates: (a) GYRO sensor: one per 30,000 h; (b) GPS: one per 100,000 h; (c) processor machine: one per 17,520 h; (d) network: one per 8,760 h. The expected time to repair any component is 168 h.

We build the resource failure-based model using the Tangram-II tool. The model has 192 states, which means that the system has 192 distinct configurations.

When solving the model for the mission time of 3 months we obtain the fraction of time the system is operational equal to 0.99971. If we consider only a single failure for any of the system’s components, the measure of interest is 0.99727. Therefore if the specifications require two 9’s of availability only five configurations needed to be tested.

6 Conclusions

In this paper we have generalised our automated test case generation approach introduced in [2] and extended in [1] by applying performability theory [3, 4].

We have exemplified our approach by presenting two empirical studies that are very similar to two large industrial systems we have worked with.

We are currently designing an approach to certify a very large mission-critical system for reliability and we are planning to use the reliability estimation algorithm introduced in this paper to generate the resource usage test cases and to derive the plots of the reliability metric as a function of system execution time. This will represent a real application of the approach described in this paper.

7 Acknowledgment

The research of E. de Souza e Silva and R.M.M. Leão is partially supported by grants from CNPq, FAPERJ and Siemens Corporate Research.

8 References