Reliability Prediction and Sensitivity Analysis Based on Software Architecture

Swapna S. Gokhale  
Dept. of CSE  
Univ. of Connecticut  
Storrs, CT 06269  
Email: ssg@engr.uconn.edu

Kishor S. Trivedi  
CACC, Dept. of ECE  
Duke University  
Durham, NC 27708  
Email: kst@ee.duke.edu

Abstract

Prevalent approaches to characterize the behavior of monolithic applications are inappropriate to model modern software systems which are heterogeneous, and are built using a combination of components picked off the shelf, those developed in-house and those developed contractually. Development of techniques to characterize the behavior of such component-based software systems based on their architecture is then absolutely essential. Earlier efforts in the area of architecture-based analysis have focused on the development of composite models which are quite cumbersome due to their inherent largeness and stiffness. In this paper we develop an accurate hierarchical model to predict the performance and reliability of component-based software systems based on their architecture. This model accounts for the variance of the number of visits to each module, and thus provides predictions closer to those provided by a composite model. The approach developed in this paper enables the identification of performance and reliability bottlenecks. We also develop expressions to analyze the sensitivity of the performance and reliability predictions to the changes in the parameters of individual modules. In addition, we demonstrate how the hierarchical model could be used to assess the impact of changes in the workload on the performance and reliability of the application. We illustrate the performance and reliability prediction as well as sensitivity analysis techniques with examples.

1 Introduction

The size and complexity of computer systems has increased more rapidly in the past decade, than our ability to design, test, implement and maintain them. Computer systems are being increasingly used in various active (controlling), and passive (monitoring) applications, and the trend will surely continue in the future. Computer system failures make newspaper headlines because at best they inconvenience people (e.g., malfunctions of home appliances), cause economic damage (e.g., interruptions of banking services), and in the extreme cases cause deaths (e.g., failures of flight control systems or medical software). The computer industry has seen uneven progress. With the steadily growing power and reliability of the hardware, software reliability has been identified as a major stumbling block in the realization of highly dependable computer systems. When lives and fortunes depend on software, assurance of its quality becomes an issue of critical concern.

The impact of the structure of an application\(^1\) on its reliability and correctness has been highlighted almost two decades ago [19, 23]. However, prevalent approaches to characterizing the behavior of software systems are black-box based, i.e., the software system is considered as a whole and only its interactions with the outside world are modeled, without looking into its internal structure. These approaches were suited to capture the behavior of largely custom, built-to-specification type of applications. Several critiques of the black-box based approaches to predict the reliability of software systems have appeared in the literature [8, 9] and some of these include the fact that they are applicable very late in the life-cycle of the software, ignore information about testing and reliabilities of the components\(^2\) of which the software is made, and do not take into consideration the architecture of the software. With the advancement and widespread use of object oriented systems design and web-based development, the use of component-based software development is on the rise. The software components can be commercially available off the shelf (COTS), developed in-house, or developed contractually.

\(^1\)Application, software system and software are used interchangeably in this paper.

\(^2\)Component and module are used interchangeably in this paper.
Thus, the whole application is developed in a heterogeneous (multiple teams in different environments) fashion, and hence it may be inappropriate to model the overall failure process of such applications using the existing software reliability growth models (black-box approach) [4]. These heterogeneous systems, where components having different workloads and failure behaviors interact, have become more of a norm than an exception [16]. Modern programs can no longer be treated as monolithic entities, but are likely to be made up of thousands or millions of little parts distributed globally, executing whenever called, acting as parts of one or more complex systems [26]. Thus, predicting the performance and reliability of an application taking into account the information about its architecture, is absolutely essential.

Goseva-Popstojanova et al. classify the different approaches to architecture-based software reliability assessment into three categories: state-based, path-based and additive [7]. Earlier research efforts in this area concentrated on the state-based approaches [2, 13, 14, 12], whereas more recently path-based [11, 30] and additive approaches [29] have been proposed. Most of the state-based techniques to architecture-based software reliability analysis rely on composite models, where the architecture of the software and the failure behavior of its components is combined into a single model. This model is then solved to assess the reliability of the software. Cheung proposed a model for the reliability evaluation of software systems when the architecture of the application is modeled by a discrete time Markov chain (DTMC) and the probabilities of individual components are known [2]. Siegrist developed a method to determine system reliability in terms of module reliabilities and transition probabilities [21, 22]. Chen et al. propose composite models to predict the reliability of different architecture styles [28]. Composite models however can be cumbersome and inconvenient because of the following two reasons:

- An important utility of architecture-based analysis is in the design phase of the software, where various competing alternatives need to be evaluated, and various decisions with respect to how many and which components should be developed in house, and which modules can be outsourced need to be made. In this phase, it is then essential to assess the impact of an individual module on the overall behavior of the software. In case of a composite model, to analyze such an impact, the combined model has to be reconstructed and re-solved to obtain revised predictions. This involves repeated construction and solution of the combined model. This is further exacerbated by the fact that in practice a software of moderate size can have several hundreds to thousands of states, which makes repeated construction and solution of the composite model both costly and time consuming.
- The second problem arises due to the stiffness of the composite model. This is because of the fact that the probability of failure of the modules is quite low, compared to the transition probabilities among the components, which makes the transitions to the failure state unlikely. Solution techniques which take into account the stiffness will need to be employed in this case [18].

Hierarchical models decompose the reliability evaluation process into two steps to overcome the problems posed by composite models. Littlewood [15] proposed a hierarchical model where the architecture of the software is represented by a semi-Markov process, and the failure behavior of each individual component is described by a Poisson process. Laprie et al. [12] develop a hierarchical method for the assessment of software and system reliability, when the architecture of the software is described by a continuous time Markov chain (CTMC) and the failure behavior of each component is described by a time-independent failure rate. One of the major drawbacks of hierarchical models is that it only provides an approximation to the composite models and hence the reliability metrics obtained using these models are not as accurate as the ones obtained using composite models. Development of accurate hierarchical models which provide very close predictions to those obtained from composite models is essential, so that the other advantages offered by the hierarchical models can be exploited. The prior research efforts also mention the potential of the architecture-based reliability models to assess the impact of an individual component on the overall reliability of the application. However, most of the efforts provide very little insight into how this can be accomplished.

In this paper, we present an accurate hierarchical model for architecture-based performance and reliability analysis of software systems. The model developed in this paper includes second-order architectural effects, and hence provides more accurate predictions. We assume that the architecture of the application is modeled using a discrete time Markov chain (DTMC), and the time spent in each component per visit and the reliability of each component is known. We develop techniques for performance and reliability prediction, as well as the identification of performance and reliability bottlenecks, and illustrate them using examples. We also develop expressions to analyze the sensitivity of the performance and reliability predictions to the changes in the parameters of individual modules. In addition, we demonstrate how the hierarchical model can be used to assess the impact of changes in the workload on the performance and reliability predictions. Thus, the contributions of the paper are two-fold: i) Development of an accurate hierarchical model for performance and reliability predictions ii) Demonstration of the use of the hierarchical
model to assess the impact of an individual module (module sensitivity) on the overall performance and reliability behavior of the software.

The layout of the paper is as follows: Section 2 presents a brief overview of discrete time Markov chains (DTMCs). Section 3 describes the methodology for performance and reliability prediction. Section 4 illustrates the methodology described in Section 3 using examples. Section 5 describes how the hierarchical approach can be used for sensitivity analysis. Section 6 briefly discusses how the performance and reliability predictions techniques developed in this paper can be applied to real-life software systems. Section 7 presents conclusions and directions for future research.

2 Overview of DTMCs

We assume that the architecture of the application is modeled using a DTMC and we present a brief overview of DTMCs in this section. The interested reader is referred to [10, 24] for a detailed study. A DTMC is characterized by its one-step transition probability matrix, $P = [p_{ij}]$. $P$ is a stochastic matrix since all the elements in a row of $P$ sum to one, and each element lies in the range $[0, 1]$.

For our purposes, DTMCs can be classified into the following two categories:

- Irreducible: A DTMC is said to be irreducible if every state can be reached from every other state.
- Absorbing: A DTMC is said to be absorbing, if there is at least one state $i$, from which there is no outgoing transition. A DTMC upon reaching an absorbing state is destined to remain there forever.

The transition probability matrix of an absorbing DTMC can be partitioned as:

$$P = \begin{bmatrix} Q & C \\ 0 & 1 \end{bmatrix}$$

where $Q$ is an $(n-m) \times (n-m)$ substochastic matrix (with at least one row sum less than 1), $I$ is an $m \times m$ identity matrix, $0$ is an $m \times (n-m)$ matrix of zeros, and $C$ is an $(n-m) \times m$ matrix, when there are $m$ absorbing states in the chain with $n$ states.

The $k$ step transition probability matrix $P^k$ has the form:

$$P^k = \begin{bmatrix} Q^k & C^k \\ 0 & 1 \end{bmatrix}$$

where the entries of matrix $C^k$ are not relevant. The $(i,j)$-th entry of matrix $Q^k$ denotes the probability of arriving in transient state $j$, starting from transient state $i$ in exactly $k$ steps. It can be shown that $\sum_{k=0}^{\infty} Q^k$ converges as $t$ approaches infinity. This implies that the inverse matrix $(I - Q)^{-1}$, called the fundamental matrix $M$, exists, and is given by:

$$M = (I - Q)^{-1} = I + Q + Q^2 + \ldots = \sum_{i=0}^{\infty} Q^i \quad (1)$$

Let $X_{i,j}$ denote the number of visits to state $j$ starting from state $i$ before the process is absorbed. The $(i,j)^{th}$ entry of the fundamental matrix $M$ represents the expected number of visits to state $j$ starting from state $i$ before it is absorbed [24]. Thus we have:

$$E[X_{i,j}] = m_{i,j} \quad (2)$$

The fundamental matrix can also be used to compute the variance of the expected number of visits [10]. Let $\sigma_{i,j}^2$ denote the variance of the number of visits to state $j$ starting from state $i$. Then, we have:

$$\sigma^2 = M(2M_{dg} - I) - M_{eq} \quad (3)$$

where $M_{dg}$ represents a diagonal matrix and $M_{eq}$ denotes the square of the fundamental matrix $M$. Thus:

$$Var[X_{i,j}] = \sigma_{i,j}^2 \quad (4)$$

3 Performance and reliability prediction

In this section we describe the methodology for performance and reliability prediction of an application when its architecture is described by an absorbing DTMC. We assume that the time spent by the application in each component per visit is a random variable with known mean and variance. Similarly, we also assume that the reliability of each component per visit is deterministic and is known. Expressions to compute the application reliability assuming component reliabilities to be random instead of deterministic is the direction of future research. We assume that the application consists of $n$ components, and has a single initial state denoted by 1, and a single absorbing or exit state denoted by $n$. The architecture of the application is given by the one-step transition probability matrix $P$. We also assume that the components fail independently of each other as well as in successive executions.

A “hierarchical method” of performance and reliability prediction is discussed, i.e., the one-step probability matrix of the DTMC representing the architecture of the application is analyzed to obtain the mean and the variance of the
number of visits to transient state \( i \) starting from the initial state 1, using the equations described in Section 2. The mean and variance of the number of visits are then combined with the expected time spent in each component per visit for performance prediction, and with the reliabilities of individual modules for reliability prediction. Thus, the analysis follows a two step approach as shown in Figure 1.

### 3.1 Performance prediction

The performance model in this scenario is a semi-Markov process [20], and the random variable representing the holding time in state \( i \) denoted by \( T_i \) or consequently the time spent in module \( i \), has a mean \( \tau_i \), and variance \( \sigma_i^2 \). In this section, we describe how the expected number of visits obtained from the embedded DTMC underlying the semi-Markov process describing the architecture of the application, can be used to predict the mean time to completion of the application. The time to completion of the application for a single run, denoted by the random variable \( T \) is given by:

\[
T = \sum_{i=1}^{n} T_i X_{1,i} \tag{5}
\]

where \( X_{1,i} \) denotes the number of visits to the transient state \( i \) starting from state 1. The number of visits to the absorbing state \( n \) is always 1, i.e., \( X_{1,n} = 1 \).

The expected or the mean time to completion of the application is given by:

\[
E[T] = E\left[ \sum_{i=1}^{n} T_i X_{1,i} \right] = \sum_{i=1}^{n} E[T_i] E[X_{1,i}] \tag{6}
\]

Thus to obtain the expected time to completion of the application, we need to obtain \( E[T_i X_{1,i}] \), which is the total expected time spent in component \( i \) during a single execution. \( T_i X_{1,i} \) is a function of the random variables \( T_i \) and \( X_{1,i} \), and hence \( T_i X_{1,i} \) is a random sum [25]. Hence \( E[T_i X_{1,i}] \) using the expressions for random sums [24] is given by:

\[
E[T_i X_{1,i}] = E[T_i] E[X_{1,i}] = \tau_i m_{1,i} \tag{7}
\]

Since the number of visits to state \( n \) is 1, \( E[X_{1,n}] = 1 \), and hence \( E[T_n X_{1,n}] = \tau_n \).

From Equation (6) and Equation (7), the expected time to completion of the application is given by:

\[
E[T] = \left[ \sum_{i=1}^{n-1} \tau_i m_{1,i} + \tau_n \right] \tag{8}
\]

The variance of the random variable \( T_i X_{1,i} \) is given by [24]:

\[
Var[T_i X_{1,i}] = Var[T_i] E[X_{1,i}] + (E[T_i])^2 Var[X_{1,i}] \tag{9}
\]

which can be re-written as:

\[
Var[T_i X_{1,i}] = \theta_i^2 m_{1,i} + \tau_i^2 \sigma_{1,i}^2 \tag{10}
\]

From Equation (10), the variance of the time to completion of the application, assuming independence among the time spent in each of the modules is given by:

\[
Var[T] = \sum_{i=1}^{n} \theta_i^2 m_{1,i} + \tau_i^2 \sigma_{1,i}^2 \tag{11}
\]

The component with the largest total expected time in a single execution, that is maximum of \( E[T_i X_{1,i}] \) is the performance bottleneck.

### 3.2 Reliability prediction

In this section we describe how the mean and the variance of the number of visits obtained from DTMC analysis can be combined with the reliabilities of individual components to predict the reliability of the application. During a single execution, the reliability of the application, assuming all components fail independently of each other as well as in the successive executions, denoted by the random variable \( R \) is given by:

\[
R = \prod_{i=1}^{n} R_i X_{1,i} \tag{12}
\]

where \( X_{1,i} \) denotes the number of visits to the transient state \( i \) starting from state 1. The number of visits to the absorbing state \( n \) is always 1, i.e., \( X_{1,n} = 1 \). We emphasize that since
the number of visits to each component is a random variable (in general, except for the last component in the execution sequence), \( R \) itself is a random variable.

The expected reliability of the application is given by:

\[
E[R] = E\left[ \prod_{i=1}^{n} R_i^{X_{i,i}} \right] = \prod_{i=1}^{n} E[R_i^{X_{i,i}}] 
\] (13)

Thus to obtain the expected reliability of the application, we need to obtain \( E[R_i^{X_{i,i}}] \), which is the expected reliability of component \( i \) for a single execution.

From the Taylor series expression of the function of a random variable as presented in Equation (28) in Appendix A, we have:

\[
E[R_i^{X_{i,i}}] = R_i^{E[X_{i,i}]} + \frac{1}{2} (R_i^{E[X_{i,i}]})(\log R_i)^2 \text{Var}[X_{i,i}] 
\] (14)

which from Equation (2) can be written as:

\[
E[R_i^{X_{i,i}}] = R_i^{m_{i,i}} + \frac{1}{2} (R_i^{m_{i,i}})(\log R_i)^2 \sigma_{1,i}^2 
\] (15)

Since the number of visits to the absorbing state \( n \) is always 1, \( E[X_{1,n}] = 1 \), and \( \text{Var}[X_{1,n}] = 0 \), and hence \( E[R_n^{X_{1,n}}] = R_n \).

Equation (13) can thus be written as:

\[
E[R] = \prod_{i=1}^{n-1} (R_i^{m_{i,i}} + \frac{1}{2} (R_i^{m_{i,i}})(\log R_i)^2 \sigma_{1,i}^2)]R_n 
\] (16)

Equation (16) incorporates the impact of second-order architectural effects which are captured by the variance of the number of visits to a component on the overall reliability of the application. Capturing of the second-order architectural effects will provide more accurate predictions that are closer to the predictions provided by a composite model. It should be noted that the only source of approximation in the model is the result of the Taylor series cut-off.

If we ignore the second-order architectural effects, as captured by the variance of the number of visits to a module, the expected reliability of component \( i \) can be given by:

\[
E[R_i^{X_{i,i}}] \approx R_i^{m_{i,i}} 
\] (17)

Hence the expected reliability of an application with \( n \) components is given by:

\[
E[R] \approx \prod_{i=1}^{n-1} R_i^{m_{i,i}} R_n 
\] (18)

The component with the lowest expected reliability in a single execution, that is the minimum value of \( E[R_i^{X_{i,i}}] \) is the reliability bottleneck.

4 Illustrations

In this section, we illustrate the methodology for performance and reliability analysis described in Section 3 using the example application shown in Figure 2. The application has 10 components and the architecture of the application is specified by an absorbing DTMC shown in Figure 2. State 1 is the initial state of the DTMC, and state 10 is the exit or the absorbing state of the DTMC. We also assume that the reliabilities of the modules and the time spent in each module per visit are known, and are summarized in Table 2.

The intercomponent transition probabilities for the application shown in Figure 2 are given in Table 1. Table 1 lists the non-zero entries of the transition probability matrix \( \mathbf{P} \) for the application shown in Figure 2. The mean \( m_{1,i} \), variance \( \sigma_{1,i}^2 \), \( E[R_i^{X_{i,i}}] \), and \( E[T_i X_{1,i}] \) for component \( i \) are summarized in Table 3. The expected time to completion of the application is 14.9897 time units. The expected reliability of the application computed using Equation (16) is 0.8280, while the reliability of the application computed using Equation (18) is 0.8264. The reliability of the application can also be obtained using a composite model shown in Figure 3. In Figure 3, states \( C \) and \( F \) correspond to the successful completion and abnormal termination of the application respectively. Analysis
of the model shown in Figure 3, gives the reliability of the application to be 0.8299 [2]. Thus we see that the methodology reported in this paper provides a close approximation to the reliability computed using composite method. The composite method involves combining the architecture of the application, and the reliabilities of the individual components into a single model. This model can then be solved to obtain performance and reliability predictions. Though the composite method produces accurate results, whereas the hierarchical method gives only approximate solutions, the latter may be preferred to the former because of its ease of use in sensitivity analysis as elaborated in the sequel.

From Table 3 we can see that component 3 has the highest reliability of $\hat{R}_3 = 0.999$ and hence is the performance bottleneck, whereas component 5 has the lowest reliability of $\hat{R}_5 = 0.950$ and hence is the reliability bottleneck.

For the same application, we now consider the reliabilities of the modules as shown in Table 4. The expected reliability of each module computed using Equation (17) and Equation (15) are also shown in the table. Using the reliabilities computed using Equation (17) (without the second order or variance effects), components 3 and 5 have the lowest reliability and would be identified as bottlenecks. However, taking into consideration the second order effects as in case of Equation (15), component 5 has lower reliability than component 3, and hence is the only reliability bottleneck. Though the difference in the reliabilities of components 3 and 5 after incorporating the effects of variance is not very significant in this case, this example nevertheless highlights the fact that ignoring this effect can lead to misleading conclusions. Such mispredictions can be very costly, since

Table 1. Intermodal transition probabilities

<table>
<thead>
<tr>
<th>Module #</th>
<th>Reliability ($R_i$)</th>
<th>Time ($T_i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.999</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.980</td>
<td>2.00</td>
</tr>
<tr>
<td>3</td>
<td>0.990</td>
<td>3.00</td>
</tr>
<tr>
<td>4</td>
<td>0.970</td>
<td>2.00</td>
</tr>
<tr>
<td>5</td>
<td>0.950</td>
<td>2.00</td>
</tr>
<tr>
<td>6</td>
<td>0.995</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>0.985</td>
<td>2.00</td>
</tr>
<tr>
<td>8</td>
<td>0.950</td>
<td>3.00</td>
</tr>
<tr>
<td>9</td>
<td>0.975</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 2. Reliability and mean execution time per visit of modules

<table>
<thead>
<tr>
<th>Module #</th>
<th>Reliability ($R_i$)</th>
<th>Time ($\tau_i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.999</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.980</td>
<td>2.00</td>
</tr>
<tr>
<td>3</td>
<td>0.990</td>
<td>3.00</td>
</tr>
<tr>
<td>4</td>
<td>0.970</td>
<td>2.00</td>
</tr>
<tr>
<td>5</td>
<td>0.950</td>
<td>2.00</td>
</tr>
<tr>
<td>6</td>
<td>0.995</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>0.985</td>
<td>2.00</td>
</tr>
<tr>
<td>8</td>
<td>0.950</td>
<td>3.00</td>
</tr>
<tr>
<td>9</td>
<td>0.975</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>0.985</td>
<td>1.00</td>
</tr>
</tbody>
</table>
sensitivity analysis and optimization of the software. During the design stage it is common that the exact values of the input parameters for the model are unknown. Sensitivity analysis helps in analyzing the influence of the change in input parameters on the performance and reliability metrics. The hierarchical model facilitates sensitivity analysis along the following three dimensions:

- Study the effect of changes in the time spent in a module per visit on the expected time to completion of the application.
- Study the effect of changes in the reliability of a module on the expected overall reliability of the application.
- Study the effect of the changes in the workload of the application which is represented by the inter-component transition probabilities on the expected time to completion and reliability of the application.

We discuss the above three situations in the following three subsections.

### 5 Sensitivity analysis

Sensitivity analysis is another important aspect in performance and reliability analysis. It is very useful for bottleneck analysis and optimization of the software. During the design stage it is common that the exact values of the input parameters for the model are unknown. Sensitivity analysis can then help in analyzing the influence of the change in input parameters on the performance and reliability metrics.

The hierarchical model facilitates sensitivity analysis along the following three dimensions:

- Study the effect of changes in the time spent in a module per visit on the expected time to completion of the application.
- Study the effect of changes in the reliability of a module on the expected overall reliability of the application.
- Study the effect of the changes in the workload of the application which is represented by the inter-component transition probabilities on the expected time to completion and reliability of the application.

We discuss the above three situations in the following three subsections.

#### 5.1 Effect of change in the time spent in a module

The impact of a change in the mean time spent in module $j$, or $\tau_j$, on the overall expected time to completion of the application can be analyzed by differentiating the expected time to completion of the application as given by Equation (8) with respect to $\tau_j$. From Equation (8), we have:

$$dE[T] = m_{1,j}d\tau_j$$

where $dE[T]$ denotes the change in the expected time to completion of the application. Equation (19) shows that the impact of a change in $\tau_j$ on the expected time to completion of the application directly depends on the expected number of visits to module $j$. The revised time to completion of the application is then given by:

$$E[T_{\text{rev}}] = E[T_{\text{orig}}] + dE[T]$$

where $T_{\text{rev}}$ is the revised time to completion of the application and $T_{\text{orig}}$ is the original time to completion of the application. Based on the expressions developed in this section, we analyze the effect of changes in the mean time spent per visit in components 3 and 5 on the expected time to completion of the application. The choice of components 5 and 3 was completely arbitrary. We vary the mean time spent in each of these components from 1.00 through 10.00 units, one at a time and compute the expected time to completion of the application. In Figure 4, the solid line shows the effect of the variation in the expected time to completion of the application when the mean time spent in module 5 per visit is varied from 1.00 to 10.00 time units. In the same figure, the dotted line shows the expected time to completion of the application, when the mean time spent in module 3 per visit is varied from 1.00 to 10.00. As can be seen from the figure, variations in the mean time spent in com-

<table>
<thead>
<tr>
<th>Module #</th>
<th>$m_{1,i}$</th>
<th>$\sigma^2_{1,i}$</th>
<th>$E[R^X_{1,i}]$</th>
<th>$E[T_{1,i}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>0.0000</td>
<td>0.9990</td>
<td>1.0000</td>
</tr>
<tr>
<td>2</td>
<td>0.9077</td>
<td>0.6444</td>
<td>0.9819</td>
<td>1.8154</td>
</tr>
<tr>
<td>3</td>
<td>0.9107</td>
<td>0.5499</td>
<td>0.9909</td>
<td>2.7321</td>
</tr>
<tr>
<td>4</td>
<td>0.4184</td>
<td>0.3928</td>
<td>0.9874</td>
<td>1.2552</td>
</tr>
<tr>
<td>5</td>
<td>1.3504</td>
<td>0.7185</td>
<td>0.9337</td>
<td>2.7008</td>
</tr>
<tr>
<td>6</td>
<td>0.2510</td>
<td>0.2319</td>
<td>0.9987</td>
<td>0.2510</td>
</tr>
<tr>
<td>7</td>
<td>0.6155</td>
<td>0.6261</td>
<td>0.9968</td>
<td>1.2310</td>
</tr>
<tr>
<td>8</td>
<td>0.8737</td>
<td>0.4225</td>
<td>0.9564</td>
<td>2.6211</td>
</tr>
<tr>
<td>9</td>
<td>0.3831</td>
<td>0.2462</td>
<td>0.9904</td>
<td>0.3831</td>
</tr>
<tr>
<td>10</td>
<td>1.0000</td>
<td>0.0000</td>
<td>0.985</td>
<td>1.0000</td>
</tr>
</tbody>
</table>
Table 4. Example II: $R_i, E[R_i X_{i,j}]$ (Equation (17)) and $E[R_i X_{i,j}]$ (Equation (15))

<table>
<thead>
<tr>
<th>Module #</th>
<th>$R_i$</th>
<th>$E[R_i X_{i,j}]$ (Equation (17))</th>
<th>$E[R_i X_{i,j}]$ (Equation (15))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9500</td>
<td>0.9500</td>
<td>0.9500</td>
</tr>
<tr>
<td>2</td>
<td>0.9000</td>
<td>0.9088</td>
<td>0.9120</td>
</tr>
<tr>
<td>3</td>
<td>0.8554</td>
<td>0.8674</td>
<td>0.8732</td>
</tr>
<tr>
<td>4</td>
<td>0.9000</td>
<td>0.9569</td>
<td>0.9590</td>
</tr>
<tr>
<td>5</td>
<td>0.9000</td>
<td>0.8674</td>
<td>0.8708</td>
</tr>
<tr>
<td>6</td>
<td>0.9000</td>
<td>0.9739</td>
<td>0.9751</td>
</tr>
<tr>
<td>7</td>
<td>0.9000</td>
<td>0.9372</td>
<td>0.9405</td>
</tr>
<tr>
<td>8</td>
<td>0.9000</td>
<td>0.9121</td>
<td>0.9142</td>
</tr>
<tr>
<td>9</td>
<td>0.9000</td>
<td>0.9604</td>
<td>0.9618</td>
</tr>
<tr>
<td>10</td>
<td>0.9500</td>
<td>0.9500</td>
<td>0.9500</td>
</tr>
</tbody>
</table>

Figure 4. Effect of variations in time spent in modules per visit on time to completion

ponent 5 per visit cause a larger variation in the mean time to completion of the application, than the variations in the mean time spent in component 3 per visit. This is consistent with Equation (19), since the expected number of visits to component 5 is higher than the expected number of visits to component 3 as shown in Table 3.

5.2 Effect of change in the reliability of a module

The impact of a change in the reliability of module $j$ or $R_j$ on the overall expected reliability of the application can be obtained by differentiating the expected reliability of the application as given by Equation (16) with respect to $R_j$. Towards this end, we first let:

$$R_i' = R_i^{m_{i,j}} + \frac{1}{2} R_i^{m_{i,j}} (\log R_i)^2 \sigma_{1,i}^2$$

Equation (16) can then be written as:

$$E[R] = \prod_{i=1}^{n-1} R_i' R_n$$

The change in the expected reliability $dE[R]$ due to a change in the reliability of module $j$ is given by:

$$dE[R] = (m_{i,j} R_j^{m_{i,j}+1} + \frac{1}{2} \sigma_{1,j}^2 (m_{i,j} R_j^{m_{i,j}} - 1) (\log R_j)^2 + \frac{2 \log R_i}{R_j R_j^{m_{i,j}}} \prod_{i=1, i \neq j}^{n-1} R_i' R_n) dR_j$$

Equation (23) demonstrates that the change in the expected reliability of the application due to the change in the reliability of module $j$ follows a complex and non-intuitive dependence on the expected number of visits to module $j$, variance of the number of visits to module $j$ and the reliability of module $j$. The revised expected reliability of the application is given by:

$$E[R_{rev}] = E[R_{orig}] + dE[R]$$

where $R_{rev}$ denotes the revised reliability of the application, and $R_{orig}$ represents the original reliability of the application. Using the expressions developed in this section, we study the effect of variations in the reliabilities of module 5 and module 7 on the expected application reliability. The choice of modules 5 and 7 was completely arbitrary. In the Figure 5, the solid line shows the variation in the expected application reliability when the reliability of module 5, i.e., $R_5$ is varied from 0.90 to 0.99. In the same figure, the dotted line shows the expected reliability of the application when the reliability of module 7, i.e., $R_7$ is varied from 0.90 to 0.99. Figure 5 shows that variations in the reliability of module 5 influence the overall reliability in a stronger...
manner than the variations in the reliability of module 7. This is because the mean and the variance of the number of visits to module 5 is higher than the mean and the variance of the number of visits to module 7, while the reliability of module 5 is lower than the reliability of module 7.

5.3 Effect of change in the workload

The change in the workload manifests as the change in the intercomponent transition probabilities of the DTMC, and influences both the expected time to completion of the application as well as the expected reliability of the application. Expressions to analyze the effect of change in workload on the overall time to completion and the reliability of the application are cumbersome, and hence we demonstrate how the hierarchical model could be used to study the impact empirically. Referring to the intercomponent transition probabilities in Table 1, we conducted two experiments. In the first one we varied \( p_{4,5} \) from 0.10 to 0.90, while maintaining \( p_{4,5} + p_{4,6} = 1.0 \). In the second one, we restored \( p_{4,5} \) back to its initial value, and varied \( p_{5,7} \) from 0.10 to 0.90, maintaining \( p_{5,7} + p_{5,8} = 1.0 \). The expected time to completion, and the expected overall reliability was computed for both the experiments. The expected overall reliability of the application when \( p_{4,5} \) is varied from 0.10 to 0.90 is shown by the solid line, whereas the expected time to completion of the application when \( p_{5,7} \) is varied between 0.10 to 0.90 is shown by the dotted line. As in the case of reliability, variations in \( p_{4,5} \) have a stronger influence on the expected time to completion of the application than variations in \( p_{5,7} \).

6 Application to software systems

The application of the performance and reliability modeling techniques developed in this paper to real-life software systems hinges on our ability to obtain the architecture information of the system (as captured by the transition probabilities among its components) as well as the performance and reliability metrics of the individual components comprising the system. The architecture and the performance and reliability information can be obtained from a variety of sources depending on the phase of the software life-cycle during which architecture-based analysis is employed. Architecture-based analysis could be employed in various phases of the software life-cycle, starting from the design phase up to and including the operational phase.

If architecture-based analysis is to be employed during the design phase, the architecture information as exemplified by the intercomponent transition probabilities can be either “guestimated”, obtained from extensive consultations with experts who are intimately familiar with the system, or from prior release of the application. The information could also be obtained from the occurrence probabilities of various scenarios based on the operational profile of the system as illustrated in [30]. The performance and reliability met-
components, which takes into account the variance of the number of visits to each module. The method can also be used to identify performance and reliability bottlenecks. In addition, we have also developed expressions to determine the sensitivity of the performance and reliability predictions to the changes in the parameters of individual modules. We also demonstrated the use of the hierarchical model to analyze the impact of changes in the workload on performance and reliability predictions. Sensitivity analysis can be particularly valuable when architecture-based analysis is being used for the evaluation of various design alternatives, or in decision making with respect to which components should be picked off the shelf, which components should be developed in-house, etc.

The techniques described in this paper rely on the fundamental assumption that the architecture of the application and the time spent in each component per visit, as well as the reliability of each component are known a priori. However, more often than not, this information is not readily available, and needs to be extracted from some other sources. We describe various methods which could be used to obtain the information necessary for architecture-based analysis in Section 6. Our current research focuses on the development and validation of methodologies to obtain the information for architecture-based analysis from a variety of sources [6]. Also, we endeavor to relax the assumption that the components fail independently of each other, and develop models which incorporate dependencies in the failures of the components.

7 Conclusions and future research

In this paper we have developed an accurate hierarchical model for performance and reliability prediction of software systems based on their architecture, the time spent in each component per visit, and the reliabilities of the individual components. The method can also be used to identify performance and reliability bottlenecks. In addition, we have also developed expressions to determine the sensitivity of the performance and reliability predictions to the changes in the parameters of individual modules. We also demonstrated the use of the hierarchical model to analyze the impact of changes in the workload on performance and reliability predictions. Sensitivity analysis can be particularly valuable when architecture-based analysis is being used for the evaluation of various design alternatives, or in decision making with respect to which components should be picked off the shelf, which components should be developed in-house, etc.

The techniques described in this paper rely on the fundamental assumption that the architecture of the application and the time spent in each component per visit, as well as the reliability of each component are known a priori. However, more often than not, this information is not readily available, and needs to be extracted from some other sources. We describe various methods which could be used to obtain the information necessary for architecture-based analysis in Section 6. Our current research focuses on the development and validation of methodologies to obtain the information for architecture-based analysis from a variety of sources [6]. Also, we endeavor to relax the assumption that the components fail independently of each other, and develop models which incorporate dependencies in the failures of the components.

References


Appendix A: Expectation of a function of a random variable

In this section, we derive an expression for the approximate mean of a function $H(X)$ of a random variable $X$, in terms of mean $E[X]$, and variance $Var[X]$ of $X$ [1]. The Taylor series approximation of $H(X)$ about the mean $E[X]$ is given by:

$$H(x) = H(E[X]) + H'(E[X])(x - E[X])$$
$$+ \frac{1}{2}H''(E[X])(x - E[X])^2$$

which suggests the approximation:

$$E[H(X)] = H(E[X]) + \frac{1}{2}H''(E[X])Var[X]$$

We now state expressions for two functions of random variable $X$, namely, $H_1(X) = bX$ and $H_2(X) = a^X$, where $a$ and $b$ are constants.

From Equation (26), we have:

$$E[H_1(X)] = E[bX] = bE[X]$$

$$E[H_2(X)] = E[a^X] = a^{E[X]} + \frac{1}{2}(\log a)^2 a^{E[X]} Var[X]$$