NISTIR 7566

Markov Chain Analysis for Large-Scale Grid Systems

Christopher Dabrowski
Fern Hunt

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 7566

Markov Chain Analysis for Large-Scale Grid Systems

Christopher Dabrowski
Software and Systems Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8530

Fern Hunt
Mathematical and Computational Sciences Division
National Institute of Standards and Technology
Gaithersburg, MD 20899-8530

April 2009

Markov Chain Analysis for Large-Scale Grid Systems

Christopher Dabrowski and Fern Hunt

Abstract: In large-scale grid systems with decentralized control, the interactions of many service providers and consumers will likely lead to emergent global system behaviors that result in unpredictable, often detrimental, outcomes. This possibility argues for developing analytical tools to allow understanding, and prediction, of complex system behavior in order to ensure availability and reliability of grid computing services. This paper presents an approach for using piece-wise homogeneous Discrete Time Markov chains to provide rapid, potentially scalable, simulation of large-scale grid systems. This approach, previously used in other domains, is used here to model dynamics of large-scale grid systems. A Markov chain model of a grid system is first represented in a reduced, compact form. This model can then be perturbed to produce alternative system execution paths and identify scenarios in which system performance is likely to degrade or anomalous behaviors occur. The expeditious generation of these scenarios allows prediction of how a larger system will react to failures or high stress conditions. Though computational effort increases in proportion to the number of paths modelled, this cost is shown to be far less than the cost of using detailed simulation or testbeds. Moreover, cost is unaffected by size of system being modelled, expressed in terms of workload and number of computational resources, and is adaptable to systems that are non-homogenous with respect to time. The paper provides detailed examples of the application of this approach and discusses future work.

Table of Contents

1. Introduction

The long-term continued commercial success of grid technology will likely depend on the emergence of large-scale, decentralized grid systems in which large numbers of service providers and consumer clients enter into service-level agreements (SLAs) [Andr2007] to allocate grid resources. Here, as in other large-scale systems with decentralized control, the interactions of many consumers and providers, can lead to emergent global system behaviors that result in unpredictable, often detrimental, outcomes [Mill2006]. The movement toward realization of large-scale grid systems is evident in developments such as commercial cloud computing, in which mass computing services are being made available for sale. Clouds and other commercially-related developments likely foreshadow eventual creation of grid compute economies that operate on market principles. Having in place analytical tools to allow understanding, and prediction, of complex system behavior will be necessary to ensure availability and reliability of grid computing services in economic settings.

For these reasons, the development of analytical tools that take into account complex systems behaviors is even now seen as a priority. In particular, tools that can predict the impact on overall system performance of changes to key system parameters are of particular importance. Previous researchers have used simulation to study behavior of grid systems that utilize different economic strategies [Chun2002], [Yeo2005], [Mill2008]. Studies of failure scenarios in grid system such as [Mills2006] have shown that small variations in key variables can lead to alternative execution paths that yield large differences in overall system performance. Although more practical than using operational grid systems as testbeds, detailed, large-scale simulation that attempts to accurately reproduce system structure and component behavior is often a computationally expensive proposition when large numbers of alternative execution paths must be considered. Moreover, computational expense increases dramatically with increase in model size, a critical factor for analysis of large-scale grid systems which, even now, can exceed 10^5 computing resources [Carr2006, Raff2006].

To remedy this situation, this paper presents an approach in which discrete time Markov chain analysis is combined with a form of rapid, scalable, simulation. This approach, previously used in other areas, is used here to model dynamics of large-scale grid systems. In this approach, a state model of the system is first derived by observing system operation and then converted into a succinct Markov chain representation in which model scale is reduced by taking advantage of the stochastic characteristics of this model. The resulting model is expressed as a set of transition probability matrices (TPMs) that succinctly summarize system dynamics over different time periods. The TPMs represent an execution path that can be changed by altering, or *perturbing*, the values of individual transition probabilities in the TPM. By systematically perturbing combinations of transition probabilities, it is possible to model alternative execution paths, each of which lead to a different evolution of a grid system over time. Among these are execution paths where failure to meet fundamental guarantees of service causes system performance to significantly degrade.

The approach presented in this paper allows expeditious investigation of a large number of alternative system execution paths and identification of paths, or scenarios, in which failure to meet service guarantees adversely affects overall system performance. In this way, the Markov chain analysis can be used to predict how a larger system will react when key service guarantees are not met. Though computational effort increases in proportion to the number of paths modeled, the cost of using Markov chains is far less than the cost of searching the same problem space through detailed simulation or use of testbeds. Moreover, computational cost is unaffected by size of system being modeled, where size is expressed in terms of workload and number of computational resources. The approach can also be adapted for cases in which transition probabilities change with (e.g. are non-homogenous with respect to) time and workload. The paper concludes with thoughts on evolving this approach to serve as a concrete tool for analysis of system complexity in large-scale grid systems.

The plan of this paper is as follows. Section 2 summarizes previous work on using Markov chain analysis and related techniques in distributed computer systems. Section 3 more precisely describes the problem being investigated through Markov chain analysis. The section defines fundamental guarantees of service that large-scale grid systems will need to provide to their customer base and which are the basis for analysis in this paper. Section 4 describes the process of creating a state model of a grid system, extending this model to be a Markov chain model, and its use in modeling system evolution. This section describes how the model is reduced in size and adapted for situations where the Markov chain is non-homogenous with respect to time. Section 5 describes the method of perturbing the Markov chain TPM to simulate alternative execution paths that violate service guarantees defined in section 3. Section 6 presents results of using the methods described in sections 4 and 5 to predict system evolution and compares these results with those produced by more detailed simulation. Section 7 presents conclusions and future work.

2. Related Previous Work

Markov Chain analysis is well established analytical tool for understanding dynamic systems behavior [Keme1976]. This section briefly reviews work on use of Markov chains, focusing on two outstanding problems relevant to a Markov model of a grid system: methods to reduce model size and perturbation analysis techniques that reduce perturbation space size.

Discrete Time Markov Chains (DTMCs) have been applied to a variety of practical problems in real-world domains. Markov chain analysis has long been used in manufacturing [Dall1992] for problems such as transient analysis of dependability of manufacturing systems [Nara1994], [Zaka1997] and deadlock analysis [Nara1990]. Li et al. [Li2008] describes recent uses of Markov chains to model split and merge production line processes [Helb2000], [Tan2000], [Helb2003], [Li2005], [Diam2006], [Liu2008]. Similarly, Li describes use of Markov chains to model part quality defects [Kim2005], [Coll2005a], [Coll2005b]. In communications networks, [Cass1990] has used Markov approaches to model mean time to failure of network components, while [Bala1994] used Markov chain analysis to model link reliability. Markov chains have been used for multi-processor computer architectures to examine fault-tolerance [Aupp1991] and performance [Chio1993], in parallel systems to investigate performance in queuing networks [Jonk1994], in real-time process control systems to model fault tolerance and performance [Triv2004], and software systems reliability measurement [Lapr1992] [Gose2001]. In grid computing, Markov chains have been used to model workload for scheduling [Song2004] and load balancing [Akio2003]. However, unlike these efforts or those that quantitatively estimate performance or reliability, this work uses Markov chain modelling to understand alternative system behaviors that may occur as a consequence of significant system-wide events or decisions: in this case, the failure to meet fundamental service guarantees for grid systems.

The combinatorial increase of the number of states in DTMC models for large problems has long been widely recognized as a barrier to practical use of Markov Chain analysis. To solve this problem, the concept of *lumping* states with similar characteristics into larger aggregated units was first introduced by Kemeny and Snell [Keme1976]. The problem has been worked on extensively since. [Sieg1992] and [Nico2004] surveyed lumping approaches that rely on model structure symmetry that can be exploited to reduce size. Among these, [Buch1992] [Buch1995] reduced size by exploiting state hierarchies in Markov models to combine states. Model structure was also leveraged to reduce Markov chain size by using group-theoretic concepts [Aupp1991], Stochastic Activity Nets [Sand1991], stochastic colored nets [Chio1993], generation of lumped models that approximate Markov properties [Bala1994], and by using a reward variable structure to identify symmetries that can lead to generation of models of reduced size [Obal2001]. In [Jaco2007], a method is proposed for using equivalence classes of eigenvectors to partition a Markov state space into lumps. Work on this problem continues. Fortunately, in the model we present, the number of states is readily reducible using the stochastic characteristics of Markov chains or the basic theorem provided in [Keme1976] that is described in section 4. While the number of states in our model did not prove to be a barrier, the size of the perturbation analysis problem did.

Perturbation analysis of discrete time Markov chains has been the topic of theoretical work in the last three decades [Schw1968]. Like the problem of model size, the size of a typical perturbation space may quickly become computationally intractable, if there are many combinations of alternative system variable values to consider. To attack this problem, [Ho1985], [Suri1987] [Suri1989] [Ho1991] and others advanced the idea of perturbation analysis of discrete event systems by calculating system performance gradients based on key decision parameters. This approach estimated the sensitivity of changes to decision parameters in order to optimize system performance. Gradient-based approaches had the potential to reduce the perturbation space because they needed to observe as few as one execution path of a system. This approach was adapted for Markov chains in [Ho1988] by estimating gradients for alternative execution paths. However, this method still required some way of determining performance estimate vectors (performance potentials) for state spaces in various problem domains, which usually necessitate extensive sampling and data gathering. In addition, questions remained about the computational complexity of gradient calculation algorithms [Suri1989].

Methods for estimating performance gradients on the basis of limited sample system executions that are applicable to subsets of Markov chain that use control policies were also explored by [Cao1998][Baxt2001], and others. In [Baxt2001], a brief summary of some of these works is provided. Most recently in Cao [Cao2005] [Cao2008], the gradient-based approach was extended to reduce problem size in Markov chain models by grouping state transitions on the basis of events to evaluate control policies. In this event-based approach, gradients were calculated by aggregating estimates of performance for alternative execution paths taken under different policies. The gradients could then be used to evaluate different control policies. The approach was thought to scale with the number of events and system size, but the issues of determining performance vectors and efficient gradient calculation still remained, and, in addition, policy iteration algorithms need to be developed. Further, not all problems were reducible to a form which allowed tractable calculation of gradients for individual policies. In sum, while gradient-based perturbation algorithms have the potential of reducing the state space, they also introduce not inconsiderable computational issues and apparently are not applicable to all Markov problems. In other words, the "no free lunch" theorem applies [Wolp1997]. Moreover, the gradient-based approach appears more immediately applicable to specifically-defined optimization problems that depend on relatively few system parameters, rather than the more general problem of assessing alternative execution paths. Yet, the potential of gradient-based approach cannot be completely ruled out and may be a factor in future work.

Instead, the approach presented in this paper avoids the computational difficulties of gradient-based methods to allow examination. The potential problem of size in Markov models of grid system dynamics is mitigated through a straightforward, readily lumpable problem representation and an intuitive, limited search strategy. While this approach does not completely solve the issue of problem size (e.g., there is no free lunch), the resulting analysis yields comparable results to more detailed simulation at a small fraction of the computational cost. It therefore constitutes a viable analytical tool for study of large-scale dynamic systems.

3. Questions to be Answered Through Perturbation of Markov Chains.

It is convenient to organize analysis of grid computing systems on the basis of certain guarantees of service that grid systems must provide to their users in order to successfully support economic activity. These guarantees constitute basic requirements for grid computing systems, which if not met, would render a grid system useless. The extent to which a grid system fulfills, or does not fulfill, these guarantees impacts system performance. The ability of Markov chain analysis to accurately model and predict how the system behaves if these guarantees are not met is an interesting and relevant question. Three guarantees may be described.

- First, a grid system must guarantee that current information about what grid computing services exist is available to users. In grid systems, this guarantee is fulfilled through service discovery mechanisms that locate needed services and make information about them available to users. The *service discovery guarantee* refers to the ability of a grid system to provide necessary information about grid computing services, including relevant updates, which users require to make decisions.
- Second, if a user has found a needed service, the service is available (not reserved for other tasks), and the user is qualified to use the service, then the grid system should allow the user to engage that service. This is referred to as the *service engagement guarantee*. To be qualified, the user must possess security and

administrative access, and must also have the economic means to afford the service. The service engagement guarantee is meant to ensure that users and providers of services who should logically cooperate, in fact do so. In most cases, engagement of a service is signified by the formation of a *service level agreement* (SLA), which reserves the service for the user and specifies a fee to be paid.

- The third guarantee is the *service fulfillment guarantee*, which simply states that once a user has engaged a service, e.g. and SLA has been formed, the terms of the related agreement should be fulfilled by both provider and user.

Understanding and predicting the consequences of not fulfilling these guarantees is an important analysis problem. Particularly important is understanding of how performance of a grid compute economy degrades as the extent of guarantee fulfillment decreases incrementally. Administrators of grid systems as well as providers and users need to understand how a different level of non-fulfillment of each guarantee is likely to affect a system. At what point of incremental increase does system performance begin to degrade rapidly? What specific actions by providers or consumers affect non-fulfillment of a particular guarantee? Answering questions like these by taking an actual production system offline to use as a testbed is impractical for obvious reasons. Simulation is a plausible alternative, and has been used successfully to estimate impacts of failure scenarios in grid systems [Mill2006, Mill2008]. However, simulation may require executing many repetitions using a detailed compute-intensive model. If there are a substantial number of system parameters to vary, then analysis may either take considerable time, be limited to a restricted number of alternative execution paths, or both. Both testbeds and simulation thus have limitations in answering these questions. Markov chain analysis provides a viable alternative for obtaining more detailed understanding of effects of not fulfilling grid service guarantees. The remainder of the paper illustrates the basic approach and shows preliminary results.

4. The Markov Chain Model

The behavior of a large-scale grid system can be modeled in terms of the computing tasks executing in the system at any time. Each task progresses through a life cycle in which it is first submitted by a user, service providers are discovered to run the task, an SLA is negotiated with selected provider(s), and the task is either executed to completion or fails to complete. The state of the grid system can be described by the states of all the tasks that are in the system at some particular time. This section first describes the state transition model for an individual task and then shows how the aggregate of many tasks states are represented in a Markov chain model. This potentially very large model can be compacted, or lumped, into a more concise representation in which the dynamics of the grid system can be studied in a meaningful way.

4.1 Representing the Lifecycle of a Task as a State Model

The lifecycle of an individual task can be represented in seven states, shown in Figure 1. This model is derived from a large-scale model of a grid system [Mill2008, Dabr2008] that simulates the system operation over an 8-hour day. Three of these states in this model—*Discovering*, *Negotiating*, and *Monitoring*, or the task execution phase—can be further decomposed into sub-states. These decompositions constitute a hierarchy of 27 additional states, for total of 34 states. The interested reader is referred to Appendix A for a description of the larger model and its logical correspondence to the seven-state model. Appendix A also provides a brief analysis of this decomposition and suggests some directions for future work. Because the 34-state model directly corresponds to, and can be transformed to and from, to the simpler seven-state model, the latter is retained for the remainder of the paper.

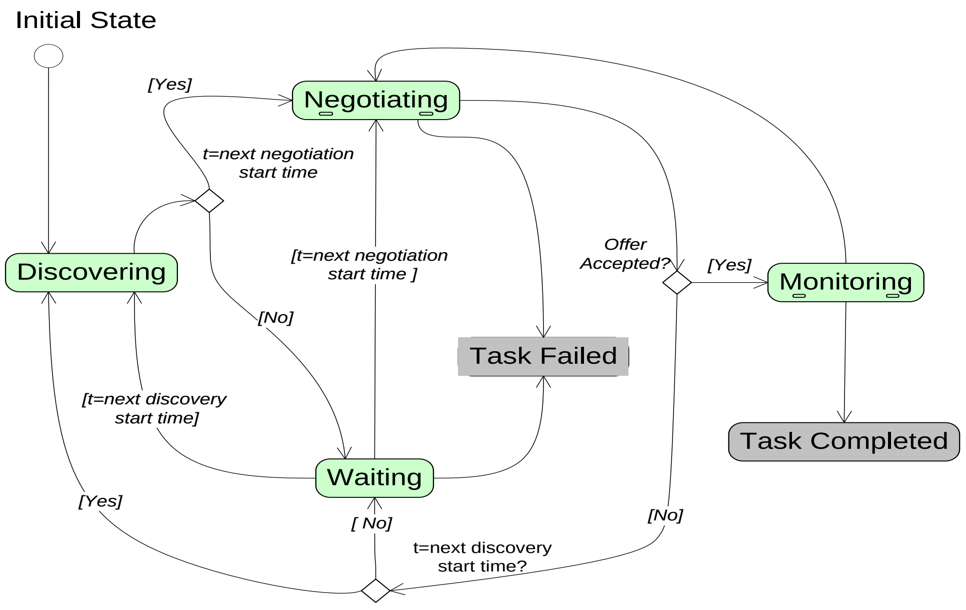

Figure 1. State model of grid compute economy simulation model described in [Mill2008].

The high-level model representation may be described as follows. In the *Initial* state, a task has not yet entered the grid system. Each task is assigned an arrival time and deadline from exponential distributions [Mill2008]. At the arrival time, the task automatically transitions to a *Discovering* state. In *Discovering*, the task client attempts to discover eligible providers with sufficient computing resources to execute the task. After discovery actions conclude, the task may either transition to *Negotiating* or *Waiting*. Tasks enter the *Negotiating* state at regular intervals. A task that has completed *Discovering* and found at least one provider enters *Negotiating* if the interval has elapsed; otherwise it goes into the *Waiting* state. In *Negotiating*, clients order discovered providers they are qualified to use, e.g., can afford, on the basis of anticipated cost. They contact each provider, one at a time, and offer an SLA to execute the task for a fee. Once a provider accepts, negotiation ceases and the task enters the *Monitoring* state, during which the task is either blocked on an execution queue or executing. If negotiations fail (i.e., no provider can be found to accept an SLA), the task goes from the *Negotiating* state to either *Discovering* or *Waiting*. As in *Negotiating*, a task enters *Discovering* at regular intervals. If negotiations fail, a task transitions from *Negotiating* to *Discovering* if the start time has arrived. Otherwise it transitions to *Waiting* and remains until the next *Discovering*, or *Negotiating*, start time.

A task that has obtained an SLA and transitioned from *Negotiating* to *Monitoring* enters the *Completed* state, if task execution is successful. If execution fails, the task falls re-enters the *Negotiating* state. Tasks may also transition into the *Failed* terminal state from either the *Negotiating* or *Waiting* states. This occurs when it becomes impossible to complete the task by its deadline, as explained in [Mill2008, Dabr2008]. Both *Completed* and *Failed* are terminal states from which tasks cannot leave once they enter.

4.2 Evolving the State Model to a Markov Chain Model

A Markov chain has the property that the probability of transition between any two states depends entirely on circumstances in the state from which the transition originates and not on the previous history of the process. More formally, given a sequence of states X_1, X_2, …… X_n, the Markov Property is given as:

$$\Pr\left(X_{n+1} = x \mid X_n = x_n, \ldots, X_1 = x_1\right) = \Pr\left(X_{n+1} = x \mid X_n = x_n\right) \qquad (1)$$

The state model depicted in Figure 1 satisfies the Markov property. Careful review of the preceding description shows that the decision to transition to another state depends only on circumstances of the state the task is currently in. These circumstances include whether a time interval has elapsed, an SLA has been secured, task execution has succeeded or failed, etc.

In a Markov chain, probabilities are associated with transitions between states. To calculate state-to-state transition probabilities, transition frequencies are first summed over a simulated eight-hour day using the model described in [Mill2008, Dabr2008]. This is done by determining where state transitions occur in executing model code and inserting counters at those places. In our experiments, frequencies were summed for all state transitions over 50 repetitions at a 75% load level over 36000 s (10 hours: 8 hours + two extra hours for late tasks). State transition probabilities were derived as follows. Given states s_i, s_j, $i, j = 1 \ldots n$ where n=7, p_{ij}, is the probability of transitioning for state i to state j, written as $s_i \rightarrow s_j$. This probability is estimated by calculating the frequency of $s_i \rightarrow s_j$, or f_{ij}, divided by the sum of the frequencies of s_i to all other states s_k, or

$$p_{ij} = \frac{f_{ij}}{\sum_{1 \le k \le n}^{n} f_{ik}} \qquad (2)$$

Here i and j may be equal, to allow for transition of a state to itself, or *self-transition*. A self-transition occurs when a task remains in a state longer than a specified interval (equal to a Markov simulation discrete time step, d_{ts}, described below). The resulting TPM is a 7 × 7 stochastic matrix, shown in Figure 2. Here rows stand for the state the transition originates from, or *from state*, and columns represent states the transition goes to, or *to state*. Each cell in a TPM represents a p_{ij}, where i and j are from and to states, respectively. As in any stochastic TPM, the transition values of all columns in a row must sum to 1.0. The only exception to this procedure involved arrival times of tasks into the grid system, described above. Here, the Markov chain process was altered to reproduce exactly the exponential arrival times of the large-scale simulation.

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0.9697	0	0.0303	0	0	0	0
Waiting	0	0.8363	0.0673	0.0918	0	0	0.0046
Disc	0	0.0355	0.6714	0.2931	0	0	0
Ngt	0	0.4974	0.0182	0.2882	0.1961	0	0.0001
Mon	0	0	0	0.0003	0.9917	0.0080	0
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

Figure 2. Summary stochastic transition probability matrix (TPM). This is a summary TPM that is weighted average on the basis of separate TPMs for five equal time period divisions over the 36000 s duration. Individual p_{ij} for each of the five time periods are weighted on the basis of relative number of transitions in their respective periods. Matrices for the five time periods appear in Appendix B.

The Markov chain and related TPM can be further classified. Careful analysis of the description of the state model in section 4.1 and the structure of the matrix in Figure 2 shows that tasks can enter the *Discovering*, *Waiting*, *Negotiating*, and *Monitoring* states multiple times, but always remain temporarily. At some point they enter either the *Completed* or *Failed* state, where they remain permanently, or are *absorbed* (These states are considered *absorbing* states, in which only self-transitions are possible). A Markov chain with these characteristics is called an *absorbing chain* [Keme1976] that can be divided into a *transient* part (the *Discovering*, *Waiting*, *Negotiating*, and *Monitoring* states), and an absorbing part (with two absorbing states, *Completed* and *Failed*). This characterization is born out in the Markov chain simulation described below while the summary TPM in Figure 2 is useful for illustrating concepts and for certain analytical studies of the system, it requires further elaboration to model non-homogeneity.

4.3 Reducing Problem Size and Handling Time Non-Homogeneity

In the large-scale simulation [Mill2008, Dabr2008] loaded at 75%, there are typically over 400 tasks, each of which progresses through the seven states. If all are modeled simultaneously, this means there are 7^{400+} possible combinations of states for all the tasks. This number is clearly too large for easy analysis. However, the stochastic nature of Markov Chains allows one to consider the distribution of the 400+ tasks among the seven states. It is easy to see that a system-wide state can be represented in terms of the proportion of the 400+ tasks allotted to particular states. In this way, it is possible to represent the system state as a seven-element vector in which the value of each vector element represents the proportion of tasks {0, 1.0} in one of the seven states. This method of representing the system state is a simplification that can be performed in addition to combining 34 states into seven, described in Section 4.1 and Appendix A.

In the large-scale simulation [Mill2008, Dabr2008], task submission times varied exponentially over the simulated day, with mean task start time at t=3600 s (end of the first hour). This distribution resulted in different workload levels at different times in the day and caused transition probabilities over the 36000s-period to vary. Therefore, different TPMs were actually in force at different times, making the system non-homogenous with respect to time. For this reason, more accurate simulation results for the transient behavior of the system were obtained by creating time-period partitions and computing a separate TPM for each period. In this experiment, frequencies were summed separately for five time periods of 7200 s each[1]. These matrices, shown in Appendix B, allow a representation of our model as a *piece-wise homogenous* Markov chain having a bounded number of pieces [Rose2004] corresponding to the time periods. The individual transition probabilities in the TPMs for these five periods can be weight averaged on basis of relative transition frequencies in each period to reproduce the summary stochastic TPM, shown in Figure 2. In the summary matrix, each probability of transition, , is computed as follows:

$$p_{ij} = w_i^1 p_{ij}^1 + w_i^2 p_{ij}^2 + \dots . w_i^{nper} p_{ij}^{nper} \qquad (3)$$

in which each w^l_i represents the weight for a row *i* in time period *l*, $l \in \{1.. \, nper\}$where *nper*=5. Each w^l_i is computed by

$$w_i^l = \sum\nolimits_{1 \le j \le n} f_{ij}^{\,l} \Big/ \sum\nolimits_{1 \le tp \le nper} \sum\nolimits_{1 \le j \le n} f_{ij}^{tp} \qquad (4)$$

where each f^{tp}_{ij} is the frequency of transition from state *i* to state *j* in time period *tp* and *n* is the dimension of the matrix (*n* = 7). While this summary matrix is useful for illustrative purposes and for computing certain theorems, the Markov simulations described below used the five time period matrices.

4.4 Using A Sequence of Markov Chain TPMs to Simulate a Dynamic System

A well-known use of stochastic TPMs in a Discrete Time Markov Chain is to describe how a dynamic system changes over time in discrete time steps each, where each step represents a fixed amount of time. In this experiment, a discrete time step is chosen to represent 85 s, or d_{ts} = 85 (which also determines length of time for identifying frequency of self-transitions above)[2]. Hence, if a time period covers a duration of d_{period} = 7200 s, each of the five time-period matrices represent S= d_{period} /*h* steps or 85 steps.

As indicated above, the system state can be summarized in a vector *v* having seven elements, where each element represents the proportion of tasks in one of the seven states. Using equation (5), a vector , which represents the system state at time step *m*, is multiplied by the TPM for the applicable time period *tp* to produce a new system state $_{+1}$, to evolve the system over a single discrete time step.

$$(Q^{tp})^T * v_m = v_{m+1}, \text{ where } tp = \textit{integral value}\ (m/S) + 1 \qquad (5)$$

[1] Different numbers of time periods were attempted, including three, 10, and 15; however, five provided the most accurate results. Devising a method of selecting an optimal number of time periods is a problem left for future work.
[2] Different values for d_{ts} were tried, ranging from 50s to 100s, with d_{ts}= 85 s providing the best results. As with the number of time periods, devising a method of selecting d_{ts} is left for future work.

where *T* indicates a matrix transpose. Starting with v_0, which represents a system state with a value of 1.0 for the *Initial* state and 0 for all others, equation (3) is repeated for 339 time steps (representing 28,800 s or a simulated 8-hour day). This results in a system state vector, v_{339}, in which the sum of the proportion of tasks in the *Completed* and *Failed* states approaches 1, while other states are at 0. A goal of Markov chain analysis is to execute this procedure with a set of time period TPMs derived from a real-world system (or, in this case, the large-scale simulation) in order to approximate the operation of that system. Figure 3 compares the proportion of tasks in the *Completed* and *Failed* states after executing the large-scale simulation in [Mill2008, Dabr2008] for 28,800 s with the values for these states in the Markov chain simulation over 339 steps (28,800 s).
This paper argues that a piecewise homogenous Markov chain can approximate the transient behavior of a real-world grid system (for which the large-scale simulation is a proxy). By applying (5) to a set of perturbed TPMs to simulate alternative evolutions of a grid system, one can model, or predict, the effects of changed system capabilities, undesirable behaviors, and events of interest. In this study, we are interested in perturbing selected rows of the TPM set to represent changes in the ability of the system to fulfill the three grid system guarantees described in section 3, and then to predict the impact of these changes on the final system state (proportion of tasks in *Completed* or *Failed* states.

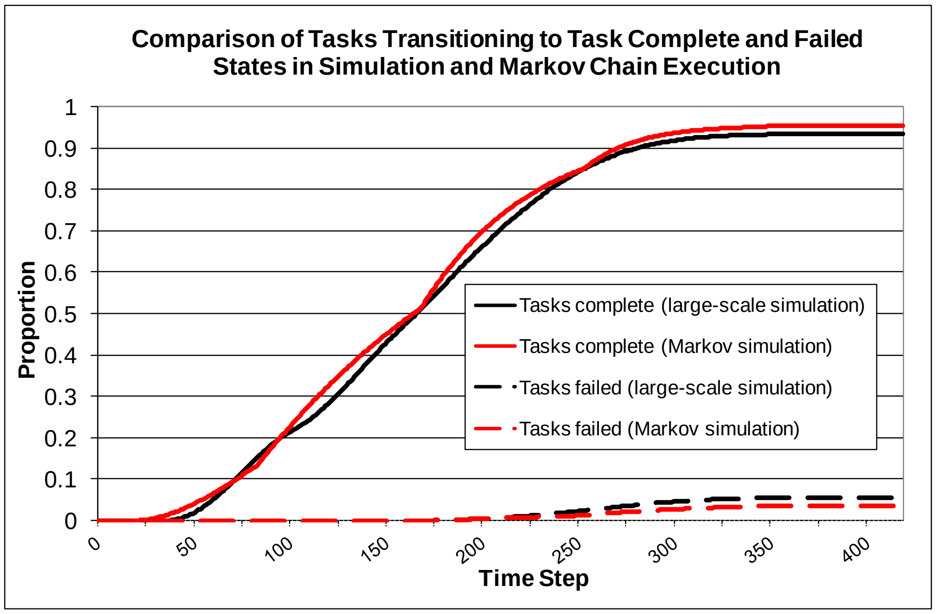

Figure 3. Comparison of system evolution of *Completed* and *Failed* states over time for the large-scale and Markov chain simulations. TPMs for 5 time periods in Appendix B were used.

5. Method of Perturbation

The algorithm for incrementally perturbing selected rows is intended to predict broad trends rather than precise outcomes. It is a limited, brute-force search that is restricted in order to conserve resources, while exploring a reasonable range of alternatives. The algorithm permits simultaneous perturbation of combinations of two rows in order to capture situations where inter-row dependencies exist. In each perturbed row, each row element, corresponding to a column, with a probability of transition greater than zero is selected in turn for incremental increase. At the same time, the transition probabilities of one or more other row elements with non-zero values are decreased by a total equal to the increase, so that all elements in the perturbed row continue to sum to one. For a set of time-period matrices, these changes are applied to each matrix in the set. Each combination of altered transition probabilities represents a different execution path that the system may take.

The algorithm requires that a user first select a *primary row*, *r*, to perturb. The *secondary rows, s*, to be perturbed are then automatically determined, as described below. The user also must select a *perturbation limit* ***L***, on how far transition probabilities can be perturbed and also select the incremental amounts by which primary and secondary rows will be perturbed. These decisions define the extent and granularity of the perturbation that will take place. An overview of the procedure is provided below. For more detail, see Appendix C which describes the algorithm in detail and provides a summary table of term definitions. This section also discusses the computational effort required to apply the perturbation algorithm to the Markov chain simulation. This effort is a small fraction of what would be necessary to explore the same set of alternative behaviors using the large-scale simulation. The next section, Section 6, then presents results of applying the perturbation algorithm and compares these results with those produced by the large-scale simulation.

5.1 Overview of Perturbation Algorithm

In the primary row, starting from numerically lowest row element, each element having a positive transition probability is used in turn to determine as the *primary increase column*, $c^{\uparrow}$. In this column, the transition probability is raised by a gradually increasing amount, m_{prim} up to the limit ***L***. These increases occur in increments defined by a *primary increase amount*, v_{prim}. At the same time, the other elements in the primary row are reduced by proportions of m_{prim} determined by weight factors, as follows. Each *non-increase column* in turn is selected as the primary column to decrement, termed a *primary sink column*, $c^{\downarrow}$. For the primary sink column, a *sink weight*, *w*, is selected from a predetermined set of sink weights called the *sinkWeightSet*. In the experiments reported here, the sinkWeightSet consisted of {0.2, 0.4, 0.6, 0.8, 1.0}. The probability of transition for $c^{\downarrow}$ is reduced by the amount $w \cdot m_{prim}$. The remainder of the weighted reduction, or $(1.0 - w)\cdot m_{prim}$, is distributed to the other *non-sink columns*. A perturbation of primary row *r* may be summarized by

$$p_{rj}^{(new)} = \begin{cases} p_{rj}^{(old)} + m_{prim} & j = c^{\uparrow} \\ \left[p_{rj}^{(old)} - w \cdot m_{prim}\right]^{+} & j = c^{\downarrow} \\ \left[p_{rj}^{(old)} - (1-w)\cdot m_{prim} \dfrac{p_{rj}^{(old)}}{\sum\limits_{k \neq c^{\uparrow}, c^{\downarrow}} p_{rk}}\right]^{+} & j \neq c^{\uparrow}, c^{\downarrow} \end{cases} \qquad (6)$$

for $p^{(old)}_{rj} > 0$ where $[a]^{+}=a$ if $a>0$ and $[a]^{+}=0$ otherwise. Similarly perturbation of the primary increase column $c^{\uparrow}$ ceases if the perturbed value would exceed 1.0. Of course, if *w* is 1.0, or if the primary increase column $c^{\uparrow}$ and primary sink column $c^{\downarrow}$ are the only columns with non-zero transition probabilities, the primary sink column bears the entire reduction. (Please see Appendix C for further details.)

The secondary row *s* can be selected on the basis of either: (a) the numeric value of the primary increase column $c^{\uparrow}$, if it is not equal to the number of the primary row, or $c^{\uparrow} \neq s$ (otherwise no secondary row is selected); and (b) by strength of association, using the total value of transition probabilities between the two states the rows represent and (if known) the number of transitions that occur between these states. The default method is (a); and this was used for the results reported below. Thus in the primary row *r*, as each primary increase column, $c^{\uparrow}$, is selected, a different secondary row is also selected. As in the primary row *r*, each positive row element in the secondary row, *s*, is selected in turn for increase, and the corresponding column is designated as the *secondary increase column*, $d^{\uparrow}$. However, in the secondary row, the perturbation is simpler--the transition probability of a secondary increase column, $d^{\uparrow}$, is raised by a *secondary increment amount*, , in 5 equal steps to produce successively perturbation amounts, , up to ***L***. As in the primary row *r*, transition probabilities in the remaining columns of the secondary row are decreased by an equal amount; though here the amount of decrease for each column is assigned in proportion to the relative value of its transition probability (similar to non-sink columns in the primary row). To summarize,

$$p_{sj}^{(new)} = \begin{cases} p_{sj}^{(old)} + m_{\sec} & j = d^{\uparrow} \\ \left[p_{sj}^{(old)} - m_{\sec} \dfrac{p_{sj}^{(old)}}{\sum\limits_{k \neq d\uparrow} p_{sk}} \right]^{+} & j \neq d^{\uparrow} \end{cases} \tag{7}$$

Each combination of variable assignments for the primary increase column, primary sink column, and sink weight in the primary row and the secondary increase column and secondary increase amount in the secondary row (if any) is considered a unique perturbation combination, labelled {*r*, $c^{\uparrow}$, $c^{\downarrow}$, *w*, *s*, $d^{\uparrow}$, }. For each perturbation combination, a separate perturbation sequence of [*L*/] steps is carried out in the primary row. In each element of the perturbation sequence, the value of the primary increase column, $c^{\uparrow}$, is successively raised by while the primary sink column, $c^{\downarrow}$, and non-sink columns, if any, are decremented as described above. For a set of time period matrices, this perturbation sequence is applied simultaneously in each matrix in the set. For each assignment of incremental values in the perturbation sequence, the Markov chain simulation procedure described in section 4.4 is carried out for a time-period matrix set. Each such execution represents a potentially different execution path for the system. The incremental increases in the perturbation sequence continue until the transition probability in the primary increase column $c^{\uparrow}$, reaches ***L*** or 1.0 in each time-period matrix. Thus if L=0.25 and = 0.01, there are 25 Markov chain simulations in a perturbation sequence for each perturbation combination of column assignments in the primary and secondary rows.

5.2 Implementing this Approach to Perturb the TPM

In this way, the effects of a reasonably wide range of perturbation combinations can be explored. Carrying out the sequence of perturbations for each perturbation combination can yield potentially interesting alternative behaviors, though only a subset can be relevant. The perturbation method was used to predict the result of failing to fulfill the three service guarantees described in section 3. To do this, we used the time-period matrix set in Appendix B (summarized by the weight-averaged matrix in Figure 2), together with the default method (a) for secondary row selection. The sink weight set and parameters values for $\boldsymbol{L}$, v_{prim}, and v_{sec} described above were also used. Applying the perturbation method resulted in generation of 2805 perturbation combinations and perturbation sequences consisting of 89,750 simulations, which required 3354 s (56 minutes) of computation time using an Intel Xeon MP processor. This is a substantial amount, but less than 0.5% of the time (205 hours) that the large-scale simulation needed to show behaviors described below in which the service guarantees were violated. Table 1 shows the total number of perturbation combinations for the rows of this matrix set. Details of perturbation combinations are provided in Appendix D. Analysis of the results of these Markov chain simulations and the execution of corresponding perturbations in the large-scale simulation is provided in the next section.

Table 1. CPU resources used for perturbing of rows of the Markov chain matrix, with secondary row selected using method (a). Primary rows perturbed to maximum value in increments as specified. Secondary rows perturbed to a maximum value of 0.25 in 0.0625 increments. The details of calculating the number of perturbation combinations and perturbation sequences is provided in Appendix D. Actual number of perturbation sequences carried out is in parenthesis.

Primary Row	L / v_{prim}	Secondary Rows	Number perturbation combinations	Number Perturbation sequences	CPU time used
1	0.031 / 0.001	3	80	2480	82.39 s
2	0.25 / 0.01	3, 4	425	10625	373.44 s
3	0.25 / 0.01	2, 4	460	11500	369.69 s
4	0.25 / 0.01 and 0.5 / 0.01	2, 3 2, 3	785 785	19625 39250	789.17 s 1480.46 s
5	0.25 / 0.01	4	270	6750	260.15 s

6. Comparing Perturbations of the Markov Chain and Large-Scale Simulation

This section presents the results of using the perturbation method described in the previous section to predict system performance when the service guarantees described in section 3 are violated. The section provides a detailed analytic comparison of these results with the results produced by the large-scale simulation.

The systematic perturbation of the TPMs revealed a wide range of behaviors. A subset of these behaviors, corresponding to a subset of the total perturbation combinations discussed above show what might occur if the service guarantees were violated. These perturbation combinations correspond to service guarantee violation scenarios of interest.

In the Markov chain model, violation of the Discovery Guarantee corresponded to a subset of perturbation combinations for rows 1-4 of the TPMs in which tasks were prevented from transitioning to the *Discovering* state. Two of these combinations are presented here.

Failure to fulfil the Service Engagement Guarantee was enacted by reducing probability of transition from the *Negotiating* state to the *Monitoring* state in row 4. Perturbation combinations that reduced this probability represented a violation scenario in which SLAs were not granted even though users and providers might be eligible for, and should be able to enter into, agreements. Violation of the Service Fulfilment Guarantee was enacted by reducing the probability of transition from the *Monitoring* state to the *Completed* state in row 5, while increasing the probability of transitioning from *Monitoring* to another state. This violation scenario corresponded to aborting a task that was either executing, or in a waiting queue. The results of these perturbations of the Markov chain are shown in graphs of perturbation sequences for relevant perturbation combinations.

To compare the results of the perturbations to the Markov chain with similar changes to the behavior of the large-scale simulation, the original model [Mill2008] was altered to simulate the effects of not fulfilling the three service guarantees. These changes to the large-scale model are described below and their effect on performance is also graphed. In what follows, the results of perturbing both the Markov chain and large-scale simulation to emulate violation of the three service guarantees are described. These results are compared in terms of how well the Markov chain simulation predicts the result of the large-scale simulation and the relative computational effort required by each method.

6.1. Discovery Guarantee

The Discovery guarantee is violated when discovery of information about grid resources is prevented by such events as widespread network transmission failures or directory malfunctions. The effects of not fulfilling the Discovery guarantee can be emulated by lowering the transition probability values from the *Initial* State to the *Discovering* state (row 1), from the *Waiting* state (row 2) to the *Discovering* state, from the *Negotiating* state to the *Discovering* state (row 4), and by increasing the probability of self-transition to the *Discovering* state (row 3). These perturbations constitute four separate scenarios for violating the Discovery guarantee. We examine each in turn.

6.1.1 Perturbation of Transition to the Discovery state in Row 1

Row 1, column 3 of the unperturbed weight-averaged matrix shows the probability of transition from the *Initial* state to the *Discovering* state. The five-period matrix set in the Appendix shows this transition occurs entirely in the first time period of the simulated day. The transition from *Initial* to *Discovering* marks the arrival of a task in the grid system and is followed immediately by an attempt to discover providers to execute the task. This case describes a situation where, for instance, incoming tasks are prevented from executing the initial discovery phase due to network attack, outage, or similar such reason. Although relatively straightforward, the case is included to show the correspondence of the Markov simulation to the large-scale simulation over a larger set of cases.

Figure 4 shows the effect of systematically lowering the probability of the transition from *Initial* to *Discovering* in the Markov chain simulation and the equivalent operation in the large-scale simulation[3]. To perturb the Markov chain simulation, column 1 of row 1 is selected as the primary increase column, while column 3, *Discovering*, is designated as the primary sink column. Since there are no other columns in row 1 that have transition probabilities greater than 0, the sink weight is 1.0. Using the default secondary column selection method, no secondary row is perturbed, since the number of the primary row and increase column are the same.

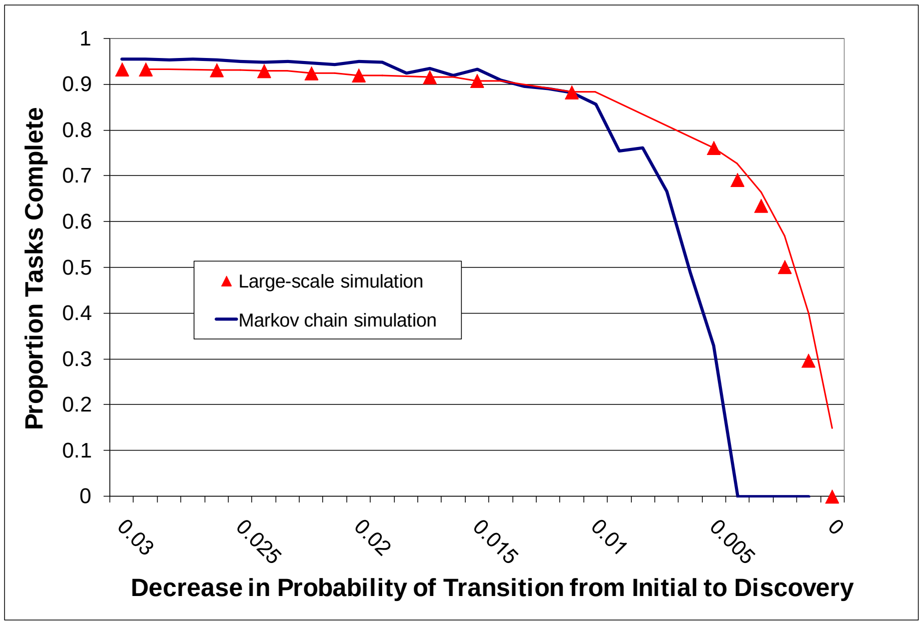

Figure 4. Proportion tasks complete in large-scale and Markov chain simulations in response to reduction in the probability of transition from *Initial* to *Discovering* (column 3 of row 1 is the sink column), while raising self-transition of *Initial* (column 1). Because values could not be obtained for all data points, a two-percent moving average trend line was generated using Microsoft Excel to draw the curve for proportion of tasks complete for the large-scale simulation.

In the large-scale simulation, the equivalent of reducing the probability of transition to the *Discovering* state was achieved by systematically increasing the amount of time each task remains in the *Initial* state, thus in effect delaying arrival of tasks into the grid system. This perturbation had the effect of *right-shifting* the arrival time distribution described above (Mills and Dabrowski 2008) and caused tasks to fail to meet their deadlines. When column 1 of row 1 was selected as the primary increase column in the Markov chain simulation, the same right shift was simulated, because recall that the Markov chain process was modified to allow task arrival to take place using the distribution derived from the large-scale simulation (Section 4.1). Right-shifting this distribution delayed transition from the *Initial* to *Discovering* state, producing the same result.

[3] Please note that in Figure 4 and some subsequent figures, decreasing values are used on the horizontal axis. This is done to show the impact of lowering probability of transition on proportion of tasks complete.

In Figure 4, curves for both large-scale and Markov chain simulations show that proportion of tasks complete decreases relatively little as the probability of transition from *Initial* to *Discovering* state is reduced from 0.03 to 0.01. Below 0.01, the proportion complete drops sharply in both cases. This reflects the effect of increasing delay of tasks leaving the *Initial* state to progress through the *Discovering*, *Negotiating*, and *Monitoring* states, so that they do not have sufficient time to execute and reach *Completed*. Although the Markov chain curve declines more steeply, it is similar to the curve for the large-scale simulation. Both show that performance will decline little until the probability of transition to *Discovering* falls below 0.01. In our experiments, the Markov chain simulation completely perturbed row 1 in 82.39 s, while, the large-scale simulation required 21.18 hours to capture the similar guarantee violation behavior.

6.1.2. Perturbation of Transition to the Discovery state in Rows 2, 3, and 4

In row 2 the probability of transition from the *Waiting* state to the *Discovering* state is shown in column 3, while in row 4, the probability of transition from *Negotiating* to *Discovering*, which occurs less frequently in the large-scale simulation, is also shown in column 3. These state transitions reflect subsequent attempts to initiate discovery operations by task clients after the first round of discovery which occurs when a task first arrives. Failure to initiate subsequent rounds of discovery constitutes violation of the Discovery guarantee which, as before, reflects events such as faults in service discovery directories or communications failures. Row 3, column 3 shows the probability of self-transition to *Discovering*, which if increased, indicates that tasks remain in the *Discovering* state for longer periods for reasons such as those listed above. This could result in prolonging the discovery state to a degree that constitutes a violation scenario for the Discovery guarantee.

In the Markov chain simulation, the first violation scenario was produced by reducing the probability of transition from *Waiting* to *Discovering* (selecting column 3 as the primary sink column with a primary sink weight of 1) while raising the probability of transition from *Waiting* to *Negotiating* (selecting column 4 as the primary increase column). In this case, row 4 (*Negotiating*) was perturbed as the secondary row since the *Negotiating* state corresponds to column 4. Equivalent behavioral changes were made to the large-scale simulation by altering the code to prevent the task client from initiating subsequent rounds of discovery. The large-scale model was iteratively executed. On each iteration, the probability of delaying the start of a subsequent discovery phase was incrementally raised. TPMs were generated for this perturbed behavior and compared with the Markov chain process.

Figure 5 shows the impact of these alterations on the large-scale simulation together with the Markov chain perturbation sequences for 15 perturbation combinations of row 2 that best captured this violation scenario. Exploring all 425 perturbation combinations for row 2 required 373.44 s of computational time, while the large-scale simulation required 12.17 hours to capture the perturbed behavior.

Figure 5 shows the proportion of tasks complete for both the Markov chain and large-scale simulations as the probability of transition from the *Waiting* to the *Discovery* state--e.g., initiation of subsequent discovery actions--decreases along the horizontal axis. In Figure 5, both the curves for the Markov chain and large-scale simulation show essentially no reduction in task completion as the probability of initiating subsequent discovery actions goes to zero. In the large-scale simulation, failing to initiate subsequent discovery does not affect task completion, because the discovery process is sufficiently efficient so that all eligible providers are found on the first discovery attempt (see Section 6.1.1). Hence, subsequent discovery actions are not actually needed, and the absence of these actions does not impact performance. The curves for the related Markov chain perturbation combinations shown in Figure 5 agree well with the large-scale simulation. If the Markov chain curves were used to make predictions, they would accurately predict the result of the large-scale simulation with relatively minor differences in value of tasks completed.

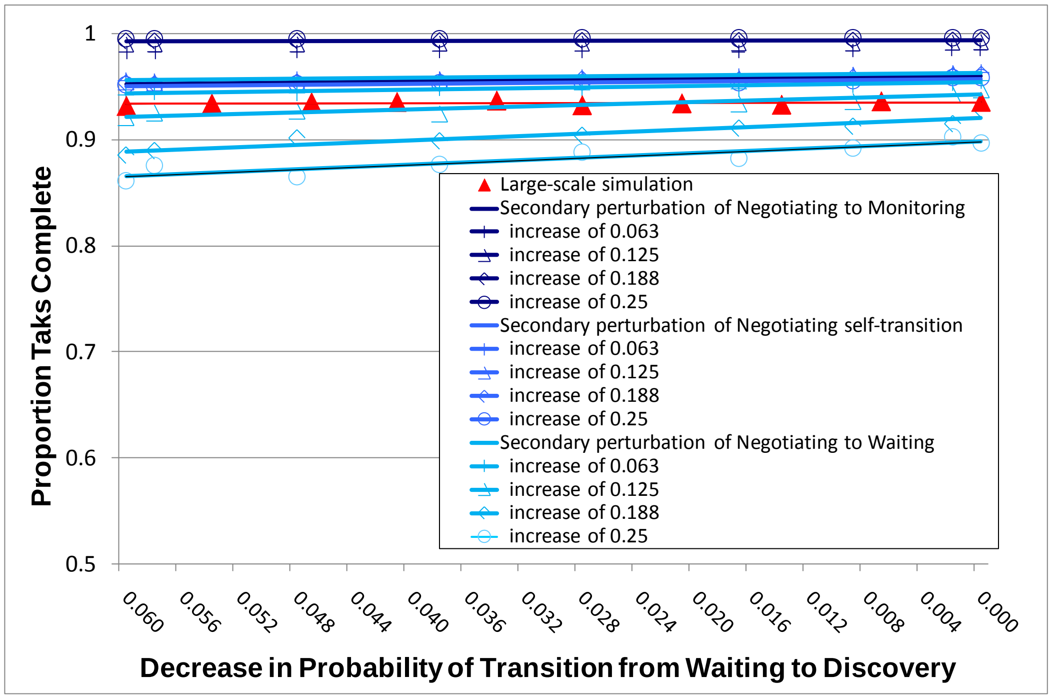

Figure 5. Proportion tasks complete in the large-scale and Markov chain simulations (shown using linear trend lines) in response to reduction in the probability of transition from *Waiting* to *Discovering* (column 3 of row 2 is the primary sink column) while raising the transition probability of *Waiting* to *Negotiating* (column 4 is the primary increase column). A linear trend line is used to draw the curve for proportion of tasks complete for the large-scale simulation.

The second of the three violation scenarios involves row 4. Here, a similar set of perturbations also may be performed to predict the result of reducing the probability of transition from *Negotiating* to *Discovering* (column 3) while raising the probabilities of transitioning from *Negotiating* to *Waiting* (column 2), *Negotiating* to *Monitoring* (column 5), and *Negotiating* self-transition (column 4). Secondary row perturbations are applied to row 2 *Waiting*, row 3 *Discovery*, and row 6 *Monitoring* to produce additional perturbation combinations that capture this violation scenario. In this case, as in the preceding case, Markov chain simulation also agrees with the large-scale simulation. Performance is not impacted by reducing probability of transition to *Discovering*, for the same reasons given above. A similar graph to Figure 5 showing this could be provided but is omitted, because the probability of transition from *Negotiating* to *Discovery* is very low (see five time-period matrix set in Appendix B). Perturbing such low values to 0 involved no more than 3 steps.

In the third violation scenario involving perturbation of row 3, Figure 6 shows that the Markov chain also predicts that increasing the probability of self-transition to the *Discovering* state (making column 3 the primary increase column) will have little impact on the number of tasks completed. The Markov chain predicts this result regardless of whether increases in self-transition to *Discovering* are offset by decreasing probability of transition to either the *Waiting* or *Negotiating* states (i.e., making either the primary sink column) and regardless of what *sink weight* is assigned. In these perturbation combinations, no secondary row perturbation occurs since the row number (3) and primary increase column number (3) are the same. In the large-scale simulation, the increase of the probability of self-transition to *Discovering* is emulated by artificially delaying the times when task clients conclude discovery operations.

`

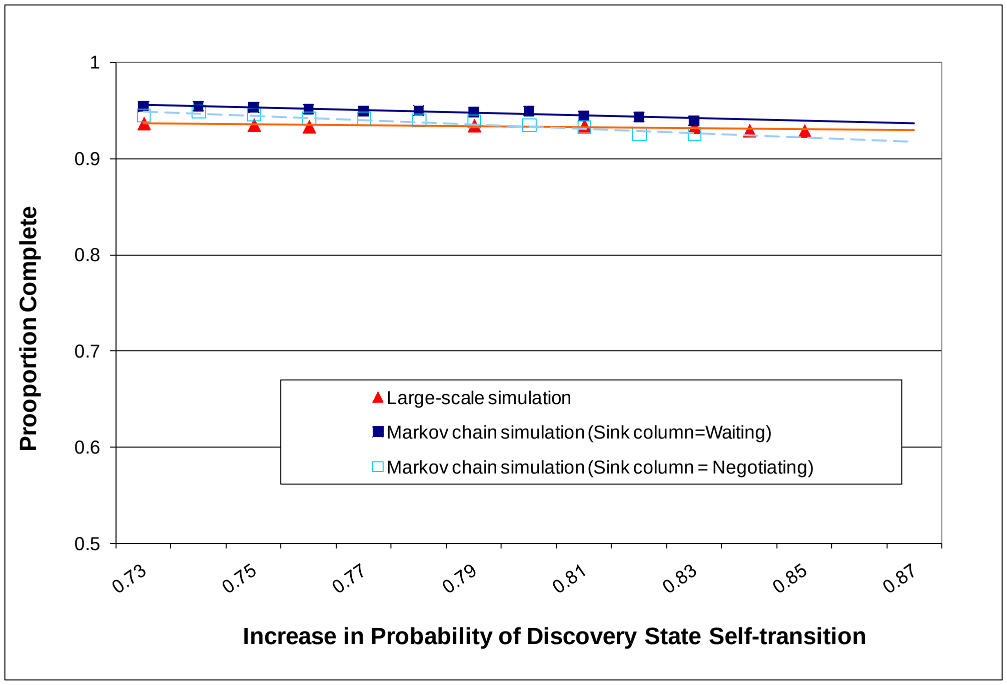

Figure 6. Proportion tasks complete in the large-scale and Markov chain simulations in response to increasing the probability of self-transition in row 3 of the *Discovering* state (column 3 becomes the primary increase column) while decreasing in probability of transition to *Waiting* and *Negotiating* states (columns 2 and 4 become sink columns with sink weights of 1.0). A linear trend line is used to draw the curve for proportion of tasks complete for the large-scale simulation.

Figure 6 shows the curves for the perturbation sequences of two Markov chain perturbation combinations for row 3. Figure 6 shows the impact on tasks complete of increasing the probability of *Discovery* self-transition and the resulting prolongation of the *Discovery* state that may occur as a result of faults. In these two perturbation sequences, the probability of transition from *Discovering* to *Waiting* (column 2) and from *Discovering* to *Negotiating* (column 4) is decreased. In the first sequence, sink weights of 0.4 and 0.6 are assigned to the two columns, while in the second sequence, weights of 0.6 and 0.4 are assigned (though other values were equally predictive). Again, the Markov chain and large-scale simulation agree. Regardless of which perturbation combination is chosen or which perturbation sequence is carried out, the proportion tasks complete remains at a high level. This again is because all necessary providers have already discovered in the initial round of discovery. The perturbed Markov chain simulation were accomplished in 369.69 s, while the large-scale simulation required 8.3 hours.

6.2. Service Engagement Guarantee

The act of engaging a service to execute a task is represented in the Markov chain by the transition from the *Negotiating* state to the *Monitoring* state. In row 4 of the TPM, the effects of non-fulfilment of the Service Engagement Guarantee can be illustrated by reducing the probability of transition. This perturbation is meant to predict the effect of reducing acceptance of agreements because users or providers fail to conclude SLAs they should enter into. Such failures could occur because either users or providers employ poor decision algorithms or communications failures. Along with decreasing the probability of transition from *Negotiating* to *Monitoring* (making column 5 of row 4 the sink column), the time-period TPM set is perturbed by increasing the probability of transition from *Negotiating* to either *Waiting*, *Discovering*, or *Negotiating*, i.e., choosing columns 2, 3, or 4 of row 3 as the primary increase columns. Choosing columns 2 or 3 involves selecting corresponding rows (rows 2 and 3) for secondary perturbation.

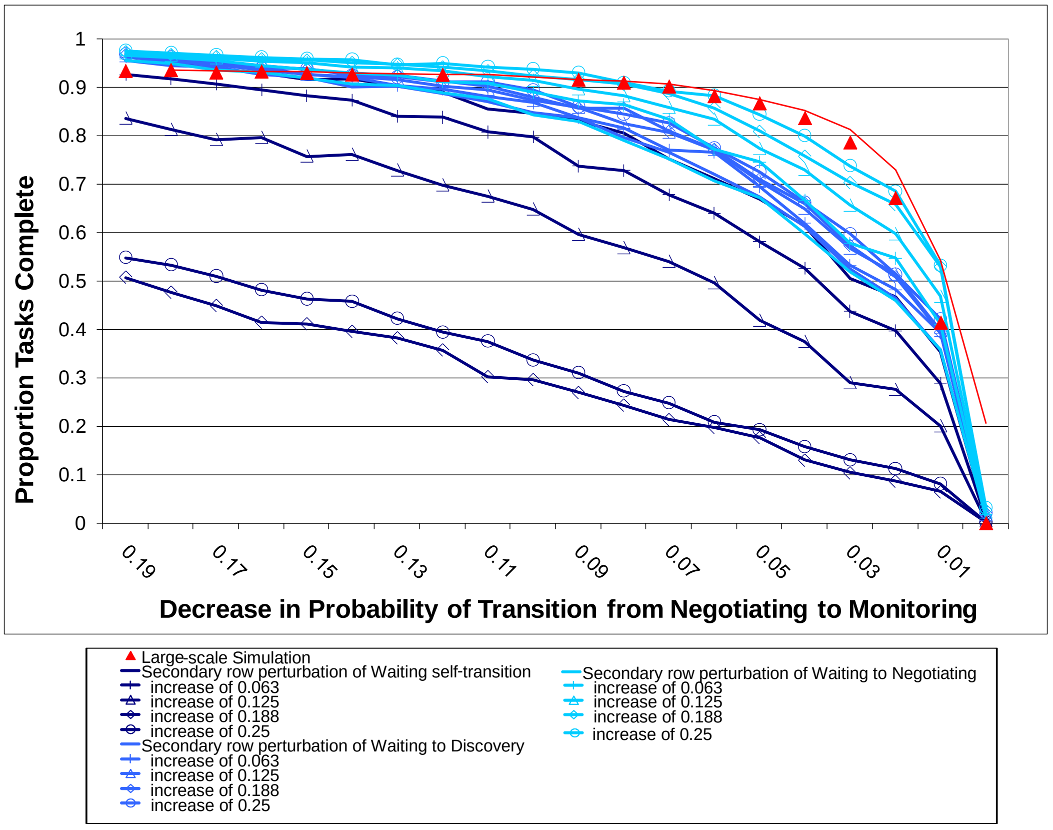

Figure 7. Proportion of tasks complete in the large-scale and Markov chain simulations in response to reducing the probability of transition from *Negotiating* to *Monitoring* (column 5 of row 4 is the primary sink column) while the transition probability to *Waiting* increases (column 2 is the primary increase column). A two-percent moving average trend line is used to draw the curve for proportion of large-scale simulation tasks complete.

Figure 7 shows curves for perturbation sequences of 15 relevant perturbation combinations for row 4 (out the 785 total) in which the probability of transition from *Negotiating* to *Waiting* is raised (column 2 is the primary increase column) while the transition probability from *Negotiating* to *Monitoring* is lowered (column 5 is the *sink column*, with a sink weight of 1). Row 2, *Waiting*, is chosen for secondary perturbation. In the large-scale simulation, the equivalent perturbation was accomplished by systematically increasing the probability that a provider rejects an agreement, the result of which is also shown in Figure 7. This figure shows that the perturbation of the Markov chain simulation is generally predictive of the large-scale simulation result. The Markov chain curves correctly predict that as the probability of transition to *Monitoring* falls to zero, i.e., users and providers fail to conclude SLAs, the proportion of tasks completed also falls to zero.

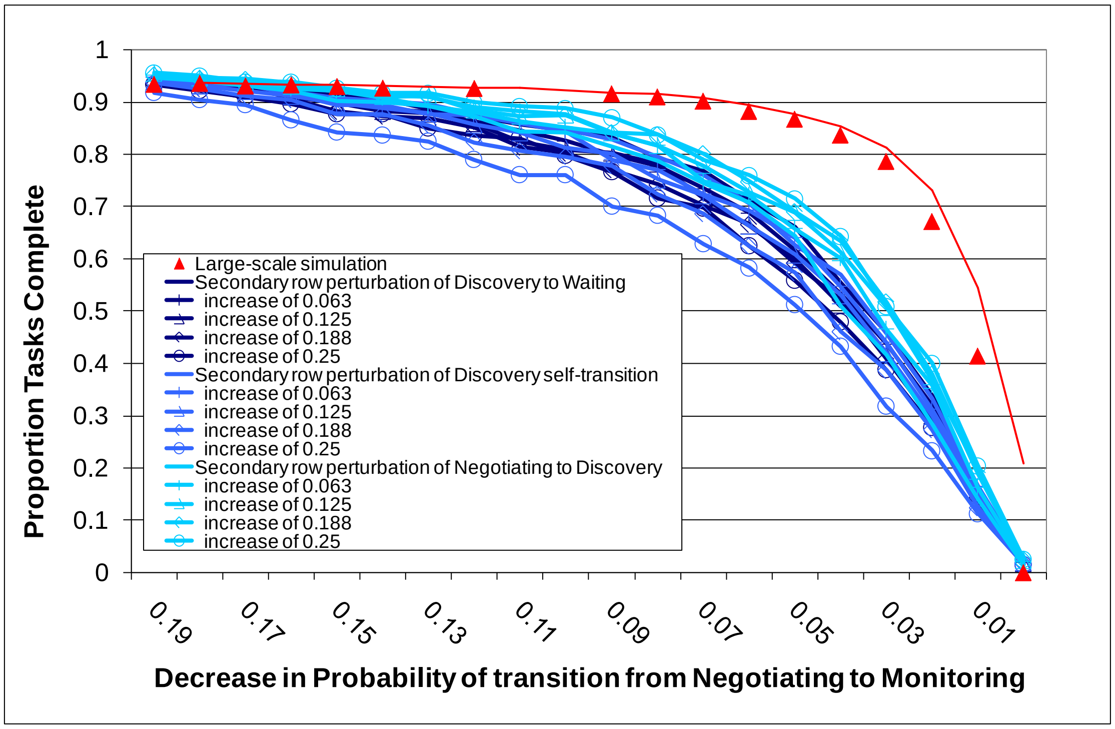

Figure 8. Proportion tasks complete in the large-scale and Markov chain simulations in response to reduction in the probability of transition from the *Negotiating* to the *Monitoring* state (column 5 in row 4 is the sink column with a sink weight of 1) while the transition probability to *Discovery* increases (column 3 is the primary increase column). A two-percent moving average trend line is used to draw the curve for proportion of tasks complete for the large-scale simulation.

In the case where secondary row perturbation of row 2 increased *Waiting* self-transition and simulated still more delay, the Markov chain curves in Figure 7 show that system performance would degrade still further. When secondary row perturbations of row 2 increased the probability of going from *Waiting* to *Negotiating* rather than increasing *Waiting* self-transition or *Waiting* to *Discovering*, the Markov chain process predicted improved system performance. This is so because, logically, transitioning to *Negotiating* increases the chances of then transitioning to *Monitoring* and then completing (i.e., the *Negotiating* state is a more direct path to completion than is *Waiting* or *Discovering*). Here secondary row perturbations also affected predictions because values of transition probabilities in row 2 of the unperturbed TPMs were close to 0 or 1, i.e. they were near extreme limits (see Figure 2 or Appendix B).

In two additional instances of this violation scenario, Markov chain perturbations of row 4, while also being generally predictive, proved to be less accurate predictors of large-scale simulation behavior—for reasons explained below. In the first, the Markov chain simulation predicts decline in tasks completed if a decrease in the probability of transitioning from *Negotiating* to *Monitoring* is offset by an increase in the transition probability from *Negotiating* to *Discovering*, rather than *Waiting*. This is shown in Figure 8. Here, column 3 of row 4 (*Discovering*) is made the primary increase column while column 5 (*Monitoring*) remains the sink column. While the Markov chain family of curves exhibits a lower task completion percentage than the large-scale simulation as probability of transition to *Negotiating* falls from 0.13 and 0.01, there is again a distinct downward trend to zero tasks completed in both cases. Here, the Markov chain curve predicts lower task completion percentage because, unlike the large scale simulation, it increases the probability of transition to *Discovering*. The immediate transition to *Discovering* simulates delay of subsequent negotiation attempts which also delayed transition to *Monitoring* and decreased chances of task completion (i.e., transition to *Discovering* is on a less direct path to completion than is transition to *Negotiating*). Further, additional rounds of discovery do not improve chances of task completion, because, as discussed in section 6.1.2, clients generally discover all providers on the first round of negotiation. Hence, increasing transition to *Discovering* either through primary row or secondary row perturbation does not improve performance.

In the second version of this violation scenario, the Markov chain is perturbed to decrease in the probability of transitioning from *Negotiating* to *Monitoring* and increase in probability of *Negotiating* self-transition (i.e. column 4 in row 4 is made the primary increase column). In this case, the Markov chain simulation also predicts a performance decline in Figure 9. Like the previous case, this violation scenario is less accurate with respect to the behavior of the large-scale simulation. As in the previous case, the Markov chain simulation shows a reduced level of performance in comparison to the large-scale simulation. This is again because the Markov chain simulation simulates prolonging the task in the *Negotiating* state with no progress to *Monitoring* and *Completed* states.

The cases shown in figures 8 and 9 illustrate situations where the accuracy of the Markov chain simulation is reduced because the choice of secondary row perturbations does not accurately model the behavior of the large-scale simulation. These cases show the importance of choosing perturbation combinations that closely correspond to the behavior of the target system. The predictiveness of the Markov chain approach is improved by focusing on the perturbation combinations known to be most accurate, if that knowledge is available. Such information may prove useful for developing a methodology of Markov chains in which the number of perturbations is kept to a minimum. Nevertheless, in all three cases discussed above, in which the probability of transition from *Negotiating* to *Monitoring* is decreased in the Markov chain simulation, the Markov chain predicts a performance decline.

The computational cost for the entire 785 perturbation combinations required to perturb all of row 4 in the Markov chain simulation was 789.17 s. Raising the perturbation limit, ***L***, to 0.5 to obtain a additional perturbation sequences needed to increase range in this case required another 1480.46 s. Here, the large-scale simulation required 41.57 hours.

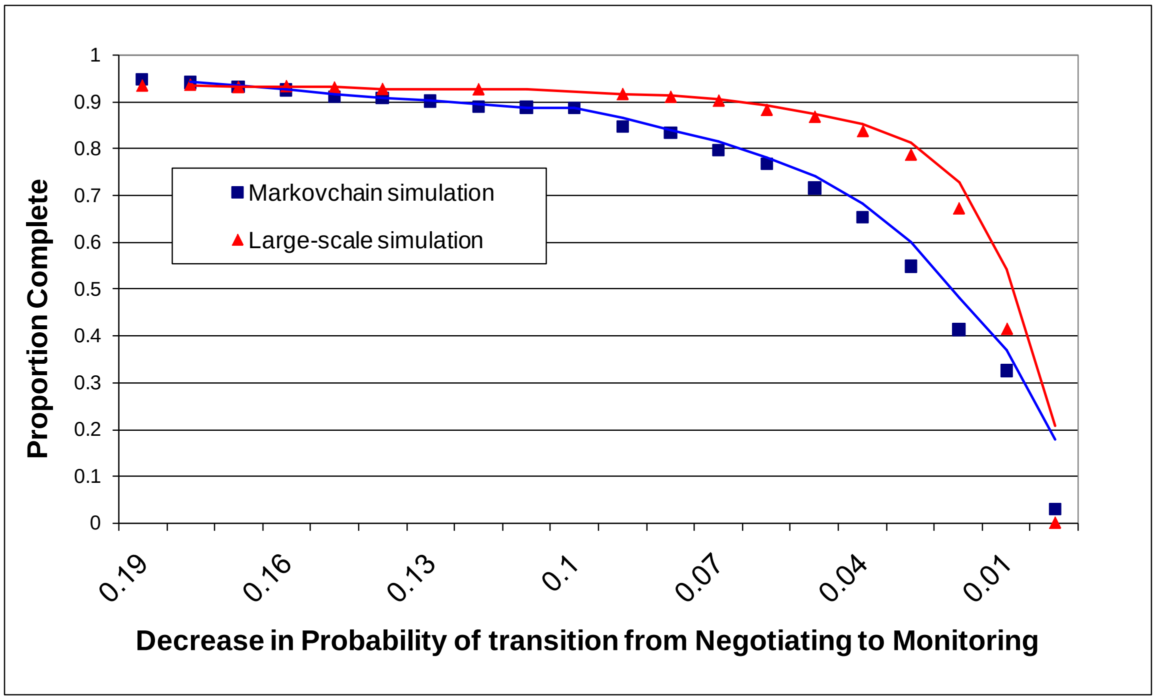

Figure 9. Proportion of tasks complete in the large-scale and Markov chain simulations in response to reduction in the probability of transition from the *Negotiating* to the *Monitoring* state (column 5 in row 4 is the sink column with a sink weight of 1) while the probability of *Negotiating* self-transition increases (column 4 is the primary increase column). Since there is no secondary row perturbation, only one curve is shown. A two-percent moving average trend line is used to draw the curve for proportion of tasks complete for the large-scale simulation.

6.3. Agreement Fulfillment Guarantee

The *Monitoring* state is entered once an SLA is concluded. Failure to fulfill an SLA can occur for many reasons: widespread network communications failures, network-wide cyber attacks and viruses, faults or failures of computing resources that are executing user tasks, of crashes of host operating system software. In the Markov chain, failure to fulfill an agreement may be modeled by increasing the probability of transition from *Monitoring* to states other than the *Completed* state; namely, either increasing the probability of transition to the *Negotiating* state (representing a task abort) or increasing the probability of self-transition in the *Monitoring* state (representing an extended delay). In the former violation scenario, a task that transitions from *Monitoring* to *Negotiating* (aborts) may recover from this setback by later obtaining another SLA, returning to the *Monitoring* state, and then completing. In this section, this violation scenario is simulated in the Markov chain by making *Negotiating*, column 4 of row 5, the primary increase column, while making *Completed*, column 6, the sink column. In the resulting perturbation combinations, secondary row perturbation is applied to the *Negotiating* row (row 4). In the large-scale simulation, the equivalent behavior change was enabled by systematically increasing the rate at which a provider aborts a queued or executing task.

As before, the large-scale simulation iterated, with the abort rate increasing on each iteration.

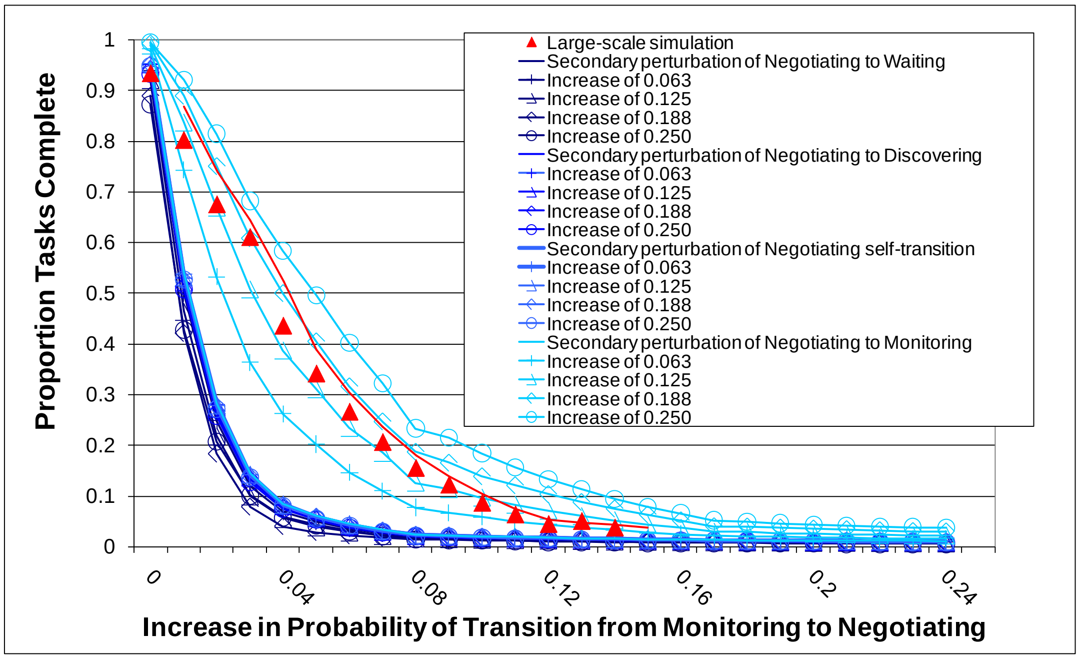

Figure 10. Proportion tasks complete in large-scale and Markov chain simulation in response to reduction in the probability of transition from *Monitoring* to *Completed* (row 5, column 6 is the sink column) while increasing the probability of transition from *Monitoring* to *Negotiating* (column 4 is the primary increase column). A two-percent moving average trend line is used to draw the curve for proportion of tasks complete for the large-scale simulation.

Figure 10 shows the resulting curves for Markov chain perturbation sequences in which the probability of transition to the *Negotiating* state is raised (i.e., column 4 is the primary increase column) as the probability of transition to *Completed* (sink column 5) falls from a weighted average of 0.008 to zero. The figure shows that this perturbation causes the proportion of tasks completed to fall dramatically. This figure shows 20 of the most relevant perturbation combinations (out of a total of 270) for the case where the *Completed* state has a sink weight of 0.2. Alternative perturbations using *Monitoring* as the sink column (not shown) produce a similar result. In all cases, the Markov chain curves show a pronounced reduction in proportion of tasks completed that is substantially predictive of the curve for the large-scale simulation, also shown in Figure 10. The figure shows that 5 of the 20 curves represent accurate approximations of the large-scale simulation result. The remainder show the distinct downward trend in tasks completed, though at markedly different slopes. In the 5 curves closest to the large-scale simulation result, the secondary row, *Negotiating* (row 4), is perturbed to raise the probability of transition to the *Monitoring* state (column 5). This effectively models a situation where tasks that fail to transition from *Monitoring* to *Completed* (abort) are later able to negotiate new SLAs, return to the *Monitoring* state, and complete—as might be expected in the real world. While only a relatively small proportion of the curves produced through Markov chain simulation are this accurate, all curves predict that if the probability of transition from *Monitoring* to *Negotiating* is raised sufficiently (i.e., the probability of task abort is high enough), system performance will drastically degrade. The Markov chain simulation required 260.2 s of execution time to process the 270 perturbation combinations for row 5. The large-scale simulation required a lengthy 122.6 hours to carry out the equivalent perturbation behavior, because repeated task aborts entailed extensive delays.

6.4 Summary of Analysis and Outstanding Issues

The preceding sections showed that perturbation of TPMs carried out through application of the method described in sections 4 and 5 does indeed generally predict how the large-scale grid system will perform when key service guarantees are violated. In the case of the Discovery guarantee, the Markov chain was found to accurately correspond to the large-scale simulation in both cases shown. For the Engagement guarantee, perturbations of the Markov chain were found to generally correspond to the results produced by the large-scale simulation in one violation scenario shown here. For the violation of the Service Fulfillment Guarantee, the perturbation of the Markov chain accurately captured the behavior exhibited in the large-scale simulation. Thus perturbation of the Markov chain was shown to be an effective predictor for all cases shown in this paper. In no case, did the Markov chain simulation produce results that contradicted the large-scale simulation. Moreover, as we have seen, the Markov chain approach achieved these results at less than 0.5% of the computational cost of the large-scale simulation. If the required data was obtained from a real-world system to create a Markov model and related TPMs, it is reasonable to believe that comparable results could be achieved.

Despite this success, important issues still remain to be resolved. The most important is scalability, which has three aspects. First is whether the approach scales with respect to the size of the system being modelled, as expressed in terms of such variables as number of entities being modelled, number of transitions taken, and workload. As section 4 has shown, the method of counting state transitions and generating transition probabilities is a straightforward arithmetic process that clearly does not depend on number of transitions. Here, scale does not hinder analysis. Second, there is the all-important issue of the size of the state model, that is, the number of states and the corresponding size of the TPM. Here, further work incorporating lumping techniques described in Section 2 will be needed. Finally, it is important to consider scalability with respect to the number of perturbations, or alternative execution paths, investigated. Despite the dramatic reduction in execution time seen for Markov chain method (< 1% of the execution time used by large-scale simulation), scalability may not be good for very large matrices or if many perturbations are needed. Follow-on research will be needed to examine this issue and to develop a methodology for systematic perturbation of Markov chains. Here, there is the possibility of extending non-linear algebra techniques and matrix methods (Stewart and Sun 1990) to generate eigensystems that can be analysed to determine what parts of the matrix are most sensitive to perturbation and thus where investigation should be focused. Despite these issues, use of the Markov chain approach entails dramatically less computational effort than large-scale simulation.

7. Conclusions

Section 6 showed that perturbation of TPMs and Markov chain simulation was generally predictive of changes to performance arising from failure to fulfill basic service guarantees provided by grid computing systems. While Markov chain analysis did not reproduce the exact performance curves generated by a detailed simulation, a carefully limited brute-force perturbation of TPMs produced a family of related curves which approximated the impact of not fulfilling service guarantees. Perturbed TPMs produced by Markov chain analysis were predictive both in cases where changes to the behavior of the large-scale simulation resulted in severe performance degradation as well as cases where changes to the large-scale simulation did not significantly impact results. Thus, it is possible to conclude that the approach to perturbing Markov chains described in this paper did indeed answer the questions posed in section 3; namely, how non-fulfillment of the three service guarantees affects performance of a large scale grid system. Moreover, the Markov chain procedure was able to perform the analysis necessary to answer this question using a small fraction of the computational resources (less than 1%, or two orders of magnitude) that was necessary for the large-scale simulation. If, instead of the large-scale simulation, a real-world system could be used as a testbed in which conditions are sufficiently controlled to allow execution of repeated trials, the contrast in time (and resource) expenditure could be much greater. The study thus shows that Markov chain analysis is a valuable tool for understanding complex system behavior in large-scale grid systems and can be used to predict performance changes that result when fundamental guarantees of service are not met.

8. References

[Akio2003] S. Akioka and Y. Muraoka. The Markov Model Based Algorithm to Predict Networking Load on the Computational Grid. *Journal of Mathematical Modelling and Algorithms*, **2**, (2003), 251–261.

[Andr2007] A. Andrieux, K. Czajkowski, A. Dan, K. Keakey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. *Web Services Agreement Specification (WS-Agreement)*. GFD.107, Open Grid Forum (May 2007).

[Aupp1991] B. Aupperle and J. Meyer. State space generation for degradable multiprocessor systems. *Twenty-First International Symposium on Fault-Tolerant Computing (FTCS-21), Digest of Papers*, (June 1991), 308-315.

[Bala1994] M. Balakrishnan and A. Reibman. Reliability models for fault-tolerant private network applications. *IEEE Transactions on Computers*, **43**, 9, (September 1994), 1039-1053.

[Baxt2001] J. Baxter and P. Bartlett. Infinite-Horizon Policy-Gradient Estimation. *Journal of Artificial Intelligence Research*, **15**, (November 2001), 319-350.

[Buch1995] P. Buchholz. Hierarchical Markovian Models: Symmetries and Reduction. *Performance Evaluation*, **22**, 1, (1995), 93-100.

[Cao1998] Cao, X. and Wan, Y. Algorithms for Sensitivity Analysis of Markov Systems Through Potentials and Perturbation Realization. *IEEE Transactions on Control Systems Technology*. vol. 6, no. 4, July 1998.

[Cao2005] X. Cao. Basic Ideas for Event-Based Optimization of Markov Systems. *Discrete Event Dynamic Systems: Theory and Applications*, **15**, (2005), 169–197.

[Cao2008] X. Cao, J. Zhang. Event-Based Optimization of Markov Systems. *IEEE Transactions on Automatic Control*, **53**, 4, (May 2008), 1076-1082.

[Carr2006] D. Carr. How Google Works. *Baseline Magazine* http://www.baselinemag.com, July 6, 2006.

[Cass1990] C. Cassandras, J. Lee, and Y. Ho. Efficient parametric analysis of performance measures for communication networks. *IEEE Journal on Selected Areas in Communications*, **8**, 9, (December 1990), 1709-1722.

[Chio1993] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad. Stochastic Well-Formed Colored Nets and Symmetric Modeling Applications. *IEEE Transactions on Computers*, **42**, 11, (November 1993), 1343-1360.

[Chun2002] B. Chun and E. Culler. User-centric performance analysis of market-based cluster batch schedulers. *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid*, (2002).

[Coll2005a] M. Colledani and T. Tolio. Impact of statistical process control (SPC) on the performance of production systems—Part 1 (small systems). *5th international conference on analysis of manufacturing systems—production management*. Zakynthos Island, Greece, (May 2005).

[Coll2005b] M. Colledani and T. Tolio. Impact of statistical process control (SPC) on the performance of production systems—Part 2 (large systems). *5th international conference on analysis of manufacturing systems—production management*. Zakynthos Island, Greece, (May 2005).

[Dall1992] Y. Dallery, S. Gershwin. Manufacturing flow line systems: A review of models and analytical results. *Queuing Systems Theory and Applications*, **12**, (1992), 3–94.

[Deri2003] S. Derisavi, H. Hermanns, and W. Sanders. Optimal state-space lumping in Markov chains. *Information Processing Letters*, **87**, 6, (September 2003), 309-315.

[Diam2006] A. Diamantidis and C. Papadopoulos. Markovian analysis of a discrete material manufacturing system with merge operations, operation-dependent and idleness failures. *Computers & Industrial Engineering*. **50**, (2006), 466–487.

[Glob2008] *The Globus Toolkit*. http://www.globus.org/toolkit/.

[Gose2001] K. Goseva-Popstojanova and K. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, **45**, 2-3, (2001), 179-204.

[Helb1997] S. Helber. Decomposition of unreliable assembly/disassembly networks with limited buffer capacity and random processing times. *European Journal of Operational Research*, **109**, 1, (August 1998), 24-42.

[Helb2000] S. Helber. Approximate analysis of unreliable transfer lines with splits in the flow of material. *Annals of Operations Research*. **93**, (2000), 217–243.

[Helb2003] S. Helber and N. Mehrtens. Exact analysis of a continuous material merge system with limited buffer capacity and three stations. In S. Gershwin, Y. Dallery, C. Papadopoulos, and J. Smith (eds.). *Analysis and Modeling of Manufacturing Systems*. Kluwer Academic, Dordrecht, (2003), 85–121.

[Ho1985] Y. Ho. A Survey of the Perturbation Analysis of Discrete Event Dynamic Systems. *Annals of Operations Research*, **3**, 1985, 393-402.

[Ho1988] Y. Ho and S. Li. Extensions of infinitesimal perturbation analysis. *IEEE Transactions on Automation Control*, **AC-33,** 5**, (**1988), 427-438.

[Ho1991] Y. Ho. *Perturbation Analysis of Discrete Event Systems*, Kluwer Academic Publishers, Boston, MA, (1991).

[Ho1997] Y. Ho and C. Cassandras**.** Perturbation analysis for control and optimization of queueing systems: An overview and the state of art. In J. Dshalalow (ed.). *Frontiers in Queueing: Models and Applications in Science and Engineering*, CRC Press, (1997), 395-420.

[Jaco2007] M. Jacobi and O. Gornerup. A Dual Eigenvector Condition for Strong Lumpability of Markov Chains. Submitted to *Arxiv preprint arXiv:0710.1986*. http://arxiv.org/abs/0710.1986, (2007).

[Jonk1994] H. Jonkers. Queueing models of parallel applications: the Glamis methodology. *Lecture Notes in Computer Science*, **794**, 123-138.

[Keme1976] J. Kemeny and J. Snell. *Finite Markov Chains*. Springer, New York, (1976).

[Kim2005] J. Kim, S. Gershwin. Integrated quality and quantity model of a production line. *OR Spectrum*, **27**, (2005), 287–314.

[Lapr1992] Laprie, J, Kanoun, K. X-ware reliability and availability modeling. *IEEE Transactions on Software Engineering*. vol.18, no.2, February 1992. pp.130-147.

[Li2005] J. Li and N. Huang. Modeling and analysis of a multiple product manufacturing system with split and merge. *Proceedings of the 2004 IEEE International Conference on Robotics and Automation (ICRA '04)*, New Orleans, LA, (April 2004) 2261-2266.

[Li2008] J. Li, D. Blumenfeld, N. Huang, and J. Alden. Throughput analysis of production systems: recent advances and future topics. *International Journal of Production Research*. To appear, (2008).

[Liu2008] Y, Liu and J. Li. Modeling and Analysis of Bernoulli Production Systems with Split and Merge. *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, (May 2008), 3618-3623.

[Mill2006] K. Mills, C. Dabrowski. Investigating Global Behavior in Computing Grids. Lecture Notes in Computer Science, **4124**, 120-136.

[Mill2008] K. Mills, C. Dabrowski. Can Economics-based Resource Allocation Prove Effective in a Computation Marketplace? *Journal o f Grid Computing*, **6**, 3, 291-311.

[Nara1990] Y. Narahari, N. Viswanadham, and K. Krishna Prasad. Markovian Models for Deadlock Analysis in Auto Manufacturing Systems. *SZdhanii*, **15**, 4 & 5, (December 1990), 343-353.

[Nara1994] Y. Narahari and N. Viswanadham. Transient Analysis of Manufacturing Systems Performance. *IEEE Transactions on Robotics and Automation*, **10**, 2, (April 1994), 230-244.

[Nico2004] D. Nicol, W. Sanders, and K. Trivedi. Model-based evaluation: from dependability to security. *IEEE Transactions on Dependable and Secure Computing*. **1**, 1, (January 2004), 48 – 65.

[Obal2001] W. Obal and W. Sanders. Measure-adaptive state-space construction. *Performance Evaluation*, **44**, 1-4, (April 2001), 237-258.

[Raff2006] D. Raffo. Grid, redundancy, and home-cooked management help site survive. *Byte and Switch*, (November 22, 2006).

[Rose2004] D. Rosenberg, E. Solan, and N. Vielle. (2004): Approximating A Sequence of Observations By A Simple Process. *The Annals of Statistics* **32** (6), (2004), 2742-2775.

[Sand1991] W. Sanders and J. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications, special issue on Computer-Aided Modeling Analysis, and Design of Communication Networks,* **9**, 1, (1991), 25–36.

[Schw1968] P. Schweitzer. Perturbation Theory and Finite Markov Chains. *Journal of Applied Probability*. **5**, 2, (August 1968), 401-413.

[Sieg1992] M. Siegle. On Efficient Markovian Modelling. *Proceedings of the QMIPS Workshop on Stochastic Petri Nets*, Sophia Antipolis, France, 213-225.

[Song2004] B. Song, C. Ernemann and R. Yahyapour. Parallel Computer Workload Modeling with Markov Chains. Lecture Notes in Computer Science, **3277**, (2004), 47-62.

[Suri1987] R. Suri. Infinitesimal perturbation analysis for general discrete event systems. *Journal of the ACM*, **34**, 3, 686-717.

[Suri1989] R. Suri. Perturbation Analysis: The State of the Art and Research Issues Explained via the GI/G/l Queue. *Proceedings of the IEEE*, **77**, 1, (January, 1989), 114-138.

[Tan2000] B. Tan and S. Karabati. Modelling and Scheduling of an Asynchronous Cyclic Production Line with Multiple Parts. *The Journal of the Operational Research Society*, **51**, 11, (November 2000), 1296-1308.

[Triv2004] K. Trivedi, S. Ramani, and R. Fricks. Recent advances in modeling response-time distributions in real-time systems. *Proceedings of the IEEE,* **91**, 7, (July 2003), 1023-1037.

[Wolp1997] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, **1,** 1, 67–82.

[Yeo2005] C. Yeo and R. Buyya Service level agreement based allocation of cluster resources: handling penalty to enhance utility. *Proceedings of the 7th IEEE International Conference on Cluster Computing*. Boston, MA, USA, (2005).

[Zaka1997] A. Zakarian and A. Kusiak. Modeling manufacturing dependability. *IEEE Transactions on Robotics and Automation*, **13**, 2, (April 1997), 161-168.

APPENDIX A. Decomposition of Selected States

This appendix is provided for readers who want to obtain additional detail about the state model. Future work may entail investigating the effects of perturbing lower-level Markov chain models for these three states. This appendix describes the decomposition of the *Discovering*, *Negotiating*, and *Monitoring* states into lower-level substates and shows the related substate decomposition diagrams. The decomposition of these three states is strictly hierarchical. In each case, transition into the containing higher-level state denotes simultaneous transition in the related substate decomposition diagram from the initial state to another substate. Similarly, in each decomposition diagram, transition to the terminal state denotes simultaneous transition out of the containing higher-level state. In each case, particular substates that are shaded identify states in which errors occur or tasks complete either properly or improperly. These substates provide a basis for diagnosing errant behaviors. All three substate diagrams have the Markov property for which stochastic TPMs can be formed. Similarly, as in the case of the higher-level state Markov chain, a global system state vector representing the lumped states of all tasks in the system can be created for each substate Markov chain. A brief analysis of the decomposition concludes the section.

A.1 Decomposition of the Discovering State

These substates describe in detail the discovery process by which a set of grid service providers are located that have the potential to execute a user task. The process described here follows the Globus Monitoring and Discovery System (MDS) [Glob2008]. In this process, a transition is taken from the initial state to the *Processing GIIS* state. In this state, the user first queries a Grid Index Information Service (GIIS) to obtain addresses of any Grid Resource Information Services (GRIS) that may have information about service providers which are capable of executing the user task. The *Processing GIIS* state encompasses the activities involved in querying one or more GIIS and processing the related responses. Inability to obtain a response from a GIIS results in transition to a *Failed GIIS State* in which the failure is recorded. A negative response from a GIIS results in transition to the *Discarding GRIS* state in which all service provider information in the GRIS is discarded and no longer available. A GIIS may return information about one or more GRIS that can be queried to obtain information about relevant service providers. In this case, a transition is taken to the *Processing GRIS* state. Here, as in the case of the GIIS, attempts to contact GRIS directories may also fail or produce perceived irrelevant responses. If a GRIS returns information about service providers that are potentially capable of running the user task, the *Storing Discovery* state is entered to record the discovery. Once all GRIS are queried and discoveries are recorded, the *Processing GRIS* substate, and the containing *Discovery* state, are exited simultaneously. This process is illustrated in Figure A.1. In this figure (and in subsequent figures in this appendix), yellow shading signifies a state that has been added to the diagram to allow tracking of failure conditions.

Two states may be further decomposed. The *Processing GIIS* state consists of four substates detailing elementary steps for sending queries to the GIIS and processing responses. The *Processing GRIS* state may be similarly decomposed into three more detailed states for querying GRIS. Both decompositions are strictly hierarchical. In total, the *Discovering* state consists of 14 substates in two lower-level hierarchical decompositions.

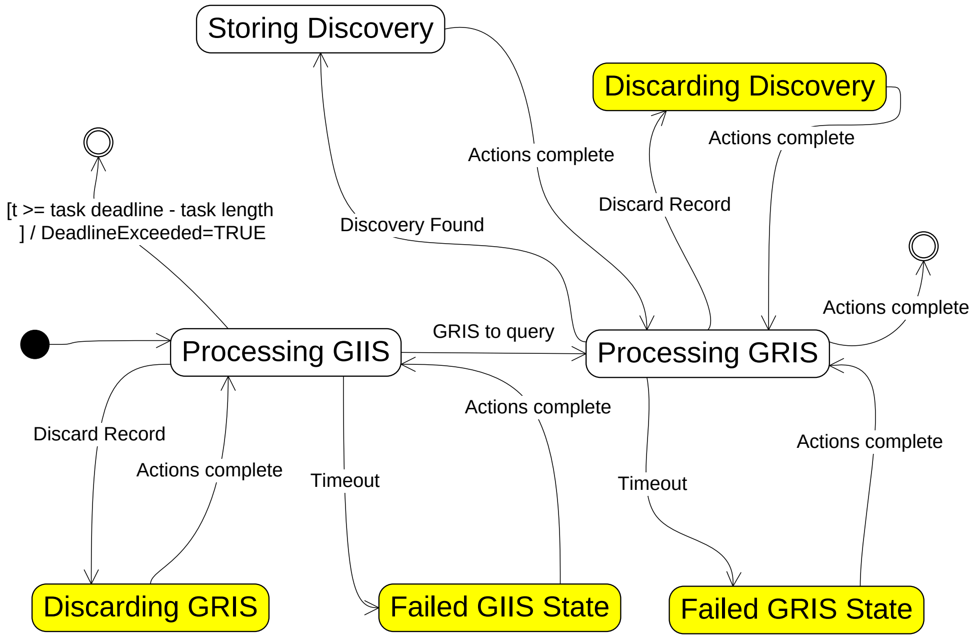

Figure A.1. Decomposition of the *Discovering* state into a substate diagram. Black circles without rings refer to initial states or entry points. Black circles with rings refer to termination or end points.

A.2 Decomposition of the Negotiating State

The substates shown in Figure A.2 identify critical aspects of the negotiation process. This process is executed independently by a client user for each service provider that was discovered during the *Discovering* phase. Upon entry into the containing *Negotiating* state, the *Processing SLA Offer* substate is simultaneously entered. In this substate, the client user carries out the actions involved in preparing an SLA offer to a provider, sending the offer, and awaiting the response. The substate may be further decomposed into another five straightforward, procedural states. These actions follow the SLA negotiation protocol required by the *WS-Agreement* standard specification [Andr2007].

More importantly perhaps, the remaining three states in Figure A.2 are intended to facilitate understanding behavior of the grid system as a whole. The *Failed Negotiation State* indicates that either the provider rejected the SLA offer or the negotiation process could not be completed due to error. The *Record Provider Rejection* state indicates the client user decided that the provider was inappropriate and terminated the procedure. The *SLA Accepted* state indicates an agreement was reached. If included in a system state vector using the procedure described in section 4.2, these three substates could be used to diagnose causes of degradation in system performance in the more detailed model. In total, the Negotiating state consists of 9 substates, including the further decomposition of the *Processing SLA Offer* substate.

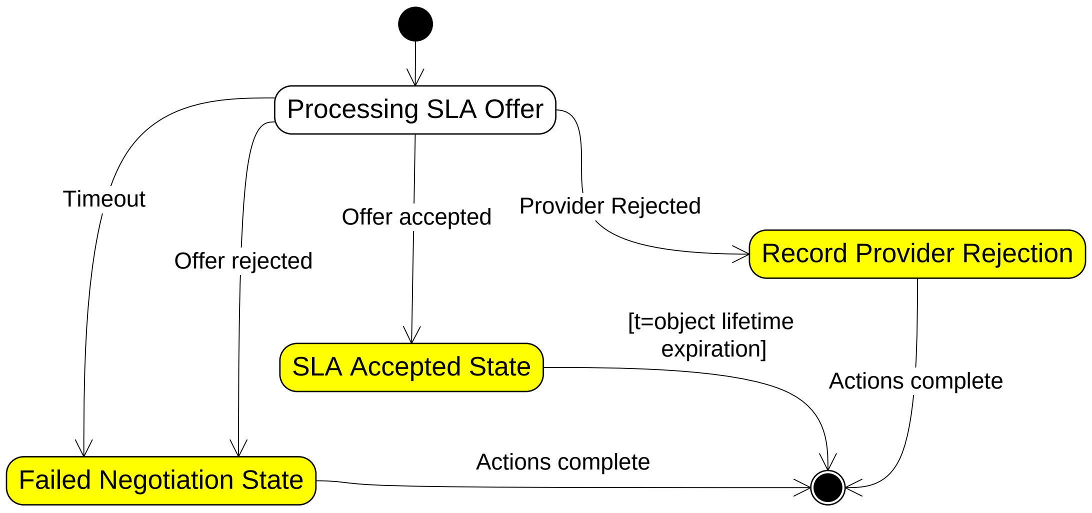

Figure A.2. Decomposition of the *Negotiating* state into a substate diagram.

A.3 Decomposition of the Monitoring State

From the point of view of the user, the *Monitoring* state decomposes into four states that describe substates that are entered once an SLA for the task has been agreed to and the task has been submitted to the provider for execution. This is shown in figure A.3 on the next page. In this figure, none of these states are further decomposed. The first, *Requesting Registration*, represents a request by the user to be notified of changes to the status of the user's task, or job. This is followed by transition to a local *Waiting* state specific to the *Monitoring* state machine (not to be confused with the *Waiting* state in the high-level state diagram in figure 1), duringwhich self-transitions may be taken when either notice of registration decision arrives. If the registration is accepted, the task returns to the *Waiting* state, where it remains until the task has begun execution. If the task completes, a transition occurs from the *Waiting* state to the *Completion Actions* state (which signifies transition to the *Task Completed* state in the high-level state diagram in figure 1). If the task is aborted or the registration request is rejected, a transition to the *Task Aborted* state occurs (which signifies transition to the *Task Failed* state in the high-level state diagram).

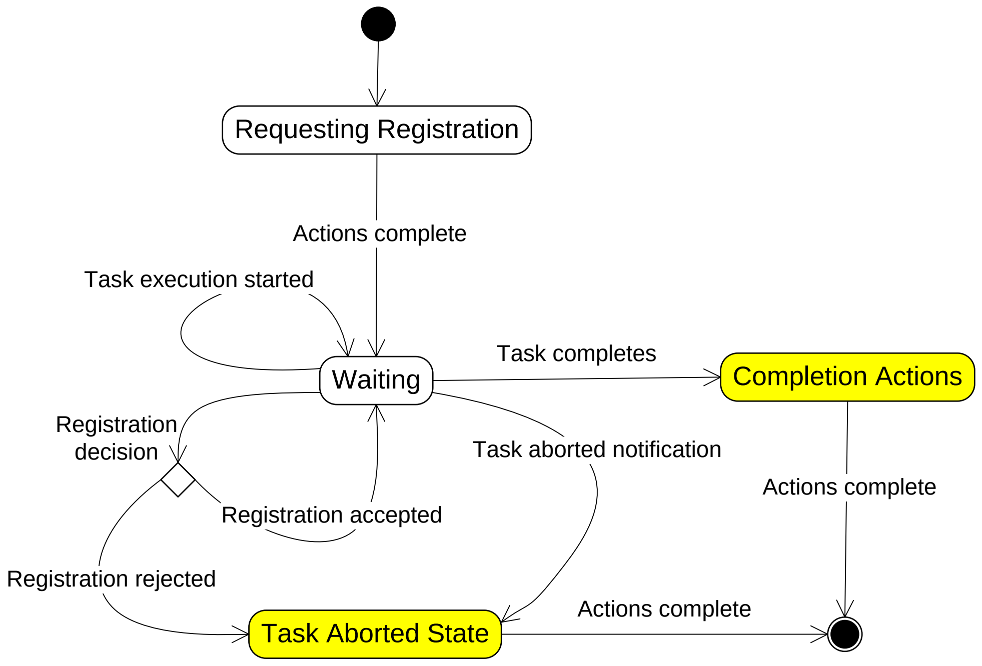

Figure A.3. Decomposition of the *Monitoring* state into a substate diagram.

A.4 Analysis of Decomposition of High-Level Model

As shown above, three states in the high-level model can be further decomposed into 27 states. There are 14 states in the decomposition of *Discovering*, nine in the decomposition of *Negotiating*, and four in the decomposition of *Monitoring*. Each of these constitutes a hierarchy of states. *Discovering* decomposes directly into seven states with two states (*Processing GIIS* and *Processing GIIS*) decomposing into 3 and 4 states respectively. If flattened, this becomes a single diagram with 12 states. Negotiating decomposes into four states, one of which (*Processing Offer*) decomposes into five. If flattened, this yields eight states. *Monitoring* decomposes simply into four states, none of which are further decomposed. If the entire 34-state structure (seven at the high level and 27 at the lower levels) were flattened, it would yield a single 28-state model. Detailed analysis of the larger model and the individual substate Markov chains may be a direction for future work. This would have two areas of interest. First, a number of more detailed states describe error situations, which if entered into by a high proportion of tasks in the system, could be predictive drastic changes at the system level. Second, the larger 28-state model could be converted into a larger TPM and used to test the scalability of methods described in this paper.

APPENDIX B. Five Time-Period Transition Matrices

These matrices, referred to in section 4, allow a representation of the model as a *piece-wise homogenous* Markov chain having a bounded number of pieces [Rose2004], where each piece corresponds to a different time period. The five matrices, shown below, were used as a basis for the perturbation experiments described in section 6. The summary matrix shown in figure 2 was produced by weight averaging these matrices on the basis of number of transitions in each time period.

Time Period 1, 0-7200 s

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0.9697	0	0.0303	0	0	0	0
Waiting	0	0.8072	0.0550	0.1378	0	0	0
Disc	0	0.0631	0.6013	0.3356	0	0	0
Ngt	0	0.3506	0.0132	0.2920	0.3442	0	0
Mon	0	0	0	0	0.9961	0.0039	0
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	0

Time Period 2, 7201-14400 s

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0	0	0	0	0	0	0
Waiting	0	0.8527	0.0722	0.0749	0	0	0.0002
Disc	0	0	0.4988	0.5012	0	0	0
Ngt	0	0.5466	0.0316	0.3770	0.0448	0	0
Mon	0	0	0	0.0001	0.9928	0.0071	0
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

Time Period 3, 14401-21600s

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0	0	0	0	0	0	0
Waiting	0	0.8494	0.0730	0.0702	0	0	0.0074
Disc	0	0	0.6407	0.3593	0	0	0
Ngt	0	0.7845	0.0143	0.1854	0.0157	0	0.0001
Mon	0	0	0	0.0010	0.9828	0.0162	0
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

Time Period 4, 21601-28800s

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0	0	0	0	0	0	0
Waiting	0	0.8340	0.0722	0.0689	0	0	0.0249
Disc	0	0	0.9991	0.0009	0	0	0
Ngt	0	0.9899	0	0	0	0	0.0101
Mon	0	0	0	0.0013	0.9317	0.067	0
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

Time Period 5, 28801-36000s

	Initial	Wait	Disc	Ngt	Mon	Comp	Fail
Initial	0	0	0	0	0	0	0
Waiting	0	0.8111	0.0720	0.0734	0	0	0.0435
Disc	0	0	0.9991	0.0009			
Ngt	0	0.9899	0	0	0	0	0.0101
Mon	0	0	0	0.0013	0.9317	0.0670	
Comp	0	0	0	0	0	1.0	0
Fail	0	0	0	0	0	0	1.0

APPENDIX C. Algorithm to Perturb Row of Transition Probability Matrix

This section provides pseudo-code for the perturbation algorithm overviewed in section 5.1. The main algorithm on the next page details the steps in determining perturbation combinations, while subsequent pseudo-code procedures describe (1) the default method of selecting secondary rows, (2) the method for evaluating perturbation sequences, and (3) the steps in multiplying matrices to carry out the Markov chain simulation. Table C.1 defines the major variables used in these procedures and discussed in the main body of this report.

Table C.1. List of term used to describe perturbation algorithm

Term	Symbol (if any)	Definition
Non-sink column	---	The column in the *primary row* in which p_{ij} values are decreased by an amount proportional to 1 – (*w* * $v_{prim,}$), where *w* is the *sink weight*. See Appendix C for further details.
Primary increase amount	v_{prim}	The amount by which the value of p_{ij} in the *primary row* is successively raised during a *perturbation sequence*.
Primary increase column	$c^{\uparrow}$	The column in the *primary row* in which p_{ij} values are increased by an amount v_{prim}..
Primary row	*r*	The TPM row selected by the analyst in which columns in the transient portion of the matrix will be perturbed.
Primary sink column	$c^{\downarrow}$	The column in the *primary row* in which p_{ij} values are decreased by an amount equal to *w* * $v_{prim,}$, where *w* is the *sink weight.*
Perturbation combination	---	A combination of value assignments for a *primary increase column*, *primary sink column*, and *sink weight* in the *primary row* and a *secondary increase column* and *secondary perturbation amount* in the *secondary row* (if any).
Perturbation Limit	***L***	The maximum amount by which the p_{ij} values of *primary* and *increase columns* can be raised by.
Perturbation sequence	---	A sequence carried out for a single *perturbation combination* in which the value of the *primary row, increase column* is successively raised by v_{prim} while the *sink column* and *non-sink columns*, if any, are decremented.
Secondary increase column	$d^{\uparrow}$	The column in the *primary row* in which p_{ij} values are increased by an amount v_{prim}..
Secondary increase amount	v_{sec}	A value, 0 ≤ 1.0, by which the *secondary increase column* is incrementally increased.
Secondary row	*s*	The TPM row in which column for which p_{ij} values are perturbed in addition to the *primary row*. By default, the secondary row is the number of the *increase column*.
Sink weight	*w*	The variable *w* whose value, 0 ≤ 1.0, determines the amount, *w* * $v_{prim,}$, that the *primary sink column* p_{ij} should be reduced by.
Sink weight set	---	A set of *sink weights*, *w*, iteratively used to produce alternative values by which the *primary sink column* is reduced by in a *perturbation combination*.

Procedure Determine Perturbation Combinations
// Initially,

- Let $\boldsymbol{np}$ be the number of time periods in a non-homogenous Markov chain.
- Let $\mathbb{Q} = \{\boldsymbol{Q^1} \ldots \boldsymbol{Q^{np}}\}$ as unperturbed set of $\boldsymbol{np}$ $n \times n$ TPMs for $\boldsymbol{np}$ time periods, where q^{tp}_{ij} is the probability of transition from state i to state j in time period tp, $1 \leq tp \leq \boldsymbol{np}$.
- Let $\mathbb{P}$ be the set of $\boldsymbol{np}$ matrices to be perturbed.
- Let *perturbable* be an $n \times n$ Boolean matrix where *perturbable* (i, j) = TRUE if for any $\boldsymbol{Q^{tp}}$ in $\mathbb{Q}$, $q^{tp}_{ij} > 0$. If *perturbable* (i, j) = TRUE, the *jth* column in row i can be perturbed.
- Select row, $\boldsymbol{r}$, as *primary row* to perturb in all matrices in $\mathbb{Q}$, where $\boldsymbol{r}$ corresponds to a transient state in the Markov chain
- Select range and granularity of perturbation
    - Set perturbation limit $\boldsymbol{L}$ (ex. $\boldsymbol{L} = 0.25$)
    - Set primary row perturbation increment amount v_p (ex. $v_{prim} = \boldsymbol{L}/25 = 0.01$
    - Set primary row *sinkWeightSet*, e.g.{0.2, 0.4, 0.6, 0.8, 1.0}
    - Set secondary row perturbation increment amount v_{sec} (ex. $v_{sec} = \boldsymbol{L}/5 = 0.625$)
    - Set *numSteps* equal to number of discrete steps to execute Markov chain (ex. 339)

```
BEGIN // Define perturbation combinations and evaluate perturbation sequence for each
FOR each column $x$ in row $\mathbf{r}$, where perturbable ($\mathbf{r}$, $x$):
    1. Set primary increase column $c^{\uparrow} = x$.
    2. FOR each column $y$ in row $\mathbf{r}$, $y \neq c^{\uparrow}$ and perturbable ($\mathbf{r}$, $y$):
          a. Set sink column $c^{\downarrow} = y$. // This is the primary sink column
          b. Set secondary row s = DetermineSecondaryRow ($\mathbf{r}$, $c^{\uparrow}$)   // Find secondary row
          c. FOR each sink weight $w$ in sinkWeightSet
               DO
                  // Check if there is secondary perturbation. If not evaluate immediately,
                  // otherwise iterate through secondary perturbation combinations and evaluate each.
                  IF $s$ = NULL    //
                  THEN EvaluatePerturbationSequence ($\mathbb{P}$, $\mathbf{r}$, $c^{\uparrow}$, $c^{\downarrow}$, $w$, $L$, $v_{prim}$, numSteps)
                  ELSE for each column $d$ in secondary row $s$, s.t. perturbable ($s$, $d$):
                     i. Set $d^{\uparrow} = d$.           // Vary and iterate over columns of secondary row
                    ii. Set $m_{sec} = 0$
                   iii. Reset $\mathbb{P} = \mathbb{Q}$,
                    iv. WHILE $m_{sec} \leq L$,      // Vary and iterate over different values of $m_{sec}$
                          a. Set $m_{sec} = m_{sec} + v_{sec}$.
                          b. FOR $P^{tp}$ in $\mathbb{P}$, $1 \leq tp \leq np$     // Perturb matrices $p^{tp}$ in $\mathbb{P}$
                             DO                                // according to equation (7)
                               1. Set $p^{tp}_{sd\uparrow} = q^{tp}_{sd\uparrow} + m_{sec}$.
                               2. IF $p^{tp}_{sd\uparrow} \geq 1.0$, CONTINUE.
                               3. FOR each column $c$ in row $s$, s.t. $c \neq d^{\uparrow}$,
                                  DO
                                     $p^{tp}_{sc} = q^{tp}_{sc} - m_{sec} \bullet \dfrac{q^{tp}_{sc}}{\sum_{s \neq d\uparrow} q^{tp}_{sj}}$ for each column $j$ in row $s$
                          c. EvaluatePerturbationSequence ($\mathbb{P}$, $\mathbf{r}$, $c^{\uparrow}$, $c^{\downarrow}$, $w$, $L$, $v_{prim}$, numSteps)
END
```

```
Procedure *DetermineSecondaryRow* (r, $c^{\uparrow}$)
// Accepts primary row r, and primary column $c^{\uparrow}$. Uses default method described in Section 5 to determine
if there is secondary row to perturb
BEGIN
IF $\mathbf{r} = c^{\uparrow}$ RETURN NULL
ELSE RETURN $c^{\uparrow}$
END

Procedure *EvaluatePerturbationSequence* ($\mathbb{P}$, r, $c^{\uparrow}$, $c^{\downarrow}$, w, L, $v_{prim}$, numSteps)
// Accepts a defined perturbation combination including primary row r, primary increase
// column $c^{\uparrow}$, primary sink row $c^{\downarrow}$, primary sink weight w, and secondary row perturbation
// if any embedded in $\mathbb{P}$. Iterates through perturbation sequence in $\mathbb{P}$, perturbing $\mathbb{P}$ by
// incrementing $m_{prim}$ on every iteration according to equation (6) and executes Markov
// chain for each. The matrix Q assumes the role of $P^{(old)}$ in equation (6). The procedure
// also contains detail on border conditions not described in main body of report.
BEGIN
1.  Set primary increment amount $m_p = 0$
2.  Set $\mathbb{Q} = \mathbb{P}$,                              // To be reset on each iteration
3.  WHILE $m_p \le \mathbf{L}$                               // Execute perturbation sequence up to L
    DO                                                  // Increment perturbed value, perturb matrix
        a.  Set $m_{prim} = m_{prim} + v_{prim}$.                 // Set and do one execution of Markov chain
        b.  Reset $\mathbb{P} = \mathbb{Q}$
        c.  FOR $P^{tp}$ in $\mathbb{P}$, $1 \le tp \le \mathbf{np}$,             // Perturb matrix set by next $m_p$
            DO
            1.  IF $(q^{tp}_{rc^{\uparrow}} + m_{prim}) > 1$, CONTINUE   // Go to next matrix if perturbed value ≥ 1
            2.  Set $p^{tp}_{rc^{\uparrow}} = q^{tp}_{rc^{\uparrow}} + m_{prim}$            // Otherwise, set primary increase column

            // Border condition #1! If no non-sink columns in this time period matrix,
            // set Let sink column bear the entire decrease. Otherwise, re-distribute
            3.  IF NOT ($\exists$ c, s.t. $q^{tp}_{rc} > 0$, and $c \neq c^{\uparrow}$, $c \neq c^{\downarrow}$)
                THEN $p^{tp}_{rc^{\downarrow}} = q^{tp}_{rc^{\downarrow}} - m_{prim}$
                ELSE
                  i. Set $p^{tp}_{rc^{\downarrow}} = q^{tp}_{rc^{\downarrow}} - w \bullet m_{prim}$          // Decrease primary sink column
                 ii. FOR each column c in row r, $c \neq c^{\uparrow}$, $c^{\downarrow}$       // Distribute 1-w to non-sink columns
                     DO
                         $p^{tp}_{rc} = p^{tp}_{rc} - (1\text{-}w) \bullet m_{prim} \bullet \frac{q^{tp}_{rc}}{\sum_{c \neq c\uparrow, c\downarrow} q^{tp}_{rc}}$ for columns j in row r

                // Border condition #2! Check for $p_{ij}$ values that may have been driven below zero. If
                // found, redistribute difference proportionally to sink and non-sink columns with $p_{ij} > 0$
                iii. FOR each column c in row r, $c \neq c^{\uparrow}$
                     IF $p^{tp}_{rc} < 0$,
                     THEN
                     1.  Set redistDiff = $| 0 - p^{tp}_{rc} |$
                     2.  Set $p^{tp}_{rc} = 0$.
                     3.  FOR each column d in row r, $d \neq c^{\uparrow}$
                         DO
                           IF $p^{tp}_{rd} > 0$
                           THEN $p^{tp}_{rd} = p^{tp}_{rd} - redistDiff \bullet \frac{q^{tp}_{rd}}{\sum_{j \neq c\uparrow} q^{tp}_{rj}}$ for columns j in row r
        d.  ExecuteMarkovChain ($\mathbb{P}$, numSteps).
END
```

```
ExecuteMarkovChain ($\mathbb{P}$, numSteps)
// Executes Markov chain for numSteps with perturbed matrix P using procedure in section 4.4.
BEGIN
        Set state vector v to initial state
        StepsInPeriod= $d_{period}$ / $d_{ts}$     // Assume $d_{period}$ is the duration of a time period while
                                             $d_{ts}$ is the duration of  time step. //
        FOR step = 1 to numSteps
           $k$ = trunc (step / StepsInPeriod) + 1
           $v = P^k \bullet v$
           Write results ($v$)
END
```

APPENDIX D. Details of Perturbation Combinations

This appendix shows the result of the method of perturbation, applied to rows 1 through 5 of the summary TPM in figure 2. The summary TPM is used because it records all transition probabilities greater than 0. The description of the application of the method results in the following perturbation combinations, of which a subset corresponds to the violation scenarios of interest. These perturbation combinations were used in the experiments described in section 6.

In row 1, there are 2 combinations of primary increase columns and sink columns. When column 1 is the increase column, there are 5 sink weights and no secondary row. Therefore, there are only 5 combinations to consider. When column 3 is the increase column, row 3 is the secondary row with 15 secondary row perturbations (three secondary increase rows times five secondary increase amounts). Column 3 as the increase column thus entails 5 x 15 = 75 perturbation combinations. The total number of perturbation combinations for row 3 is 5 + 75 = 80. If =0.25 and ***L*** = 0.25, each perturbation combination has a perturbation sequence consisting of a maximum of 25 Markov chain simulations. Thus, there would be a maximum of 80 x 25, or 2000 simulations. However, to get more precise results that better corresponded to the actual observed values in row 1, was changed to 0.001 and ***L*** was changed to 0.031. This meant that each of the 80 perturbation sequences now had 31 simulations. This yielded 80 × 31, or 2480 simulations in 82.39 s.

In row 2, there are 9 combinations of increase columns and sink columns alone for columns 2, 3, 4, and 8. Column 8 is discounted as a sink column because of its already low value. There were a total of 425 perturbation combinations with secondary row perturbation.

- When column 2 is the increase column, columns 3 and 4 are sink columns. No secondary row may be selected, because the number of the primary and secondary rows would be the same. Since columns 3 and 4 entail five sink weights each, there are 10 combinations to consider.
- When column 3 is the increase column, columns 2 and 4 are sink columns with row 3 selected as the secondary row. Columns 2 and 4 entail five sink weights each. Row 3 has three possible secondary increase columns, each with five possible secondary increase amounts. Thus there are 2 sink columns × 5 sink weights × 3 secondary increase columns × 5 secondary increase amounts = 150 perturbation combinations.
- When column 4 is the increase column, columns 3 and 4 are sink columns with row 4 selected as the secondary row. Columns 3 and 4 again entail five sink weights each. As a secondary row, row 4 has five possible increase columns, each with five possible increase amounts. Thus there are 2 sink columns × 5 sink weights × 5 secondary increase columns × 5 secondary increase amounts = 250 perturbation combinations when column 4 is the increase column.
- When column 8 is the increase column, columns 2, 3 and 4 are sink columns. No secondary row may be selected, because this column is not in the transient portion of the matrix. Since columns 2, 3 and 4 entail five sink weights each, there are 15 combinations to consider.

Thus for all selections on increase columns for row 2, there are 10 + 150 + 250 + 15 = 425 perturbation combinations. If v_{prim}=0.25 and $\boldsymbol{L}$ = 0.25, each perturbation combination has a maximum of 25 Markov chain simulations for a total of 10625, which took 373.44 s to execute.

In row 3, there are also 6 combinations for increase columns and sink columns in the primary row. There were a total of 460 perturbation combinations with secondary row perturbation.

- When column 2 is the increase column, columns 3 and 4 are sink columns with row 2 selected as the secondary row. Columns 3 and 4 entail five sink weights each. As a secondary row, row 2 has four possible secondary increase columns, each with five possible secondary increase amounts. Thus there are 2 sink columns × 5 sink weights × 4 secondary increase columns × 5 secondary increase amounts = 200 perturbation combinations.
- When column 3 is the increase column, columns 2 and 4 are sink columns and no secondary row may be selected. Since columns 3 and 4 entail five sink weights each, there are 10 combinations to consider.
- When column 4 is the increase column, columns 2 and 3 are sink columns with row 4 selected as the secondary row. Columns 2 and 3 again entail five sink weights each. Row 4 has five possible increase columns, each with five possible increase amounts. Thus there are 2 sink columns × 5 sink weights × 5 secondary increase columns × 5 secondary increase amounts = 250 perturbation combinations when column 4 is the increase column.

Thus for all selections on increase columns in row 3, there are 10 + 200 + 250 = 460 perturbation combinations. If v_{prim}=0.25 and $\boldsymbol{L}$ = 0.25, each perturbation combination has a maximum of 25 Markov chain simulations for a total of 11500, which took 369.69 s to execute.

In row 4, there are also 16 combinations of increase columns and sink columns alone for columns 2-5 and 8. Again column 8 is discounted as a sink column because of its already low value. There were a total of 785 perturbation combinations with secondary row perturbation.

- When column 2 is the increase column, columns 3, 4 and 5 are sink columns with row 2 selected as the secondary row. Columns 3, 4, and 5 entail five sink weights each. As a secondary row, row 2 has four possible secondary increase columns, each with five possible secondary increase amounts. Thus there are 3 sink columns × 5 sink weights × 4 secondary increase columns × 5 secondary increase amounts = 300 perturbation combinations.
- When column 3 is the increase column, columns 2, 4, and 5 are sink columns with row 3 selected as the secondary row. Columns 2, 4, and 5 entail five sink weights each. Row 3 has three possible secondary increase columns, each with five possible secondary increase amounts. Thus there are 3 sink columns × 5 sink weights × 3 secondary increase columns × 5 secondary increase amounts = 225 perturbation combinations.

- When column 4 is the increase column, columns 2, 4, and 5 are sink columns and no secondary row may be selected. Since columns 2, 4, and 5 entail five sink weights each, there are 15 combinations to consider.
- When column 5 is the increase column, columns 3, 4, and 5 are sink columns with row 3 selected as the secondary row. Columns 3, 4, and 5 entail five sink weights each. Row 5 has three possible secondary increase columns, each with five possible secondary increase amounts. Thus there are 3 sink columns $\times$ 5 sink weights $\times$ 3 secondary increase columns $\times$ 5 secondary increase amounts = 225 perturbation combinations.
- When column 8 is the increase column, columns 2-5 are sink columns. No secondary row may be selected, because this column is not in the transient portion of the matrix. Since columns 2-5 entail five sink weights each, there are 20 combinations to consider.

Thus for all selections on increase columns in row 4, there are 1300 + 225 + 15 + 225 + 20 = 785 perturbation combinations. If v_{prim}=0.25 and $\boldsymbol{L}$ = 0.25, each perturbation combination has a maximum of 25 Markov chain simulations for a total of 19625, which took 789.16 s to execute. Subsequently, to provide greater range of the perturbation, $\boldsymbol{L}$ was increased to 0.5 and a perturbation sequence of 39250 simulations was carried out. This took 1480.46 s to execute.

In row 5, there are thus 6 combinations of increase columns and sink columns for columns 4-6. There were a total of 270 perturbation combinations with secondary row perturbation.

- When column 4 is the increase column, columns 5 and 6 are sink columns with row 4 selected as the secondary row. Columns 5 and 6 entail five sink weights each. As a secondary row, row 4 has five possible increase columns, each with five possible increase amounts. Thus there are 2 sink columns $\times$ 5 sink weights $\times$ 5 secondary increase columns $\times$ 5 secondary increase amounts = 250 perturbation combinations when column 4 is the increase column.
- When column 5 is the increase column, columns 4 and 6 are sink columns and no secondary row may be selected. Since columns 4 and 6 entail five sink weights each, there are 10 combinations to consider.
- When column 6 is the increase column, columns 5 and 6 are sink columns. Here again no secondary row may be selected because column 6 is outside the transient portion of the matrix. Since columns 5 and 6 entail five sink weights each, there are 10 combinations to consider.

Thus for all selections on increase columns in row 5, there are 150 + 10 + 10 = 270 perturbation combinations. If v_{prim}=0.25 and $\boldsymbol{L}$ = 0.25, each perturbation combination has a maximum of 25 Markov chain simulations for a total of 6750 possible, which took 260.15 s to execute. The approximately 84230 perturbation sequences for these perturbation combinations are executed in about 56 minutes. To execute the equivalent of these sequences with the simulation program would require in excess of 1 week, conservatively.