Proceedings of the 2011 Pressure Vessels & Piping Division (PVPD) Conference
July 7-11, 2011, Baltimore, MD, USA

PVP2011-57683

IDENTIFYING FAILURE SCENARIOS IN COMPLEX SYSTEMS BY PERTURBING MARKOV CHAIN MODELS

Christopher Dabrowski
National Institute of Standards and Technology
Gaithersburg, MD, USA

Fern Hunt
National Institute of Standards and Technology
Gaithersburg, MD, USA

ABSTRACT

In recent years, substantial research has been devoted to monitoring and predicting performance degradations in real-world complex systems within large entities such as nuclear power plants, electrical grids, and distributed computing systems. Special challenges are posed by the fact that such systems operate in uncertain environments, are highly dynamic, and exhibit emergent behaviors that can lead to catastrophic failure. Discrete Time Markov chains (DTMCs) provide important tools for analysis of such systems, because they represent dynamic behavior succinctly, provide a means to measure uncertainty, and can be used to make quantitative measurements of the potential for change to system performance. Moreover, DTMCs can be extended to be time-inhomogeneous, i.e. to represent behavior that varies over long durations. To date, DTMCs have been proposed for tasks such as fault detection and long-term condition equipment monitoring in real-world complex systems. However, the scope of these models has generally been restricted to describing states and state transitions that directly concern fault conditions or states of degradation. Less work has been done on using DTMCs to represent a more complete range of states a system may enter into during normal operation. Of special interest are sequences of states that involve failure scenarios, in which a system evolves from a normal operating state into undesirable state that leads to widespread performance degradation. Unfortunately, use of large DTMCs often involves large search spaces, a problem which in part motivates our work. This paper describes progress made on developing an approach for using larger, more detailed DTMC models of operational complex systems to uncover potential failure scenarios. The approach uses a combination of methods to perturb a DTMC, simulate alternative system evolutions, and identify scenarios in which a system proceeds from normal operation to failure. Key to the approach is the use of graph theory techniques to reduce the size of the search space involved in exploring alternative behaviors. We show how graph theory techniques can be used to identify critical state transitions which can be perturbed to simulate performance degradation. Using critical transitions, it is also possible to estimate the rate of performance degradation and to understand how this rate is likely to change in response to increased failure incidence. Examples are provided of the use of this approach on a DTMC of significant size to identify failure scenarios in a distributed resource allocation system.

I. INTRODUCTION

In recent years, substantial research has been devoted to monitoring and predicting performance degradations in real-world complex systems within large entities such as nuclear power plants, electrical grids, and distributed computing systems. Such systems pose special challenges because they operate in uncertain environments, are highly dynamic, and may exhibit emergent behaviors that lead to catastrophic failures.

Markov chains provide a basis for modeling complex system dynamics, because they can represent dynamic behavior succinctly, provide a means to capture uncertainty, and can model evolution of system behavior over long durations. To date, Markov chain models have been proposed for tasks such as fault detection [1–4] and long-term condition monitoring of equipment in complex systems [5–8]. Generally, such models have been concerned with in-depth analysis of specific processes within larger systems, focusing on fault conditions and related stages of degradation. While such concise, focused models lend themselves to tractable analysis, they are often less able to capture the broader operational context in which a system degrades from normal operation into a failed state. A model that represents a more complete range of states that a system may enter can facilitate a more detailed understanding of how failures evolve. Such a model can also provide a basis for exploring alternative scenarios and reveal unsuspected potential failures. However, analysis of more detailed, comprehensive models is often complicated due to their substantial size.

In this paper, we describe an approach for using a Discrete Time Markov chain (DTMC) to investigate dynamic system behavior and identify potential failure scenarios in which system-wide performance collapses. The approach employs a combination of analysis techniques, which have heretofore not been used together in order to analyze DTMCs of significant size. First, the DTMC describes a detailed set of states that a system may enter and through which it can progress along different paths. The model is at a sufficient level of detail so as to provide a basis for pinpointing how a system may degrade from normal operation into a failed state. Second, a set of transition probability matrices (TPMs) can be used to simulate the advance of the system through these states in discrete time steps. Each TPM represents a different time period and contains probabilities for transitioning between states that are specific to that period. The TPMs are sequenced to enable simulation of system evolution over time, thus making the DTMC *time-inhomogeneous*. Third, the model can be perturbed to change the probabilities of transitions between states in order to explore alternative execution paths. This allows the analyst to see how a system might behave when certain events occur or conditions of interest arise. Using this information, the analyst can identify *failure scenarios* in which system performance may be affected catastrophically. By failure scenarios, we mean the occurrence of real-world events that lead to system performance collapses, which can be represented by perturbing the DTMC. The perturbed DTMC can then be made to simulate the altered performance of the system as the incidence of failure increases in order to obtain a quantitative estimate of the rate of performance decline and to identify thresholds beyond which performance collapses. This enables a more precise understanding of how and where failures may occur and their potential consequences. Fourth, we demonstrate the application of graph theory concepts to identify critical state transitions in the DTMC, which when suitably perturbed, reveal the potential for performance collapse and the existence of failure scenarios. To reduce the amount of search needed to find critical transitions, we employ minimal s-t cut set analysis on the directed graph of the DTMC. We argue that this method is far more computationally efficient than exhaustive examination of a large space of alternative executions and enable tractable perturbation of larger, more detailed DTMCs. Further, this method can be used to reveal previously unsuspected potential failures, which may be difficult to detect using other methods.

To demonstrate the use of this combined technique, we created a detailed, time-inhomogeneous, *absorbing* DTMC [9] for a cloud computing system. The focus of the DTMC is the process of requesting and obtaining computing resources from the cloud. The DTMC consists of 39 states and 139 potential transitions that encompass both normal and abnormal operations. This DTMC is based on a discrete event large-scale simulation model described in [10], which we also use as a proxy for a real-world system in this study. This 39-state model is far larger than previous models we have analyzed [11–13], which consisted of only seven states. To this DTMC, we apply our combined method to find likely failure scenarios, which we verify in the large-scale model. We consider the extent to which the failure scenarios identified using our method can be used to predict failure in the large-scale model.

This paper first reviews previous research on Markov chain models for monitoring and analysis of complex systems (Sec II). This is followed by an overview of the cloud system domain and a description of a state model for the process of requesting resources from the cloud (Sec. III). We then describe the method by which the time-inhomogenous DTMC of the cloud system was created (Sec. IV) and show how a critical state transition in this example DTMC may be perturbed to reveal potential failure scenarios (Sec. V). We then show how graph theory concepts may be used to identify the most critical state transitions in the sample problem (Sec. VI). This is followed by a discussion of the effectiveness of the set of methods we use, their current strengths as analytical tools, and areas where further work is needed (Sec. VII), and conclusions (Sec VIII). While the approach presented here employs minimal s-t cut set analysis as its basis, elsewhere we describe the use of spectral methods for eigendecomposition to identify critical state transitions [13] and algorithms for exhaustive search of a Markov chain transition probability matrix [12].

II. PREVIOUS WORK

The method discussed here should be distinguished from the well-known use of Discrete Time Markov chains (DTMCs) as tools for providing quantitative measures of system performance and reliability. DTMCs have been used in this manner in a variety of real-world domains. In manufacturing, DTMCs have been used for problems such as for analysis of dependability of manufacturing systems [14], split and merge production line processes and part quality defects [15]. Markov chain analysis has been used to model mean time to failure in communications networks [16], link reliability [17], as well as to examine fault-tolerance and performance in multi-processor computer architectures [18, 19], real-time process control systems [20], and software systems [21, 22]. In grid computing, DTMCs have been used to model workload for scheduling [23] and load balancing [24]. In most of these efforts, DTMCs have been used to quantitatively estimate performance or reliability. Perhaps most closely related to our work are [25, 26], where a control loop is employed to mitigate message delay in a communications network and a Markov chain model is used to represent and measure delay. Our work differs, because instead of measuring reliability, we use DTMCs to examine alternative execution paths for dynamic systems in order to identify failure scenarios in which performance degrades catastrophically.

In the nuclear power plant domain, Markov chain models have also been used to make quantitative estimates of equipment reliability [27, 28]. In addition, DTMCs have been used as tools for fault detection and diagnosis [1, 2, 4, 29–31] and for long-term condition monitoring of equipment [5, 7, 8, 32, 33]. For example, Hidden Markov models have been used to represent the behavior of a target system, in which the current state of the system and the direction in which it is evolving are estimated using external signals and data [29, 31]. Again, our approach differs, because it is geared toward discovering critical state transitions in a DTMC model, which when perturbed, reveal execution paths that lead system failures.

Perturbation analysis of DTMCs has also been a topic of extensive theoretical [34–36] and computational study [37, 38], but for different purposes than we intend. For example, some researchers [39–41] used system performance gradients that are based on key decision parameters in order to perturb Markov models. While gradient-based approaches showed potential for modeling performance change, some issues involving computation of gradients required further research [41]. In addition, gradient-based approaches seem geared for system optimization, rather than for examining alternative execution paths and identifying situations in which performance degrades.

An important research problem in perturbing Markov models is managing the size of the perturbation space, i.e. the number of alternative execution paths to explore. This problem is of central concern in our work and has also been addressed by previous researchers. For example, performance gradients were seen as having the advantage of reducing a potentially large perturbation space because, in some cases, only a few execution paths need to be observed in order to compute the gradient [39, 40]. In other work on performance gradients [41, 42], problem size was reduced by grouping state transitions on the basis of events.

In addition, more general solutions have been proposed for reducing the size of Markov models which could potentially be used to reduce the size of a perturbation space as well. For instance, in [9], the concept of lumping states with similar characteristics into larger aggregations was introduced. Since then, various lumping approaches have been explored that use model structure and symmetry to reduce size [43–45]. Other methods for reducing model size are based on group-theoretic concepts [18], Stochastic Activity Nets [46], stochastic colored nets [19], use of reward variable structures to identify symmetries [47], and use of eigenvector equivalence classes to partition a Markov state space into lumps [48]. Because these approaches may conflict with our goal of pursuing a more detailed model, we rely on different strategies. First, we rely on the inherent stochastic characteristics of Markov chains to provide a succinct representation of the system being modeled, as described in Section IV. More importantly, to reduce the size of the perturbation space, we employ graph theoretic concepts to identify critical state transitions, which if perturbed, reveal the failure scenarios of most interest.

III. A DTMC OF A CLOUD COMPUTING SYSTEM

In this section, we describe the cloud computing system that we will analyze. We provide a state model for the process of requesting and obtaining resources from the cloud. This process constitutes a request lifecycle that, as mentioned previously, is the focus of our analysis. The state model for the request lifecycle was derived by observing a large-scale discrete event simulation model [10] of a cloud system. This large-scale model can represent the actions of to 10^5 distributed nodes and also serves as a proxy for a real-world system in our analysis. In the next section, we discuss the transformation of state model for the request process into a time-inhomogeneous DTMC. In Section V, we explain how this DTMC can be perturbed to identify the potential for performance degradation.

The Discrete Time Large-Scale Simulation

Cloud computing systems are Internet-based distributed systems that provide on-demand computing resources to users for a fee. The large-scale discrete event simulation model of the cloud system [10] simulates in detail how user requests for computing resources are processed and how resources are allocated to requests. We first provide background on the cloud system and then describe a state model of the request lifecycle.

A cloud computing system consists of a *cloud controller* and a set of computing clusters which contain the computing resources, i.e., processors, memory, and disk space. These resources reside on network nodes that are interconnected within the cluster. The large-scale model [10] assumes a set of users, a single cloud controller, a set of clusters subordinate to the cloud controller, and nodes within the cluster that reside on different platforms. User, cloud controller, clusters, and nodes all communicate via the HTTP protocol. Each user periodically submits a *request* to the cloud controller for access to computing resources for a specified period of time. Each request is for a quantity of *virtual machines*, or *VMs*. A VM is a set of processers, together with memory and disk space that is organized as a single unit that can be accessed via a URL. From the user’s standpoint, a VM appears as a single functioning computing platform. The cloud system advertises a finite number of VM types, which can be requested by users. Each user request specifies a minimum and maximum number of each VM type desired. When the cloud controller receives a request, it passes it to its clusters, which each respond with estimates of their ability to satisfy the request. A cluster may respond that (a) it can fully satisfy the request, i.e., provide a *full grant* (allocate the maximum number for each desired VM type); (b) it can partially satisfy the request, i.e., provide a *partial grant* (allocate at least the minimum number of VMs for each requested VM types, but less than the maximum for one or more VM types); or (c) there are not enough resources to available (*NERA*) to meet the minimum for one or more of the requested VM types. The model is restricted in that a request must be implemented on a single cluster and cannot be apportioned across multiple clusters. If there are one or more clusters that can support a full or partial grant, the controller selects one cluster on which to implement the request. The resources for the VMs are then allocated to the user on that cluster, and the user is provided with the URLs to access the requested VMs. For this access, the user pays the cloud a fee per unit time the VMs are held. Of course, the resources necessary to provide the minimum number of requested VMs may sometimes be unavailable on any cluster, in which case the request is denied. The user may then resubmit the request later or give up after some number of retries. To provide the behavior described above, the large-scale model incorporates known cloud system algorithms. Probability distributions are used to determine request arrival times, request resource requirements, and the duration over which users hold VMs. The operation of the model is controlled by a large set of input parameters, including the number of users, different user types, request frequency, as well as the number of clusters, their constituent nodes, and the number and type of VMs provided.

Overview of the State Model for the Request Lifecycle

The state model describes the lifecycle of a single request and forms the basis for analysis in the DTMC. The request lifecycle begins at the time the request is formulated by the user and ends when the request is either granted or when the user must give up after repeated denials. Figure 1 provides an overview of the request lifecycle. The figure shows the major phases of the lifecycle as large-grained, decomposable states. These states are connected by arrows which indicate the circumstances under which a request transitions between the major phases. In subsequent sections, we decompose these phases and provide detailed state models for each.

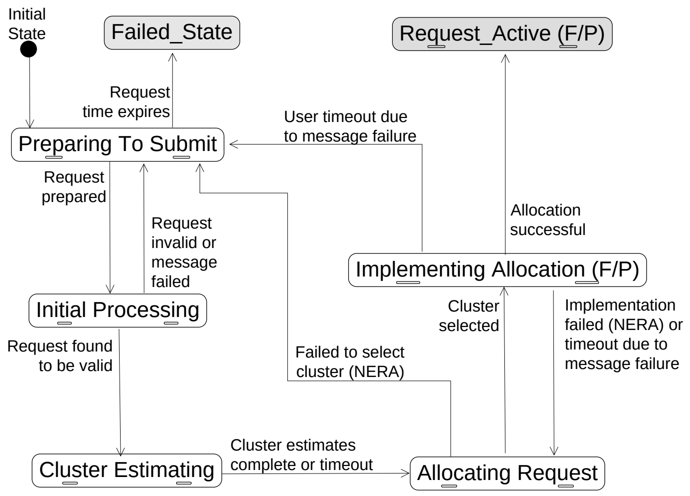

Initial State
Failed_State
Request_Active (F/P)
Request time expires
User timeout due to message failure
Preparing To Submit
Request prepared
Request invalid or message failed
Allocation successful
Implementing Allocation (F/P)
Initial Processing
Cluster selected
Implementation failed (NERA) or timeout due to message failure
Request found to be valid
Failed to select cluster (NERA)
Cluster estimates complete or timeout
Cluster Estimating
Allocating Request

Figure 1. Overview of major phases in the lifecycle of a request. Each phase is decomposed into more detailed states as described below.

A request begins in an *Initial State*, from which it enters the *Preparing to Submit* phase. During this phase, the user formulates the request and makes one or more attempts to submit the request to the cloud controller. In the subsequent *Initial Processing* phase, the request is received by the controller and checked for validity. Invalid requests are returned to the *Preparing to Submit* phase for correction by the user. If the request is valid, the *Cluster Estimating* phase is entered, in which the cloud controller contacts the clusters in order to obtain estimates of the ability of each cluster to satisfy the request, to which clusters respond as described above. When either all responses have been received or the response period ends, the request enters the *Allocating Request* phase, in which the controller selects a cluster on which to implement the allocated request. If such a cluster is found, the request transitions to the *Implementing Allocation* (*F*/*P*) phase. In this phase, the selected cluster assigns VMs to its subordinate nodes and boots them, so as to be consistent with the previous estimate the cluster provided. Note that this phase contains the annotation (*F/P*), which indicates whether the implementation is for a full (*F*) or partial (*P*) grant. This differentiation indicates two distinct states that the request may enter (i.e., the request can enter a state in which it is being processed as a full grant or partial grant, but not both). This (*F/P*) notation is used to make the model diagrams more concise—and will be retained throughout the paper. If no cluster can be found that can provide a full or partial grant, a transition is taken back to *Preparing to Submit* phase for further consideration by the user. If the request enters the *Implementing Allocation* (*F*/*P*) phase and implementation operations succeed, the *Request_Active (F/P)* state is entered, at which point the process is complete. The *Request_Active (F/P)* notation again signifies the existence of two mutually exclusive states that indicate whether a full (*F*) or partial (*P*) grant has been made.

Finally, during the *Implementing Allocation* (*F*/*P*) phase, the implementation action may fail, because the resources used in the cluster's estimate have failed or have otherwise become unavailable. In this case, the request returns to the *Allocating Request* phase for the cloud controller to find a new cluster. In some cases, the implementation succeeds and the controller grants the request, but the message notifying the user fails, in which case the user is unaware of the grant. When this happens, the user, who is waiting for a response, eventually reaches the end of the wait period and issues a *timeout*. In the Markov model, a timeout event causes the state of the request to transition back to the *Preparing to Submit* phase to restart. A request that has been repeatedly resubmitted and failed may finally enter the *Failed_State*, indicating the user has given up. Both the *Failed_State* and the *Request_Active* (*F*/*P*) states are terminal states that signify the end of the request lifecycle. In the DTMC, these are the absorbing states.

Decomposing the Major Phases into a State Space

Decomposition of the major phases in Fig. 1 yields the actual states of the request lifecycle of the DTMC model. In all, there are 39 states and 139 potential state transitions that could be taken. The states denote situations in which the request is subjected to various actions by the user, the cloud controller, or its subordinate clusters. The states are chosen, as much as possible, to denote circumstances and conditions of both normal and abnormal operation in order to enable potential causes of performance degradation to be identified. For instance, some states indicate that a message concerning the request is being transmitted between the cloud controller and its clusters. Such states are important in identifying causes of performance degradation when messages are delayed or lost.

The description of the entire DTMC is lengthy. Therefore, we focus only on the decomposition of the *Cluster Estimating* phase and those parts of the subsequent *Allocating Request* phase that are directly related. While the results of applying our approach for identifying critical transitions and identifying failure scenarios will be provided for the entire cloud model, we will discuss only the analysis of this phase in the body of the paper. Annex A contains the decomposition of the remaining phases, which the interested reader is invited to consult.

Details of Cluster Estimating

The *Cluster Estimating* phase begins when the cloud controller contacts its clusters to request estimates of their ability to satisfy the user request. Each cluster then estimates its ability to satisfy the request. In the subsequent *Allocating Request* phase, the cloud controller reviews these estimates and selects a cluster to implement the request. Here we describe how the Markov model represents the evolution of the state of the request as it is processed by the controller and its clusters.

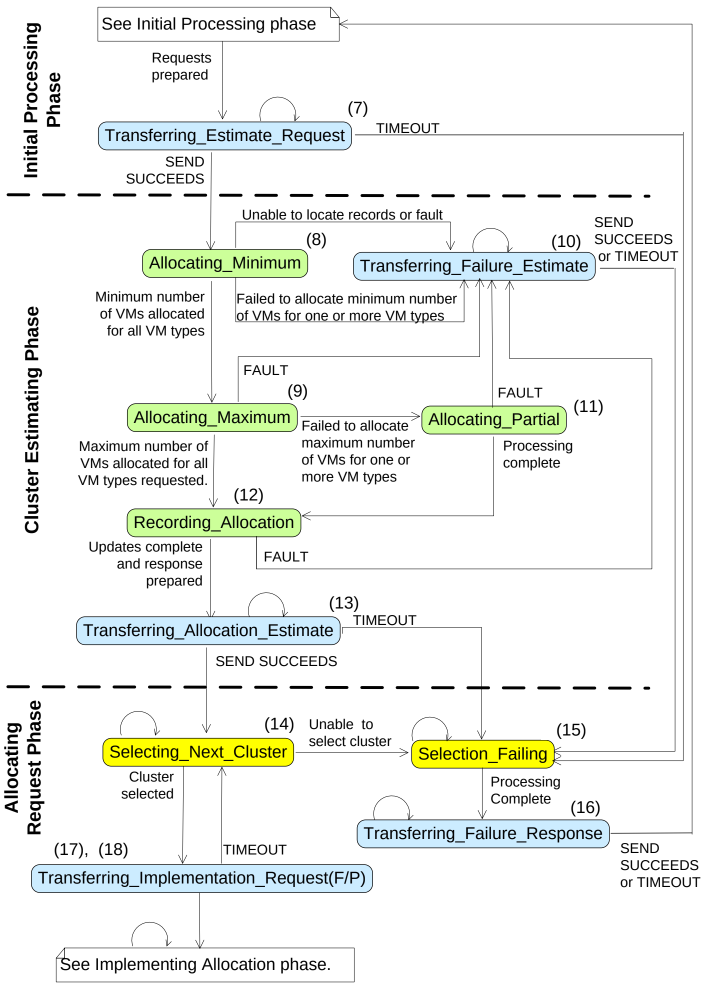
In *Cluster Estimating*, a cluster may provide an estimate that it either: (a) can fully satisfy the request; (b) can partially satisfy the request, or (c) cannot satisfy the request, because a NERA situation exists. In the large-scale simulation, each cluster makes this determination independently. In the Markov model, the actions of the clusters are represented as an aggregated response by all the clusters to the controller. An aggregated response is necessary, because all cluster responses pertain to the state of a single request. Aggregating responses also preserves the stochastic characteristics of the Markov chain representation of a system state, as explained in Section IV.

Figure 2 shows the DTMC state space for the *Cluster Estimating* phase and related parts of the *Allocating Request* phase. In the large-scale simulation, a cluster begins the estimation process by accessing an internal database and retrieving records that describe utilization of its constituent nodes. Using these records, the cluster first determines if a minimum allocation can be supported. If the minimum number cannot be allocated for even one VM Type, the cluster returns a failure response to the cloud controller. In the Markov model, initiation of the *Cluster Estimating* phase is represented by a transition into the *Allocating_Minimum* state. The transfer of a failure response is represented by the *Transferring_Failure_Estimate* state. The request transitions to this state from *Allocating_Minimum* when no cluster can estimate a minimum allocation. Otherwise, the following takes place.

In the large-scale simulation, if a cluster can allocate the minimum number for all the VM Types in a request, it then attempts to allocate the maximum number requested. If it succeeds, a full allocation estimate is recorded in its data structures, and the cluster returns this estimate to the cloud controller. In Fig. 2, these events are represented in the DTMC by a transition of the request state to the *Allocating_Maximum* state, followed by a transition to the *Recording_Allocation* state, and then to the *Transferring_Allocation_Estimate* state.

In the large-scale simulation, a cluster that is processing a request for more than one VM type may be able to allocate the minimum number requested for all VM types but be unable to allocate the maximum number. In this event, the cluster attempts to apportion available resources in such way as to allocate an equal number of VMs for each requested VM type using an iterative round robin procedure, such that the requirement for each VM type is satisfied equally to the greatest extent possible. Usually after this process completes, the request can no longer be fully satisfied, and the cluster returns an estimate to the cloud controller that it can only provide a partial allocation (though in rare cases, the cluster may succeed in restoring the full allocation). As Fig. 2 shows, these events are captured in the Markov model by a transition of the request from the *Allocating_Maximum* state to the *Allocating_Partial* state, followed by a transition to the *Recording_Allocation* state, and finally to the *Transferring_Allocation_Estimate* state.

In the large-scale simulation, a hardware or software fault may occur at any stage in the estimation process, which may cause the estimation operation to be aborted. The Markov model represents fault events in all the states in the *Cluster Estimating* phase with transitions to the *Transferring_Failure_Estimate* state. For example, to represent the occurrence of a fault prevents that prevents access to node utilization records, a transition is shown from the *Allocating_Minimum* state to the *Transferring_Failure_Estimate* state.



Initial Processing Phase
See Initial Processing phase
Requests prepared
(7)
Transferring_Estimate_Request
TIMEOUT
SEND SUCCEEDS
Cluster Estimating Phase
Unable to locate records or fault
(8)
Allocating_Minimum
(10)
Transferring_Failure_Estimate
SEND SUCCEEDS or TIMEOUT
Minimum number of VMs allocated for all VM types
Failed to allocate minimum number of VMs for one or more VM types
FAULT
(9)
Allocating_Maximum
Failed to allocate maximum number of VMs for one or more VM types
FAULT
Allocating_Partial
(11)
Maximum number of VMs allocated for all VM types requested.
Processing complete
(12)
Recording_Allocation
Updates complete and response prepared
FAULT
(13)
Transferring_Allocation_Estimate
TIMEOUT
SEND SUCCEEDS
Allocating Request Phase
(14)
Selecting_Next_Cluster
Unable to select cluster
Selection_Failing
(15)
Cluster selected
Processing Complete
(16)
Transferring_Failure_Response
(17), (18)
TIMEOUT
Transferring_Implementation_Request(F/P)
SEND SUCCEEDS or TIMEOUT
See Implementing Allocation phase.

Figure 2. Decomposition of state space for the *Cluster Estimating* phase. States in which the request is being processed by clusters are shown in green, while states being processed by the cloud controller are in yellow. States in which information about the request is being transmitted are in blue. Transitions to states in external phases are indicated by notes. States of the *Allocating Request* phase below the dashed horizontal line are shown that are relevant to the narrative. (See Annex A for further details). For each state, row numbers from Fig. 3 are provided in brackets. In addition to the state transitions shown in the diagram, all states in the *Cluster Estimating* phase may transition to the *Selection_Failing* state in the event a timeout is issued by the Cloud Controller. User timeout events, which involve transition to the *Preparing to Submit* phase are omitted to simplify the discussion.

Similarly, in the large-scale simulation, the time taken to complete an estimate may exceed the timeout limit of the cloud controller, due to internal faults that cause the cluster to fail and be unable to respond. In this case, the controller declares a timeout and considers only estimates of clusters that responded. In the Markov model, this event is represented by a transition from the state that the request was in when the timeout occurred to the *Allocating Request* phase, as described below. Conceptually, a request could be in any state in the *Cluster Estimating* phase when a timeout event occurs. A complete representation of all transitions that could be taken as a result of the timeouts would render Fig. 2 difficult to read, so we show only the transitions from the *Transferring_Estimate_Request* and *Transferring_Allocation_Estimate* states. Likewise, transitions representing timeouts declared by users are also omitted. (However, all timeout transitions are accounted for in the TPM for the DTMC. See Fig. 3 below).

Cluster Selection in the *Allocating Request* Phase

In the Markov model, the request enters this phase via a transition from the *Transferring_Allocation_Estimate* state or from the *Transferring_Failure_Estimate* state. Because individual cluster results are aggregated, if no cluster can allocate the minimum number of VM types, then the state of the request follows a path in Fig. 2 from the *Allocating_Minimum* state to the *Transferring_Failure_Estimate*, and then to the *Selection_Failing* state. The *Selection_Failing* state represents a circumstance in which the cloud has determined that it cannot satisfy the user's request and is preparing a message notifying the user of this result. The transmission of this message is represented in the Markov model by the *Transferring_ Failure_Response* state. (Note that the *Selection_Failing* state would also be entered if estimate request messages to all clusters failed, or if responses from clusters all fail.)

However, if one or more clusters can respond with an affirmative estimate that indicates a full or partial grant is possible, the request transitions to the *Transferring_ Allocation_Estimate* state and then to the *Selecting_Next_ Cluster* state in Fig. 2. The latter state corresponds to the actions of the cloud controller in selecting one of the responding clusters to implement the request. The controller then sends a message to the selected cluster telling it to implement the request. In the DTMC, this action is represented by a transition from the *Selecting_Next_Cluster* state to one of the two *Transferring_Implementation_Request* (*F/P*) states.

During the selection operation, the cloud controller may be unable to find a cluster to host the request, because either message transmissions failed or the implementation operation on the cluster failed. If no cluster can implement the request, the controller sends a failure message to the user. In the Markov model, this situation is represented by transitions from the *Selecting_Next_Cluster* state to the *Selection_Failing* state and then to *Transferring_Failure_Response* state. This sequence is also shown in Fig. 2. This path could also be taken if software process faults forced the cloud controller to abort the cluster selection process, as we describe further below. (Annex A contains further details of the *Allocating Request* and *Implementing Allocation* (*F/P*) phases.)

Finally, some states in Fig. 2 show transitions back to themselves. These are referred to as *self transitions*, and denote situations where processes dwell in states for a period that exceeds a discrete time step of a DTMC (explained below). Self transitions model situations where the request process remains in a state for a prolonged period, such as may occur when a delayed message transmission leads to a timeout. The DTMC represents all self transitions that have been observed or could potentially occur in the large-scale model.

IV. TRANSFORMNG THE STATE MODEL INTO A DTMC

A Markov chain has the property that the probability of transition between any two states depends entirely on 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* states:

$$\Pr(X_{n+1} = x \mid X_0 = x_0, X_1 = x_1, \ldots, X_n = x_n) = \Pr(X_{n+1} = x \mid X_n) \quad (1)$$

The 39 states and 139 potential state transitions in the state model were reviewed to ensure that the design adhered to this principle. Where necessary, the DTMC was restructured to ensure that the Markov property held.

Calculating Probabilities of Transition

In a Markov chain, probabilities are associated with transitions between states. A DTMC simulates the evolution of system state by state transitions that occur at discrete time steps. To calculate the state-to-state transition probabilities, we used the discrete event large-scale simulation [10]. We observed the operation of the large-scale model and summed transition frequencies over a simulated duration. The summation was done by determining where state transitions occur in the large-scale model code and inserting counters at those places[1]. State transition probabilities were derived as follows. Given states s_i, s_j, $i,j = 1\ldots n$ where n=39, 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 states s_k, or

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

Here i and j may be equal, to allow for a self transition, which is counted if the request process remained in a state longer than a discrete time step, which for this problem was chosen to be 100 s. The resulting TPM is a 39 × 39 stochastic matrix, shown in Fig. 3, where rows stand for the state the transition originates from, or the *from state*, *i*, and columns represent states the transition goes to, or the *to state*, *j*. Each matrix element in a TPM contains a p_{ij}. As in any stochastic TPM, the transition values of all columns in a row must sum to 1.0.

[1] Note that as part of this operation, transitions relating to cluster estimates for the same request were combined to provide an aggregate representation of the transition probability of a request. See Sec. III.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
1	Initial	0.995	0.005	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	Thinking	0	0.962	0.038	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	Submitting	0	ε	0.873	0.122	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.0001
4	Transferring_User_Request	0	0	0.022	ε	0.978	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	Initiating_Request_Session	0	0	ε	0	0	1-2ε	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	Preparing_Cluster_Estimate_Requests	0	0	ε	0	0	0	1-2ε	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	Transferring_Estimate_Request	0	0	ε	0	0	0	ε	0.993	0	0	0	0	0	0	0.007	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	Allocating_Minimum	0	0	ε	0	0	0	0	0	0.248	0.752	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	Allocating_Maximum	0	0	ε	0	0	0	0	0	0	ε	0.464	0.536	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	Transferring Failure_Estimate	0	0	ε	0	0	0	0	0	0	ε	0	0	0	0	1-2ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	Allocating Partial	0	0	ε	0	0	0	0	0	0	ε	0	1-3ε	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	Recording_Allocation	0	0	ε	0	0	0	0	0	0	ε	0	0	1-3ε	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	Transferring_Allocation_Estimate	0	0	ε	0	0	0	0	0	0	0	0	0	0	1-2ε	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	Selecting_Next_Cluster	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0.168	0	0.402	0.429	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	Selection Failing	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	ε	1-3ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	Transferring_Failure_Response	0	0	0.952	0	0	0	0	0	0	0	0	0	0	ε	0	0.048	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	Transferring_Implementation_Request (F)	0	0	ε	0	0	0	0	0	0	0	0	0	0	0.012	0	0	ε	0	0.133	0	0.855	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	Transferring_Implementation_Request (P)	0	0	ε	0	0	0	0	0	0	0	0	0	0	0.012	0	0	0	ε	0	0.053	0	0.934	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	Queued_for_Implementation (F)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	ε	0	1-3ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	Queued_for_Implementation (P)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	ε	0	1-3ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	Verifying_Allocation (F)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0.821	0.061	0	0	0	0	ε	0	0	0.118	0	0	0	0	0	0	0
22	Verifying_Allocation (P)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	ε	0.684	0	0	0	0	ε	0	0	0.313	0	0	0	0	0	0	0
23	Launching_Instances (F)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0.485	0	0.496	0.018	0	0	0	0	0	0	0	0	0	0	0
24	Launching_Instances (P)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0.317	0	0.587	0.096	0	0	0	0	0	0	0	0	0	0
25	Reallocating_VM_Instances (F)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	1-5ε	0	ε	0	ε	0	ε	0	0	0	0	0	0	0	0	0	0
26	Reallocating_VM_Instances (P)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	1-4ε	0	ε	0	0	ε	0	0	0	0	0	0	0	0	0	0
27	Recording_Launch (F)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ε	1-3ε	0	0	0	0	0	0	0	0	0
28	Recording_Launch (P)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ε	0	1-3ε	0	0	0	0	0	0	0	0
29	Rolling_Back_Implementation	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1-2ε	0	0	0	0	0	0	0
30	Transferring_Implementation_Success (F)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ε	0	0	1-3ε	0	0	0	0	0	0
31	Transferring_Implementation_Success (P)	0	0	ε	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ε	0	0	1-3ε	0	0	0	0	0
32	Transferring_Implementation_Failure	0	0	ε	0	0	0	0	0	0	0	0	0	0	1-2ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ε	0	0	0	0	0	0	0
33	Preparing_Grant (F)	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1-ε	0	0	0	0
34	Preparing_Grant (P)	0	0	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1-ε	0	0	0
35	Transferring_Grant (F)	0	0	0.028	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.077	0	0.895	0	0
36	Transferring_Grant (P)	0	0	0.014	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.038	0	0.948	0
37	Request_Active (F)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
38	Request_Active (P)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
39	Failed_State	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 3. Summary TPM computed as a weighted average of time period TPMs. Rows indicate states from which transitions originate, while numbered columns indicate destination states. Shaded matrix elements with values > ε indicate transitions that were taken during the execution of the large-scale system simulation. Matrix elements with values of ε indicate transitions whose unperturbed transition probabilities are less than ε, where ε = $1.0e10^{-6}$ for over 27000 hours of simulated operation of the large-scale simulation under normal conditions. Values are rounded to three significant digits

The Markov chain described in this paper can be further classified as an *absorbing chain* [9]. This is because in the model shown in Fig. 2 and in Annex A, a request process can enter and remain in 36 of the 39 states temporarily, but always exits from these states. At some point the process enters either the *Request_Active* (*F/P*) or *Failed* absorbing state, where it is considered to have completed. Once the request is in an absorbing state, only self-transitions are possible over any future duration. While the summary TPM shown in Fig. 3 is useful for providing an overview of the dynamics of the system and for illustrating concepts, it requires further elaboration to capture time inhomogeneity.

Stochastic Behavior and Time Inhomogeneity

In a large cloud system there may be many requests that are being processed concurrently, each of which can be in any of the 39 states at a given time. If these numbers are very large, it will be impossible to model a system state in which all requests are accounted for simultaneously and individually. However, as we will show below, the stochastic nature of a Markov chain makes it possible to represent the concurrent progress of requests through the 39 states in terms of the proportion of requests that are allotted to each state. In this way, it is possible to represent the system state as a 39-element vector in which the value of each vector element represents the proportion of requests in one of the 39 states. This simple method of representing the system state facilitates problem analysis.

In a real-world cloud system that operates over a long duration, as well as in the large-scale simulation of a cloud, the frequency of request submission may vary significantly over time. Availability of computing resources or network congestion will vary with time, and both affect the rate of transition between states. To capture changes in such system dynamics over time, different TPMs must be computed for different times. We therefore sum transition frequencies in the large-scale model by creating time-period partitions and compute a separate TPM for each period.

These time period TPMs allow representation of our model as an inhomogeneous Markov chain, also referred to as a *piece-wise homogenous* Markov chain [49], in which there are a bounded number of pieces that correspond to different time periods. To compute the inhomogeneous Markov chain, frequencies from the large-scale cloud system simulation were summed separately for 27000 time periods of *3600* s, or one hour each. The 27000 time-period matrices captured cloud system dynamics over the 27000-hour duration. To provide a basis for the perturbations to be described below, the first 16 hours of the 27000-hour duration were used. The transition probabilities for the first 16 hours are represented in the summary TPM in Fig. 3. To produce this matrix, the transition probabilities for the first 16 of the 27000 time period TPMs were weight averaged on the basis of the relative transition frequencies in each period[2]. The summary TPM in Fig. 3 shows all transitions that could potentially be taken. Transitions likely to occur only under extreme conditions (or when the model is perturbed) are noted by matrix elements with values of ε, which indicates that their probability of occurring during a simulated 27000-hour execution of the DTMC is $\varepsilon < 1/(27000$ time periods × 36 time steps per period$) = 1.0e10^{-6}$. The choice of ε is based on the idea that the probability of an "extreme" event, i.e. an event that is unlikely to be observed in T time steps of the simulation is $O(1/T)$. Such extreme events can be caused by failures, which we discuss below.

Markov Simulation

A well-known use of stochastic TPMs in a DTMC is to simulate how a dynamic system evolves over a time in discrete time steps, where each step represents a fixed time duration. Here, we provide details on this method, which we refer to as *Markov simulation*. A more formal description appears in [13].

In this investigation, a discrete time step was chosen to represent 100 s, or $h = 100$. The Markov simulation was applied to the first 16 time periods of the total 27000 time period duration. Since a time period covers a duration of $d_{period} = 3600$ s, each of the 16 time-period TPMs represents $S = d_{period}/h$, or 36, steps. In Markov simulation, the state of the system is summarized at any time step in a 39-element state vector, which we denote as v. In v, each of the 39 elements represents the proportion of requests in one of the 39 states of the DTMC. The elements of the state vector v_m , which represents the system state at time step m, are ordered so as to correspond to the numbered states in Fig. 3. Thus the first element in v_m contains the proportion of tasks in the *Initial* state at time step m, the second contains the proportion of tasks in the *Thinking* state, and so forth. Using equation (3), the vector v_m is multiplied by the TPM, Q^{tp}, for the applicable time period tp to produce a new system state v_{m+1}, in order to evolve the system state over a single discrete time step. That is,

$$(Q^{tp})^T * v_m = v_{m+1}, \text{ where } tp = \textit{integral value}\,(m/S) + 1 \qquad (3)$$

where T indicates a matrix transpose. Starting with v_1, which represents a system state with a value of 1.0 for the *Initial* state (see Fig. 1) and 0 for all others (i.e, all requests begin in the *Initial* state), equation (3) is repeated for 576 time steps (representing 57,600 s, or the 16 simulated hours)[3]. This results in a system state vector, v_{576}, in which the sum of the proportion of requests is distributed over the 38 states other than the *Initial* state (i.e., all states have transitioned from *Initial*). In an absorbing chain, requests eventually transition into an absorbing state, i.e. the *Request_Active* (*F*/*P*) or *Failed* state in this model, where they remain permanently. Thus, a measure of system performance as the simulation progresses is the proportion of requests that enter the *Request_Active* (*F*/*P*) states, which represents requests that have received full or partial grants. On the other hand, a performance collapse may be simulated when a large proportion of requests enter the *Failed* state, or when they are otherwise unable to enter *Request_Active* (*F*/*P*).

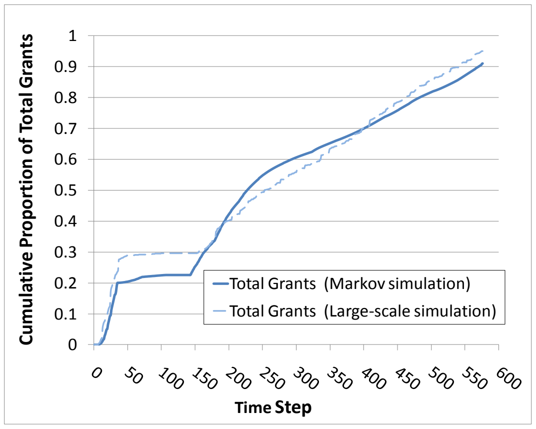


Figure 4a. Comparison of total grants for Markov and large-scale simulations over first 16 time periods of the simulated duration, where each time period consisted of 36 time steps of 100 s each.

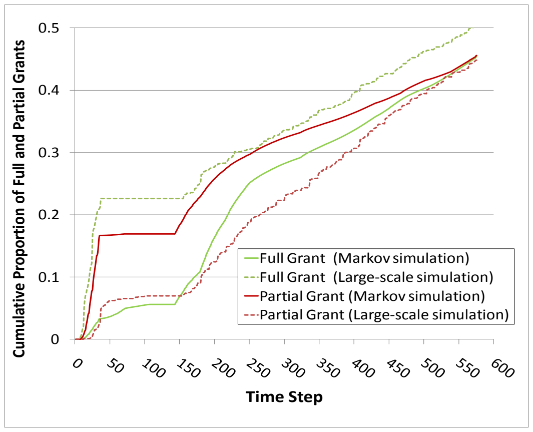


Figure 4b. Comparison of full and partial grants for Markov and large –scale simulations over first 16 time periods of the simulated duration.

Fig. 4 compares the results of applying Markov simulation to the results produced by the large-scale simulation over the first 16 time periods. Figure 4a shows the growth of total grants (full

[2] In the summary matrix, each weight-averaged probability of each transition, p_{ij}, is computed as follows

$$p_{ij} = w_i^1 p_{ij}^1 + w_i^2 p_{ij}^2 + \ldots . w_i^{nper} p_{ij}^{nper}$$

in which each w_i^l represents the weight for a row i in time period l, $l \in \{1..\, nper\}$where $nper$ is the number of time periods over which the summary matrix is computed. Each w_i^l is computed by

$$w_i^l = \sum_{1 \le j \le n} f_{ij}^l \Big/ \sum_{1 \le tp \le nper} \sum_{1 \le j \le n} f_{ij}^{tp}$$

where each f^{tp} is the frequency of transition from state i to state j in time period tp and n is the dimension of the matrix.

[3] Note that each repetition of equation (3) is an independent operation that is applied to a proportion (fraction) of the total number of requests in a state. Repeated application of (3) results in distribution of this proportion across other states as defined by the state transitions in the TPM.

and partial) in both simulations, while Fig. 4b shows the growth full and partial grants separately. The two figures show that the Markov simulation more closely approximates the large-scale simulation with respect to total grants (Fig. 4a). However, there is a greater discrepancy when full and partial grants are considered separately (Fig. 4b). We will return to this issue in Section V, when discussing the results of the perturbations. We also note that the Markov simulation runs faster. The results shown in Fig. 4 were produced by a single execution of the large-scale simulation [10] which required almost 3 minutes, while the Markov simulation took about 4 s. Thus, the Markov simulation was 25 times faster than the large-scale simulation.

V. PERTURBING STATE TRANSITIONS TO MEASURE THE EFFECTS OF FAILURE

In this section, we show that the perturbation of a related set of state transitions coupled with Markov simulation can be used to identify failure scenarios that reveal potential catastrophic performance degradations. In so doing, we assess the extent to which these techniques can be used to make predictions. In the next section, we describe a method for using graph theory concepts to identify all sets of critical state transitions in the directed graph of a Markov chain that can be perturbed to identify failure scenarios.

Consider a situation where the clusters are unable to make estimates of the minimum VMs to be allocated. This could occur, for example, because resource databases on the clusters have become inaccessible due to a software or hardware fault. In the large-scale simulation, this failure would then lead the clusters to abort the estimation computation and return failure estimates to the cloud controller. In Fig. 2 (see Sec. III), this failure scenario corresponds to a reduced ability to transition from the state *Allocating_Minimum* to the state *Allocating_ Maximum*. We write this transition, *Allocating_Minimum* → *Allocating_Maximum*, for notational convenience. In the TPM shown in Fig. 3, the perturbation of *Allocating_Minimum* → *Allocating_Maximum* is modeled by lowering the probability of transition from the *Allocating_Minimum* state to the *Allocating_Maximum* state, i.e., lowering the value of column 9 in row 8, or TPM element {8, 9}. Since transition probabilities in a row of a stochastic matrix must sum to 1, the transition probabilities of one or more columns other than column 9 in row 8 must be raised by a corresponding amount. Because error handling procedures would require the cluster to return failure estimates in this scenario, the obvious choice would be to raise the transition probability of column 10, *Allocating_Minimum* → *Transferring_Failure_Estimate*, or TPM element {8, 10}. Thus, a perturbation applied to all time-period TPMs, in which the related TPM element {8, 9} is systematically lowered by increments while TPM element {8, 10} is raised, serves to model how the increasing incidence of internal database failure affects the proportion of requests that receive either full or partial grants. Fig. 5 shows the TPM elements that are affected by this perturbation.

		8	9	10
8	Allocating_Minimum	0	0.248 ↓	0.752 ↑
9	Allocating_Maximum	0	0	ε
10	Transferring Failure_Estimate	0	0	ε

Figure 5. TPM elements from Fig. 3 that were raised and lowered as a result of perturbation shown in Fig. 6a.

Fig. 6a shows a Markov simulation in which this perturbation is applied to the TPM set for the first 16 hours (16 time periods) of the 27000-hour simulated duration described in Section IV. Fig. 6a shows the predicted effect on the total proportion of requests that received grants (both full and partial) as the probability of transition of *Allocating_Minimum* → *Transferring_Failure_Estimate* (the value of TPM element {8, 10}) is raised in increments of 0.01, while the transition probability of *Allocating_Minimum* → *Allocating_Maximum* is lowered (the value in TPM element {8, 9} is decreased). In Fig. 6a, the horizontal axis shows the increase in the transition probability of *Allocating_Minimum* → *Transferring_Failure_ Estimate*. The left vertical axis shows units for the proportion of requests that were granted, while the right vertical axis shows units by which the transition probability of *Allocating_Minimum* → *Allocating_Maximum* is reduced.

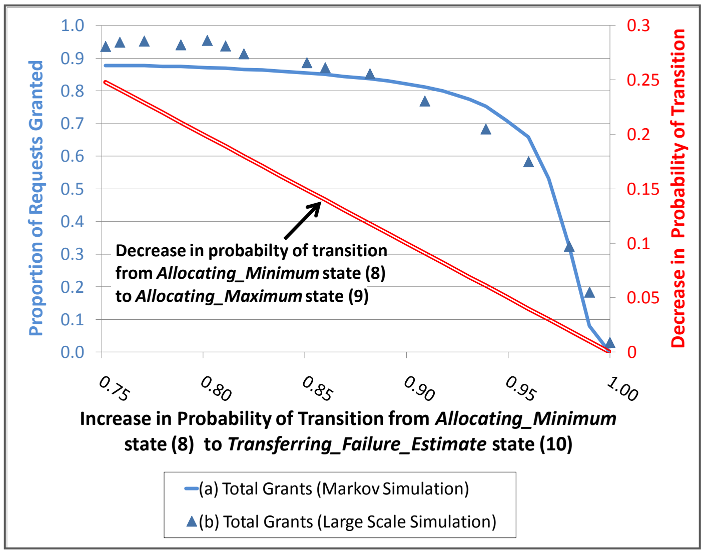


Figure 6a. Decline in total requests granted due to cluster allocation estimation failure (a) as estimated by perturbing the DTMC and using Markov simulation; and (b) as actually computed in the discrete time large-scale simulation. The DTMC is perturbed to raise the probability of transition for *Allocating_Minimum* → *Transferring_Failure_ Estimate* (TPM element {8, 10} and lower the probability of transition for *Allocating_Minimum* → *Allocating_Maximum* (TPM elements {8, 9}. The increase of this probability of transition for *Allocating_ Minimum* → *Transferring_Failure_Estimate* is indicated by the horizontal axis, while the left vertical axis provides unit values used to describe the proportion of requests granted. The right vertical axis provides units for the decrease in probability of the state transition *Allocating_Minimum* → *Allocating_Maximum*.

To test whether the failure scenario identified using the DTMC actually caused a catastrophic performance collapse in the discrete time large-scale simulation, the failure of clusters to make minimum allocation estimates was introduced into the large-scale simulation. The proportion of requests granted was recorded as the probability of allocation estimation failure was increased from 0 (no occurrence of failure) to 1 (all attempts to make minimum allocation estimates by clusters failed). Fig. 6a shows the result of simulating this failure in the large-scale simulation for comparison. To equate the Markov and large-scale simulations, the change in transition probabilities was recorded in the large-scale simulation as the incidence of allocation estimation failure in clusters increased[4].

Close examination of Fig. 6a shows that the Markov simulation could be used to accurately predict that total grants will fall to 0 in the large-scale simulation as the incidence of failure rises. In Fig. 6a, the perturbation of the Markov chain also estimates that the drastic decline in total grants will not occur until the incidence of failure reaches a fairly high rate. This information might be used to an analyst to judge that the threshold beyond which performance collapses occurs when the probability of transition for *Allocating_Minimum* → *Transferring_Failure_Estimate* exceeds 0.80 (on the horizontal axis). In this way, the Markov chain could be used to make a quantitative estimate of system behavior and to identify a threshold failure rate, beyond which system performance is likely to collapse.

However, in Fig. 6b, we also see that Markov chain perturbation can be less accurate in modeling the impact of the perturbation on full grants and partial grants separately. The figure shows that the Markov chain perturbation underestimates the rate of full grants and overestimates the rate of partial grants in the large-scale simulation in response to failure. It is likely that the inaccuracy stems from the fact that the perturbation approach described here currently is more suited to predicting the impact of changes to localized parts of the Markov chain, which directly interact with each other. The approach is less able to predict the impact of a perturbation on state transitions that are further apart in the graph of the DTMC. We will return to the discussion of this issue.

Keeping this caveat in mind, the example shows that perturbation of a large, detailed Markov chain can be used to determine if failures to specific system components can lead to severe performance degradations of a system and to estimate the quantitative extent and rate of decline. Moreover, the Markov simulation executes faster than the large-scale simulation. In the former, each perturbation level took on the average of 1.65 s, while the latter required an average of 49.7 s. In [11], we found that Markov simulation was generally two-orders of magnitude faster than large-scale simulation in perturbing a similar problem.

[4] To record transition probabilities in the large-scale simulation, counters were inserted as described in Sec. IV. The recorded transition probabilities are dependent on, but distinct from, the rate at which data structure failures occur.

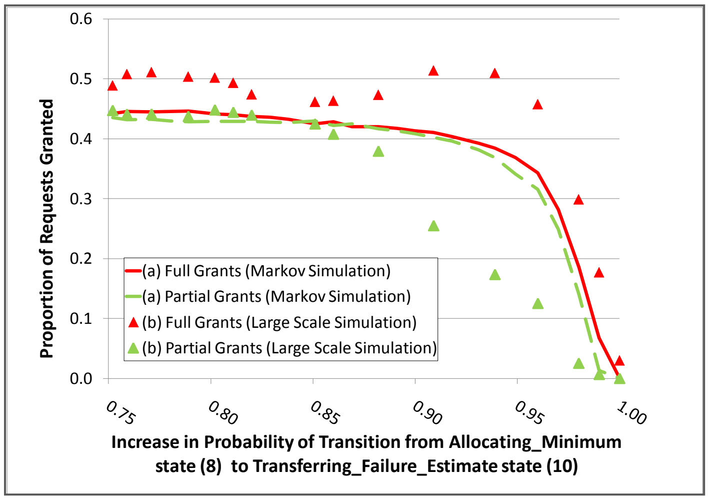


Figure 6b. Decline in full and partial grants, due to cluster allocation estimation failure (a) as estimated by perturbing the DTMC and using Markov simulation; and (b) as computed in the discrete time large-scale simulation. The DTMC is perturbed to raise the probability of transition for *Allocating_Minimum* → *Transferring_Failure_Estimate* (TPM element {8, 10} and lower the probability of transition for *Allocating_Minimum* → *Allocating_Maximum* (TPM elements {8, 9}.

VI. USING MINIMAL S-T CUT SET ANALYSIS TO REDUCE THE PERTURBATION SPACE

The previous example illustrated how a DTMC model of a dynamic system can be perturbed to identify failure scenarios and predict resulting performance changes. However, the example involved perturbation of only one possible combination of state transitions among many. In the DTMC discussed in this paper, there are potentially many other such combinations that may have to be examined in order to find all critical state transitions where perturbation leads to drastic performance declines. Of these, many combinations will consist of transitions that originate from different states (have different *from states*), and thus would involve perturbation of more than one row of the TPM in Fig. 3. Since it is highly unlikely that all critical transitions could be found by visual inspection of large Markov chain, other means are necessary.

Consider a complete examination of all rows of the TPM in Fig. 3, where for each row we perturb all possible combinations in which the transition probability of one column is raised while the transition probabilities of one or more other non-zero columns in the same row is lowered to 0 (as was done in the preceding example in Fig. 6a). If the *Initial state* and the three absorbing states are excluded, there are $n-4$ possible columns from which increase and decrease columns are to be selected, where $n = 39$. For one column of increase and one column of decrease, the number of feasible perturbations (corresponding to feasible scenarios) is 115 (depending on assumptions made as to which columns can be perturbed). However, if we consider perturbing different combinations of rows together in order to find combinations of state transitions in different rows which together are critical, the figure increases by a factor of

$$\binom{n-4}{r}$$

where n is the dimension of the matrix and r is the number of rows to be considered in combination. Using this formula, there would be 595 combinations of two rows to examine. If we consider that each row has approximately 3 sets of perturbations to examine (*integral value* (115/35) = 3), there would be 9 combinations of perturbations to examine for each pair of rows, which would require an estimated 5355 perturbations of the kind described in the example in Section V. (If we consider three rows in combination, approximately 58,905 additional perturbations would need to be carried out.) Clearly, a brute force search to find all critical state transitions is likely to be infeasible for a large problem. Yet in a large TPM, it is necessary to examine many rows to find those individual state transitions, and combinations of transitions which can be perturbed to reveal performance degradations.

Therefore in this section, we introduce an approach in which graph theory concepts are used to avoid exhaustive search and enable tractable perturbation of a large DTMC. Specifically, we use minimal s-t cut set analysis to identify critical state transitions which can be directly perturbed, as was done in the preceding example, to reveal the potential for performance collapses. Minimal s-t cut set analysis is a technique that can be used to find all combinations of state transitions, which if removed from a directed graph of a Markov chain, cut all paths from the *Initial* state to a desired absorbing state—in this case, both the *Request_Active (F)* state for full grants or the *Request_Active (P)* state for partial grants, denoted as *Request_Active (F/P)*. In what follows, we first review graph theory concepts necessary to the understanding of minimal s-t cut set analysis and then show how this technique can be used to identify sets of critical state transitions that most affect system performance. Through examples, we show that systematic perturbation of these sets of transitions can be used to identify failure scenarios, in which the rate of performance decline can be estimated as the occurrence of failure increases.

Definitions

In graph theory, a graph G (V, E) consists of a set of vertices V connected by edges from the set E. A directed graph is a graph in which edges can be traversed in only one direction. A Markov chain is a directed graph, in which vertices correspond to states and directed edges correspond to state transitions. A directed path through this graph is a sequence of state transitions from one state to another. In this problem, the directed paths of most interest are non-cyclic paths that lead from the *Initial* state to one of the two desired absorbing states, denoted *Request_Active (F/P)*. A set of one or more edges, which if removed, disconnects all paths between two vertices s and t is referred to as an *s-t cut set* [50]. An s-t cut set is a *minimal s-t cut set* if removal of any edge from the cut set reconnects s and t. By finding minimal s-t cut sets consisting of state transitions that disconnect the *Initial* and *Request_Active (F/P)* states, it is possible to know where reducing the related transition probabilities to 0 can halt the progress of requests to completion.

Fig. 7 shows an example of a minimal s-t cut set from the *Allocating Request* phase (see Fig. 2), which is directly related to the perturbation discussed in the preceding section (see Fig. 6a). This cut set consists of a single state transition, *Allocating_Minimum* → *Allocating_Maximum)*, or TPM element {8, 9}. Disconnecting the directed graph of the Markov chain at this point prevents the progression of requests to either of the two absorbing states, *Request_Active (F/P)*, as demonstrated by the perturbations shown in Fig. 6a in which the probability of this transition is reduced to 0.

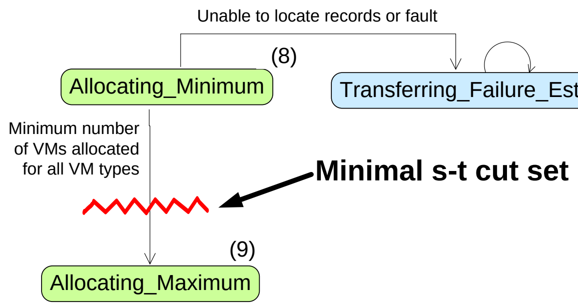

Unable to locate records or fault
(8)
Allocating_Minimum
(10)
Transferring_Failure_Estimate
Minimum number of VMs allocated for all VM types
Minimal s-t cut set
(9)
Allocating_Maximum

Figure 7. A minimal s-t cut set consisting of a single state transition from *Allocating_Minimum* → *Allocating_Maximum* from Fig. 2 (In the TPM in Fig. 3, this cut set involves TPM element {8, 9}). For each state, related row numbers in Fig. 3 are provided.

Identifying Critical State Transitions Using Minimal s-t Cut Set Analysis

A number of algorithms have been developed for enumerating all minimal s-t cut sets between two vertices in a directed graph [51–53]. All require considerable computational effort for large graphs, and we will return to the question of scalability of various approaches for cut set generation in the discussion section (see Sec. VII). However, for purposes of illustrating the use of minimal s-t cut analysis on the problem described in this paper, we implemented the algorithm of [51] to find all cut sets in the directed graph of the Markov chain presented in this paper. This algorithm has been proven to be able to find all minimal s-t cut sets between two vertices s and t in directed graph. The algorithm was applied to a directed graph of our Markov chain, in which the *Initial State* was designated as vertex s and the absorbing states *Request_Active (F/P)* (38) and (39) were together designated as vertex t, so that a cut set included state transitions that cut all paths to both states. All potential state transitions that could be taken were included in this directed graph. The algorithm generated 159 minimal s-t cut sets, which ranged in size from one state transition to 8. A reasonable assumption in analyzing this set of transitions is that in this domain, as in most large domains, the most critical cut sets will consist of a small number of transitions [54], since small combinations represent events that are more likely to occur and impact the system being modeled. Therefore, we used the implemented algorithm of [51] to generate minimal s-t cut sets and then sorted them on the basis of the number of transitions they contain, in ascending order.

Tables 1 and 2 show the 33 minimal s-t cut sets generated using the implemented algorithm of [51] which consist of one or two state transitions. Analysis of cut sets with three transitions is omitted for lack of space. Table 1 contains 10 cut sets having one transition. Table 2 contains 23 cut sets that have two transitions. In both tables, the total probability of transition of cut set members is used as a secondary ordering criterion[5]. Each cut set entry lists the state transitions in the cut set using the state numbers given in Fig. 3 for identification. Each cut set entry lists the figures in which the transitions in the cut set appear. Table 2 also shows the number of states from which the transitions in the set originate, i.e., the number of *from states*.

Table 1. Ranking of minimal s-t cut sets of the directed graph of the Markov that consist of one transition. For each cut set entry, member state transitions are listed as TPM elements from Fig. 3, together with the total probability of transition, and the identifying number of the figures in which the transitions in the cut set appear.

	Set of member transitions from Fig. 3	Total Probabilty	Figure Reference
1-1	{1, 2}	0.001	A.1
1-2	{2, 3}	0.025	A.1
1-3	{3, 4}	0.124	A.1
1-4	{8, 9}	0.264	2
1-5	{4, 5}	0.978	A.2
1-6	{6, 7}	0.978	A.2
1-7	{7, 8}	0.990	2
1-8	{13, 14}	0.991	2, A.2
1-9	{5, 6}	0.995	A.2
1-10	{12, 13}	1.000	2

Each cut set in these tables can be perturbed in the same manner as the example discussed in Section V and shown in Fig. 6a, with the exception that in many cut sets, multiple transitions are involved. By reducing the probability of transition of *all* state transition in any multiple-transition cut set to 0, the flow of requests to the absorbing states *Request_Active* (*F*/*P*) is also reduced to 0. Thus, each minimal s-t cut set identifies a set of critical state transitions. As in the example in Fig. 6a, perturbation of all transitions can be used to estimate likely rates of performance degradation and to identify performance thresholds. Below, we will provide two examples.

It is possible to further narrow down which cut sets in Tables 1 and 2 are of most interest by leveraging domain expertise about whether the transitions in the cut set concern states in which the request is being processed by the cloud controller, clusters, or network. The single-transition cut sets of Table 1 occur in the *Preparing to Submit*, *Initial Processing*, and *Cluster Estimating* phases. Cut set #1-1 consists of the transition from the *Initial state*. Cutting this transition has obvious consequences. Cut sets #1-2 and #1-3 consist of state transitions from the *Preparing to Submit* phase that relate to possible human error (see Annex A). Cut sets #1-5, #1-7, and #1-8 concern transitions that relate to network transmission, and so may be of value in analysis of the effect of network failures. Cut sets #1-6 and #1-9 describe component failures in the *Initial Processing* phase. Cut set #1-4 can be used to model the effect of faults in cluster components that lead to failure to allocate the minimum numbers of VMs for requests during the *Cluster Estimating* phase. This cut set is shown in Fig. 7. The result of perturbing the single state transition in the cut set was discussed in Section IV and is shown in Fig. 6a. Cut set #1-10 is similar to cut set #1-4 because it can be used to model the effects of not being able to update the cluster's internal databases and complete the estimation operation (see Fig. 2).

[5] Note that the total probability of transition for all state transitions in a cut set may exceed 1. This is because a minimal s-t cut set may consist of state transitions involving multiple states, rather than a single state. In a stochastic matrix, only the transition probabilities of state transitions from a single state must sum to 1.

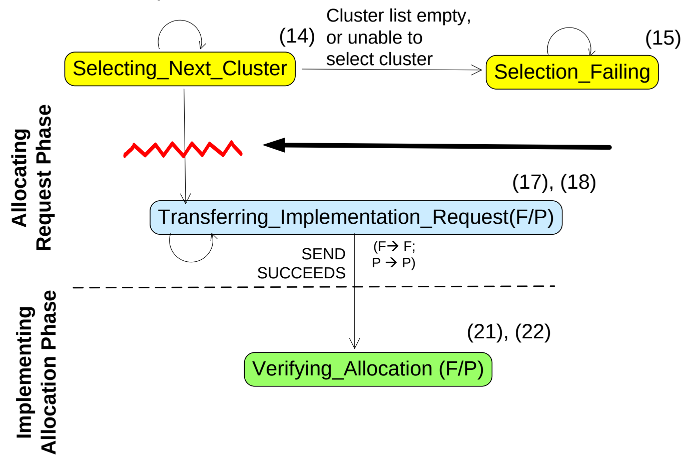
Table 2. Ranking of minimal s-t cut sets that consist of two transitions. For each cut set entry, member state transitions are listed as TPM elements from Fig. 3, together with the number of *from state*s the transitions originate from, the total probability of member transitions, and the identifying number for the figure in which the transitions in the cut set appear.

	Set of member transitions from Fig. 3	Number of From States	Total Probabilty	Figure Reference
2-1	{14, 17} {14, 18}	1	0.895	2, A.3
2-2	{9, 11} {9, 12}	1	1.000	2
2-3	{9, 12} {11, 12}	2	1.395	2
2-4	{23, 27} {36, 38}	2	1.438	A.4
2-5	{23, 27} {31, 34}	2	1.499	A.4
2-6	{23, 27} {28, 31}	2	1.507	A.4
2-7	{23, 27} {34, 36}	2	1.507	A.4
2-8	{35, 37} {36, 38}	2	1.861	A.4
2-9	{31, 34} {35, 37}	2	1.922	A.4
2-10	{30, 33} {36, 38}	2	1.924	A.4
2-11	{28, 31} {35, 37}	2	1.930	A.4
2-12	{34, 36} {35, 37}	2	1.930	A.4
2-13	{27, 30} {36, 38}	2	1.931	A.4
2-14	{33, 35} {36, 38}	2	1.931	A.4
2-15	{30, 33} {31, 34}	2	1.985	A.4
2-16	{27, 30} {31, 34}	2	1.993	A.4
2-17	{31, 34} {33, 35}	2	1.993	A.4
2-18	{28, 31} {30, 33}	2	1.993	A.4
2-19	{30, 33} {34, 36}	2	1.993	A.4
2-20	{27, 30} {28, 31}	2	2.000	A.4
2-21	{27, 30} {34, 36}	2	2.000	A.4
2-22	{28, 31} {33, 35}	2	2.000	A.4
2-23	{33, 35} {34, 36}	2	2.000	A.4

The cut sets in Table 2 each have two state transitions. Cut set #2-1 consists of the two transitions from *Selecting_Next_Cluster* to the states *Transferring_Implementation_Request* (*F*/*P*) (TPM elements {14, 17} {14, 18}). These transitions are related to the *Allocating Request* phase and were discussed in

Section III. Cut sets #2-2 and #2-3 consist of two transitions from the *Cluster Estimating* phase, which was also discussed in Section III. The remainder, cut sets, #2-4 through #2-23, occur in the *Allocating Request* and *Implementing Allocation* (*F*/*P*) phases (see Annex A for the latter). For all two-transition cut sets, system performance is driven toward 0, by reducing the probabilities of transition for both state transitions in the cut set.

For the two-transition cut sets in Table 2, we can also use information about relationships between transitions in the cut set and the location of the cut set in the Markov chain graph to determine which cut sets might be of most interest to examine. Of special interest are two-transition cut sets in which both transitions originate from the same state, i.e., have the same *from state*. An example of this is cut set #2-1, in which both transitions originate from the *Selecting_Next_Cluster* state. Another example is cut set #2-2, which consists of the transitions *Allocating_Maximum* → *Allocating_Partial* {9, 11} and *Allocating_Maximum* → *Recording_Allocation* {9, 12}. In cut set #2-2, both transitions also originate from the same state, *Allocating_Maximum*. Cut sets #2-1 and #2-2 could used to model the effects of component failures that can be related to their common *from states*.



Cluster list empty, or unable to select cluster
(14)
(15)
Selecting_Next_Cluster
Selection_Failing
Allocating Request Phase
(17), (18)
Transferring_Implementation_Request(F/P)
SEND SUCCEEDS
(F→ F; P → P)
Implementing Allocation Phase
(21), (22)
Verifying_Allocation (F/P)

Figure 8. Minimal s-t cut set #2-1, which consists of the two state transitions, *Selecting_Next_Cluster* → *Transferring_Implementation_Request (F/P)*, from Fig. 2. For each state, related row numbers in Fig. 3 are provided. (In the TPM in Fig. 3, these are TPM elements {14, 17} and {14, 18}). Note that the transition arrow between these two states denotes both transitions from *Transferring_Implementation_Request (F/P)* using the compacted representation used in this paper.

As an example of a failure scenario that involves a two-transition cut set, consider cut set 2-1, which is shown in Fig. 8. This cut set could be used to model the effects of faults, which impair the cloud controllers' ability to send messages that instruct a cluster to implement a request (see Sec. III and Annex A(3) for details on the *Allocating Request* phase). Such a fault could be caused by a virus or other malfunction, which has corrupted internal databases that contain cluster addresses. Because of this fault, the controller may be able to select a cluster, but is unable to send the implementation request message and so must abort the selection process. In the Markov chain, this failure scenario corresponds to a reduced ability to transition from the *Selecting_Next_Cluster* state to either the *Transferring_Implementation_Request* (*F*) or to the *Transferring_Implementation_Request* (*P*) state, which are the two state transitions in cut set 2-1.

In the TPM shown in Fig. 3, the perturbation of *Selecting_Next_Cluster* → *Transferring_Implementation_Request* (*F/P*) is modeled by lowering the probability of transition from the *Selecting_Next_Cluster* state to the *Transferring_Implementation_Request* (*F/P*) states, i.e., lowering the values in columns 17 and 18 in row 14, or TPM elements {14, 17} and {14, 18}. In compensation, we choose to raise the transition probability of column 15, *Selecting_Next_Cluster* → *Selection_Failing*, or TPM element {14, 15}. This choice is logical since in the large-scale simulation, error handling procedures require the cloud controller to return a failure (NERA) response to the user when cluster selection fails. Thus, a perturbation applied to all time-period TPMs, in which the related TPM elements {14, 17} and {14, 18} are systematically lowered by increments while TPM element {14, 15} is raised, serves to model how the increasing incidence of internal database failure affects the proportion of requests that receive either full or partial grants. Figure 9 shows the TPM elements that are affected by the perturbation of the transitions shown in the cut set in Fig. 8.

		14	15	16	17	18
14	Selecting_Next_Cluster	ε	0.168↑	0.000	0.402↓	0.429↓
15	Selection_Failing	ε	ε	1-3ε	0.000	0.000
16	Tranferring_Failure_Response	ε	0.000	0.048	0.000	0.000
17	Transferring_Implementation_Request (F)	0.012	0.000	0.000	ε	0.000
18	Transferring_Implementation_Request (P)	0.012	0.000	0.000	0.000	ε

Figure 9. TPM elements from Fig. 3 that were raised and lowered as a result of perturbation shown in Fig. 10a

Fig. 10a shows the impact of this perturbation on total requests granted. As in the previous example in Section V, this perturbation was carried out using Markov simulation over the initial 16 hours (16 time periods) of the 27000-hour simulated duration. To equate the Markov and discrete-time large-scale simulations as before, the equivalent message initiation failure was also introduced into the large-scale simulation. In the latter, the proportion of requests granted was recorded as the probability of failure was increased from 0 (no occurrence of failure) to 1 (all attempts to by the cloud controller to send implementation messages to clusters fail). Similarly, the change in probabilities of the related transitions identified above was also recorded in the large-scale simulation as the incidence of introduced failure was increased. Fig. 10a compares the decline in requests granted that occurred in the Markov and large-scale simulation. The horizontal axis shows the increase in transition probability of *Selecting_Next_Cluster* → *Selection_Failing*. The left vertical axis shows units for proportion of requests granted, while the right vertical axis shows units for the decrease in the transition probability of *Selecting_Next_Cluster* → *Transferring_Implementation_Request* (*F/P*).

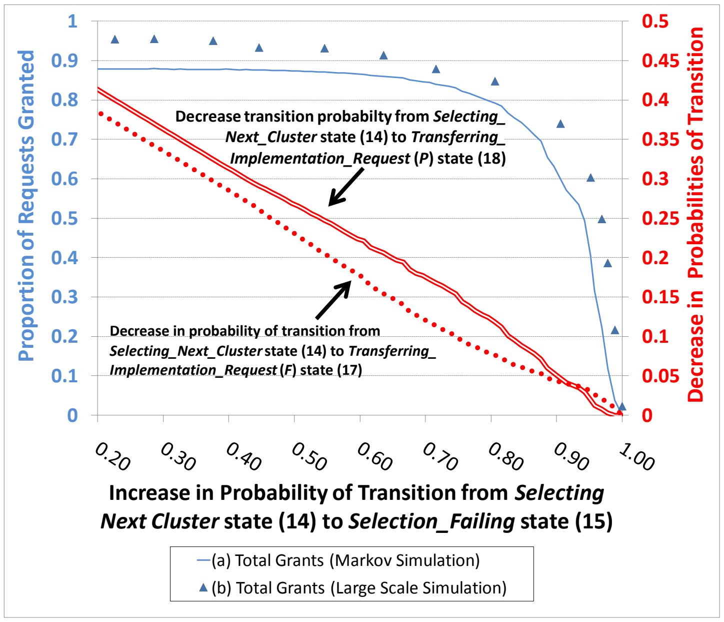


Figure 10a. Decline in total requests granted due to cloud controller database failure (a) as estimated by perturbing the DTMC and using Markov simulation; and (b) as actually computed in the discrete time large-scale simulation. The DTMC was perturbed to raise the probability of transition for *Selecting_Next_Cluster* → *Selection_ Failing* (TPM element {14, 15}) and lower the transition probabilities of *Selecting_Next_Cluster* → *Transferring_Implementation_Request (F/P)* (TPM elements {14, 17} and {14, 18}). Both transitions are lowered at the same rate. The increase of the probability of transition for *Selecting_Next_Cluster* → *Selection_Failing* is indicated by the horizontal axis. The left vertical axis provides unit values for the proportion of requests granted. The right vertical axis provides units for the decrease in transition probability of *Selecting_Next_Cluster* → *Transferring_Implementation_Request (F/P)*

As in Fig. 6a, Fig. 10a shows that the perturbation of the Markov chain could be used to predict that total grants will fall to 0 as the incidence of failure rises. As before, the perturbation of the DTMC also roughly estimates the rate of decline. The figure shows that a drastic decline in total grants will not occur until the incidence of failure causes the transition probability of *Selecting_Next_Cluster* → *Selection_Failing* to exceed a threshold of about 0.80 (on the horizontal axis). However, as is obvious in Fig. 10a, the perturbation of the DTMC underestimates the resilience of the large-scale simulation. In the latter, the rate of total grants is slightly higher and the collapse in system performance is postponed until the incidence of failure is greater than in the Markov simulation. Yet, the overall shape of the two curves is similar, as it was in the previous example in Section V.

In Fig. 10b, we again see that Markov chain perturbation can be less accurate in modeling the impact of the perturbation on full grants and partial grants separately. In this case, the inaccuracy stems from the fact that in the large-scale simulation, the probability of transition for *Selecting_Next_Cluster* → *Transferring_Implementation_Request (F)* and *Selecting_ Next_Cluster* → *Transferring_Implementation_Request (P)* decline at different rates in response to increased incidence of failure. This difference in rate of decline cannot be predicted by the perturbation described in Fig. 9. Instead, the Markov chain perturbation must be iterated to model the effects of different combinations of relative rates of decline for these two transitions. Figure 10b, shows three such combinations: one is accurate; two are not. As in the first example, it is likely that the inaccuracy in Fig. 10b stems from the fact that the perturbation approach described here currently can only be used to predict the impact of changes to localized parts of the Markov chain, which directly interact with each other. In the large-scale simulation, the simulated failure prevents implementation of allocations on clusters which in turn results in lower utilization of cluster resources. Although the overall grant rate declines as a result, the greater availability of resources means that when grants can be made, full grants are more likely. Modeling this type of non-local interaction in a DTMC may require knowledge of dependencies between state transitions that are not members of the same cut set and that involve states that are distant from each other in the DTMC graph. Our current perturbation approach cannot do this, and future work is necessary to determine if the Markov chain paradigm can be extended to overcome this limitation.

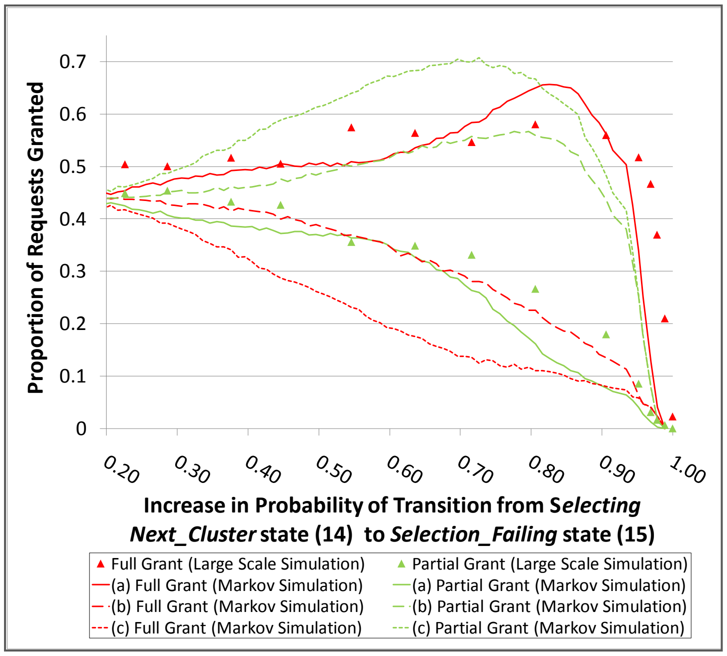


Figure 10b. Decline in full and partial requests grants, due to cloud controller database failure as (a) as estimated by perturbing the DTMC and using Markov simulation; and (b) as computed in the discrete time large-scale simulation. The DTMC was perturbed as described for Fig. 10a. Three combinations of relative rates of decline are shown for TPM elements {14, 17} and {14, 18}: (a) 0.25 and 0.75; (b) 0.5 and 0.5; and (c) 0.75 and 0.25.

Cut sets #2-4 through #2-23 from Table 2 contain pairs of state transitions that originate in different states. Using Fig. 2, it is possible to classify these cut sets on the basis of whether their states relate to network transmission or relate to functions of either the cloud controller or clusters. Thus, cut sets #2-8, #2-9, and #2-10 pertain to states that may be used to model network failure. Cut set #2-3 concerns states that can be used to describe component failure within clusters during the *Cluster Estimating* phase. Cut sets #2-6 and #2-20 can be used to describe possible component failures during *Implementing Allocation* (*F*/*P*) phase. Cut sets #2-3, #2-6, and #2-20 pertain to states which all relate to processes that are either within the clusters (#2-3) or within the cloud controller (#2-6, and #2-20), but not within both (see Annex A). Cut sets #2-7, #2-21, and #2-22 also relate to the *Implementing Allocation* (*F*/*P*) phase, but in each, one member state transition is related to the cloud controller while the other is related to the clusters. The analyst is left to choose which would be of most importance to examine.

The perturbation of the transitions in cut sets #2-4 through #2-23 is accomplished in the same manner as described above, except that transition probabilities are reduced to 0 for two state transitions that have different *from states*. As an example, we focus on the perturbation of cut set #2-3, which consists of two state transitions from Fig. 2. These transitions are *Allocating_Maximum* → *Recording_Allocation* (TPM element {9, 12} in Fig. 3) and *Allocating_Partial* → *Recording_Allocation* (TPM element {11, 12} in Fig. 3). Cut set #2-3 is shown in Fig. 11.

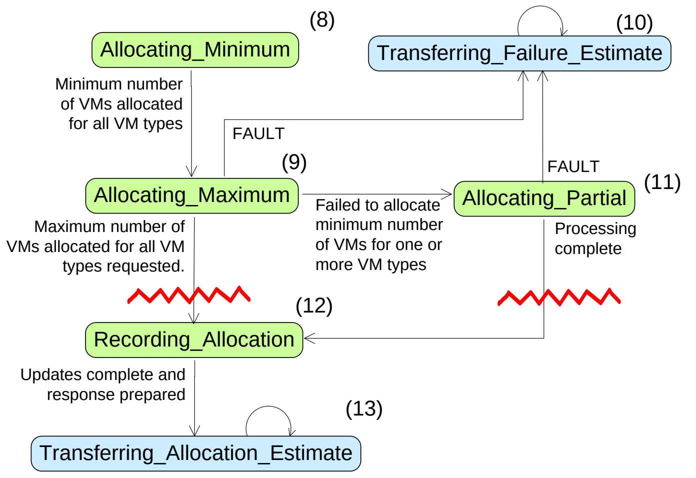

(8)
Allocating_Minimum
(10)
Transferring_Failure_Estimate
Minimum number of VMs allocated for all VM types
FAULT
(9)
FAULT
Allocating_Maximum
Failed to allocate minimum number of VMs for one or more VM types
Allocating_Partial
(11)
Maximum number of VMs allocated for all VM types requested.
Processing complete
(12)
Recording_Allocation
Updates complete and response prepared
(13)
Transferring_Allocation_Estimate

Figure 11. Minimal s-t cut set #2-3, consisting of two state transitions (denoted by jagged red lines): *Allocating_Maximum* → *Recording_Allocation* (TPM element {9, 12} in Fig. 3) and *Allocating_Partial* → *Recording_Allocation* (TPM element {11, 12}). For each state, related row numbers for the TPM in Fig. 3 are provided.

This cut set may be related to a failure scenario in which viruses or other faults cause widespread software process failures in clusters, which prevent completion of VM allocation estimates. In this scenario, clusters are successful in allocating the minimum number of VMs requested, but cannot allocate the maximum number. However, the cluster is still operational and is able invoke alternative round robin procedures, which also fail. Further software malfunctions then cause the initial estimate of a minimum allocation to be lost, and so the cluster returns a failure estimate to the controller. To model this scenario in the DTMC involves perturbation of the two rows in Fig. 3, which directly represent the two transitions in the cut set. These rows are shown in Fig. 12.

		9	10	11	12
9	Allocating_Maximum	0	ε	0.464 ↑	0.536 ↓
10	Transferring Failure_Estimate	0	ε	0.000	0.000
11	Allocating Partial	0	ε ↑	0.000	1-3ε ↓
12	Recording_Allocation	0	ε	0.000	0.000

Figure 12. TPM elements from Fig. 11 that were raised and lowered as a result of perturbation shown in Fig. 13.

In row 9 of Fig. 12, the transition probability of the first state transition in the cut set, *Allocating_Maximum* → *Recording_Allocation* (TPM element {9, 12} in Fig. 3) is decreased, while at the same time the probability of *Allocating_Maximum* → *Allocating_Partial* (TPM element {9, 11}) is increased. This perturbation represents the failure to allocate the maximum number of VMs requested and the subsequent invocation of alternative procedures. Simultaneously in row 11, the probability of the second transition, *Allocating_Partial* → *Recording_Allocation* (TPM element {11, 12}), is decreased, while the probability of transition for *Allocating_Partial* → *Transferring_Failure_Estimate* (TPM element {11, 10}) is increased. The second perturbation models the failure of the alternative procedures and the resulting failure response to the cloud controller.

The systematic increases and decreases of transition probabilities in the two rows were synchronized such that the transition probabilities approached 0 simultaneously. The perturbation was again carried out using Markov simulation over the first 16 hours (16 time periods) of the simulated duration discussed in Section IV. As before, this failure was also introduced into the discrete time large-scale simulation and the resulting changes to transition probabilities were recorded. The results are also plotted in Fig. 13. As in the two previous examples, perturbation of the DTMC could be used to accurately predict that total requests granted will ultimately fall to 0 as the incidence of this failure increases. The shape of the Markov simulation curve for total grants also indicates that grants will remain at a fairly high level, until probability of taking the failure transition exceeds 0.85 in both states (rows) being perturbed. However, the perturbation once again underestimates the resilience of the large-scale simulation. The underestimation is likely due to the reasons described earlier, and which we discuss further in Section VII. Nevertheless, perturbation of the DTMC does provide a quantitative estimate of the performance of the target system in this failure scenario. We omit the analysis of the comparative decline in full and partial grants, since it is similar to the two previous examples.

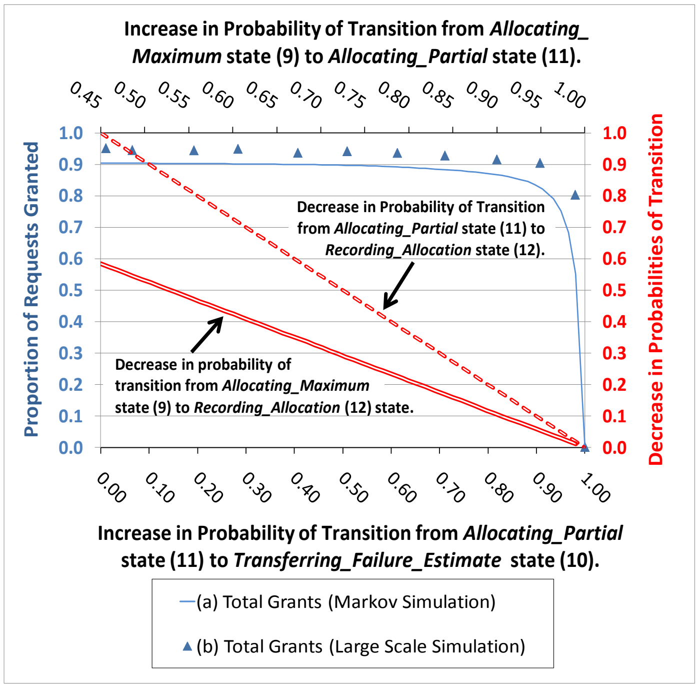


Figure 13. Decline in total requests granted due to failure of allocation components in the *Cluster Estimating* phase (a) as predicted by perturbing the DTMC and using Markov simulation; and (b) as actually computed in the large-scale simulation. The failure is modeled by perturbing the two state transitions in cut set #2-3 which relate to TPM elements {9, 12} {11, 12} (See Fig. 12). Labeled red curves show the decrease in the transition probabilities for these two TPM elements, which are plotted against the right vertical axis. The top horizontal axis shows the increase in the transition probability for *Allocating_Maximum* → *Allocating_Partial* (TPM element {9, 11} as the probability of transition for *Allocating_Maximum* → *Recording_ Allocation* (TPM element {9, 12}) is decreased. The bottom horizontal axis shows the increase in the probability of transition for *Allocating_Partial*→ *Transferring_Failure_Estimate* (TPM element {11, 10} as the probability of transition for *Allocating_Partial* → *Recording_Allocation* (TPM element {11, 12}) is decreased. Blue curves show the resulting decrease in requests granted as estimated using the DTMC and as actually occurred in the large-scale simulation. These curves are plotted against the left vertical axis.

VII. DISCUSSION

The preceding sections showed how a methodology which combines a DTMC representation, Markov simulation, and minimal s-t cut set analysis could be used to identify failure scenarios in a complex distributed system and to estimate the rate of performance degradation in response to increased occurrence of failures. The DTMC, though extensive, provides a succinct representation of the structure and dynamics of a system under study. Markov simulation can be applied to the DTMC to show how the system evolves over time. To determine which perturbations cause drastic performance degradations, the DTMC is formulated as a directed graph and minimal s-t cut set analysis is used to identify cut sets that disconnect the *Initial state* from both of the desired absorbing states, *Request_Active* (*F*/*P*). The use of minimal s-t cut set analysis to guarantee interruption of the flow between two vertices in a graph is well-known result of graph theory research [50-54]. Thus, systematically reducing the probability of transition for the state transitions in these cut sets toward 0 prevents requests from reaching the absorbing states. The transitions in the cut set can thus be identified as critical transitions, which serve as a basis for describing potential failure scenarios. In previous work [12], we reported the results of experiments which showed that minimal s-t cut set analysis could be used to find all critical state transitions in an absorbing DTMC for a much smaller grid computing system at 1/100th the computation cost of large-scale simulations. This exhaustive analysis need not be repeated for the problem described in this paper, as there is not the space for it. However, for the problem described in this paper, we estimate that perturbation of the 33 one- and two-transition cut sets would also require about 1/100th of the computational effort needed to examine 5470 perturbation sequences using exhaustive search (see of Section VI) in order to find the same sets of critical transitions. These savings increase even more dramatically if combinations of three critical transitions are considered. Though further research is necessary, it is our belief that both in [12] and in this study, we have described an analytical approach that can aid in understanding where and how catastrophic failures may occur in complex systems. The results to date have shown that the approach is tractable for the types of problems we have examined. Moreover, the approach can be combined with domain expertise about how specific minimal s-t cut sets relate to the architecture of the cloud system in order to focus on cut sets that should be examined in more detail (which can further increase computational savings). In large problems, minimal s-t cut set analysis can serve to reveal potential failure points that could be overlooked by walk-throughs of the state model.

In addition to identifying the points where failure is most likely to cause the system to collapse, the Markov simulation capability allows quantitative estimates about the rate of performance decline. This allows the analyst to identify thresholds beyond which performance collapses drastically. The potential accuracy of the estimates and identified thresholds was illustrated in the preceding examples. However, the examples also showed that while the perturbation methodology used here could potentially be used to predict how failure impacts the rate and extent of aggregate decline in requests granted, it is unable to predict how different types of grants (full and partial) are likely to be affected. As discussed earlier, this inability may occur because a Markov chain, as defined here, can only consider information in its present state to determine which transition to take. To make precise predictions about the impact of failure on full or partial grants may depend on information available in states that were entered earlier in the progress of requests and that are distant from each other in the DTMC graph. Thus, the Markov chain, in its current form, cannot represent a situation where software failures in the cloud controller cause a change in cluster resource allocation behavior that favors full grants over partial. Extending the approach presented here to handle such situations is a topic for future research into techniques which represent and utilize inter-state dependencies or which can represent state memory, i.e., higher-order Markov chains [55, 56]. This latter point is of particular interest. Although, as explained above, great care was taken to ensure that the DTMC presented here obeyed equation (1), it was also clear that the model could be expanded to include states that require memory of the history of processes. Though this expansion increases the complexity of the DTMC, it has the potential to enable more detailed analysis of causality, to enable more precise predictions, and to generally enrich the model. Evolution of the DTMC to identify new states of importance is the target system being modeled has been found to be both natural and desirable in other domains [1, 2]. We plan to pursue this avenue in the future. An area of particular importance to the nuclear power plant domain is modeling of system evolution over long durations, as would be required to represent the aging of equipment [5, 7, 8, 32, 33]. Here, the use of sets of TPMs provides a basis for capturing such time-inhomogenous, long term processes. The time duration being modeled can be extended by both increasing the number of time period TPMs and the duration of each time period (see Section IV). Further research would be needed to apply the methods described in this paper to the nuclear power plant domain and construct an appropriate DTMC that captures the dynamics of aging.

Another important question that requires further research is the scalability of minimal s-t cut set analysis. Cut set enumeration algorithms are known to be computationally expensive for large problems. For instance in [12], a Markov chain with 50 states, though sparse, was found to contain over 10^8 minimal s-t cut sets on paths between the initial and absorbing states. Further, computational characteristics of directed graphs are not well understood and remain a topic for future work. The TPM for the problem presented here is also a sparse matrix, which in its current form, contains sufficiently few cut sets so that enumeration is possible. However, because the DTMC has a large number of states and state transitions, and because it is grounded in the semantics of a real-world domain, the model provides a good test bed for the methods we have been working with thus far. To find minimal s-t cut sets in still larger problems, we have adapted a node contraction algorithm, which while not guaranteed to find all minimal s-t cut sets in a directed graph, can be controlled to bound computational cost. Efficient implementations of this algorithm for undirected graphs can run in $O(n^2)$ time [57]. We plan to expand the size of the current problem to add additional states and state transitions—and then apply the node contraction algorithm, and perhaps other techniques, to find critical transitions. It will also be interesting to learn how well domain-specific methods employed above for categorizing cut sets will work for larger problems.

Finally, it is important to briefly discuss alternative analysis approaches of interest that are not based on finding critical state transitions. For instance, rather than finding individual points of failure, it may be desirable to use the Markov model to understand the effect of system-wide events such as network instability. For this, it is necessary to perturb states that relate to message transmission between users, cloud controllers, and cluster. The perturbation software we have written is being extended to increase the number of states being perturbed simultaneously, and we plan such experiments in the future. Another alternative direction that could be taken is to restructure the problem definition so that the model is an ergodic chain, rather than an absorbing chain. This can be done by modeling users who continually make requests over time. Thus, rather than entering absorbing states, individual requests return to the *Preparing to Submit* phase, where they may re-emerge as new requests. Studying the problem as an ergodic chain may lead to discovery of favorable and unfavorable stationary distributions that the system may reach, which could yield new insights into behavior of cloud systems.

VIII. CONCLUSIONS

In this paper, we have presented a new approach for using a Discrete Time Markov chain (DTMC) to provide an understanding of dynamic system behavior and to identify failure scenarios that lead to catastrophic performance degradations. The approach employs several analysis techniques whose combined use has heretofore not been reported: the use of a detailed DTMC to provide an in-depth model of complex system dynamics; the use of a time-inhomogeneous DTMC representation to simulate evolution of the system state over time; the use of model perturbation to understand potential failure scenarios and predict the rate at which performance is likely to degrade, and the use of graph-theoretic methods to reduce search for critical state transitions in a large DTMC, which when suitably perturbed, reveal failure scenarios. We have shown how a detailed DTMC of a complex system can be formulated so that the application of this overall approach can be used to understand how a system might behave when unfavorable events occur or adverse conditions arise. Application of this method could be used to reveal unsuspected potential failures, which might be difficult to detect through non-automated means. Examples provided in this paper show the strengths of this approach in finding critical transitions that can be used to identify failure scenarios. In addition, we have discussed the current limitations of this approach and identified avenues for future research. It is our hope that this paper will provide useful ideas to other researchers working on the simulation and monitoring of complex systems, and ultimately, that the work will lead to the development of effective monitoring and prediction tools for real-world systems.

REFERENCES

[1] Smyth, P., 1993, “Hidden Markov models and neural networks for fault detection in dynamic systems,” *Proceedings of the 1993 IEEE-SP Workshop on Neural Networks for Signal Processing*, Linthicum Heights, MD, USA, pp. 582–592.

[2] Smyth, P., 1994, "Markov monitoring with unknown states," *IEEE Journal on Selected Areas in Communications*, 12(9), pp. 1600–1612.

[3] Ying, J., Kirubarajan, T., Pattipati, K., and Patterson-Hine, A., 2000, "A hidden Markov model-based algorithm for fault diagnosis with partial and imperfect tests," *IEEE Transactions on Systems, Man, and Cybernetics*, 30(4), pp. 463–473.

[4] Lee, J., Kim, S., Hwang, Y., and Song, C., 2004 "Diagnosis of mechanical fault signals using continuous hidden Markov model," *Journal of Sound and Vibration*, 276 (3-5), pp. 1065–108.

[5] Hatzipantelis, E., Murray, A., and Penman, J., 1995, "Comparing hidden Markov models with artificial neural network architectures for condition monitoring applications," *Fourth International Conference on Artificial Neural Networks*, pp.369–374.

[6] Bunks, C., McCarthy, D., and Al-Ani, T., 2000, "Condition-Based Maintenance Of Machines Using Hidden Markov Models," *Mechanical Systems and Signal Processing*. 14 (4), pp. 597–612.

[7] Miao, Q., and Makis, V., 2007, "A comparison study of support vector machines and hidden Markov models in machinery condition monitoring," *Journal of Mechanical Science and Technology*, 21, pp. 607–615.

[8] Xu, Z., Ji, Y., and Zhou, D., 2008, "Real-time Reliability Prediction for a Dynamic System Based on the Hidden Degradation Process Identification," *IEEE Transactions on Reliability*, 57(2), pp. 230–242.

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

[10] Mills, K., Filliben, J., and Dabrowski, C., 2011, "Sensitivity Analysis of Koala, an Infrastructure Cloud Simulator," to appear in the *Proceedings of the 4th International Conference on Cloud Computing*, IEEE, Washington, D.C., July 4-9, 2011.

[11] Dabrowski, C., and Hunt F., 2009, "Using Markov Chain Analysis to Study Dynamic Behavior in Large-Scale Grid Systems," *Seventh Australasian Symposium on Grid Computing and e-Research (AUSGRID 2009)*, Wellington, New Zealand.

[12] Dabrowski, C., Hunt F., and Morrison, K., 2010, *Improving the Efficiency of Markov Chain Analysis of Complex Distributed Systems*, NISTIR 7744, National Institute of Standards and Technology, Gaithersburg, MD.

[13] Hunt, F., Morrison, K., and Dabrowski, C., 2011, "Spectral Based Methods That Streamline the Search for Failure Scenarios in Large-Scale Distributed Systems," Unpublished manuscript, National Institute of Standards and Technology.

[14] Zakarian, A., and Kusiak, A., 1997, "Modeling manufacturing dependability," *IEEE Transactions on Robotics and Automation* 13(2), pp. 161–168.

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

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

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

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

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

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

[21] Laprie, J., and Kanoun, K., 1992, "X-ware reliability and availability modeling," *IEEE Transactions on Software Engineering* 18(2), pp. 130–147.

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

[23] Song, B., Ernemann, C., and Yahyapour, R., 2004, "Parallel Computer Workload Modeling with Markov Chains," *Lecture Notes in Computer Science*, 3277, pp. 47–62.

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

[25] Wu, J., and Deng, F., 2006, "Finite Horizon Optimal Control of Networked Control Systems with Markov Delays," Proceedings of the Sixth World Congress on Intelligent Control and Automation, pp. 4513–4517.

[26] Feng, D., Wencai, D., and Zhi, L., 2009, "New Smith Predictor and Nonlinear Control for Networked Control Systems," Proc. of the International MultiConference of Engineers and Computer Scientists, (Volume II), pp. 1148–1153.

[27] Noriega, H., Saldanha, P., and Frutuoso e Melo, P., 1999, "Reliability Appraisal in a PWR Auxiliary Feed-Water System Under Aging Point Processes," Transactions of the Fifteenth International Conference on Structural Mechanics in Reactor Technology (SMiRT-15), Volume III, Seoul, Korea, pp. 127–134.

[28] Fleming, K., 2004, "Markov models for evaluating risk-informed in-service inspection strategies for nuclear power plant piping systems," *Reliability Engineering and System Safety*, 83, pp. 27–45.

[29] Kwon, K., Kim, J., and Seong, P., 2002, "Hidden Markov Model-Based Real-Time Transient Identifications in Nuclear Power Plants," *International Journal of Intelligent Systems*, 17, pp. 791–811.

[30] Ying, J., Kirubarajan, T., Pattipati, K., Patterson-Hine, A., 2000, "A hidden Markov model-based algorithm for fault diagnosis with partial and imperfect tests.," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Review*, 30(4), pp.463-473.

[31] Zhu, H., Zhang, C., and Yue, X., 2010, "Fault Diagnosis of Nuclear Power Equipment Based on HMM-SVM and Database Development," *Advanced Materials Research*, *Volumes 139–141*, pp. 2532–2536.

[32] Bunks, C., McCarthy, D., and Al-Ani, T., 2000, "Condition-Based Maintenance Of Machines Using Hidden Markov Models," *Mechanical Systems and Signal Processing*. 14(4), pp. 597-612.

[33] Grimberg, R. et al., 2007, "Hidden Markov Chain Model for Lifetime Prediction of Pressure Tubes in PHWR Nuclear Power Plant," *Sixth International Conference on NDE in Relation to Structural Integrity for Nuclear and Pressurized Components*, Budapest, Hungary.

[34] Schweitzer, P., 1968, "Perturbation Theory and Finite Markov Chains," *Journal of Applied Probability*. 5(2), pp. 401–413.

[35] Delebecque, F., 1983, "A Reduction Process for Perturbed Markov Chains," *SIAM Journal of Applied Mathematics*, 43, pp. 325–250.

[36] Hassin, R., and Haviv, M., 1992, "Mean Passage Times and Nearly Uncoupled Markov Chains," *SIAM Journal of Discrete Mathematics*, 5(3), pp. 386–397.

[37] Meyer, C., 1989, "Stochastic Complementation, Uncoupling Markov Chains, and the Theory of Nearly Reducible Systems," *SIAM Review*, 31(2), pp. 240–272.

[38] Stewart, W., and Dekker, M., 1994, *Numerical Solution of Markov Chains*. Princeton University Press, Princeton, New Jersey.

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

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

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

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

[43] Siegle, M., 1992, "On Efficient Markovian Modelling," *Proceedings of the QMIPS Workshop on Stochastic Petri Nets*, Sophia Antipolis, France, pp. 213–225.

[44] Buchholz, P., 1995, "Hierarchical Markovian Models: Symmetries and Reduction, *Performance Evaluation*," 22(1), pp. 93–100.

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

[46] Sanders, W., and Meyer, J., 1991, "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), pp. 25–36.

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

[48] Jacobi, M., and Gornerup, O., 1986, "A Dual Eigenvector Condition for Strong Lumpability of Markov Chains," Submitted to Arxiv preprint arXiv:0710.1986.

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

[50] Tsukiyama, S., Shirakawa, I., Ozaki, H., and Ariyoshi, H., 1980, "An Algorithm to Enumerate All Cut Sets of a Graph in Linear Time per Cutset," *Journal of the ACM*, 27(4), pp. 619–632.

[51] Provan S., and Ball M., 1984, "Computing Network Reliability in Time Polynomial in the Number of Cuts," *Operations Research*, 32(3), pp. 516–526.

[52] Lin, H., Kuo, S., and Yeh, F., 2003, "Minimal cutset enumeration and network reliability evaluation by recursive merge and BDD," *Proceedings of the 8th IEEE International Symposium on Computers and Communications*, Kemer-Antalya, Turkey, pp. 1341– 1346.

[53] Yan L., Taha H., and Landers T., 1994, "A Recursive Approach for Enumerating Minimal Cutsets in a Network," *IEEE Transactions on Reliability*, 43(3), pp 383–387.

[54] Karger, D., 2001, "A Randomized Fully Polynomial Time Approximation Scheme for the All-Terminal Network Reliability Problem," *SIAM Review*, 43(3), pp. 499–522.

[55] Konrad, A., Joseph, A., Ludwig, R., and Zhao, B., 2001, *A Markov-Based Channel Model Algorithm for Wireless Networks*. University of California, Berkeley. Report No. UCB/CSD-01-1142.

[56] Kanal, L., and Sastry A., 1978, "Models for Channels with Memory and their Applications to Error Control," *Proceedings of the IEEE*. 66(7), pp.724–743.

[57] Karger, D., and Stein, C., 1996, "A New Approach to the Minimum Cut Problem, *Journal of the ACM*. 43, pp. 601–640.

ANNEX A

DETAILS OF STATE SPACES FOR ADDITIONAL PHASES OF CLOUD MODEL

In the annex, we provide further description of phases omitted from the body of the paper, but which some readers may find to be of value. The annex also serves to document the size and complexity of our model. In all annex figures, states colored in blue pertain to network transmission, those in violet relate to the actions of the human operator as she or he is submitting the request to the cloud system. States colored in yellow represent the state of the request when it is being acted upon by the cloud controller and those in green indicate that the request is being processed by the clusters.

(1) *Preparing to Submit*. During this phase, shown in Figure A.1, the user prepares the request message and submits it to the cloud controller. Initially, when a start time arrives in the discrete event large-scale model (See [10]), a *Thinking* (or waiting) state is entered. In this state, the user is deciding on the content of the request and waiting for an opportune time to submit. In the large-scale model, this state may persist for an extended period of time. Once the *Thinking* state ends, the request transitions to the *Submitting* state, during which the user prepares and sends the message containing the request to the cloud controller. Once the message is sent, the request enters a *Transferring_User_Request* state, where it remains until the message is either received by the cloud controller (see Annex A(2) for the *Initial Processing* phase) or the message fails. In the event of failure, the request re-enters the *Submitting* state. Note there are two transitions from the *Transferring_User_Request* state to the *Submitting* state indicating that the message has failed. These transitions are triggered by the events *Undelivered Message* and *timeout*. The two transitions denote alternative circumstances which may occur that cause messages to fail. When failure occurs for either of these reasons and the *Submitting* state is re-entered, the user may then attempt to re-submit the message. If the transmission is unsuccessful after repeated resubmissions, the request re-enters the *Thinking* state for reconsideration by the user. In the event that more failures occur, the request repeats the transitions between the *Thinking* and *Submitting* state until a predetermined *request time period* (chosen at random) has elapsed, at which time the request is deemed to have failed and enters the *Failed_State*.

Figure A.1 also shows the *Transferring_Failure_Response* state. This state represents the transmission of a message from the cloud controller to the user, which notifies the user that no cluster can be found to satisfy the request (See Annex A(3) on the *Allocating Request* phase for a description of the circumstances under which this message is sent.) The request transitions from *Transferring_Failure_Response* to *Submitting*. Note that the model indicates this transition is taken whether the message from the cloud controller succeeds or fails. When the transition is taken due to message failure, this indicates that a user-initiated timeout has occurred, and the state of the request has reverted to the user.

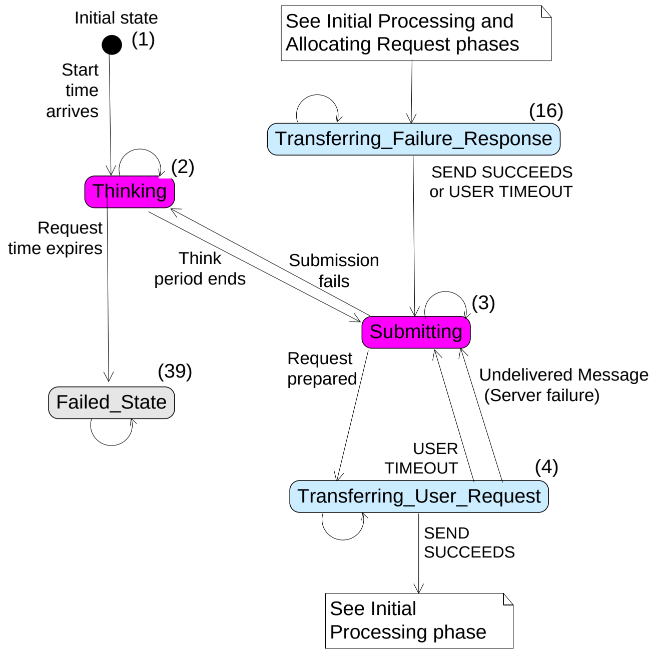

Initial state
(1)
Start time arrives
See Initial Processing and Allocating Request phases
(16)
Transferring_Failure_Response
SEND SUCCEEDS or USER TIMEOUT
(2)
Thinking
Request time expires
Think period ends
Submission fails
(3)
Submitting
Request prepared
Undelivered Message (Server failure)
(39)
Failed_State
USER TIMEOUT
(4)
Transferring_User_Request
SEND SUCCEEDS
See Initial Processing phase

Figure A.1 State diagram for *Preparing to Submit* phase. States where the request is being processed by the user are shown in purple. States indicating that information about the request is being transmitted across the network are shown in blue. The *Failed* absorbing state is in gray. For each state, related row numbers from the TPM in Fig. 3 are provided.

(2) *Initial Processing*. Once the request message is received by the cloud controller, it enters the *Initial Processing* phase. The state space for this phase is shown in Figure A.2. Entry into this phase is signified by a transition from the *Transferring_User_Request* state to the *Initiating_Request_Session* state. The latter represents the state of the request as it is being checked for validity and correctness. In the large-scale simulation, a request that fails this check is returned to the user. In the Markov model, this failure is represented by a transition to the *Transferring_Failure_Response* state, from which the request re-enters the *Preparing to Submit* phase, as described above. If a request passes the check in the large-scale simulation, the cloud controller prepares messages to each of its subordinate clusters, requesting an estimate of the extent to which each cluster can satisfy the request. The controller then waits for clusters to respond for a predetermined period of time. In the Markov model, the corresponding behavior is represented by the entry of the request into the *Preparing_Cluster_Estimate_Requests* state, followed by a transition to the *Transferring_*

Estimate_Request state. Normally, the request then enters the *Cluster Estimating* phase, as described in Sec III.

However, failures may interfere with the progress of the request. In the large-scale simulation, the cloud controller may, in rare situations, not be able to find an eligible cluster. This may occur, if for instance, communication failures or other malfunctions isolate the cloud controller. Under these circumstances, the controller returns a failure message to the user. In the Markov model, this event is represented by a transition from the *Preparing_Cluster_Estimate_Requests* state to the *Transferring_Failure_Response* state. In the large-scale simulation, a message requesting an estimate from the cluster may fail. If this happens, the cloud controller becomes aware of the failure only after its wait period has expired, at which point the controller issues a timeout and determines what action to take next. In the Markov model, the failure of this message from all clusters is represented by a transition from the *Transferring_Estimate_Request* state to the *Selection_Failing* state, as was described in Section III above and is discussed further below.

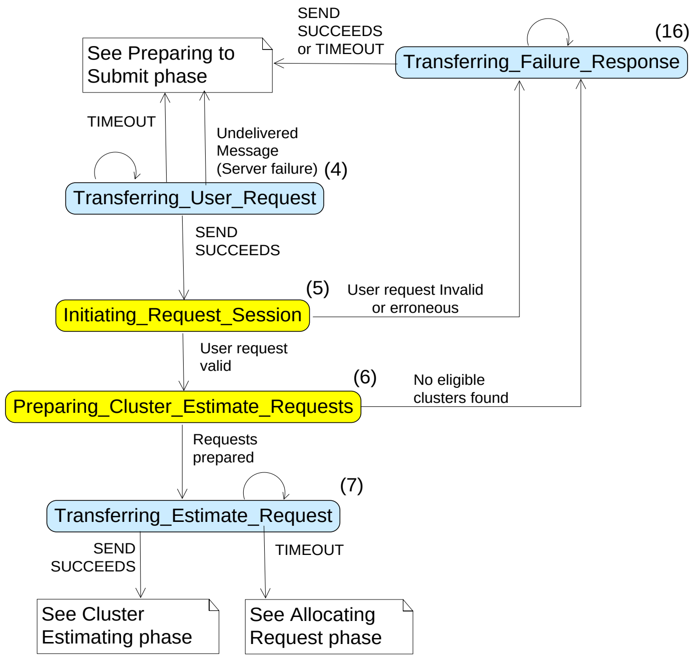

SEND
SUCCEEDS
or TIMEOUT
(16)
See Preparing to Submit phase
Transferring_Failure_Response
TIMEOUT
Undelivered Message (Server failure)
(4)
Transferring_User_Request
SEND SUCCEEDS
(5)
User request Invalid or erroneous
Initiating_Request_Session
User request valid
(6)
No eligible clusters found
Preparing_Cluster_Estimate_Requests
Requests prepared
(7)
Transferring_Estimate_Request
SEND SUCCEEDS
TIMEOUT
See Cluster Estimating phase
See Allocating Request phase

Figure A.2 State diagram of *Initial Processing* phase. States in yellow indicate the request is being processed by the Cloud Controller. States in blue indicate the request is in a state of transmission between the user and the controller or between the user and the clusters. For each state, related row numbers from the TPM in Fig. 3 are provided.

(3) *Allocating Request*. This phase begins when either (a) all clusters have responded to the cloud controller following the *Cluster Estimating* phase (see Sec III), or (b) the wait period for cluster responses has ended and the cloud controller declares a timeout. Figure A.3 shows the state diagram of this phase. At the beginning of the phase, the cloud controller determines which, if any, of the subordinate clusters that have responded affirmatively should implement the request. If no cluster has responded and indicated that a partial or full allocation is possible, the request is returned to the user, with an indication that not enough resources were available (NERA) to satisfy the request. Failure by a cluster to provide an affirmative estimate may be due to any combination of the following three events: (a) a determination by the cluster that it does not have the resources (NERA) and a response to the controller to this effect; (b) failure of the estimate request message issued by the cloud controller; or (c) failure of the estimate response message from the cluster. In the Markov model, events (a–c) are represented by aggregated transitions to the *Selection_Failing* state as follows. When event (a) occurs, a transition from *Transferring_Failure_Estimate* to *Selection_Failing* is taken. When event (b) occurs, a transition from *Transferring_Estimate_Request* state to *Selection_Failing* occurs. When event (c) occurs, a transition from *Transferring_Allocation_Estimate* to *Selection_Failing* is taken. As explained in Section III, when events (a–c) occur for all clusters, the request state is aggregated to the *Selection_Failing* state. From the *Selection_Failing* state, a transition is taken to the *Transferring_Failure_Response* state, where the state of the request reverts to the *Preparing to Submit* phase (i.e., back to the control of the user).

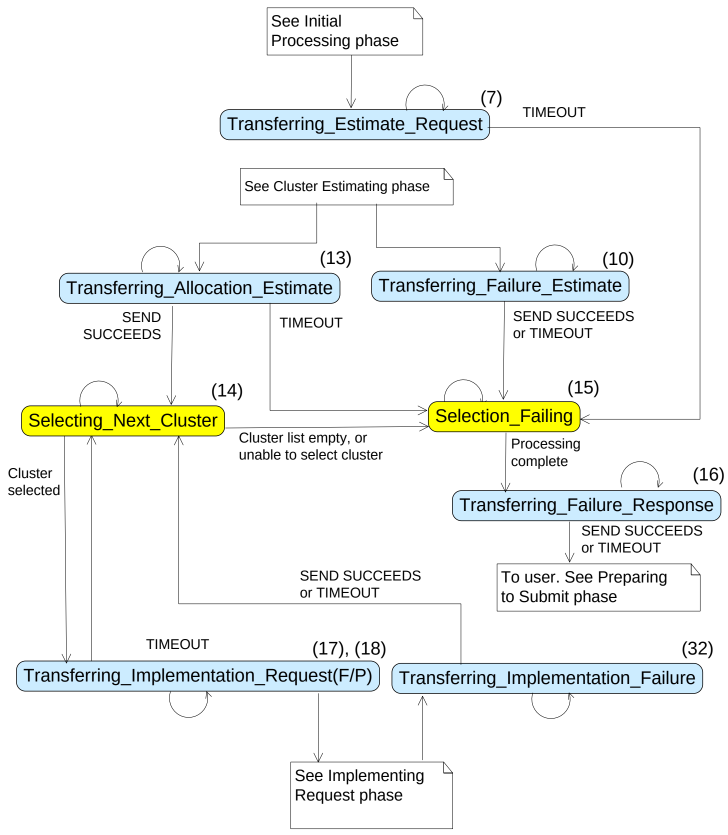

See Initial Processing phase
(7)
TIMEOUT
Transferring_Estimate_Request
See Cluster Estimating phase
(13)
(10)
Transferring_Allocation_Estimate
Transferring_Failure_Estimate
SEND SUCCEEDS
TIMEOUT
SEND SUCCEEDS or TIMEOUT
(14)
(15)
Selecting_Next_Cluster
Selection_Failing
Cluster list empty, or unable to select cluster
Processing complete
Cluster selected
(16)
Transferring_Failure_Response
SEND SUCCEEDS or TIMEOUT
To user. See Preparing to Submit phase
SEND SUCCEEDS or TIMEOUT
TIMEOUT
(17), (18)
(32)
Transferring_Implementation_Request(F/P)
Transferring_Implementation_Failure
See Implementing Request phase

Figure A.3 State diagram for *Allocating Request* phase. States in which the request is being processed by the cloud controller are shown in yellow. States in which information about requests in being transmitted are blue. In addition to the state transitions shown in the diagram, any state may transition to the *Submitting* state in the *Preparing to Submit* phase to represent a user timeout. For each state, related row numbers from the TPM in Fig. 3 are provided.

In the large-scale simulation, if there are clusters that have responded positively, the cluster responses are sorted using one of several evaluation criteria (see [10] for details]). The highest-ranked cluster in the sorted list is sent a message indicating that the cluster should implement the request, i.e., reserve and physically boot the VMs. In the Markov model, these events correspond to a transition from the *Transferring_Allocation_ Estimate* state to the *Selecting_Next_Cluster* state. Selection by the controller of a cluster to implement the allocation corresponds to transition to the *Transferring_Implementation_ Request* (*F*/*P*) state.

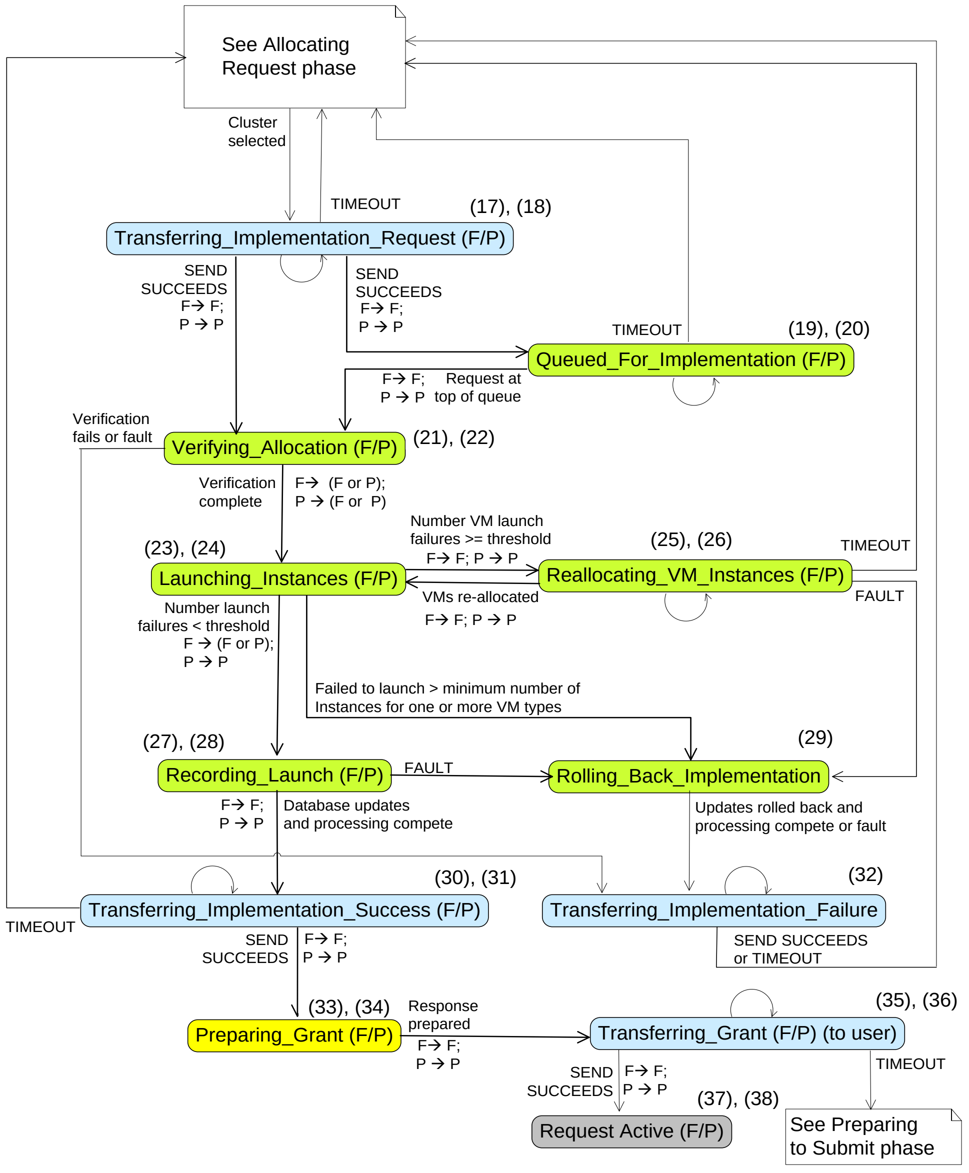
Once the selected cluster receives the message, the *Implementing Allocation* (*F*/*P*) phase is entered. In the large-scale simulation, if all goes well in this phase (described in Annex A(4) below), the cluster will respond to the cloud controller indicting that implementation has succeeded, and the controller will relay the message to the user (see below). However, the implementation message to the cluster may fail. Alternatively, the internal state of the chosen cluster may change, which causes the cluster to be unable to implement the request, for reasons described below in Annex A(4). The cloud controller becomes aware of the failure either because the cluster responds and indicates that it can no longer implement the request or through the timeout process described above. If either circumstance occurs, the cloud controller then selects the next most highly-ranked cluster on its sorted list. If implementation of the next selected cluster fails, the process repeats until either success is achieved or until no cluster can be found. In the latter event, the controller must inform the user that not enough resources were available to satisfy the request. The situation where the controller ultimately fails to find a cluster on which to implement the request is represented by a transition from the *Selecting_Next_Cluster* state to the *Selection_Failing* state, from which a transition is taken to the *Transferring_Failure_Response* state, as explained previously

(4) *Implementing Allocation* (*F*/*P*). Successful receipt of the implementation message by a cluster is represented in the Markov model by a transition from the *Transferring_ Implementation_Request* state to the states of the *Implementing Allocation* (*F*/*P*) phase. In this phase, the selected cluster attempts to reserve the resources for the VMs in the request and to physically initiate the VMs on the computing nodes that contain those resources. The process is depicted in Figure A.4. The state space of this phase consists of 11 states through which there are numerous paths that the request may take, including paths taken as a result of high workload levels and failures.

In the large-scale simulation, if there are no processing problems, the cluster receives the request and verifies the availability of the resources used to make its original estimate in the *Cluster Estimating* phase. If verification succeeds, the cluster sends messages to the nodes containing the resources for the requested VMs, causing the nodes to initiate booting, or *launching*, of the VMs. Normally, a request for which a full allocation was previously estimated by the cluster is implemented as a full request, and a partial request estimate is implemented as a partial request. In the Markov model, this process is represented by a transition from the *Transferring_ Implementation_Request* (*F*/*P*) state to the *Verifying_ Allocation* (*F*/*P*) state to indicate consistency with the previous allocation estimate. Recall the *Transferring_Implementation_ Request* (*F*) and *Transferring_Implementation_Request* (*P*) are considered two separate states, one for full and one for partial allocation. Her, the F→F; P→P notation indicates that *Transferring_Implementation_Request* (*F)* transitions to *Verifying_Allocation* (*F*) state, and the *Transferring_ Implementation_Request* (*P*) state transitions to the *Verifying_ Allocation* (*P)* state. Following successful verification, a transition is taken to one of the *Launching_Instances* (*F*/*P)* states, which under normal operating conditions (with no failures) also follows the F→F; P→P restriction. If launch operations in the large-scale simulation are successful, the action is recorded by the cluster and a confirmation message is sent to the cloud controller, which then prepares and transfers a grant message to the user. At this point, the request is active and the related resources are available to the user. The Markov model represents these events by transitions from *Launching_ Instances* (*F*/*P)* to *Recording_Launch* (*F*/*P),* followed by transition from *Recording_Launch* (*F*/*P)* to *Transferring_ Implementation_Success* (*F*/*P*), and then by transitions to *Preparing_Grant* (*F*/*P*), *Transferring_Grant* (*F*/*P*), and finally by transition to the absorbing states *Request_Active* (*F*/*P*). In this sequence of transitions, the F→F; P→P restriction is observed under normal operating conditions.

However, these problem-free paths may not be possible. There are circumstances in the large-scale simulation under which the request must be processed along other paths through this phase. In what follows, we review these situations. First, under heavy workload conditions, implementation of some requests may be significantly delayed. In the DTMC, this is represented by transitions from the *Transferring_ Implementation_Request* (*F*/*P*) states to the *Queued_for_ Implementation* (*F*/*P*) states. Exit from the queue is represented by transitions to the *Verifying_Allocation* (*F*/*P*) states, with F→F; P→P restrictions observed. The request may then resume its path through this phase to one of the two absorbing states.

A different problem occurs if resources used in the original cluster estimate subsequently become unavailable, so that the request cannot be implemented in accordance with the original estimate. This situation may be detected by the cluster during its initial verification. If, as a result, the verification completely fails and the request can no longer be implemented, the cluster terminates the implementation process and sends a failure message to the cloud controller, which must then select another cluster (see *Allocating Request* phase). In the Markov model, this event is represented by a transition from the *Verifying_ Allocation* (*F*/*P*) states (depending on whether a full or partial allocation is being processed) to the *Transferring_ Implementation_Failure* state, which then transitions to the *Selecting_Next_Cluster* state in the *Allocating Request* phase.



See Allocating Request phase
Cluster selected
TIMEOUT
(17), (18)
Transferring_Implementation_Request (F/P)
SEND SUCCEEDS F→ F; P → P
SEND SUCCEEDS F→ F; P → P
TIMEOUT
(19), (20)
Queued_For_Implementation (F/P)
F→ F; P → P Request at top of queue
Verification fails or fault
Verifying_Allocation (F/P)
(21), (22)
Verification complete
F→ (F or P); P → (F or P)
Number VM launch failures >= threshold F→ F; P → P
(23), (24)
Launching_Instances (F/P)
(25), (26)
Reallocating_VM_Instances (F/P)
TIMEOUT
VMs re-allocated F→ F; P → P
FAULT
Number launch failures < threshold F → (F or P); P → P
Failed to launch > minimum number of Instances for one or more VM types
(27), (28)
Recording_Launch (F/P)
FAULT
Rolling_Back_Implementation
(29)
F→ F; P → P
Database updates and processing compete
Updates rolled back and processing compete or fault
(30), (31)
Transferring_Implementation_Success (F/P)
(32)
Transferring_Implementation_Failure
TIMEOUT
SEND SUCCEEDS
F→ F; P → P
SEND SUCCEEDS or TIMEOUT
(33), (34)
Preparing_Grant (F/P)
Response prepared
F→ F; P → P
(35), (36)
Transferring_Grant (F/P) (to user)
SEND SUCCEEDS
F→ F; P → P
TIMEOUT
(37), (38)
Request Active (F/P)
See Preparing to Submit phase

Figure A.4 State diagram for *Implementing Allocation* phase. States in which the request is being processed by the cluster are shown in green. States where the request is being processed by the cloud controller are in yellow. States in which information about requests in being transmitted are blue. The diagram shows transitions to the *Allocating Request* phase that signify timeouts declared by the cloud controller for three states involving message transmission, the *Queued_for_Implementation* state, and the *Reallocating_VM_Instances* (*F*/*P*) states. Other state transitions that represent timeouts declared by the cloud controller or user are omitted to simplify the diagram. In addition, any state may transfer to the *Submitting* state in the *Preparing to Submit* phase via user timeout. For each state, related row numbers from the TPM in Fig. 3 are provided.

During verification in the large scale simulation, it is possible that changes in the state of cluster resource availability are detected which force a full grant request to be reduced to the status of a partial grant request. In rare cases, if availability increases, a partial grant may be elevated to a full grant. In the Markov model, these events are represented by transitions from *Verifying_Allocation* (*F*) state to the *Launching_Instances* (*P)* state and from the *Verifying_Allocation (P*) state to the *Launching_Instances* (*F) state*. The switches from full to partial and vice versa are represented by the F → (F or P) and P → (P or F) notation on the transitions from the *Verifying_Allocation* (*F/P*) state.

Yet another problem occurs in the large-scale simulation, if failure of needed VM resources becomes apparent only during launch operations. This circumstance causes the cluster to attempt to re-allocate the missing VMs for the request. If the re-allocation is fully successful, the implementation process may then complete normally. However, if re-allocation is not fully successful, a request may be reduced from full to partial allocation status, or a previously estimated partial allocation may lose VMs. Note that repeated re-allocation attempts are possible, ending only when either success is achieved or when re-allocation attempts can no longer find more VM resources that can be used. In the worse case, repeated re-allocations and launch attempts result in minimum levels of requested VMs no longer being met. In this case, the cluster is forced to abort the operation, roll back previous successful launches, and return failure to the cloud controller. In the Markov model, a fully successful re-allocation process is represented by a transition from *Launching_Instances* (*F*/*P*) state to the *Reallocating_ VM_Instances* (*F*/*P*) and back to *Launching_Instances* (*F*/*P*) with the F→F; P→P restriction observed in both directions. A case where a full allocation has been reduced to a partial allocation after repeated re-allocation attempts is represented by a transition from *Launching_Instances* (*F*) to *Recording_ Launch* (*P*). A case where repeated re-allocation cannot restore minimum levels of VMs is represented by a transition from *Launching_Instances* (*F*/*P*) state to the *Rolling_Back_ Implementation* state, and then a transition to the *Transferring_ Implementation_Failure* state. In all cases, the decision to cease re-allocation attempts is represented as residing in the *Launching_Instances* state in accordance with the operation of the large-scale simulation.

Finally, the DTMC represents the effects of catastrophic software and hardware process failures on the implementation process. In the large-scale simulation, such events could force the cluster to abort the implementation process, roll back any allocation commitments that have been made, and return failure. Figure A.4 represents such abort events that occur in the *Launching_Instances* (*F*/*P*), *Recording_Launch* (*F*/*P*), and *Reallocating_VM_Instances* (*F*/*P*) states with transitions to the *Rolling_Back_Implementation* state, followed by a transition to the *Transferring_Implementation_Failure* state. In some cases, further process errors could prevent successful roll back operations, and this is possibility is accounted for in the transition from *Rolling_Back_Implementation* state to the *Transferring_Implementation_Failure* state. In the most extreme cases, process failure may cause the cluster to go down. In such cases, roll back operations not only fail to complete, but the cluster is unable to provide any response to the cloud controller. When this happens, a timeout may be declared by the cloud controller or, in some cases, by the user. Figure A.4 explicitly represents changes in state created by timeouts declared by the cluster controller for states in which timeout events are considered to be the most likely. For all other states, transitions caused by timeout declared by the cloud controller or user have been omitted to prevent the diagram from being unreadable, but are nevertheless understood to exist.