RC24985 (W1004-120) April 30, 2010
Computer Science

Interacting Markov Chain Based Hierarchical Approach for Cloud Services

Rahul Ghosh[1], Kishor S. Trivedi[1], Vijay K. Naik[2], DongSeong Kim[1]

[1]Department of ECE
Duke University
Durham, NC 27708

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Interacting Markov Chain Based Hierarchical Approach for Cloud Services

Rahul Ghosh,
Kishor S. Trivedi
Dept. of ECE, Duke University
Durham, NC 27708
{rg51, kst}@ee.duke.edu

Vijay K. Naik
IBM T. J. Watson Research Center
Hawthorne, NY 10532
vkn@us.ibm.com

DongSeong Kim
Dept. of ECE, Duke University
Durham, NC 27708
dk76@ee.duke.edu

ABSTRACT

In this paper, we describe an analytical modeling approach for performance and dependability analysis of cloud provided services. We use infrastructure-as-a-service as an example of a cloud based service, where service availability and provisioning delays are key SLAs to users while cost reductions are important to service providers. A novelty of our approach is in reducing the complexity of the analysis by dividing the overall model into multiple interacting sub-models and then obtaining the overall solution by iteration over sub-model solutions. This results in a high fidelity model that is scalable. We summarize our modeling approach, showcase our ongoing work through initial results and outline future avenues of research using such an approach.

Keywords

Interacting Markov chains, Analytical models, Fixed-Point Iteration, Performability, Cloud

1. INTRODUCTION

Distributed nature of cloud based services is amenable to uphold SLA guarantees through automated reliable service delivery. However, because of large complex system architecture, diverse client demands and unexpected failures, evaluating the quality of a cloud service is difficult. Hence, developing a general approach for performance and dependability analysis applicable to various cloud services is important. This paper presents an hierarchical analytic modeling approach for performance, availability and performability analysis of a cloud services using interacting Markov chains. To analyze a large scale cloud system, system model solution is composed using iteration of interacting sub-models. Such an approach provides a model that is tractable, scalable and yet of high fidelity [14]. Using job rejection probability and mean provisioning time as the two measures of cloud service quality, we evaluate the effects of changes in job-arrival, service time, and the effects of system capacity on the quality of a cloud service.

System Model and Assumptions. We consider infrastructure-as-a-service provided from a cloud. Amazon EC2 [1] and IBM SBDTC [2, 3] are examples of such services. In such systems, Virtual Machines (VMs) with specific CPU, RAM, and Disk capacity are provisioned after creating an instance of a pre-built OS images. The VMs are deployed on Physical Machines (PMs) each of which may be shared by multiple VMs. We assume that the PMs are grouped into three pools - hot (i.e., running), warm (turned on, but not ready) and cold (turned off). VMs in sleep state can be readily deployed on hot servers with minimum provisioning delay. Instantiating a VM from an image and provisioning it on a running warm server needs additional provisioning time. Servers in the cold pool are turned-off when not in use and need additional startup time before a VM can be deployed on such a server. In the following discussion we use the term job to mean a user request for provisioning a VM and making it available for use by a cloud user.

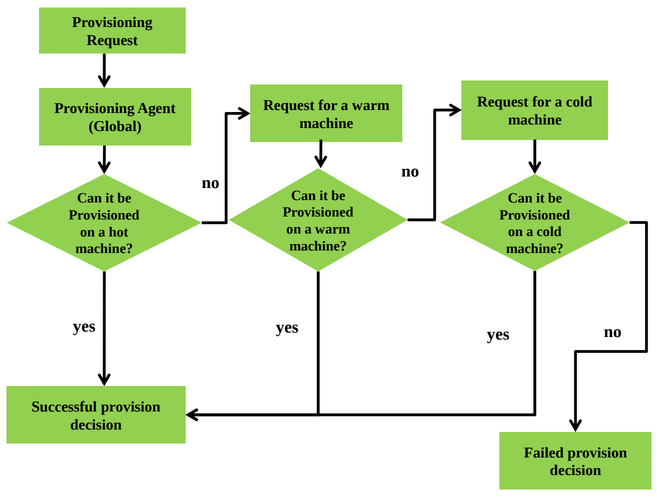
We assume that all requests are homogeneous where each request is for one VM running with a single CPU core. User requests (i.e., jobs) are submitted to a global provisioning engine that processes requests on a FCFS basis as follows. The request at the head of the queue is provisioned on a hot server if there is capacity to run the VM on one of the hot servers. If all hot machines are serving jobs, a PM from warm pool is used for provisioning the requested VM. If all warm machines are busy, a PM from cold pool is used. If none of these servers are available, the request is rejected (service unavailable). When a running job exits, the capacity used by that VM is released and becomes available for provisioning the next job. Failures in PMs can lead to reduction in available capacity and repairs lead to refurbishing the available capacity. Figure 1 shows the provisioning decision steps for a job. For the above described scenario, we investigate the effects of job arrival, job service time and available system capacity on the provisioning delays and the service availability to user requests.

2. PROPOSED APPROACH

This section describes hierarchy of the developed models and interaction among the models. We show how the models are decomposed and individual model solutions are utilized to obtain the overall solution.

2.1 Pure Performance Analysis

In pure performance analysis, we do not consider failure of physical machines. To analyze the performance of cloud



Provisioning Request
Provisioning Agent (Global)
Can it be Provisioned on a hot machine?
no
Request for a warm machine
Can it be Provisioned on a warm machine?
no
Request for a cold machine
Can it be Provisioned on a cold machine?
yes
yes
yes
no
Successful provision decision
Failed provision decision

Figure 1: Work-flow of provisioning decision process

services in an hierarchical fashion, we decompose the cloud services in three major steps (Figure 2).

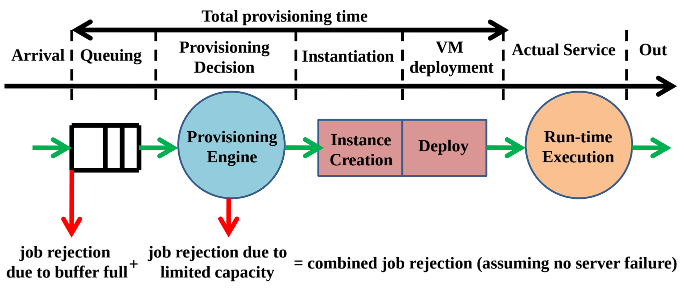


Figure 2: Time-line showing different steps to access a cloud service.

They are (1) provisioning decision, (2) resource allocation and (3) run-time execution. We create three separate Markov models to analyze these events - (1) provisioning decision model, (2) resource usage model and (3) run-time model respectively. These models are described below.

(1) Provisioning decision model. To quantify provisioning decision process, we design a continuous time Markov chain (CTMC) as shown in Figure 3. We assume that job arrival is a homogenous Poisson process with parameter λ. However, we can extend our model to capture other arrival processes (e.g., non-homogenous Poisson process [6], Markov modulated Poisson process [10]) as well. We also assume that provisioning engine takes a provisioning decision for one job at a time. States in the model in Figure 3 are represented in the form (i, s). If no job is currently undergoing provisioning decision, s is set to zero and state (0,0) captures this event. For all other states, one job is undergoing provisioning decision. s = 'H' suggests that provisioning engine is looking for atleast one hot server, which can accept the job for provisioning. Similar meanings are suggested by s = 'W' (or 'C'), when a provisioning engine is trying to take a decision whether atleast one of the warm (or cold) servers, is ready to accept the job for provisioning. We assume a finite length provisioning decision queue where a job inserted in the queue waits for its provisioning decision until decisions are taken for all jobs ahead of it in the queue. Value of i denotes number of jobs currently waiting in the queue.

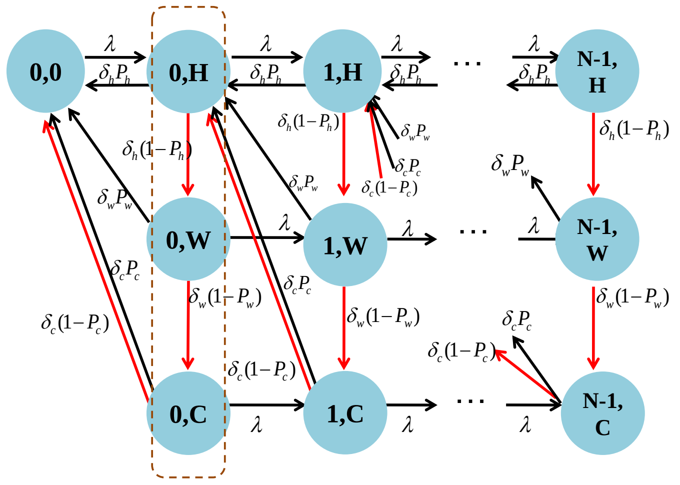


Figure 3: Provisioning decision model. Dotted rectangle represents provisioning decision step of a single job.

In our model, queue length N is the sum total of number of cores across all servers. Thus, we increase queue length dynamically to accommodate more jobs when system capacity is increased.

Dynamics of the Markov model in Figure 3. State $(0,0)$ represents the situation when no job is currently undergoing provisioning decision and no job is waiting in the queue. From here, with a job arrival rate λ, system goes to $(0, H)$, where the provisioning engine tries to take a decision if there is atleast one hot server available to provision this job. In $(0, H)$ three types of events can happen - (a) job can be accepted to be provisioned on one hot server, (b) job can not be accepted to be provisioned on any hot server and (c) arrival of a new job. Under event (a), a transition occurs from $(0, H)$ to $(0, 0)$ with a rate of transition $\delta_h P_h$. $1/\delta_h$ is the mean delay to decide whether atleast one server from hot pool can be used for resource provisioning. P_h is the probability that atleast one hot server can accept a job for resource provisioning. If no hot server is ready to accept the job (event (b)), transition occurs from $(0, H)$ to $(0, W)$ with a rate $\delta_h(1 - P_h)$. While a job is undergoing a provisioning decision, arrival of a new job will trigger a transition from state $(0, H)$ to $(1, H)$. State $(1, H)$ models the situation that one job is undergoing provisioning decision and one job is waiting in the queue . Similar transitions can happen in $(0, W)$. Here, provisioning engine tries to take a decision if atleast one warm server is available to provision the job (which could not be provisioned on any hot server). If no warm server is available, a transition occurs from $(0, W)$ to $(0, C)$. From $(0, C)$, system goes to $(0, 0)$ with the assumption that the job is out of the queue, once it receives a provisioning decision from cold pool. If a job can be accepted in the cold pool, transition occurs at a rate $\delta_c P_c$. If the job can not be accepted due to limited capacity in the

Table 1: Reward rates for different components of job rejection probability assuming no server failure

Measures	Reward rates	Expected steady state reward rates
Job rejection probability due to buffer full. (P_{block})	1 for states (N-1,H), (N-1,W) and (N-1,C); 0 for other states	$\pi_{(N-1,H)} + \pi_{(N-1,W)} + \pi_{(N-1,C)}$
Job rejection probability due to limited capacity. (P_{drop})	$\frac{\delta_c(1-P_c)}{\lambda}$ for states (i,C); where, i=0,1,...,N-1; 0 for other states	$\sum_{i=0}^{(N-1)}(\frac{\delta_c(1-P_c)\pi_{(i,C)}}{\lambda})$

cold pool, transition occurs at a rate $\delta_c(1-P_c)$. Jobs can also be rejected at the tail of the queue if the buffer is full.

Outputs of this model are job rejection probability (P_{reject}) (due to buffer full and due to limited capacity) and mean overall provisioning delay ($<t_{prov}>$) conditional upon the job being accepted. These output measures are computed by assigning proper reward rates [11, 15] to each state of the CTMC in Figure 3 and then by computing the steady state expected reward rates. Let $r_{(i,s)}$ be the reward rate assigned to state $(i,s). If \pi_{(i,s)}$ is the steady state probability that the system will be in state (i,s), expected steady state reward rate is given by $\sum_{(i,s)} r_{(i,s)}\pi_{(i,s)}$. First we compute job rejection probability (P_{reject}). P_{reject} is the sum of two probabilities - (a) job rejection probability due to buffer full (P_{block}) and (b) job rejection probability due to insufficient capacity (P_{drop}). Computations of these quantities are shown in Table 1. Next, overall mean provisioning delay ($<t_{prov}>$) is the sum of two quantities - (1) mean provisioning decision time conditional upon the job not being rejected and (2) mean provision completion time (actual provisioning step which includes VM configuration). Reward rate assignments to compute two components of $<t_{prov}>$ are shown in Table 2, where $1/\beta_h$, $1/\beta_w$ and $1/\beta_c$ denote the mean time to configure VM on a hot, warm and cold server respectively.

(2) Resource usage models. We assume that individual users request for one VM which runs on one CPU core of a server. We quantify the resources of a server by the number CPU cores. Resource usage models capture assignment of CPU cores to successfully provisioned jobs and keep track of available number of CPU cores. We design separate resource usage models for hot, warm and cold pool of servers. Figure 4 shows resource usage model of each hot server in a hot pool. Hot pool model is a disjoint set of such hot server models as shown in Figure 5.

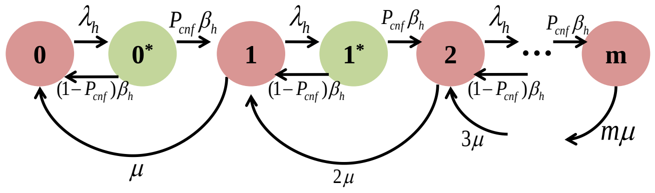


Figure 4: Resource usage model for each hot server.

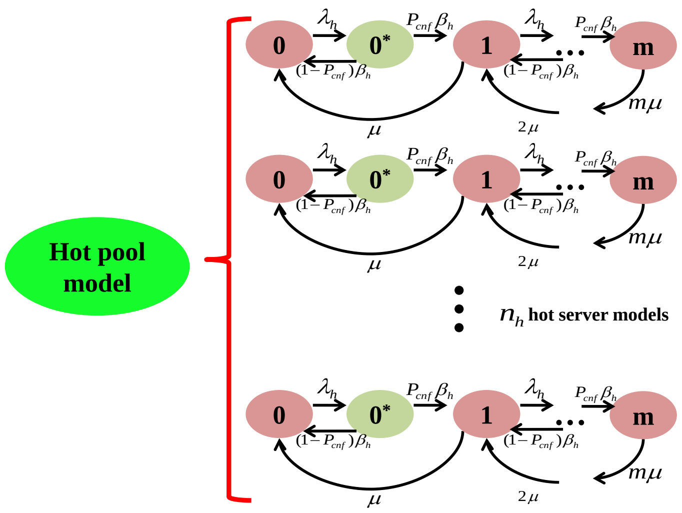


Figure 5: Hot pool model is the disjoint/non-communicating set of n_h hot server models.

States of the CTMC in Figure 4 can be separated into two classes - allocated states and configuration states. Configuration states capture instance creation and deployment of a new VM on a physical machine. Allocated states capture physical resource holding by an already provisioned job. Both states are indexed by the number of CPU cores currently allocated to provisioned jobs. However, configuration states are distinguished by a superscript on the index. Assuming total n_h servers in hot pool, effective arrival rate λ_h to each of the hot servers is as computed in (1).

$$\lambda_h = \frac{\lambda(1-P_{block})}{n_h} \quad (1)$$

Observe, P_{block} is is computed from provisioning decision model. From state 0, after an arrival occurs, system goes to state 0^*, which is a configuration state. In 0^*, a VM is configured to provision the accepted job with a probability of successful configuration P_{cnf}. Time taken to execute the configuration step is assumed to be exponentially distributed with mean $1/\beta_h$. A successful configuration process triggers a transition to state 1 (denoting one CPU core being allocated) with a rate of transition $P_{cnf}\beta_h$; while an unsuccessful configuration takes the system to state 0 with transition rate $(1-P_{cnf}\beta_h)$. We assume that there are m CPU cores in each server. A server can not accept a job for provisioning if all the m cores are already allocated. Though for simplicity we assume all hot servers have identical number of cores,

Table 2: Reward rates for different components of overall mean provisioning delay assuming no server failure

Measures	Reward rates	Expected steady state reward rates
Mean provisioning decision time conditional upon the job not being rejected	0 for state (0,0) $\frac{(i+1)}{\lambda E_1}$ for all other states of the form (i,s), with i=0,1,...,N-1 and s=H, W, C; where $E_1 = 1 - (\pi_{(N-1,H)} + \pi_{(N-1,W)} + \pi_{(N-1,C)}) - \sum_{i=0}^{(N-1)}(\frac{\delta_c(1-P_c)\pi_{(i,C)}}{\lambda})$	$\frac{\sum_{i=0}^{(N-1)}(i+1)(\pi_{(i,H)}+\pi_{(i,W)}+\pi_{(i,C)})}{\lambda E_1}$; where, $E_1 = 1 - (\pi_{(N-1,H)} + \pi_{(N-1,W)}$ $+\pi_{(N-1,C)}) - \sum_{i=0}^{(N-1)} \frac{\delta_c(1-P_c)\pi_{(i,C)}}{\lambda}$
Mean provision completion time (actual provision completion time, which includes VM configuration)	0 for state (0,0); $\frac{\delta_h P_h}{\beta_h(\delta_h+\lambda)E_1}$ for states (i,H), $\frac{\delta_w P_w}{\beta_w(\delta_w+\lambda)E_1}$ for states (i,W), $\frac{\delta_w P_w}{\beta_w(\delta_w+\lambda)E_1}$ for states (i,C), where i = 0, 1, ..., N-1 and E_1 is given previous row.	$\frac{\delta_h P_h}{\beta_h(\delta_h+\lambda)E_1} \sum_{i=0}^{N-1} \pi_{(i,H)} +$ $\frac{\delta_w P_w}{\beta_w(\delta_w+\lambda)E_1} \sum_{i=0}^{N-1} \pi_{(i,W)} +$ $\frac{\delta_c P_c}{\beta_c(\delta_c+\lambda)E_1} \sum_{i=0}^{N-1} \pi_{(i,C)}$

servers with different number of cores can be modeled by creating multiple such Markov models. After solving each hot server model, we can compute the steady state probability that all CPU cores on a hot server are busy and denote this probability by $\pi_m^{(H)}$. Assuming independence among the hot servers, probability that atleast one server in a hot pool can accept a job for resource provisioning is obtained as the output of hot pool model. This probability is denoted by P_h and presented in (2).

$$P_h = 1 - (\pi_m^{(H)})^{n_h} \tag{2}$$

CTMC for each warm server is shown in Figure 6. Warm pool model is a disjoint set of n_w warm server models. While warm server model is similar to hot server model, there are few aspects which differentiate between hot and warm server resource usage. Effective arrival rate to each warm server is λ_w as given in (3).

$$\lambda_w = \frac{\lambda(1 - P_{block})(1 - P_h)}{n_w} \tag{3}$$

This is because, jobs arrive to warm server pool only if they can not be provisioned in any of the hot servers. Moreover, when the first job arrives to a warm machine, the machine requires some additional startup work to make the warm server ready to use. This is represented by an intermediate state 0^{**}.

Mean time to make a warm machine ready for use, is assumed to be exponentially distributed with mean $1/\gamma_w$. This is followed by VM configuration on a warm server with a mean time of $1/\beta_w$. Notice that, once the first job is provisioned on the warm server, mean time to configure VMs for subsequent jobs is same as that of a hot server i.e. $1/\beta_h$. Warm pool model is a disjoint set of warm server models and its output is the probability that atleast one server in a warm pool is ready to accept a job for resource provisioning. This probability is shown in (4). $\pi_m^{(W)}$ in (4) denotes the steady state probability of all the CPU cores of a warm server being consumed for job provisioning.

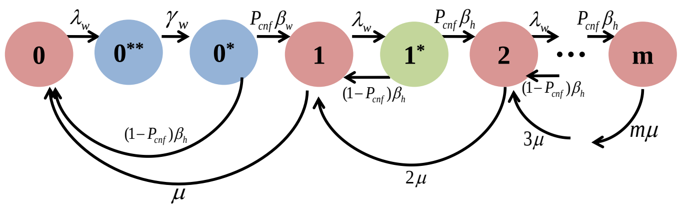


Figure 6: Resource usage model for each warm server.

$$P_w = 1 - (\pi_m^{(W)})^{n_w} \tag{4}$$

Similar model for each of the cold servers is shown in Figure 7 and cold pool model is the set of n_c such models.

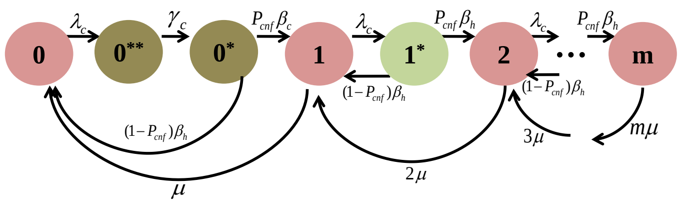


Figure 7: Resource usage model for each cold server.

Main differences between the warm and cold server models are - (i) effective arrival rates (λ_w vs. λ_c), (ii) rate at which startup process is executed (γ_w vs. γ_c) and (iii) initial VM configuration rates (β_w vs. β_c). Expression for effective arrival rate to each cold server is given in (5).

$$\lambda_c = \frac{\lambda(1-P_{block})(1-P_h)(1-P_w)}{n_c} \qquad (5)$$

Output of the cold pool model is the probability that atleast one server in a cold pool is ready to accept a job for resource provisioning. This is shown in (6), where $\pi_m^{(C)}$ is the steady state probability of all the cores in cold server being allocated for job provisioning.

$$P_c = 1 - (\pi_m^{(C)})^{n_c} \qquad (6)$$

(3) Run-time model. Once a job is successfully provisioned, it utilizes the resources until its execution is complete. Run-time models help to determine the mean time for a job completion.

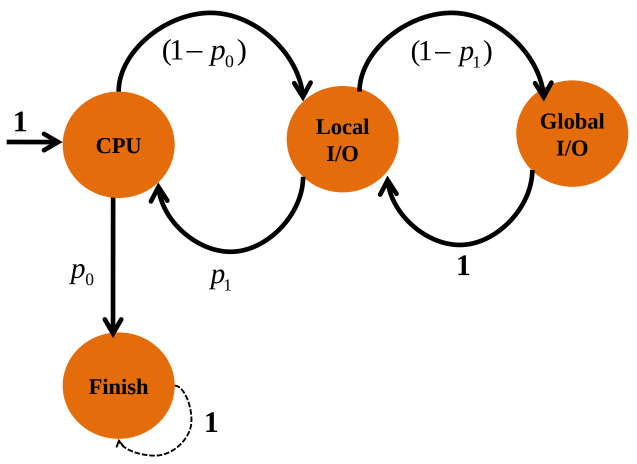


Figure 8: Run-time model for each job type.

We design a Discrete Time Markov Chain (DTMC) (Figure 8) to capture the details of job execution. From CPU, a job under going execution, can finish with a probability p_0 or go for some local I/O operations with probability $(1-p_0)$. Similar transition can happen from local I/O to global I/O with a probability $(1-p_1)$ or from local I/O to CPU with probability p_1.

$$\frac{1}{\mu} = \frac{1}{p_0\mu_c} + \frac{(1-p_0)}{p_0 p_1 \mu_l} + \frac{(1-p_0)(1-p_1)}{p_0 p_1 \mu_g} \qquad (7)$$

Assuming mean service time of CPU, local I/O and global I/O to be $1/\mu_c$, $1/\mu_l$ and $1/\mu_g$ respectively, we can compute mean job service time $(1/\mu)$ as shown in (7).

2.2 Pure Availability Analysis

Main assumptions. We summarize our main assumptions below.

(1) For simplicity, in the current version of our model, we consider only physical server failures [12] and repairs.
(2) Time to failure of each hot server is exponentially distributed with mean $1/\gamma_{hot}$. Time to failure of each warm server is exponentially distributed with mean $1/\gamma_{warm}$. $1/\gamma_{warm}$ is typically 2-4 times higher than $1/\gamma_{hot}$. Cold servers do not fail.
(3) Failure of a server in one pool also triggers migration of a server from other pools to replace the failed server. If a warm server fails, the failed server undergoes repair and is replaced by a server from the cold pool (if there is atleast one server available in cold pool). When a hot server fails, the failed server is first tried to be replaced by a warm server. If no warm sever is available, replacement is attempted by migrating a cold server to hot pool. After migration, state of a server (hot/warm/cold) becomes similar as the state of other servers in the pool where it has been migrated. For example, a warm server migrated to a hot pool becomes a hot server. We assume that the migration process instantaneous (e.g., very fast software based migration) for simplicity of model analysis.
(4) A shared repair policy is assumed. Time to repair is exponentially distributed with mean $1/\mu_r$. This mean time to repair (MTTR) is same for hot and warm servers. Once a failed server is repaired, it is returned to the original pool where it belonged before failure. Any server which was borrowed to replace the failed server is also returned to its original pool.

Model Description. To describe the availability model, we take a simple example, where each pool (hot, warm and cold) has two servers initially. CTMC model for availability analysis for this example, is shown in Figure 9.

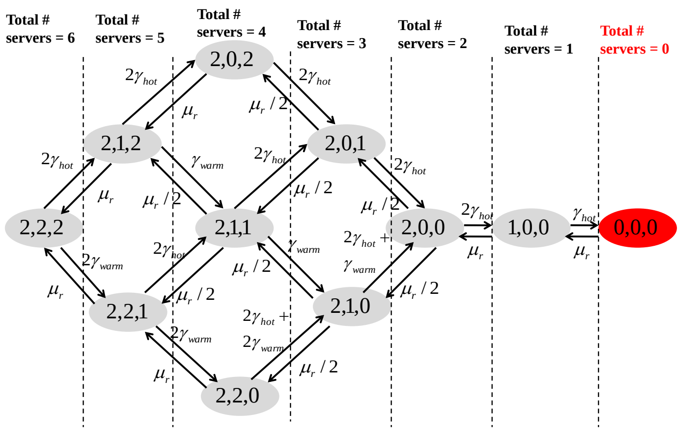


Figure 9: An instance of availability model with two servers in each pool initially. Red colored state $(0,0,0)$ denotes all servers have failed i.e. cloud system is unavailable.

State index is denoted by (i, j, k), where i, j, k are the number of hot, warm and cold servers respectively. Initially, the system is in state (2, 2, 2), where two kinds of failure events can happen - (i) a hot server fails or (ii) a warm server fails. If a hot server fails, system goes to state (2, 1, 2) with a rate of transition $2\gamma_{hot}$ (since time to failure is determined by the minimum of time to failure of individual servers). Observe, instantaneously a warm server has been moved to hot pool and number of warm servers decreases by one. Hence, number of available hot servers does not change. Once this hot server is repaired, warm server is returned to warm pool and system comes back to state (2, 2, 2) with a transition rate μ_r. Similarly, failure of a warm server in state (2, 2, 2) will take the system to state (2, 2, 1), where a cold server is migrated to warm pool. System moves from (2, 2, 1) to (2, 1, 1) upon failure of a hot server. However, failure of a warm server in state (2, 2, 1) triggers a transition to state (2, 2, 0), where all the cold servers have been migrated to warm pool. In state (2, 2, 0), model makes a transition to state (2, 1, 0) with a rate of transition $(2\gamma_{hot} + 2\gamma_{warm})$ due to failure of either hot or warm server. Note that, system can go to state (2, 1, 0) from state (2, 1, 1) as well, due to failure of a warm server and migration of a cold server to warm pool. From state (2, 1, 0), system can return to either state (2, 1, 1) or state (2, 2, 0) with same rate of transition $\mu_r/2$ because of a shared repaired policy. If the hot server is repaired, system goes to state (2, 2, 0) and returns the warm server to warm pool. Repair of a warm server will take the system to state (2, 1, 1) from (2, 1, 0) by returning the cold server to cold pool. A cold server can also migrate to hot pool, upon failure of a hot server. Example of such transition is shown from state (2, 0, 2) to (2, 0, 1), where no warm server is available to replace the failed hot server. From state (2, 0, 0) onwards, there is no more server available in either warm or cold pool. Hence, successive hot server failures finally make the entire system unavailable when system reaches to state (0, 0, 0). If steady state probability of being in state (0, 0, 0) is denoted by $\pi^{(a)}_{(0,0,0)}$, steady state availability (A) of the cloud system is given by (8).

$$A = 1 - \pi^{(a)}_{(0,0,0)} \tag{8}$$

General version of the availability model is shown in Figure 10.

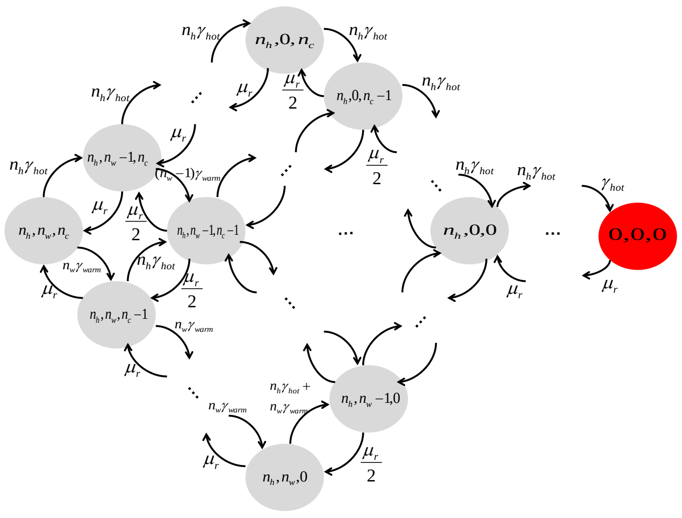


Figure 10: General version of availability model.

2.3 Model Interactions, Performability

Combination of performance and availability models lead to performability models. Combined model and interactions of sub-models are shown in Figure 11. The model hierarchy can be divided into two broad classes - one pure availability model and multiple pure performance models (provisioning decision model, resource usage model and run-time model). We briefly describe the interactions among these models here.

Table 3: Meaning of notations used in Figure 11

Notations	Meaning
P_{block}	Probability that a job is rejected due to buffer full.
P_{reject}	Combined job rejection probability due to buffer full and limited capacity assuming no server failure.
$< t_{prov} >$	Mean overall provisioning delay of a job conditional upon not being rejected (assuming no server failure).
P_h, P_w, P_c	Probability that a server in a class of pool can accept a job for resource provisioning. Subscripts '*h, w, c*' denote hot, warm and cold respectively.
$\frac{1}{\mu}$	Mean time to complete a job or mean resource holding time for a job.

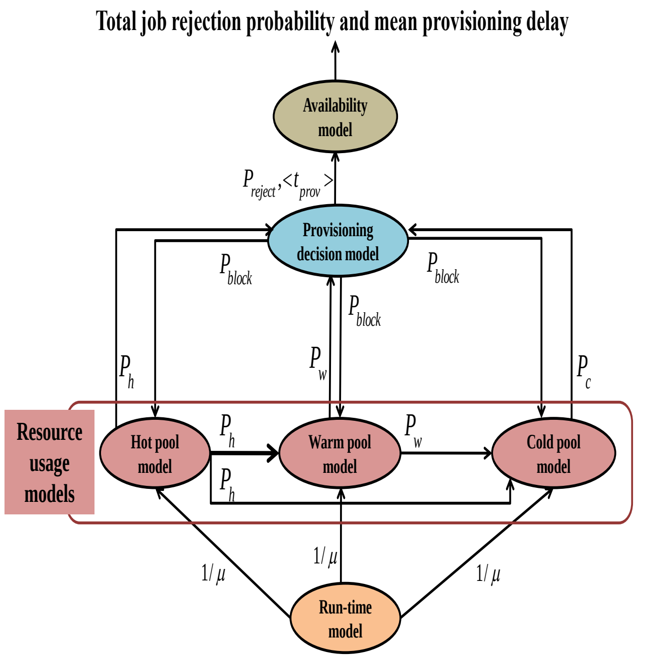


Figure 11: Interactions among the models in four level model hierarchy. Interactions have been shown strictly in terms of model outputs. Meanings of the notations are shown in Table 3

Interactions among sub-models are shown as an import graph in Figure 11. For a particular type of a job, run-time model provides mean resource holding time ($1/\mu$). This output is utilized as an input parameter to each type (hot, warm or cold) of the resource usage model. Resource usage models compute the probabilities (P_h, P_w and P_c) that at least one server in a particular pool (hot, warm and cold, respectively) can accept a job for provisioning. These probabilities are used as input parameters to the provisioning decision model. Provisioning decision model computes blocking probability (P_{block}) due to buffer full, rejection probability due to limited capacity (P_{drop}), P_{reject} (sum of P_{block} and P_{drop}) and mean overall provisioning delay ($< t_{prov} >$) conditional upon the job not being rejected (assuming no server failure). In the top-most availability model, ($< t_{prov} >$, P_{reject}) are used as reward rates to compute total job rejection probability and mean provisioning delay. Observe, there are dependencies among the sub-models. P_{block} computed in the provisioning decision model is used as an input parameter in resource usage models. However, to solve the provisioning decision model, outputs from resource usage models (P_h, P_w, P_c) are needed as input parameters. This cyclic dependency is resolved via fixed-point iteration using the method of successive substitution [8, 13].

3. NUMERICAL RESULTS

We evaluated cloud services under two metrics - (1) job rejection probability and (2) mean provisioning time. Though we investigated our models under variety of settings, due to space limitations, we present only important results here. Specifically, we show the effect of changing job arrival rates,job service time and system capacity (number of servers in each pool). We assumed exponential distribution for inter-arrival times, service times and server failures. All models were solved using SHARPE [16] software package. Assumed values of key parameters are shown in Table 4

Table 4: Values of key parameters

Symbol	Meaning	Value
$1/\delta_h$, $1/\delta_w$, $1/\delta_c$	mean delay to decide whether atleast one server from a particular pool (hot,warm or cold) can be used for resource provisioning.	3 seconds
P_{cnf}	probability that VM configuration (instantiation and deployment) is successful	0.95
$1/\beta_h$	mean time to VM configuration (instantiation and deployment) on a hot server	5 minutes
$1/\beta_w$	mean time to VM configuration (instantiation and deployment) on a warm server	7.5 minutes
$1/\beta_c$	mean time to VM configuration (instantiation and deployment) on a cold server	15 minutes
$1/\gamma_w$	mean time to prepare a warm machine ready for use	20 seconds
$1/\gamma_c$	mean time to prepare a cold machine ready for use	7.5 minutes
$1/\gamma_{hot}$	mean time to a hot server failure	1000 hours
$1/\gamma_{warm}$	mean time to a warm server failure	3500 hours
$1/\mu_r$	mean time to repair a server	1 hour

Job rejection probability. Figure 12 shows, at a fixed mean service time, increasing arrival rate increases job rejection probability. However, the effect is more pronounced if the system capacity is low.

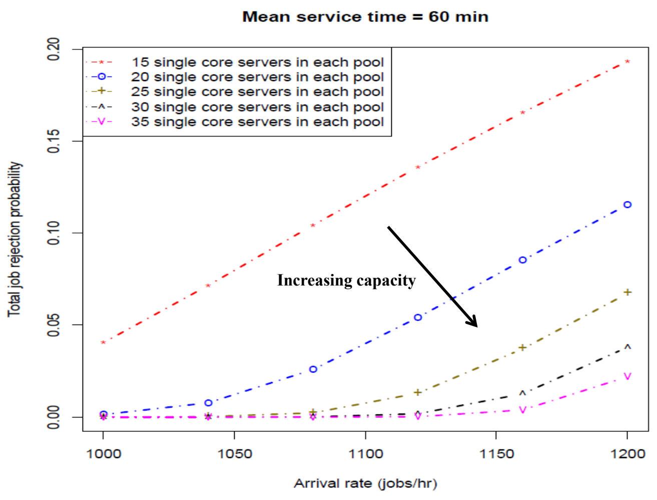


Figure 12: Effect of variation in arrival rate on job rejection probability at a fixed mean service time.

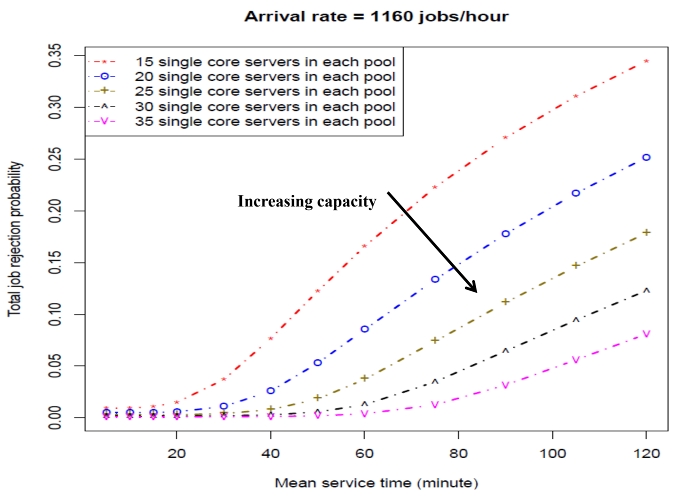


Figure 13: Effect of variation of mean service time on job rejection probability at a fixed arrival rate.

Similar effects of increasing system capacity on job rejection probability is also shown in Figure 14, where arrival rate is kept constant. Increasing mean service time at a fixed arrival rate also increases job rejection probability as shown in Figure 13 and Figure 14.

Mean provisioning time. Figure 15 shows with increasing arrival rate, mean provisioning time increases. In the lower arrival rate region (arrival rate less than 1100 jobs/hr as in Figure 15), benefits of increasing system capacity can be observed. However, in the higher arrival rate region (arrival rate around 1160 jobs/hr), mean provisioning time increases with increasing capacity. This is because, in our provisioning decision model, we dynamically increase provisioning decision queue length with increasing system capacity.

When the arrival rate is high at a fixed mean service time, increasing queue length will allow more jobs in the pro-

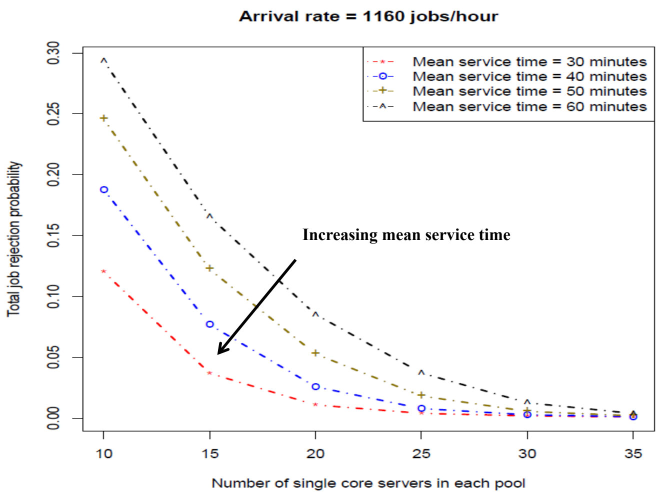


Figure 14: Effect of variation of system capacity on job rejection probability at a fixed arrival rate.

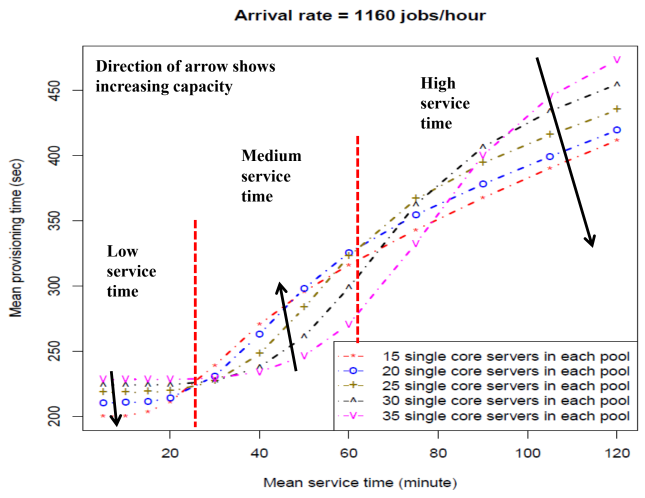


Figure 16: Effect of variation of mean service time on mean provisioning time at a fixed arrival rate.

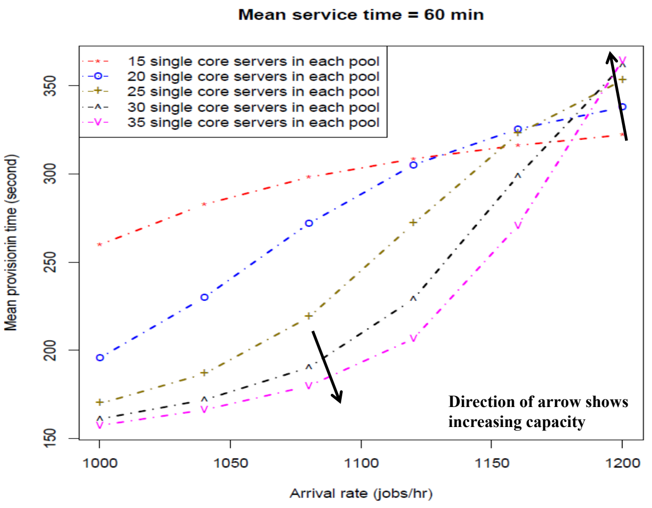


Figure 15: Effect of variation in arrival rate on mean provisioning time at a fixed mean service time.

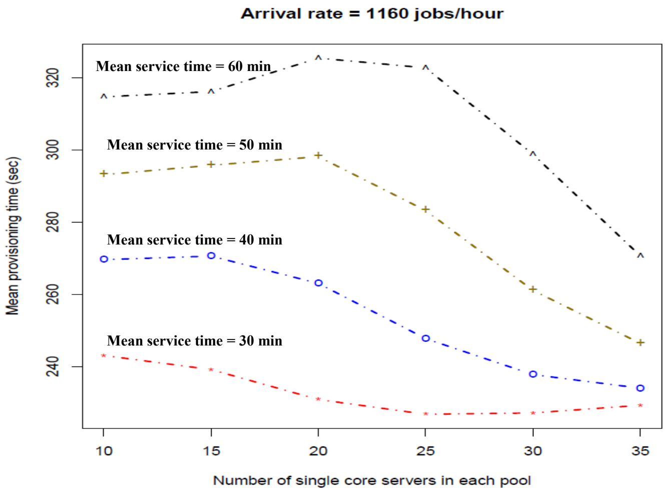


Figure 17: Effect of variation of system capacity on mean provisioning time at a fixed arrival rate.

visioning decision queue which results in increasing mean provisioning time. This inter-play between system capacity and provisioning decision queue length leads to interesting performance trade-offs. The effect is more interesting if we vary mean service time at a fixed but high value of arrival rate. Depending upon service requirements of the jobs, mean service time can be subdivided in three categories - low, medium and high. Figure 16 shows, when the mean service is low, lower capacity systems achieve better mean provisioning time. This is because, jobs are taken out of the provisioning decision queue very fast and hence, a smaller sized queue reduces mean provisioning time. For medium range mean service time, effect of having larger system capacity more than compensates the effect of having a larger sized queue. As a result, having more servers in each pool reduces mean provisioning time. Finally, when the mean service time is high, effect of increased queue length is pronounced once again. Similar effects are also shown in Figure 17 at the same arrival rate. Consider the case, when the number of severs in each pool is increased from 15 to 20. If mean job service time is 30 minutes, mean provisioning time is reduced. However, if the mean job service time 60 minutes, mean provisioning time increases. Further increase in system capacity, reduces mean provisioning time. Results corresponding to mean service time of 60 minutes in Figure 17 can be compared with the mean provisioning time values corresponding to an arrival rate of 1160 jobs/hr in Figure 15. Similar phenomena have been represented in both plots.

4. FUTURE RESEARCH

Capacity planning. Our initial results show that effect of varying system capacity is tightly coupled with a variety of system/job parameters (provisioning queue length, arrival rate, mean service time). In future, we want to solve the following question. Given an expectation on the demand of various different configurations of VMs, how many physical resources should the service provider have over a period of time so that it can uphold the cloud SLA requirements and do not fall short of resources?

Cost optimization of services and resources. In our current model, all the workloads have equal priority. Real clouds should be able to handle a mix of high and low priority workloads. In this context, we want to answer the following optimization question [4]. Given a characteristic of workload mix, what is the capacity that we should acquire over a period of time so that - (a) migration cost in case of private cloud and (b) overall cost for both public and private cloud is minimized while SLA is upheld?

Energy profiling. Our models can be exploited to compute energy consumptions [9] in a cloud environment. Depending on the granularity of energy consumptions (joules per core or joules per server), we can attach different reward rates to corresponding models (resource usage model or availability model). From this, we can compute total steady state or transient energy consumption in a typical cloud scenario.

Heterogenous requests. Results presented in this paper have been obtained under the assumption of identical request types (e.g. service time of each request were assumed to be same). It will be interesting to look at the cloud services with heterogeneity in the requests [5]. We want to vary both the number of cores/storages/VMs being requested by the job as well as vary execution time.

Detailed analysis of downtime. In our current approach, we only consider physical machine failures to denote aggregate failure. We plan to investigate different types of hardware, software, configuration and manual failures. Especially, failure of VMs are of interest [7]. We will also analyze effect of different repair strategies on system availability.

Dealing with non-exponential distributions Though our current models assume exponential distributions for job inter-arrival time, service time, server failure etc., we will remove this assumption in our future work. Specifically, we will use semi-Markov processes and Markov regenerative processes to model and solve such problems.

Model verification and validation. We plan to cross-validate our models in multiple ways. Our immediate goal is to develop a single stochastic Petri-net based model for the entire system. This will help us to understand the approximation error (if any) due to hierarchical model decomposition. Finally, we want to validate our results against simulative methods and controlled experimentations.

5. CONCLUSIONS

We investigated and evaluated cloud service measures such as job rejection probability and mean provisioning time through high fidelity analytical models. Our broad study of performability analysis of cloud services and systems helped us to identify that hierarchical models (interacting Markov chains) are the most useful ones to solve problems for a wide range of cloud scenarios. Using such approach, we can reduce the model complexity significantly and can incorporate many realistic features simultaneously. We believe, these initial findings will be very useful to solve specific problems related to cloud computing such as capacity planning for public and private cloud systems and optimization of cloud services and resources.

ACKNOWLEDGMENTS

Research by Duke authors (Ghosh, Kim and Trivedi) was supported in part under a 2009 IBM Faculty Award and in part by NSF under Grant NSF-CNS-08-31325. In particular, the authors would like to thank Dan Dias and Murthy Devarakonda from IBM Research for their support and encouragement for this work.

6. REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2.

[2] IBM Cloud Computing. http://www.ibm.com/ibm/cloud/.

[3] IBM SBDTC. http://www-180.ibm.com/cloud/enterprise/beta/dashboard.

[4] J. H. et. al. An on-line, business-oriented optimization of performance and availability for utility computing. *IEEE JSAC*, 2005.

[5] P. Garbacki and V. Naik. Efficient resource virtualization and sharing strategies for heterogeneous grid environments. In *Proceedings IM*, 2007.

[6] M. Grottke and K. S. Trivedi. Truncated non-homogeneous Poisson process models - properties and performance. *Opsearch (India)*, 42(4):310–321, November 2005.

[7] D. S. Kim, F. Machida, and K. S. Trivedi. Availability modeling and analysis of a virtualized system. In *PRDC*, 2009.

[8] V. Mainkar and K. S. Trivedi. Sufficient conditions for existence of a fixed point in stochastic reward net-based iterative models. *IEEE Transaction on Software Engineering*, 1996.

[9] B. Nogueira, P. Maciel, K. S. Trivedi, and R. Ferraz. Estimation of energy consumption and performance in embedded system platforms. *in preparation*, 2010.

[10] H. Okamura, T. Dohi, and K. S. Trivedi. Markovian arrival process parameter estimation with group data. *IEEE/ACM Transactions on Networking*, 17(4):1326–1339, August 2009.

[11] N. Sato and K. Trivedi. Accurate and efficient stochastic reliability analysis of composite services using their compact Markov reward model representations. In *IEEE SCC*, 2007.

[12] W. E. Smith, K. S. Trivedi, L. Tomek, and J. Ackeret. Availability analysis of multi-component Blade Server system. *IBM Systems Journal*, 2008.

[13] L. Tomek and K. Trivedi. Fixed-point iteration in availability modeling. In *FTCS*, 1991.

[14] K. Trivedi, D. Wang, D. Hunt, A. Rindos, W. Smith, and B. Vashaw. Availability modeling of sip protocol on ibm websphere. In *PRDC*, 2008.

[15] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Wiley, 2001.

[16] K. S. Trivedi and R. Sahner. SHARPE at the age of twenty two. *ACM Sigmetrics Performance Evaluation Review*, 36(4):52–57, March 2009.