Analysis of Distributed Storage Systems w/ Continuous Time Markov Chains (CTMCs)

Introduction

This lab will walk you through an interesting use of CTMCs in the analysis of data centers. The explosion of data on the internet resulted, naturally, in an explosion of data centers. Massive data centers of constantly failing nodes pose interesting challenges for us EECS ninjas. As a result, distributed storage is a very active area of research these days. Hope you enjoy!

In [2]:
```
#standard imports, and potentially useful functions
from __future__ import division
import random
import numpy as np
from numpy import zeros
import matplotlib.pyplot as plt
import scipy
from numpy.linalg import svd, norm

def nullspace(A, atol=1e-13, rtol=0):
    A = np.atleast_2d(A)
    u, s, vh = svd(A)
    tol = max(atol, rtol * s[0])
    nnz = (s >= tol).sum()
    ns = vh[nnz:].conj().T
    return ns

def nSample(distribution, values, n):
    rand = [random.random() for i in range(n)]
    rand.sort()
    distribution = distribution.copy()
    samples = []
    samplePos, distPos, cdf = 0, 0, distribution[0]
    while samplePos < n:
        if rand[samplePos] < cdf:
            samplePos += 1
            samples.append(values[distPos])
        else:
            distPos += 1
            cdf += distribution[distPos]
    return samples

%matplotlib inline
```

Part 0: Reading

If you haven't already, read Chapter 13.5 of Probability in EECS on CTMCs. The chapter is concise, so don't worry if you can't quite grasp everything on your first pass. This lab will walk you through everything you need to know about CTMCs, but the book will be a good place to start. You may also want to very briefly review Reed-Solomon erasure codes form CS 70. Hopefully this will be a nice review of last week's material :)

Part 1: CTMCs

There are several differences between Continuous Time Markov Chains and their discrete time counterparts, but a good place to start is by thinking of a CTMC as a DTMC where the transitions between states can happen at any time, and that the transition times are exponentially distributed. We will explore this more thoroughly through the following example.

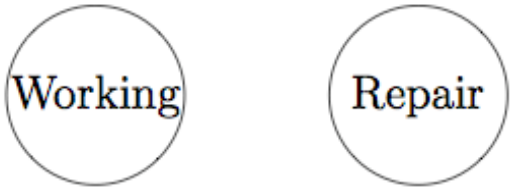


Figure 1

Let us consider a single node in a data center that can exist only in two possible states: either it is *working* or it is *in repair*. If the data storage node is in a *working* state, the data stored on the node is secure and the node itself is happily responding to user read/write requests. If the node is in a *repair* state, some problem has occured and the data stored on the node has become corrupted (data loss). Let us further model the transitions between the two states as a Markov process (i.e. satisfiying the Markov property).

If we model the process in discrete time and only consider transition probabilities between the two states, the result is the uninformative model presented below:

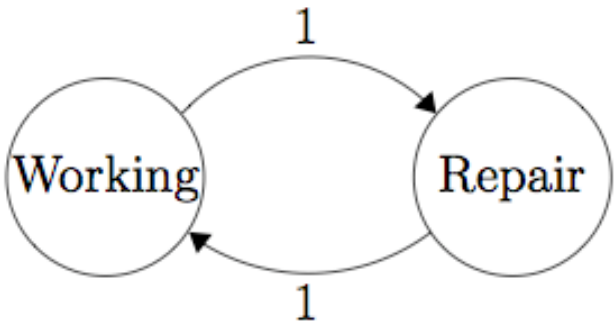


Figure 2

This model obfuscates the time it takes to transition between states, traditionally called the *firing times*. We would hope that our data storage node would take a very long time before failing, and that it would take comparatively little time to be fully repaired once it has failed.

There can be more sophisticated discrete time models (which you might want to think about when simulating CTMCs later in the lab), but why would we limit ourselves to a discrete time model for something which is clearly occuring in continuous time, and no immediate discrete time analog presents itself? Thus let us consider how we might construct a continuous time model.

Markov Property in Continuous Time

Let us consider the Markov process \(\{X_t, t \geq 0\}\) over a countable or finite state space \(\mathcal{X}\). In Satisfying the Markov Property, we have that ### \[P[X_{t+\epsilon} = i \ | \ X_t, X_u:u < t] = P[X_{t+\epsilon} = i \ | \ X_t] \ \ \ \forall \ i \ \in \ \mathcal{X}\]

Let us use this property to answer the following question and help us integrate the notion of *firing times* into our model:

1. How long does our process remain in a given state? Suppose \(X_0 = x\). What is the distribution of \(T_{x}\), the time of departure from state \(x\)? Prove that this departure time will be memoryless i.e. that \(Pr(T_x > s + r \ | \ T_x > s) = Pr(T_X > r)\), for \(s, r \geq 0\). Conclude that the departure times must be exponentially distributed (the only continuous distribution which satisfies the memorylessness property).

Hint: For this proof you will need to use the *time invariance* property of Markov Chains we have described in this class thus far. In DTMCs, time invariance says \(P(X_{n+1} = x \ | \ X_{n}) = P(X_1 = x \ | \ X_0)\), and for CTMCs we have \(P(X_t = x, t \in [s,s+r] \ | \ X_s = x) = P(X_t = x, t \in [0,r] \ | \ X_0 = x)\). Processes which satisfy this property make up an important class of stochastic process known as *stationary* processes

Your answer here

We now know that the departure times will be exponentially distributed and can return to our example from earlier. We have to tools to create a useful model for the state of our data storage node. When a node is working, its failure is exponentially distributed with rate \(\lambda\), at which point it will be under repair. The repair time is exponentially distributed with rate \(\mu\), which will return the node to a working condition. The rate of failure \(\lambda\) will be a function of the hardware and software in the storage node (which are prone to crashes and bugs), and \(\mu\) will be a function of the time it takes to detect the failure, fix it, and replace the data that was originally on the node. We would hope that our storage center nodes would satisfy \(\mu >> \lambda\).

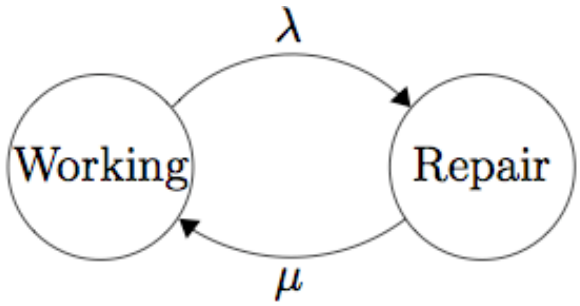


Figure 3

Suppose we had a slightly different CTMC where sometimes a node fails irreparably and ends up in a *Dead* state.

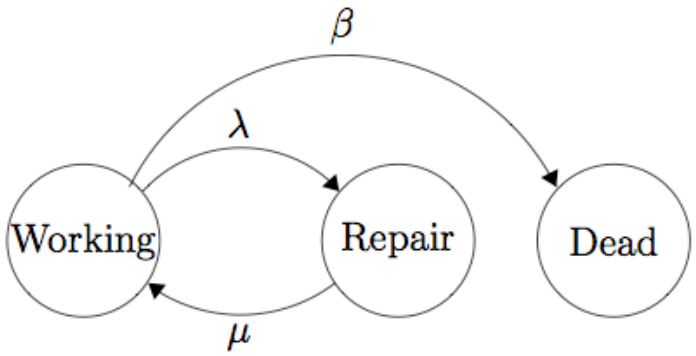


Figure 4

Use Figure 4 and your intuition to answer question 2:

2. What is the rate of transition out of state 'Working' in the CTMC in Figure 4?

3. What is the conditional probability of transition to state 'Repair' given that our CTMC is exiting state 'Working'? What is the conditional probaility of transition to state 'Dead'?

Your answer here

Specifying the Rate Matrix \(Q\)

The rate matrix \(Q\) of the CTMC in Figure 3 (assuming Working as state 1 and Repair as state 2) is #### \ [\begin{bmatrix} -\lambda & \lambda \\ \mu & -\mu \end{bmatrix}\]

In general, the \(i,j^{th}\) element of the rate matrix \(Q_{ij}\) specifies the rate at which the chain is leaving state \(i\) and entering state \(j\) for \(i \neq j\). The diagonal elements of the matrix \(Q_{ii} = -\sum_{j} Q_{ij}\). Why is the rate matrix represented as such? Intuitively, the rate matrix in continuous time represents the rates of flow in and out of the states in a CTMC, or the average rates of transition in and out of each state. If the rate of flow out of state \(i\) is \(\sum_{j} Q_{ij}\), then the rate of flow into state \(i\) must be the same, and hence we write \(Q(i,i) = -\sum_{j} Q_{ij}\), denoting the flow from all other states into state \(i\). We can then determine the invariant distribution through conservation of flow (not unlike Kirchoff's Current Law).

Finding the Invariant Distribution (if one exists)

Let \(\pi = [\pi(\text{Working}), \ \ \pi(\text{Repair})]\) denote the vector of probabilities of being in a *Working* or *Repair* state respectively. We can then use this conservation of flow idea to solve for the invariant distribution of the initial markov chain: #### \[\lambda \cdot \pi(\text{Working}) = \mu \cdot \pi(\text{Repair}) \] #### \[\pi(\text{Working}) + \pi(\text{Repair}) = 1 \] #### If this conservation of rate does not make sense to you, stare at it for a while and it will.

Let us plug in the numbers \(\lambda = \frac{1}{10}\) and \(\mu = 10\), which says that the time until node failure is exponentially distributed with mean time 10 years, and the time for repair is exponentially distributed with mean time 1.2 months. In this example, repair happens 100 times faster than failure on average. We can calculate the invariant distribution:

\[\frac{\pi(\text{Working})}{\pi(\text{Repair})} = \frac{\mu}{\lambda} = 100 \implies \frac{1 - \pi(\text{Repair})}{\pi(\text{Repair})} = 100 \implies \pi(\text{Repair}) = \frac{1}{101}, \ \ \ \pi(\text{Working}) = \frac{100}{101}\]

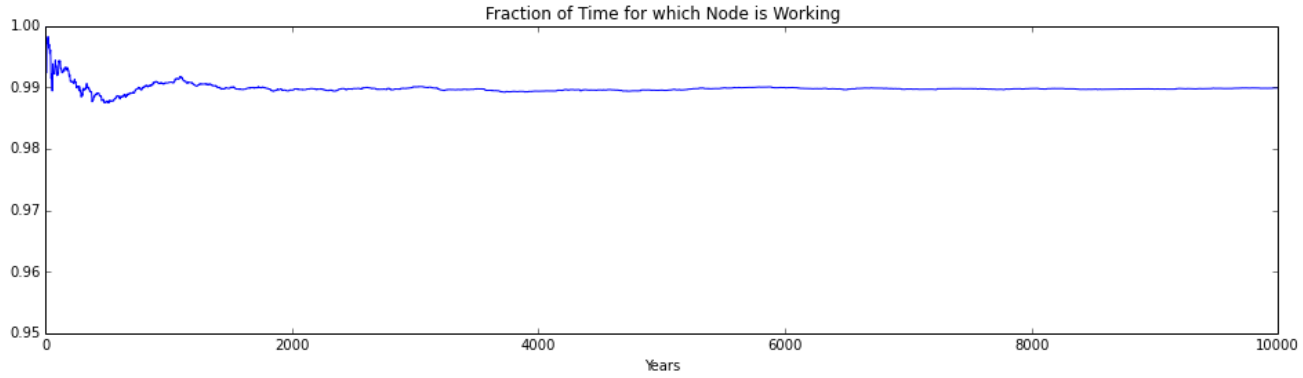
In general, we can calculate the invariant distribution of a CTMC by solving the following system of equations: #### \[\pi Q = 0 \] #### \[1^\intercal \pi = \sum_i \pi(i) = 1 \]

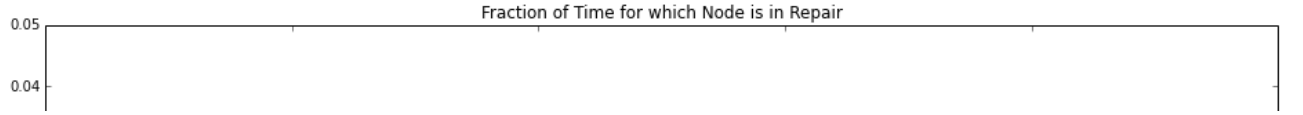
The follwing code simulates the CTMC in Figure 3 when \(\lambda = \frac{1}{10}\) and \(\mu = 10\) for 10,000 years and shows that the long term fraction of time spent in the two state *Working* and *Repair* does indeed converge to the values calculated above when \(\lambda = \frac{1}{10}\) and \(\mu = 10\).

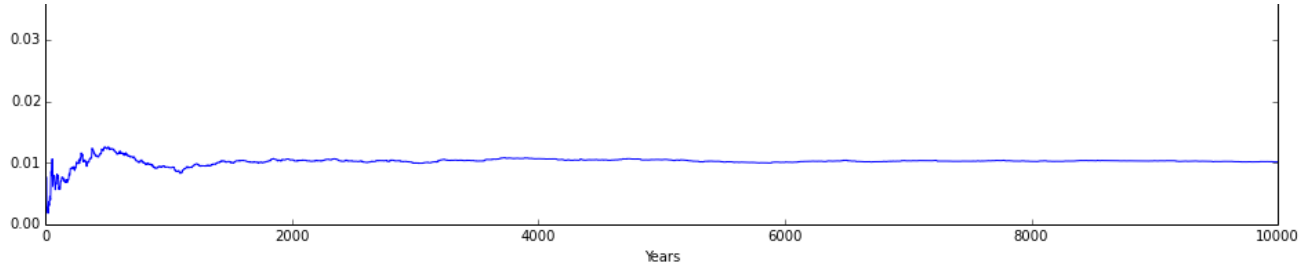
In [98]:
```
# Note: this is probably not how you should write your code for larger state s
paces; it doesn't scale well.

lambd = 1.0 / 10.0 # Node failure occurs at rate of approximately 1 every 10 y
ears
mu = 10.0 # Node repair occurs at a rate of approximately 1 every 1.2 months

TotalTime = 0.0; WorkTime = 0.0; RepairTime = 0.0
```

```
WorkFrac = np.array([]);RepairFrac = np.array([]) # Fraction of time spent working and in repair respectively
Work = 0; Repair = 1; state = Work # We start the simulation with a working node
while TotalTime < 1.0e4: # Simulating for 10,000 years
    rate = [lambd,mu][state]
    JumpTime = random.expovariate(rate) # JumpTime ~ Exp(rate)
    TotalTime += JumpTime
    if state is Work:
        WorkTime += JumpTime
    else:
        RepairTime += JumpTime

    RepairFrac = np.r_[RepairFrac, RepairTime / TotalTime]
    WorkFrac = np.r_[WorkFrac, WorkTime / TotalTime]

    state = [Repair,Work][state]

fig = figure(figsize = (16,4))
plot(np.linspace(0,1e4,len(WorkFrac)),WorkFrac)
title('Fraction of Time for which Node is Working')
ylim([0.95,1])
xlabel('Years')
fig = figure(figsize = (16,4))
plot(np.linspace(0,1e4,len(WorkFrac)),RepairFrac)
title('Fraction of Time for which Node is in Repair')
ylim([0,0.05])
xlabel('Years')
print "Overall Fraction of Time Spent Working: ", WorkFrac[-1]
print "Overall Fraction of Time Spent in Repair: ", RepairFrac[-1]
```

```
Overall Fraction of Time Spent Working:  0.989900161219
Overall Fraction of Time Spent in Repair:  0.010099838781
```







4. Determine the rate matrix \(Q\) and calculate the invariant distribution of the following CTMC. Simulate the Markov Chain and see that the fraction of time spent in each state converges to \(\pi\). There exists something called "the embedded Markov Chain," \(\tilde{Q}\), which may be useful for this simulation, if you want to make things a bit more efficient.

\(\tilde{Q}_{ij}\) encodes the conditional probability of transitioning from state \(i\) to state \(j\) given that the CTMC is transitioning out of state \(i\) (http://en.wikipedia.org/wiki/Continuous-time_Markov_chain#Embedded_Markov_chain). What you are effectively doing is only looking at the process when an event occurs.

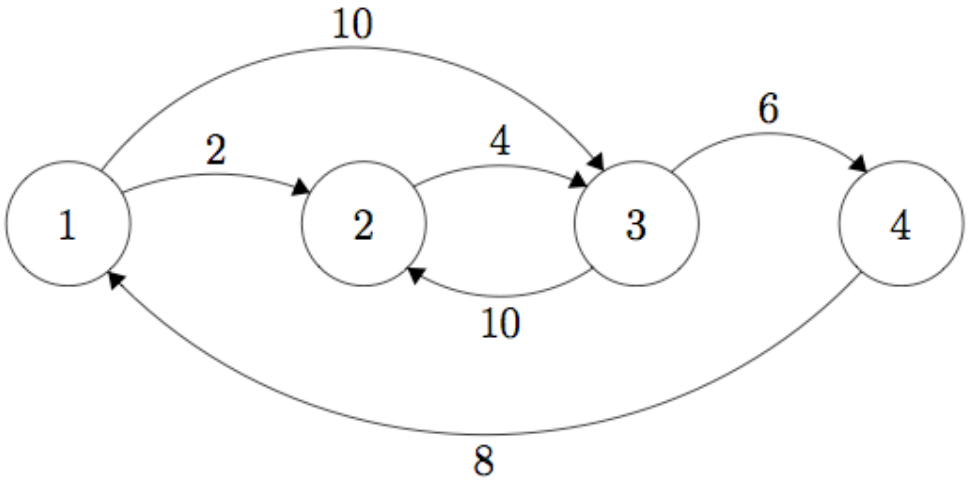


Figure 5

Your answer here

Part 2: Erasure Codes

Suppose our storage system has an object made up of \(k\) source data symbols that we would like to protect with a Reed-Solomon erasure code. We encode them into \(n\) total encoded symbols, giving us \(r = n - k\) repair symbols. Recall that Reed-Solomon codes satisfy the Maximum Distance Seperable (MDS) property, that the \(k\) source symbols can be recovered from any \(k\) of the total \(n\) encoded symbols. We denote this as an \([n,k,r]\) MDS code, and can be largely seen as a black box for this lab. The important thing to note is that if there are ever less than \(k\) symbols available, the the data is lost permanently; otherwise, the data can always be recovered.

Part 3: MTTLD Analysis in Data Center

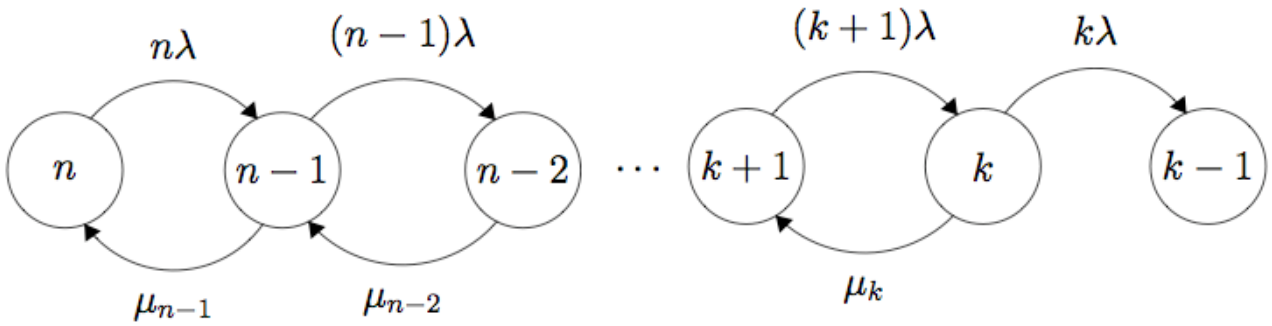


Figure 6

Our storage system maintains source data at the granularity of objects. The source data is partitioned into objects as it arrives, and is stored within the storage system. Each object is partitioned into \(k\) source fragments. An erasure encoder is used to generate \(r=n-k\) additional repair fragments from the \(k\) source fragments, and each of these \(n=k+r\) source and repair fragments are stored in a different node.

The Continuous Time Markov Chain depicted above describes the life of one object in the storage system. Every object starts in state \(n\), where every fragment of the data is available. As nodes in the storage system go through the process of failure and repair, there may at some times be fewer than \(n\) fragments avaialable. For example, if a node with one fragment of an object in state \(n\) fails, then it will lose a fragment and fall into state \(n-1\) until it has been repaired or another fragment of data is lost, whichever comes first. The state of the object in the above CTMC is equal to the number of fragments of an object that are currently avaialable. You will notice that once a piece of data has \(k-1\) fragments available, further fragments can no longer be generated (hence there are no arrows leaving state \(k-1\)). When there are only \(k-1\) fragments of an object available in the storage system, that piece of data is considered lost, and Facebook, Google, Amazon, Dropbox, Apple, Microsoft, EMC, NetApp, Akamai, Rackspace, etc. all agree that losing data is a very bad thing.

Thus, we can use a metric known as Mean Time to Loss of Data (MTTLD) to compare different storage and repair policies and pick the ones which gives us the most reliability at the lowest cost. We can derive this metric from the CTMC above as follows:

1. Calculate the expected escape time of the CTMC, or equivalently, the expected hitting time of state \(k-1\) starting from state \(n\). This will give us the mean time to loss of a particular piece of data.
2. Divide that number by the total number of objects in the storage system to find the overall MTTLD. It is important to note that this calculation assumes that each of the objects in our system are acting independently.

5. Determine the general form of the balance equations to calculate the mean hitting time to a certain state in a CTMC. Let \(T_l\) denote the first time the CTMC hits state \(l\). Then write the balance equations needed to calculate \(\mathbb{E}[T_l \ | \ X_0 = i]\). Express your equations in terms of the \(i,j^{\text{th}}\) components of the rate matrix and the embedded Markov Chain matrix, \(Q_{ij}\) and \(\tilde{Q}_{ij}\) respectively.

Hint: Use the balance equations to calculate hitting times for a DTMC as your guide. Don't forget to incorporate the expected time spent at each state. The final form of your solution should be: \[\mathbb{E}[T_l \ | \ X_0 = i] = \begin{cases} \underline{\hspace{3cm}} & \text{if $i \neq l$} \\ \underline{\hspace{3cm}} & \text{if $i = l$} \end{cases}\]

6. Determine an expression for the Mean Time to Loss of an object (the expected escape time of a single instance of the Markov Chain in Figure 6).

Assume \(\mu >> \lambda\) and \(\mu >> 1\)

Your answer here

Congratulations! You now know how to model (mini) data centers! You should use this knowledge to help Ben with `Bitdiddlers, Inc.` He'll need it.

References:

[1] The Facebook Paper - XORing Elephants

[2] The Google storage paper - Availability in Globally Distributed Storage Systems

[3] The MSFT Azure paper - Erasure Coding in Windows Azure Storage