EP2200 Course Project – 2012
Project II - Simulating a Spotify Server

Ioannis Glaropoulos

November 26, 2012

1 Introduction

Considering the available information regarding the user demand and the system capacity, we can address questions related to the *performance* of a queuing system, often evaluated based on metrics such as response time, duration of service, service denial etc. Unless, however, the models for arrival and service processes are of particular kinds, we are not able to conduct the desired performance evaluation based on analytic tools from Queuing Theory. In such cases we evaluate the system performance by means of simulation.

In this project we consider the case of a Spotify back-end server, used in this well-known on-line media-streaming service. We are given a set of models that describe the user demand and the service capabilities of the back-end server. We aim at measuring the system performance by simulating its behavior towards the user input demand. We are, then, able to answer whether and under which circumstances the system meets its service quality requirements.

2 Scenario and System Architecture

2.1 Spotify Back-end Site Description

We consider a Spotify *back-end site* [1] that is responsible for music delivery to the Spotify customers. The structure of the Spotify back-end site is depicted in Fig. 1. The site includes a set of *storage servers*, on which the *music files* (i.e. songs) are stored. The number of available storage servers is denoted by C.

The total number of songs in the system is denoted as N. The set of songs is denoted by $\mathcal{N}$. We assume that each song, N_i, is stored in a single storage server. $\mathcal{C}_i \in \{1, ..., C\},\ i = 1, 2, ..., N$ denotes the *allocation* of song N_i, i.e. it indicates the server on which the music file is stored. The *song allocation vector*, $\mathcal{C} = \{C_i\}_{i=1,...N}$ contains the information about how the songs are allocated on the different storage servers.

2.2 Client Requests & Server Response

The Spotify users – belonging to a large population – generate client *requests* to the back-end site, i.e. requests to download (stream) the music files. A client request for a music file is collected by the Spotify *manager*, which forwards it

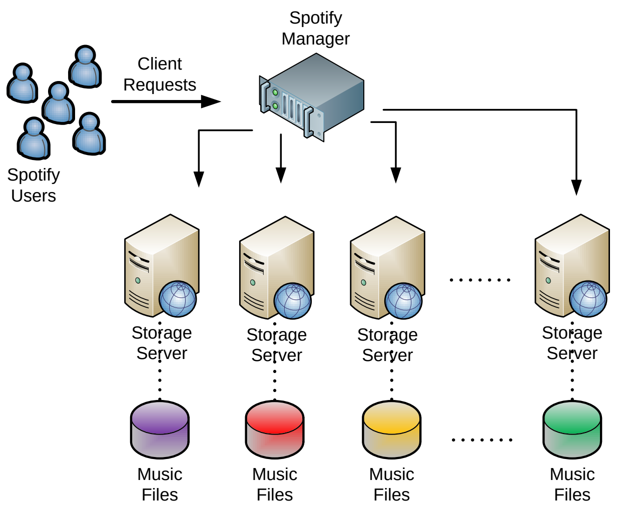

Figure 1: Architecture of the Spotify back-end server

immediately to a storage server on which the respective music file is stored. We model the arrival process of the client requests to the Sptify Manager with a Poisson time-invariant process with intensity λ requests per second.

The requests are queued-up in the *buffer* of the storage server and are served with *first-come-first-served* policy. The buffers of the storage servers can hold at most B client requests. A server can not serve client requests in parallel. The *service* of a request involves either *disk* or *cache memory* access, depending on whether the music file is stored in the hard disk of the server, or in its cache memory. We assume, reasonably that the service time is lower in the case a file is stored in the cache.

A song is stored in the cache of a server with probability p_C and with probability $1 - p_C$ it is stored on the disk. If a song is stored on the cache, the *service time* of a request for this song requires a fixed time of T_S^C seconds, otherwise, it requires an exponentially distributed amount of time with a mean of $\overline{T}_S^H$ seconds.

We define the *response time*, T_R, of a client request as the total time between the arrival of a request at the Spotify back-end server and the time when this request starts being served by the corresponding storage server.

2.3 Song Popularity

The *popularity*, q_i, of a music file, N_i, is defined as the probability that a random client request involves this particular music file, that is:

$$q_i = \Pr\{\text{Request is for song } N_i\} \in (0, 1), \quad i = 1, ..., N. \tag{1}$$

We assume that the popularities of the arriving client requests are *i.i.d* random variables. The *popularity vector*, $\mathcal{Q}$ contains the popularities of all songs, i.e.:

$$\mathcal{Q} \triangleq \{q_1, ...q_N\},$$

where it, clearly, holds:

$$\sum_{i=1}^{N} q_i = 1.$$

2.4 Parameter Setting

We list in Table 1 the parameter setting for the considered scenario.

Request Arrival Rate (λ)	5000 sec^{-1}
Service Time (cache) (T_S^C)	10^{-4} sec
Service Time (disk) (T_S^H)	$\sim exp(\mu), \quad 1/\mu = \overline{T}_C^H = 10^{-3}$ sec
Number of Servers (C)	5
Buffer Capacity (B)	100
Number of Songs (N)	10^3
Cache Availability(p_C)	0.25

Table 1: List of parameters for the numerical solutions

In addition, the file `file_popularities.txt` contains the popularity vector $\mathcal{Q}$, while file `song_allocation.txt` contains the song allocation vector $\mathcal{C}$.

3 System Simulation

You will design and implement a program that simulates the above system. You are allowed to use any programming development environment, but you are strongly encouraged to use **Matlab**$^{\text{TM}}$and take advantage of its statistic toolbox that includes libraries for probability distributions and random generators.

Before starting with the programming it is advisable to draw a *pseudo-code* of the simulator, and use it to verify that your simulator actually works as it is required. Deliver your pseudo-code along with your answers in the following Section.

Once your implementation is finished, simulate the above system for a period of time that is sufficiently long, in order to extract statistically significant results. You could think of letting the simulator run for some initial warm-up period and start collecting results only after this period. In this way you can be confident that the system steady-state has been reached.

4 Performance Evaluation

1. Queuing System First define the queuing systems that model the Spotify back-end and are to be simulated. Give the Kendall notations and the block diagrams. Define the arrival processes and the service time distributions, together with their parameters. Motivate your decisions.

2. Response Time Simulate the queuing systems and measure the performance of the Spotify back-end in terms of the response time. For each storage server derive and plot the empirical distribution (CDF) of the response time of the requests delegated to this particular server. Measure the average response time for each server.

3. The Effect of Caching Simulate the system while gradually decreasing p_C until it becomes 0 (no cashing) and compare the resulting average response time per server with the ones of the original system. Explain the result.

Consider $p_C = 0$. Consider, also, the approximation of having infinite buffer capacity at the storage servers. Based on these give the approximate CDF of the system response time, using known analytic results. Compare with what you get by simulating the system. Are the results similar?

4. Service Denial For the original system measure the performance of the Spotify back-end in terms of service denial, i.e. the percentage of client requests that are dropped by the storage servers, because their buffer is full. Give the probability of service denial for each storage server and evaluate the result.

5. Load Balancing Calculate the *sample variance* ($\hat{\sigma}^2$) of the average response times at each storage server:

$$\hat{\sigma}^2 = \frac{1}{C}\sum_{j=1}^{C}\left(T_C^j - \hat{\mu}_{T_C}\right)^2, \tag{2}$$

where T_C^j is the measured average response time for server j and $\hat{\mu}_{T_C}$ is the sample mean of the average response times considering all storage servers:

$$\hat{\mu}_{T_C} = \frac{1}{C}\sum_{k=1}^{C} T_C^k. \tag{3}$$

What is the reason for having different average response times at the servers? Suggest a different song allocation at the storage servers, $\mathcal{C}'$, that – in your opinion – reduces the variations between the response times. Motivate your answer!

Consider your proposed song allocation vector, $\mathcal{C}'$, simulate the system with the same client request demand (λ) and song popularity vector, $\mathcal{Q}$, and measure the new average response times for the storage servers. Compare with the ones measured for the original system and evaluate whether your proposal was good or not.

5 Submission instructions and grading

- Submission deadline: **Monday, January 4, 2013**. Electronic submission to **ioannisg at kth.se**, or printed copy to STEX. Do not forget to add your name.
- You are allowed to solve the problem in groups, however, you have to prepare a project report on your own. Reports including the same text will be disqualified.

- You need to submit a report of a maximum of 4 printed pages. The report needs to contain a description of the solution of the problems, including the motivation of the models used, and the pseudo-code on which the simulator is built. Discuss your results. Are they reasonable?
- You need to use software tools to get the results, we propose Matlab, but we accept all solutions, e.g., you can even program everything in C. You should not include your codes in the report.
- Check the grammar of the report. There are good tools available to do that. Make sure that performance graphs are possible to interpret. Give the dimensions and units of the axes.

Grading: pass or fail. To pass the moment, you need to show that you made a serious attempt to solve the problems. The best 20% of the submissions receive 5 extra points on the exam in December. Extra points will not be considered at the make-up exam or at later exams. These best 20% reports will be published on the course web.

Would you have any questions, contact John at **ioannisg at kth.se**.

References

[1] R. Yanggratoke, G. Kreitz, M. Goldmann, and R. Stadler, "Predicting response times for the spotify backend," in *International Conference on Network and Service Management (CNSM)*, October 2012.