Predicting Response Times
for the Spotify Backend

Rerngvit Yanggratoke[*], Gunnar Kreitz[†], Mikael Goldmann[†], and Rolf Stadler[*]
[*]ACCESS Linnaeus Center, KTH Royal Institute of Technology, Sweden
[†]Spotify, Sweden
Email: {rerngvit,stadler}@kth.se{gkreitz,migo}@spotify.com

Abstract—We model and evaluate the performance of a distributed key-value storage system that is part of the Spotify backend. Spotify is an on-demand music streaming service, offering low-latency access to a library of over 16 million tracks and serving over 10 million users currently. We first present a simplified model of the Spotify storage architecture, in order to make its analysis feasible. We then introduce an analytical model for the distribution of the response time, a key metric in the Spotify service. We parameterize and validate the model using measurements from two different testbed configurations and from the operational Spotify infrastructure. We find that the model is accurate—measurements are within 11% of predictions— within the range of normal load patterns. We apply the model to what-if scenarios that are essential to capacity planning and robustness engineering. The main difference between our work and related research in storage system performance is that our model provides distributions of key system metrics, while related research generally gives only expectations, which is not sufficient in our case.

Index Terms—Key-value store, distributed object store, performance modeling, system dimensioning, performance measurements, response times, streaming media services

I. INTRODUCTION

The Spotify service is a peer-assisted system, meaning it has a peer-to-peer component to offload backend servers, which are located at three sites (Stockholm, Sweden, London, UK, and Ashburn, VA). While the Spotify backend servers run a number of services, such as music search, playlist management, and social functions, its core service is audio streaming, which is provided by the Spotify storage system [1]. When a client plays a music track, its data is obtained from a combination of three sources: the clients local cache (if the same track has been played recently), other Spotify clients through peer-to-peer technology, or the Spotify storage system in a backend site [1].

Low latency is key to the Spotify service. When a user presses "play", the selected track should start "instantly." To achieve this, the client generally fetches the first part of a track from the backend and starts playing as soon as it has sufficient data so that buffer underrun ("stutter") will be unlikely to occur. Therefore, the main metric of the Spotify storage system is the fraction of requests that can be served with latency at most t for some small value of t, typically around 50 ms. (We sometime use the term response time instead of latency in this paper.)

The Spotify storage system has the functionality of a (distributed) key-value store. It serves a stream of requests from clients, whereby a request provides a key and the system returns an object (e.g., a part of an audio track). In this paper, we present an analytical model of the Spotify storage architecture that allows us to estimate the distribution of the response time of the storage system, as a function of the load, the storage system configuration and model parameters that we measure on storage servers. The model centers around a simple queuing system that captures the critical system resource (i.e., the bottleneck), namely, access to the server's memory cache and disk where the objects are stored.

We validate the model (1) for two different storage system configurations on our laboratory testbed, which we load using anonymized Spotify traces, and (2) for the operational Spotify storage system, whereby we utilize load and latency metrics from storage servers of the Stockholm site. We find that the model predictions are within 11% of the measurements, for all system configurations and load patterns within the confidence range of the model. As a consequence, we can predict how the response time distribution would change in the Stockholm site, if the number of available servers would change, or how the site in a different configuration would handle a given load. Overall, we find that a surprisingly simple model can capture the performance of a system of some complexity. We explain this result with two facts: (1) the storage systems we model are dimensioned with the goal that access to memory/storage is the only potential bottleneck, while CPUs and the network are lightly loaded; (2) we restrict the applicability of the model to systems with small queues —they contain at most one element on average. In other words: our model is accurate for a lightly loaded storage system. Interestingly, the model captures well the normal operating range of the Spotify storage system. As our measurements show, increasing the load beyond the confidence limit of our analysis can lead to a significant increase in response times for a large fraction of requests.

The paper is organized as follows. Section II briefly describes the Spotify storage system. Section III contains the analytical model of the system that suits our purpose. Section IV describes our work on the evaluation of the model and the estimation of the model parameters, both
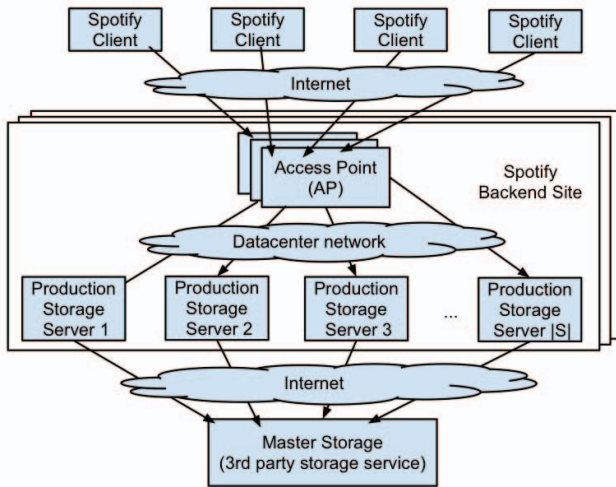
Fig. 1. Spotify Storage Architecture

Fig. 2. Simplified architecture as a basis for the performance model

on the lab testbed and on the Spotify operational system. Section V gives examples of applying the model. Section VI discusses related work and Section VII contains our conclusions and future work.

II. THE SPOTIFY BACKEND STORAGE ARCHITECTURE

We give a (simplified) overview of the part of the Spotify backend responsible for music delivery to the clients. Its architecture is captured in Figure 1. Each Spotify backend site has the same layout of storage servers, but the number of servers varies. The master storage component is shared between the sites. When a user logs in, the client connects to an *Access Point* (AP) using a proprietary protocol. Through the AP, the client can access the backend services including storage. The client maintains a long-lived TCP connection to the AP, and requests to backend services are multiplexed over this connection.

Spotify's storage is two-tiered. A client request for an object goes to *Production Storage*, a collection of servers that can serve most requests. The protocol between the AP and Production Storage is HTTP, and in fact, the Production Storage servers run software based on the caching Nginx HTTP proxy [2]. The objects are distributed over the production service machines using consistent hashing of their respective keys [3]. Each object is replicated on three different servers, one of which is identified as the primary server for the object. APs route a request for an object to its primary server. If the primary server does not store the requested object, the server will request it from one of the replicas. If they do not store it, the request will be forwarded over the Internet to *Master Storage* (which is based upon a third-party storage service) and the retrieved object will subsequently be cached in Production Storage. When the primary server of an object fails, an AP routes a request to one of the replicas instead.

While the presentation here centers on music delivery, we remark that the storage system delivers additional
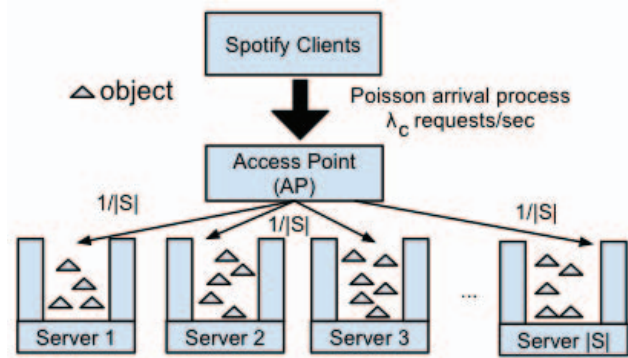
data to Spotify clients, in particular images (e.g., cover art for albums) and advertisements. We also point out that the number of requests that a client makes to the backend storage for each track played varies significantly, due to local caching and peer-to-peer functionality. As a consequence, the request rates presented in this paper do not correspond to the actual number of tracks played in the Spotify system.

III. AN ANALYTICAL MODEL OF THE STORAGE ARCHITECTURE

In order to make a performance analysis feasible, we develop a simplified model of the Spotify storage architecture (Figure 1) for a single Spotify backend site, the result of which is shown in Figure 2. First, we omit Master Storage in the simplified model and thus assume that all objects are stored in Production Storage servers, since more than 99% of the requests to the Spotify storage system are served from the Production Storage servers. Second, we model the functionality of all APs of a site as a single component. We assume that the AP selects a storage server uniformly at random to forward an incoming request, which approximates the statistical behavior of the system under Spotify object allocation and routing policies. Further, we neglect network delays between APs and storage servers, because they are small compared to the response times at the storage servers. In the following, we analyze the performance of the model in Figure 2 under steady-state conditions and Poisson arrivals of requests.

A. Modeling a single storage server

In the context of a storage system, the critical resources of a storage server are memory and disk access, which is captured in Figure 3. When a request arrives at a server, it is served from memory with probability q and from one of the disks with probability $1 - q$. Assuming that the server has n_d identical disks, the request is served from a specific disk with probability $1/n_d$. We further assume that requests arrive at the server following a Poisson process with rate λ. (All rates in this paper are measured in requests/sec.) We model access to memory or a disk as an $M/M/1$ queue. We denote the service
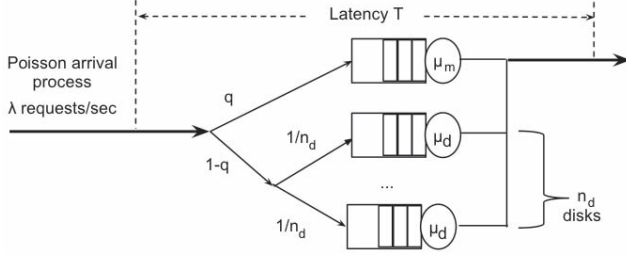
Fig. 3.   Queuing model of critical resources on a server

rate of the memory by μ_m and that of the disk by μ_d, whereby $\mu_m \gg \mu_d$ holds.

Based on Figure 3, we compute the latency T for a request on a server, which we also refer to as response time. (In queuing systems, the term *sojourn time* is generally used for T.) The probability that a request is served below a specific time t is given by $Pr(T \leq t) = qPr(T_m \leq t) + (1-q)Pr(T_d \leq t)$, whereby T_m and T_d are random variables representing the latency of the request being served from memory or a disk, respectively. For an $M/M/1$ queue in steady state with arrival rate λ, service rate μ, and latency T, the formula $Pr(T \leq t) = 1 - e^{-\mu(1-\lambda/\mu)t}$ holds [4]. Therefore, we can write

$$
\begin{aligned}
Pr(T \leq t) \quad = \quad & q(1 - e^{-\mu_m(1-\lambda_m/\mu_m)t}) \\
& + (1-q)(1 - e^{-\mu_d(1-\lambda_d/\mu_d)t}),
\end{aligned}
$$

whereby λ_m is the arrival rate to the memory queue and λ_d to a disk queue. Typical values in our experiments are $t \geq 10^{-3}$ sec, $\lambda_m \leq 10^3$ requests/sec, and $\mu_m \geq 10^5$ requests/sec. We therefore approximate $e^{-\mu_m(1-\lambda_m/\mu_m)t}$ with 0. Further, since $\lambda_m = q\lambda$ and $\lambda_d = (1-q)\lambda/n_d$ hold, the probability that a request is served under a particular latency t is given by

$$
Pr(T \leq t) = q + (1-q)(1 - e^{-\mu_d(1-(1-q)\lambda/\mu_d n_d)t}). \quad (1)
$$

B. Modeling a storage cluster

We model a *storage cluster* as an AP and a set S of storage servers, as shown in Figure 2. The load to the cluster is modeled as a Poisson process with rate λ_c. When a request arrives at the cluster, it is forwarded uniformly at random to one of the storage servers. Let T_c be a random variable representing the latency of a request for the cluster. We get that $Pr(T_c \leq t) = \sum_{s \in S} \frac{1}{|S|} Pr(T_s \leq t)$, whereby T_s is a random variable representing the latency of a request for a storage server $s \in S$.

For a particular server $s \in S$, let $\mu_{d,s}$ be the service rate of a disk, $n_{d,s}$ the number of identical disks, and q_s the probability that the request is served from memory. Let $f(t, n_d, \mu_d, \lambda, q) := Pr(T \leq t)$ defined in equation 1. Then, the probability that a request to the cluster is served under a particular latency t is given by

$$
Pr(T_c \leq t) = \frac{1}{|S|} \sum_{s \in S} f(t, n_{d,s}, \mu_{d,s}, \frac{\lambda_c}{|S|}, q_s). \quad (2)
$$

The above model does not explicitly account for the replications of objects. As explained in Section II, the Spotify storage system replicates each object three times. The AP routes a request to the primary server of the object. A different server is only contacted, if the primary server either fails or does not store the object. Since both probabilities are small, we consider in our model only the primary server for an object.

IV.   EVALUATION OF THE MODEL ON THE LAB TESTBED AND THE SPOTIFY OPERATIONAL ENVIRONMENT

A. Evaluation on the lab testbed

The first part of the evaluation is performed on our the KTH lab testbed, which comprises some 60 rack-based servers interconnected by Ethernet switches. We use two types of servers, which we also refer to as small servers and large servers (Table I).

1) Single storage server: In this series of experiments, we measure the response times of a single server as a function of the request rate, and we estimate the model parameters. This allows us to validate our analytical model of a single server, as expressed in equation 1. The requests are generated using a Spotify request trace, and they follow a Poisson arrival process. A request retrieves an object, which has an average size of about 80 KB.

The setup consists of two physical servers, a load injector and a storage server, as shown in Figure 4 (top). Both servers have identical hardware; they are either small or large servers. All servers run Ubuntu 10.04 LTS. The load injector has installed a customized version of HTTPerf [5]. It takes as input the Spotify trace, generates a stream of HTTP requests, and measures the response times. The storage server runs Nginx [2], an open-source HTTP server.

Before conducting an experiment, we populate the disk of the storage server with objects. A small server is populated with about 280K objects (using 22GB), a large one with about 4,000K objects (using 350GB). Each run of an experiment includes a warm-up phase, followed by a measurement phase. During warm up, the memory is populated with the objective to achieve the same cache hit ratio as the server would achieve in steady state, i.e., after a long period of operation. The memory of the small server holds some 12K objects, the memory of the large server some 750K objects. We use a Spotify request trace for the warmup, with 30K requests for the small server and 1,000K requests for the large server. The measurement runs are performed with a different Spotify trace than the warmup runs. A measurement run includes 30K requests for the small server and 100K requests for the large server.

We perform a series of runs. The runs start at a request rate of 10 and end at 70, with increments of 5, for the small server; they start at a rate of 60 and end at 140,
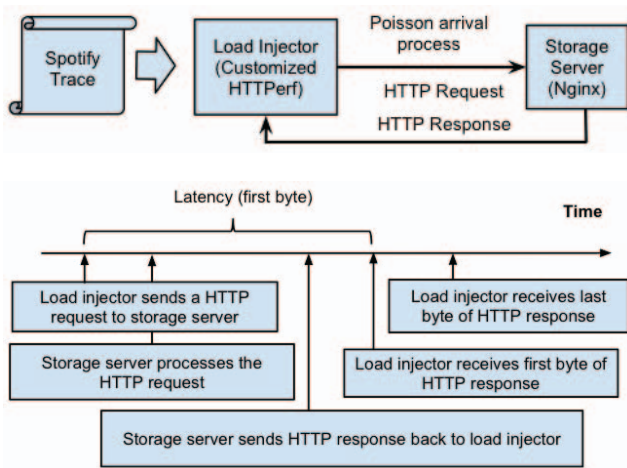
Fig. 4. Setup and measurement of request latency for a single server.

with increments of 20, for the large server. The response time for a request is measured as indicated in Figure 4 (bottom). Figures 6a and 6b show the measurement results for three selected latencies for the small and large server, respectively. The vertical axis gives the fraction of requests that have been served within a particular latency. The horizontal axis gives the rate of the request arrival process. All measurement points in a figure that correspond to the same request rate result from a single run of an experiment.

The figures further include the model predictions in forms of solid lines. These predictions come from the evaluation of equation 1 and an estimation of the model parameters/confidence limits, which will be discussed in Section IV-C.

We make three observations regarding the measurement results. First, the fraction of requests that can be served under a given time decreases as the load increases. For small delays, the relationship is almost linear. Second, the model predictions agree well with the measurements below the confidence limit. In fact, measurements and models diverge at most 11%. A third observation can not be made from the figures but from the measurement data. As expected, the variance of the response times is small for low requst rates and becomes larger with increasing rate. For instance, for the small server, we did not measure any response times above 200 msec under a rate of 30; however, at the rate of 70, we measured several response times above 1 sec.

2) A cluster of storage servers: In this series of experiments, we measure the response times of a cluster of storage servers as a function of the request rate, and we estimate the model parameters. This allows us to validate our analytical model of a cluster as expressed in equation 2. Similar to the single-server experiments, the requests are generated using a Spotify request trace, and they follow a Poisson arrival process.

We perform experiments for two clusters: one cluster with small servers (one load injector, one AP, and five storage servers) and one with large servers (one load

Small server	Specification
Model	Dell Power edge 750 1U server
RAM	1GB
CPU	Intel(R) Pentium(R) 4 CPU 2.80GHz
Harddisk	Single disk 40GB
Network Controller	Intel 82547GI Gigabit Ethernet Controller

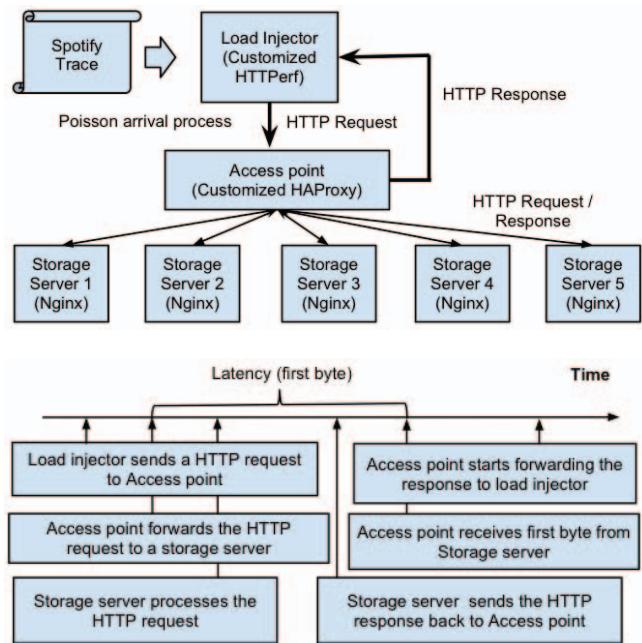| Large server	Specification
Model	Dell PowerEdge R715 2U Rack Server
RAM	64GB
CPU	two 12-core AMD Opteron(tm) processors
Harddisk	Single disk - 500GB
Network Controller	Broadcom 5709C Gigabit NICs

TABLE I
SERVERS ON THE KTH TESTBED

Fig. 5. Setup and measurement of request latency for clusters.

injector, one AP, and three storage servers). The testbed setup can be seen in Figure 5 (top). The software setup of the load injector and storage servers have been discussed above. The AP runs a customized version of HAProxy [6] that forwards a request to a storage server and returns the response to the load injector.

Before conducting an experiment, we populate the disks of the storage servers with objects: we allocate each object uniformly at random to one of the servers. We then create an allocation table for request routing that is placed in the AP. This setup leads to a system whose statistical behavior closely approximates that of the Spotify storage system. During the warmup phase for each run, we populate the memory of all storage servers in the same way as discussed above for a single server. After the warmup phase, the measurement run is performed. Driven by a Spotify trace, the load injector sends a request stream to the AP. Receiving a request from the load injector, the AP forwards it to a storage server according to the allocation table. The storage server processes the request and sends a response to

(a) Single small server

(b) Single large server

(c) Cluster of small servers

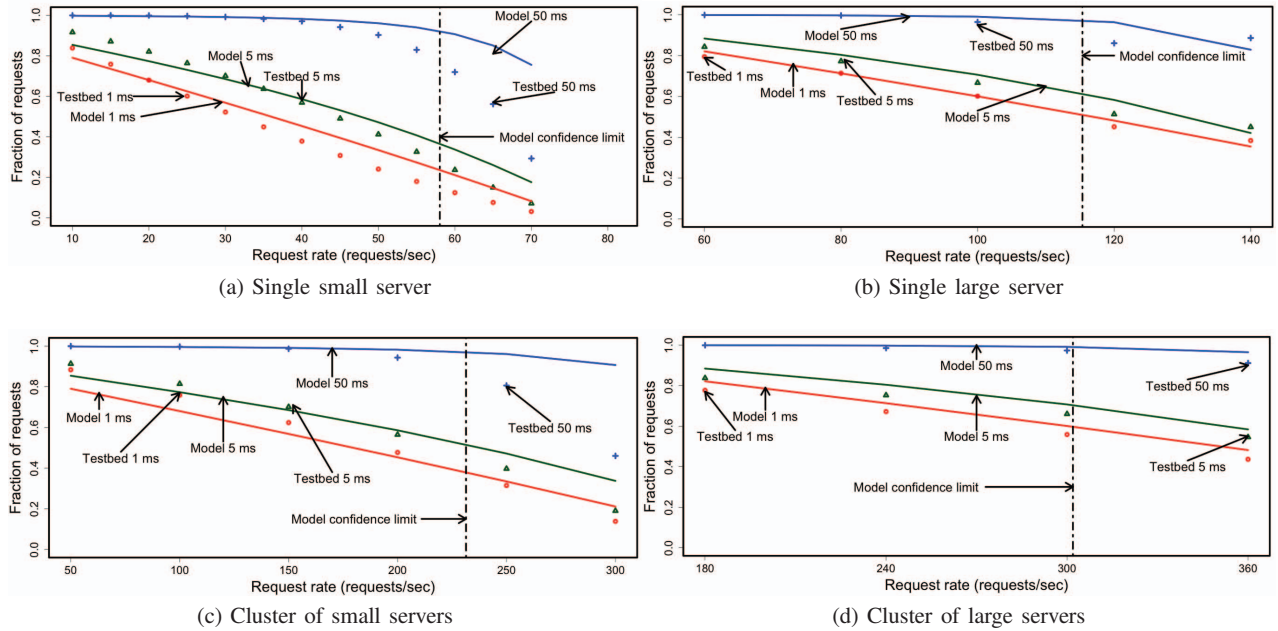(d) Cluster of large servers

Fig. 6. Lab testbed measurements and model predictions

the AP, which forwards it to the load injector. The response time of each request is measured as shown in Figure 5 (bottom). Regarding dimensioning, the number of allocated objects per server is similar to the one in the experiments discussed above involving single servers. The same is true regarding the number of objects cached in memory, the number of requests for a warmup run, and the number of requests for a measurement run.

For each experimental run, a request stream is generated at a certain rate, and, for each request, the response time is measured. The runs start at a request rate of 50 and end at 300, with increments of 50, for the cluster of small servers; they start at a rate of 180 and end at 360, with increments of 60, for the cluster of large servers. Figures 6c and 6d show the measurement results for three selected latencies for the cluster of small and large servers, respectively. The figures further include the model predictions in form of solid lines. The predictions are obtained from equation 2 and model parameters, discussed in Section IV-C. Our conclusions from the experiments on the two clusters are similar to those on the single servers: the fraction of requests that can be served under a given time decreases as the load increases. The relationship is almost linear; the slopes of the curves decrease slightly with increasing request rate. Further, the measurements and models diverge at most 9.3% below the confidence limit.

B. Evaluation on the Spotify operational environment

For this evaluation, we had access to hardware and direct, anonymized measurements from the Spotify operational environment. The single server evaluation has been performed on a Spotify storage server, and the

cluster evaluation has been performed with measurement data from the Stockholm backend site.

1) Single storage server: We benchmark an operational Spotify server with the same method as discussed in Section IV-A1. Such a server stores about 7.5M objects (using 600GB), and a cache after the warmup phase contains about 375K objects (using 30GB). (The actual capacity of the Spotify server is significantly larger. We only populate 600GB of space, since the traces for our experiment contains requests for objects with a total size of 600GB.) For a run of the experiment, 1000K requests are processed during the warm-up phase, and 300K requests during the measurement phase. The runs start at a request rate of 100 and end at 1,100, with increments of 100. Figure 7a shows the measurement results for three selected latencies for the Spotify operational server.

The qualitative observations we made for the two servers on the KTH testbed (Section IV-A1) hold also for the measurements from the Spotify server. Specifically, the measurements and model predictions diverge at most 8.45%, for request rates lower than the model confidence limit.

2) Spotify storage system: For the evaluation, we use 24 hours of anonymized monitoring data from the Stockholm site. This site has 31 operational storage servers. The monitoring data includes, for each storage server, measurements of the arrival rate and response time distribution for the requests that have been sent by the APs. The measurement values are five-minutes averages. The data includes also measurements from requests that have been forwarded to the Master Storage, but as stated in Section III, such requests are rare, below

1% of all requests sent to the storage servers.

Some of the servers at the Stockholm site have a slightly different configuration from the one discussed above. These differences have been taken in account for the estimation of model parameters. Figure 7b presents the measurement results in the same form as those we obtained from the KTH testbed. It allows us to compare the performance of the storage system with predictions from the analytical model. Specifically, it shows measurement results and model predictions for three selected latencies, starting at a request rate of 1,000 and ending at 12,000, with increments of 1,000. The confidence limit is outside the measurement interval, which means that we have confidence that our analytical model is applicable within the complete range of available measurements.

We make two observations. First, similar to the evaluations we performed on the KTH testbed, the measurements and the model predictions diverge at most 9.61%. This is somewhat surprising, since this operational environment is much more complex and less controllable for us than the lab testbed. For instance, for our testbed measurements, (1) we generate requests with Poisson arrival characteristics, which only approximates arrivals in the operational system; (2) on the testbed we use identical servers, while the production system has some variations in the server configuration; (3) the testbed configurations do not consider Master Storage, etc.

Furthermore, the measurements suggested that the fraction of requests under a specific latency stays almost constant within the range of request rates measured. In fact, our model predicts that, the fraction of requests served within 50 msec stays almost constant until the confidence limit, at about 22,000 requests/sec. Therefore, we expect that this site can handle a much higher load than observed during our measurement period, without experiencing a significant decrease in performance when considering the 50 msec response-time limit. A response time of up to 50 msec provides the user experience that a selected track starts "instantly".

C. Estimating model parameters / confidence limit

We determine the model parameters for the single server, given in equation 1, namely, the service rate of a disk μ_d, the number of identical disks n_d, and the probability that a request is served from memory q. While n_d can be read out from the system configuration, the other two parameters are obtained through benchmarking. We first estimate the average service time T_s of the single disk through running iostat [7] while the server is in operation (i.e. after the warm-up phase), and we obtain $\mu_d = 1/T_s$. We estimate parameter q as a fraction of requests that have a latency below 1 msec while the server is in operation. Figure 8 shows the measured values for q, for different server types and request rates. We observe a significant difference in parameter q between the testbed servers(small and large server) and Spotify operational server. We believe that

Parameter	Small server	Large server	Spotify server
μ_d	93	120	150
n_d	1	1	6
α	0.0137	0.00580	0.000501
q_0	0.946	1.15	0.815

TABLE II
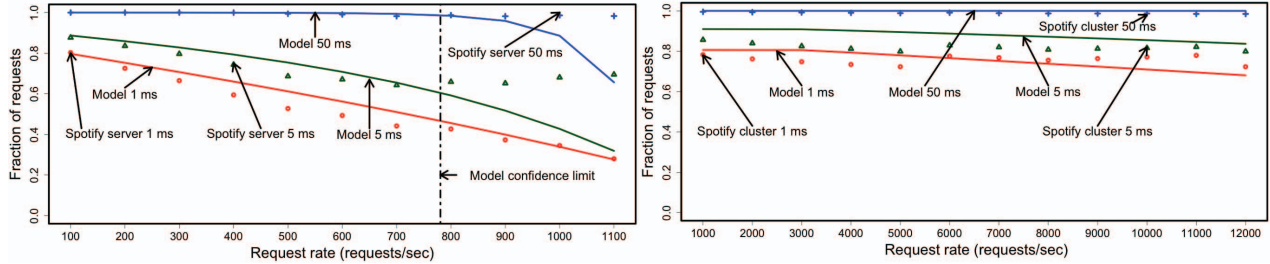MODEL PARAMETERS FOR A SINGLE STORAGE SERVER

this is because software and hardware of the operational server is highly optimized for serving Spotify traffic while the testbed servers are general-purpose servers and configured with default options.

Based on the results in Figure 8 and other measurements, we approximate, through least-square regression, q with the linear function $q = -\alpha\lambda + q_0$, whereby λ is the request rate. All model parameters of a single storage server are summarized in Table II.

We now compute the *model confidence limit for the single server*, i.e., the maximum request rate below which we feel confident that our model (i.e., equation 1) applies. Through extensive testing, we found that our model predictions are close to the measurements from the real system, as long as the average length of any disk queue is at most one. From the behavior of an M/M/1 queue, we know that the average queue length for one of the disks is $L_d = \frac{\lambda_d}{\mu_d - \lambda_d}$. Applying the linear approximation for q and setting $L_d = 1$, simple manipulations give the model confidence limit λ_L for a single server as the positive root of $\alpha\lambda_L^2 + (1 - q_0)\lambda_L - \frac{1}{2}\mu_d n_d = 0$. The confidence limits in Figures 6a, 6b, and 7a are computed using this method. As can be observed, increasing request rates beyond the confidence limits coincides with a growing gap between model predictions and measurements, specifically for the latency of 50 msec, which is an important value for the Spotify storage system.

The model parameters for a cluster, appearing in equation 2, contain the model parameters of each server in the cluster. Therefore, if the model parameters for each server are known, then the parameters for the cluster can be obtained.

We now discuss the *model confidence limit for the cluster*, i.e., the maximum request rate to the cluster below which we have confidence that the model predictions are close to actual measurements, under the assumption that we know the confidence limit for each server. The allocation of objects to primary servers in the Spotify storage system can be approximated by a process whereby each object is placed on a server uniformly at random, weighted by the storage capacity of the server. Therefore, the number of objects allocated to servers can vary, even for a cluster with homogeneous servers. The distribution of the number of objects on servers can be modeled using the balls-and-bins model [8]. If the server contains a large number of objects, as in our system, the expected load on the server is proportional to the number of objects. To compute the confidence limit for the cluster, we must know the load of the highest

(a) Single Spotify Storage server

(b) Cluster of Spotify Storage servers

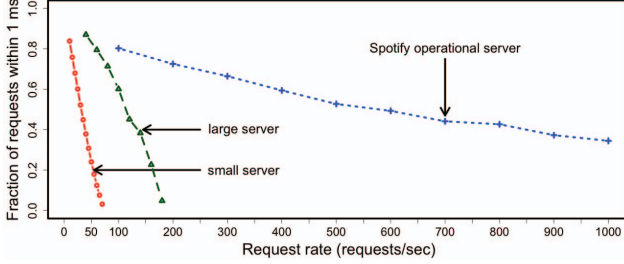Fig. 7. Spotify operational environment measurements and model predictions

Fig. 8. Estimating the parameter q

loaded server. A result from the analysis of the balls-and-bins model states that, when m balls are thrown independently and uniformly at random into n bins and $m \gg n \cdot (\log n)^3$ can be assumed, then there is no bin having more than $M = m/n + \sqrt{\frac{2m \log n}{n}(1 - \frac{1}{\beta}\frac{\log \log n}{2 \log n})}$ balls with high probability, for any $\beta > 1$ [9]. We apply this result by interpreting balls as request rates and bins as servers. By doing so, we obtain the confidence limit $\lambda_{L,c}$ of the cluster as a function of the minimum λ_L of the confidence limits of all servers and the number of servers $|S|$. We can conclude that the confidence limit for the cluster is the smaller root of $\frac{1}{|S|^2}\lambda_{L,c}^2 + (\frac{2\lambda_L}{|S|} - \frac{2 \log |S| K_{\beta,|S|}}{|S|})\lambda_{L,c} + \lambda_L^2 = 0$, whereby $\beta = 2$ and $K_{\beta,|S|} = 1 - \frac{1}{\beta}\frac{\log \log |S|}{2 \log |S|}$. The confidence limits in Figures 6c and 6d are computed using this method. Similar to the case of the single server, the model predictions can diverge significantly from the measurements for rates beyond the confidence limits.

V. APPLICATIONS OF THE MODEL

We apply the analytical model to predict, for the Spotify storage system at the Stockholm site, the fraction of requests served under given latencies for a load of 12,000 requests/sec, which is the peak load from the dataset we used. While our evaluation has involved 31 servers, we use the model to estimate response time for configurations from 12 to 52 storage servers. The result is shown in Figure 9a. The confidence limit is 17 servers. Above this number, we have confidence that the model applies. We observe that for each latency curve in the figure the slope decreases with increasing number of servers. This means that adding additional servers to

the storage system of, say, 20 servers result in a much larger reduction of response time than adding servers to the storage system of, say, 50 servers.

Second, we predict the fraction of requests served under specific response times for a storage system with 25 servers. We consider a scenario where the load varies from 1,000 to 20,000 requests/sec. The result is shown in Figure 9b. The confidence limit is 18,000 below which our model applies. We observe that the slope of all curves in the figure is almost zero between 1,000 to 3,000 requests/sec, beyond which it starts decreasing. We can predict that increasing the load on the storage system from 1,000 to 3,000 requests/sec does not have any measurable impact on performance, while we expect that an increase from 1,000 to 15,000 requests/sec clearly will. Our model also predicts that, for a response time limit of 50 msec, the fraction of requests remains almost unchanged for rates between 1,000 and 18,000 requests/sec.

VI. RELATED WORK

Substantial research has been undertaken in modeling the performance of storage devices (see, e.g., [10]–[12]). Our work differs from these, since our modeling work is on the systems level and thus captures aspects of an entire system. Several works present performance models of storage systems [13]–[15]. However, to the best of our knowledge, none of them discusses and validates models for predicting the latency *distribution* of requests to a real storage system. The authors in [13] present a performance model for Ursa Minor [16], a robust distributed storage system. Their model allows them to predict the average latency of a request, as well as the capacity of the system. A second performance model of a storage system is presented in [14]. In this work, expected latency and throughput metrics can be predicted for different allocation schemes of virtual disks to physical storage devices. The authors of [15] discuss in that paper a performance model for predicting the average response time of an IO request when multiple virtual machines are consolidated on a single server.

The development and evaluation of distributed key-value stores has been an active research area. While
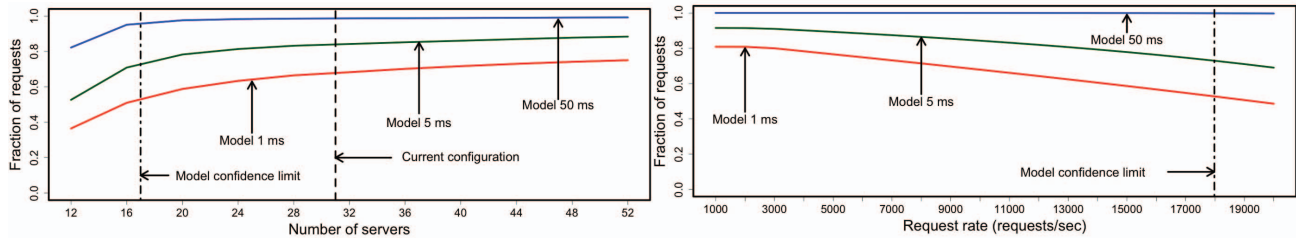
(a) Varying the number of servers in the storage system for a load of 12,000 requests/sec   (b) Varying the load in the storage system for 25 storage servers

Fig. 9.   Applications of the model to system dimensioning

these systems generally provide more functionality than the Spotify storage system, to our knowledge, no performance model has yet been developed for any of them. In contrast to Spotify's storage system design, which is hierarchical, many advanced key-value storage systems in operation today are based on a peer-to-peer architecture. Among them are Amazon Dynamo [17], Cassandra [18], and Scalaris [19]. Facebook's Haystack storage system follows a different design which is closer to Spotify's. Most of these systems use some forms of consistent hashing to allocate objects to servers. The differences in the designs of the systems are motivated by their respective operational requirements, and they relate to the number of objects to be hosted, the size of the objects, the rate of update, the number of clients, and the expected scaling of the load.

VII. Discussion

We make the following contributions with this paper. First, we introduce an architectural model of a distributed key-value store that simplifies the architecture and functionality of the Spotify storage system. We then present a queuing model that allows us to compute the response time distribution of the storage system. Further, we estimate the confidence range for this model, under the assumption that the system is lightly loaded. Second, we perform an extensive evaluation of the model, first on our testbed and later on the Spotify operational infrastructure. This evaluation shows that the model predictions are accurate, with the error of at most 11%.

The reported errors result from the fact that we use a simple model to describe, on an abstract level, the behavior of a complex distributed system. This simplicity is a virtue insofar as it allows us to predict metrics like the response time distribution without much mathematical and computational effort. The downside is that applicability of the model is restricted to a lightly loaded system; however, this corresponds to the operational range of the Spotify storage system (see further discussion below). To increase the accuracy of the model, or to extend the range of load patterns for which it can make predictions, one needs to refine the system model. Such refinements can include, modeling

the specific software process structure in a server, the OS scheduling policies, heterogeneous hardware, the detailed request routing policy in the cluster, as well as the real arrival process and service discipline of the queuing model. The difficulty will be to identify refinements that significantly increase the accuracy of the model while keeping it simple and tractable.

As it turns out, the confidence range of our model covers the entire operational range of the load to the Spotify storage system. As we have validated through experimentation, the performance of the system deteriorates when the load significantly surpass the model-predicted confidence limit. Lastly, applying our model, we predict for a specific Spotify backend site that the system could handle the peak load observed during a specific day with fewer servers, or, alternatively, that the system with 25 servers could handle a significantly higher load than observed, without noticable performance degradation (for important response time limit, which is 50 msec).

This work is important to Spotify, since latency is the key performance metric of its storage system. The main reason for this is that estimating latency distributions is essential to guarantee the quality of the overall service. Note that recent performance studies on storage systems cover only expected latencies, which is not sufficient for our case. The validity of the results in this paper goes beyond the scope of Spotify's technology. In fact, our approach can be applied to similar types of distributed key-value stores and other services that rely on them, such as video streaming.

As for future work, we plan to develop a subsystem that continuously estimates the model parameters at runtime, taking into account that the resources of the storage servers may be used by other processes than object retrieval, for instance, for maintainence or system reconfiguration. Based on such a capability, we envision an online performance management system for a distributed key-value store like the Spotify storage system.

REFERENCES

[1] G. Kreitz and F. Niemelä, "Spotify – large scale, low latency, P2P music-on-demand streaming," in *Peer-to-Peer Computing*. IEEE, 2010, pp. 1–10.

[2] I. Sysoev, "Nginx," http://nginx.org/.

[3] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *STOC*, F. T. Leighton and P. W. Shor, Eds. ACM, 1997, pp. 654–663.

[4] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[5] D. Mosberger and T. Jin, "httperf - a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, Dec. 1998.

[6] W. Tarreau, "Haproxy," http://haproxy.1wt.eu/.

[7] S. Godard, "iostat," http://linux.die.net/man/1/iostat.

[8] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, Jan. 2005.

[9] M. Raab and A. Steger, ""Balls into Bins" - A Simple and Tight Analysis," in *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, ser. RANDOM '98. London, UK: Springer-Verlag, 1998, pp. 159–170.

[10] J. Garcia, L. Prada, J. Fernandez, A. Nunez, and J. Carretero, "Using black-box modeling techniques for modern disk drives service time simulation," in *Simulation Symposium, 2008. ANSS 2008. 41st Annual*, april 2008, pp. 139 –145.

[11] A. Lebrecht, N. Dingle, and W. Knottenbelt, "A performance model of zoned disk drives with i/o request reordering," in *Quantitative Evaluation of Systems, 2009. QEST '09. Sixth International Conference on the*, sept. 2009, pp. 97 –106.

[12] F. Cady, Y. Zhuang, and M. Harchol-Balter, "A Stochastic Analysis of Hard Disk Drives," *International Journal of Stochastic Analysis*, vol. 2011, pp. 1–21, 2011.

[13] E. Thereska, M. Abd-El-Malek, J. Wylie, D. Narayanan, and G. Ganger, "Informed data distribution selection in a self-predicting storage system," in *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, june 2006, pp. 187 – 198.

[14] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, and M. Uysal, "Pesto: online storage performance management in virtualized datacenters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 19:1–19:14.

[15] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick, "Io performance prediction in consolidated virtualized environments," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 295–306, Sep. 2011.

[16] M. Abd-El-Malek, W. V. Courtright, II, C. Cranor, G. R. Ganger, J. Hendricks, A. J. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. R. Sambasivan, S. Sinnamohideen, J. D. Strunk, E. Thereska, M. Wachs, and J. J. Wylie, "Ursa minor: versatile cluster-based storage," in *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies - Volume 4*, ser. FAST'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 5–5.

[17] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[18] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[19] T. Schütt, F. Schintke, and A. Reinefeld, "Scalaris: reliable transactional p2p key/value store," in *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, ser. ERLANG '08. New York, NY, USA: ACM, 2008, pp. 41–48.