Quota: Rationing Server Resources in Peer-Assisted Online Hosting Systems

Fangming Liu[†], Ye Sun[†], Bo Li[*†], Baochun Li[‡]
[†]*Hong Kong University of Science & Technology*, [‡]*University of Toronto*
{lfxad, yesun, bli}@cse.ust.hk, bli@eecg.toronto.edu

***Abstract*—Online hosting systems are designed to provide versatile and convenient platforms for content hosting and sharing, and have rapidly become a favorite among users over the Internet. To guarantee adequate levels of service quality while conserving prohibitive server costs, such systems are designed to integrate peer bandwidth contributions with strategic server resource provisioning in a complementary and transparent manner.**

Due to the large number of users in real-world online hosting systems, it is not feasible to satisfy the resource requirements of all users. This paper seeks to explore the design space of new protocols to allocate scarce server resources—including both storage space and bandwidth—in peer-assisted online hosting systems. The focus is on the problem of resource allocation with the presence of an increasingly large number of users using bandwidth, and the ensuing larger number of files using server storage space. The objective is to maximize the use of limited server storage and bandwidth resources to guarantee adequate levels of service quality, with respect to file availability and downloading performance, while taking full advantage of peer assistance. We identify a number of unique challenges involved in such systems, and propose our design of resource allocation protocols to address these challenges, based on both mathematical analysis and practical implementations. Using real-world data sets that we have collected, we evaluate our protocol design through extensive experimental studies from different perspectives, which demonstrate the effectiveness of our design and offer a number of practical guidelines.

I. Introduction

Online hosting systems allow users to upload files of any size using a web-based interface, and has gained remarkable popularity over the Internet. In more traditional online hosting systems, such as *RapidShare*, files are uploaded to dedicated servers maintained by the service provider, to be shared among a group of interested users. These services simply return a URL that can be shared to other users (*e.g.*, in discussion forums), who can then download the file at a later time. Due to the simplicity and versatility of its user interface, this type of file sharing has rapidly become a favorite among users, overtaking peer-to-peer (P2P) file sharing services of the previous generation, such as BitTorrent.

The architectural and protocol design of online hosting systems needs to balance two extremes of the cost-performance tradeoff. On one end of the spectrum, server-based solutions such as content distribution networks (CDNs) can provide better reliability and service quality, yet they suffer from prohibitive costs of server bandwidth and storage. *Rapidshare*, one of the most well-known online hosting systems, deployed a total of 1500 terabytes of online storage in its data centers in Asia alone. Reportedly, their annual bandwidth cost hovers around \$15M – \$20M. On the other end of the spectrum, P2P file sharing protocols, such as BitTorrent, have shown good scalability and robustness; however, they have no guarantees on file availability, and the downloading performance can become unsatisfactory, when files are not actively served by peers alone.

To achieve a favorable spot of tradeoff between the two extremes, online hosting systems have evolved towards a seamless integration of peer assistance and server bandwidth provisioning, in a complementary and transparent manner. In such peer-assisted online hosting systems, a large number of users will be sharing an even larger number of files, resulting in an exorbitant volume of bandwidth being used at any given time. In such a scenario, server resources—including both storage space and bandwidth—are scarce and provided at a premium. The critical challenge, which so far has not been addressed in any other papers in the literature, is how such scarce resources are to be allocated to the end users and their files.

In this paper, we present *Quota*, our design and analysis of a series of configurable server resource allocation protocols to be practically used in peer-assisted online hosting systems. In response to a number of unique challenges involved in such systems, the design of *Quota* attempts to make the best use of scarce server storage and bandwidth resources, so that adequate levels of service quality can be guaranteed. The quality of online hosting services is evaluated in term of file availability and downloading performance. Rather than a specific protocol, *Quota* seeks to explore the entire spectrum of the design space, and proposes a framework with tunable design knobs to offer flexibility to adapt to a wide range of protocol design preferences. To evaluate the effectiveness of *Quota*, we carry out extensive analytical and experimental studies from various perspectives, based on real-world data sets that we have collected from a peer-assisted online hosting system operating live at a large scale.

The remainder of this paper is organized as follows. In Sec. II, we discuss our contributions in the context of related work. In Sec. III, we present the general architecture of peer-assisted online hosting systems, along with a number of unique challenges. Sec. IV presents our design and analysis of *Quota*,

[*]The research was support in part by grants from RGC under the contracts 615608, and 616207, by a grant from NSFC/RGC under the contract N_HKUST603/07, by a grant from HKUST under the contract RPC06/07.EG27.

including both storage and bandwidth allocation strategies. Sec. V presents our extensive experimental studies to examine various aspects of our design. Finally, we conclude the paper in Sec. VI.

II. Related Work

Traditional peer-to-peer (P2P) file sharing systems have received significant research attention in the literature. Yang and Veciana [1] use a branching process model to study the service capacity of BitTorrent-like file sharing systems in the transient regime, as well as a Markovian model for the steady-state analysis. To overcome the computation problem in [1], Qiu and Srikant [2] develop a deterministic fluid model to obtain simple expressions for the average file download time in the steady-state. B. Fan *et al.* [3] have analytically characterized the tradeoff between performance and fairness in BitTorrent-like protocols and derived relevant design implications. While recognizing the significance of these prior efforts as well as other modeling works on pure P2P environments for file sharing (*e.g.*, [4]–[6]), our work differs substantially from them, as there exist important differences between P2P file sharing and peer-assisted online hosting systems.

P2P file sharing systems do not use servers to store actual file content, as all files are exchanged among users. As a result, they have no guarantees on file availability, and files being downloaded may become unavailable at any time when all "seeds" (peers with a complete copy of the file) leave the system. For example, by examining peer arrival rates and downloading performance with traces from popular trackers of BitTorrent, Guo *et al.* [7] have shown that file availability in BitTorrent could deteriorate quickly due to the exponentially decreasing peer arrival rate and the lack of seeds. Tian *et al.* [8] observed through an analytical model that, the tit-for-tat (TFT) mechanism used in BitTorrent cannot improve file availability or prevent system-wide failures caused by the sudden departure of a completed peer.

In contrast, peer-assisted online hosting systems couple peer assistance and strategic server provisioning in a complementary manner, in order to improve file availability and downloading performance. Along such a direction, a relevant work [9] has extended the model in [2] to discuss the motivation and effects of server participation in BitTorrent-like P2P file sharing systems. However, their service models, as well as most existing studies, are still restricted to the case of *a single file*. Online hosting systems, in sharp contrast, concurrently serve a large number of users, and a even larger number of files uploaded by these users. More recently, based on real-world traces, Wu *et al.* [10] have proposed an online server capacity provisioning algorithm for multi-channel live P2P streaming systems. In contrast, *Quota* is customized for peer-assisted online hosting systems, with its own application features and performance concerns.

There have been no prior work that devotes attention to the protocol design space on dedicated online hosting servers to allocate its premium and scarce resources among a large number of files. *Quota* proposes a unified and general framework to explore the full spectrum of the protocol design space in peer-assisted online hosting systems.

III. Online Hosting: Architecture

In general, the architecture of peer-assisted online hosting systems represents a seamless integration of the following two fundamental components.

Peers. Each end user is treated as a peer, inheriting the terminology of pure P2P systems. There are two types of peers: those who upload content (files) to hosting servers (referred to as *uploading peers*), and those who download only (referred to as *downloading peers*). Group of peers participating in the distribution of each file cooperatively serve one another depending on their available bandwidth and content. Analogous to existing P2P systems such as BitTorrent, this can be achieved through a set of P2P mechanisms, including peer topology construction, data delivery, incentives, *etc*. Without loss of generality, we encapsulate these mechanisms as *peer assistance*, as they collectively seek to take advantage of peer bandwidth contributions to mitigate server bandwidth costs.

Hosting servers. Servers are maintained by the service provider, not only provide online storage for a huge volume of files uploaded by peers, but also cooperate with content distribution to maintain service quality of files when they are not actively served by peers alone. In order to improve file availability and downloading performance while conserving prohibitive bandwidth and storage costs on servers, there are two fundamental challenges associated with the server protocol design:

- ▷ Very different from BitTorrent, the number of files, of both large and small sizes, being concurrently shared in peer-assisted online hosting systems is very large. *How can the service provider make better use of a limited pool of server storage to selectively store contents that are as valuable to users as possible?*
- ▷ As such a large number of shared files are being served with highly diverse popularity, *how can the service provider make better use of a limited amount of server bandwidth to satisfy a potentially large number of requests across these files?*

IV. *Quota:* Design and Analysis

In response to the above challenges, in this section we propose our design of *Quota*, a framework for server resource allocation protocols in peer-assisted online hosting systems. Specifically, by mathematically analyzing the fundamental problems behind server protocol design, we explore two sets of server strategies in *Quota*: *server storage and replacement* and *server bandwidth allocation*, with both theoretical insights and practical guidelines for service providers and designers.

A. General Model and Performance Metrics

We first present our mathematical model based on [2], [9] for peer-assisted online hosting systems, which allows us to analyze different server strategies with respect to various performance metrics. Different from previous modeling works on pure P2P environments with single-file sharing, our model takes into account both multiple files of different popularity and sizes, and server involvement with scarce storage and bandwidth resources, in order to capture the essential aspects of practical peer-assisted online hosting systems.

Without loss of generality, suppose there are a total of M concurrent files to be shared in a peer-assisted online hosting system, represented as a set $\mathcal{M} = \{1, 2, \ldots, M\}$. For any file $i \in \mathcal{M}$, relevant notations and assumptions in our system model are summarized as follows:

$x_i(t)$: The number of peers who are downloading file i in the system at time t. $\overline{x_i}$ is the equilibrium value of $x_i(t)$.

$y_i(t)$: The number of peers who have finished downloading file i but have not yet left the system at time t. $\overline{y_i}$ is the equilibrium value of $y_i(t)$.

λ_i: The arrival rate of new peers in file i. We assume that peers arrive according to a Poisson process.

f_i: The size of file i. Given a limited total server storage capacity F provisioned by the service provider, the collection of files $\mathcal{F} \subseteq \mathcal{M}$ stored on the server shall satisfy $\sum_{i\in\mathcal{F}} f_i \leq F \leq \sum_{i\in\mathcal{M}} f_i$.

S_i: The server bandwidth assigned to file i. $S = \sum_{i\in\mathcal{F}} S_i$ is the total amount of server bandwidth provisioned by the service provider.

μ: The uploading bandwidth of a given peer. We assume that all peers have the same uploading bandwidth.

c: The downloading bandwidth of a given peer. We assume that all peers have the same downloading bandwidth and $c \geq \mu$.

θ_i: The rate at which peers abort the download of file i.

γ_i: The rate at which peers who have finished downloading file i leave the system.

η_i: The file sharing effectiveness of file i. According to [2], [9], η is defined as the fraction of the upload capacity of peers that is being utilized, with values in $[0, 1]$.

We are primarily interested in several important performance metrics that characterize "good" online hosting systems from different perspectives. *First*, with a limited pool of server storage for a potentially large number of files of diverse popularity and sizes, a service provider may intend to selectively store a collection of files $\mathcal{F}$ that can *attract and serve as many users as possible*. This can be simply represented by the system-wide peer arrival rate $\lambda = \sum_{i\in\mathcal{F}} \lambda_i$. Alternatively, this also reflects the file availability achieved by a certain design of server storage and replacement strategy. *Second*, a service provider may also intend to maintain *as high downloading performance as possible*, in order to improve the user experience. This is typically represented by the system-wide average downloading rate $d = \sum_{i\in\mathcal{F}} \overline{x_i} d_i / \sum_{i\in\mathcal{F}} \overline{x_i}$, where d_i denotes the average downloading rate of file i in steady state. Its correlation with the server bandwidth allocation strategy will be derived in Sec. IV-C. *Third*, combining the above two factors gives the system-wide throughput in term of the aggregate downloading rate $D = \sum_{i\in\mathcal{F}} \overline{x_i} d_i$.

B. Server Storage and Replacement Strategies

Essentially, given a constrained server storage capacity $F (\leq \sum_{i\in\mathcal{M}} f_i)$, a server storage and replacement strategy in *Quota* determines which set of files $\mathcal{F} \subseteq \mathcal{M}$ to be stored on the server, when presented with a large number of files $\mathcal{M}$ with diverse popularity and sizes. The problem can be formulated as a classical 0-1 *knapsack problem* with respect to different objective functions as follows.

To attract and serve the maximum number of users in term of the system-wide peer arrival rate λ, we need to solve the following optimization problem:

$$\begin{aligned} \text{Maximize} \quad & \lambda = \sum_{i\in\mathcal{M}} \lambda_i w_i \qquad (1) \\ \text{Subject to:} \quad & \sum_{i\in\mathcal{M}} f_i w_i \leq F, \\ & w_i \in \{0,1\}, i \in \mathcal{M}, \end{aligned}$$

where $w_i \in \{0, 1\}$ denotes whether a file $i \in \mathcal{M}$ is selected to be stored on the server (*i.e.*, the target set $\mathcal{F}$) or evicted from the server due to the storage capacity constraint.

On the other hand, substituting the aggregate downloading rate D for the objective function gives the following optimization problem, which aims to store those files that can achieve the maximum system-wide throughput. In particular, let T_i denotes the average downloading time of file i in steady state, then we have $\overline{x_i} d_i = \overline{x_i}(f_i/T_i) = \lambda_i f_i$ based on Little's Law.

$$\begin{aligned} \text{Maximize} \quad & D = \sum_{i\in\mathcal{M}} \overline{x_i} d_i w_i = \sum_{i\in\mathcal{M}} \lambda_i f_i w_i \qquad (2) \\ \text{Subject to:} \quad & \sum_{i\in\mathcal{M}} f_i w_i \leq F, \\ & w_i \in \{0,1\}, i \in \mathcal{M}. \end{aligned}$$

Both of the above problems (1) and (2) are NP-complete [11]. Though they can be solved using a dynamic programming algorithm with a complexity of $O(|\mathcal{M}|F)$ [11], it is not efficient enough to be used in practical large-scale systems. Furthermore, it is not suitable to be used for the eviction or replacement operation on server storage, in face of not only the dynamic evolution of user interests on currently stored files, but also a continuous flow of newly uploaded files from users.

To this end, in *Quota*, we design a simple framework of server storage and replacement strategy as described in Algorithm 1, which follows two principles as the following: *First*, the strategy should be *simple yet practical*, in the sense that simplicity and efficiency are more of a concern in real-world system implementations and operations, even though at a cost of acceptable sub-optimal solution. Hence, our strategy adopts a simple greedy algorithm to approximately approach the optimal solution with a complexity of $O(|\mathcal{M}| lg |\mathcal{M}|)$. Specifically, each file i is associated with a *profit-to-weight index* H_i, wherein the *profit* is related to specific objectives and the *weight* is related to the file size. Files are ranked in descending order, by the profit-to-weight index H_i, and those files with relatively higher ranks are preferentially stored on the server. Alternatively, it can simply and efficiently identify those files with lower ranks, and perform evictions or replacements whenever necessary. *Second*, rather than solely focusing on a particular aspect, our design unifies both of the important aspects indicated by (1) and (2) into a general framework $H_i = \lambda_i(f_i)^{-k}$, with a tunable knob $k \in [0, 1]$ providing flexible design choices.

For instance, when k is customized to 0, it becomes a *unpopular-first-eviction* strategy that ranks files merely by their popularity (*i.e.*, $H_i = \lambda_i$), and preferentially evicts/replaces those with least user interests. Such an intuitive strategy essentially tends to maximize the system-wide throughput as revealed by (2); and it is typically used in traditional server-based online hosting systems such as *Rapidshare*. When k is customized to 1, it makes a balanced consideration between file popularity and size (*i.e.*, $H_i = \lambda_i/f_i$), leading to a strategy to approach the maximum system-wide peer arrival rate (thus better system-wide file availability) as revealed by (1). Its practical implication is to store large files only if substantial user interests and popularity persist, in order to avoid excessive use of server storage. By tuning k in between 0 and 1, *Quota* can implement various degree of throughput and availability requirements.

Algorithm 1 A Framework of Server Storage and Replacement Strategy in *Quota*

```
1: Monitoring a profit-to-weight index for each file, $H_i, \forall i \in \mathcal{M}$, where $k \in [0,1]$ is a tunable parameter controlled by the service provider:
       $H_i \leftarrow \lambda_i (f_i)^{-k}$.
2: Sorting files in decreasing order according to their profit-to-weight indices.
3: Obeying a greedy strategy to determine which set of files to be stored on or evicted from the server storage:
       $\mathcal{F} \leftarrow$ Files with higher ranks are preferentially stored; alternatively, files with lowest ranks are preferentially evicted/replaced.
```

Remark: The above framework can be flexibly applied in practical systems with regard to the following aspects. *First*, to adapt to the evolution of user interests, H_i can be dynamically updated; and file ranking can be performed periodically in either a fine or coarse grained manner, under the control of the service provider. *Second*, for the eviction or replacement operation, a service provider can either start from the files with lowest ranks until a certain volume of files are evicted, or customize a threshold of H_i below which are the candidate files for eviction*. *Third*, as illustrated above, different strategies can be implemented by adjusting k. We will demonstrate that simple strategies in *Quota* can work well in practice, with evidences from our extensive performance evaluation in Sec. V.

*In real-world online hosting services, the decision on server storage and eviction may also be affected by the service agreement between the service provider and users, *e.g.*, whether it is offered free of charge or as a billable service. In this paper, we do not include such specific details.

C. Server Bandwidth Allocation Strategies

As a potentially large number of files are being concurrently shared in the system, it is essential to understand how to make better use of limited server bandwidth resources to guarantee adequate levels of downloading performance. Formally, given a specific total amount of server bandwidth S, a server bandwidth allocation strategy determines how to assign the bandwidth to each file, S_i, $\forall i \in \mathcal{F}$, *s.t.* $\sum_{i\in\mathcal{F}} S_i \leq S$, in order to guarantee a desired level of system-wide average downloading rate d.

Proposition 1: Given a certain server bandwidth allocation strategy across files, S_i, $\forall i \in \mathcal{F}$, *s.t.* $\sum_{i\in\mathcal{F}} S_i \leq S$, the system-wide average downloading rate d in steady state can be expressed as

$$d = \sum_{i\in\mathcal{F}} \lambda_i f_i \left(\sum_{i\in\mathcal{F}} \frac{\lambda_i (1 - \frac{S_i \nu_i}{\lambda_i \mu \eta_i})}{\nu_i (1 + \frac{\theta_i}{\nu_i})} \right)^{-1}, \tag{3}$$

where $\frac{1}{\nu_i} = \frac{1}{\eta_i}(\frac{f_i}{\mu} - \frac{1}{\gamma_i})$.

Proof: Based on [9], when the system is in steady state, the number of peers who are downloading file i can be expressed as follows:

When the downloading bandwidth is the constraint, *i.e.*, if $c\overline{x_i} \leq \mu(\eta_i \overline{x_i} + \overline{y_i}) + S_i$, we have

$$\begin{aligned} \overline{x_i} &= \frac{\lambda_i}{\theta_i + \frac{c}{f_i}} \\ \overline{y_i} &= \frac{\lambda_i}{\gamma_i + \frac{\gamma_i \theta_i f_i}{c}}. \end{aligned} \tag{4}$$

When the uploading bandwidth is the constraint, *i.e.*, if $c\overline{x_i} \geq \mu(\eta_i \overline{x_i} + \overline{y_i}) + S_i$, we have

$$\begin{aligned} \overline{x_i} &= \frac{\lambda_i}{\nu_i(1 + \frac{\theta_i}{\nu_i})} - \frac{S_i}{\mu \eta_i (1 + \frac{\theta_i}{\nu_i})} \\ \overline{y_i} &= \frac{\lambda_i}{\gamma_i(1 + \frac{\theta_i}{\nu_i})} - \frac{S_i \theta_i}{f_i \gamma_i \eta_i (1 + \frac{\theta_i}{\nu_i})}, \end{aligned} \tag{5}$$

where $\frac{1}{\nu_i} = \frac{1}{\eta_i}(\frac{f_i}{\mu} - \frac{1}{\gamma_i})$.

In accordance with most of the recent Internet access technologies and measurement studies on existing P2P systems [7], we assume that $c \geq \mu$ and peers will stay in the system only for a short random period of time after completing the download. Hence, the uploading bandwidth of peers in the system is most likely the constraint and we shall focus on the case of Eq. (5). Furthermore, to guarantee the corresponding condition $c\overline{x_i} \geq \mu(\eta_i \overline{x_i} + \overline{y_i}) + S_i$, the amount of server bandwidth provisioned to file i by the service provider shall satisfy:

$$S_i \leq \lambda_i \left(\frac{\frac{c - \mu\eta_i}{\nu_i} - \frac{\mu}{\gamma_i}}{1 + \frac{\theta_i}{\nu_i} + \frac{c - \mu\eta_i}{\mu\eta_i} + \frac{\theta_i}{\gamma_i \mu \eta_i}} \right). \tag{6}$$

To calculate the average downloading time T_i for peers downloading file i in steady state, we can use Little's Law as:

$$\frac{\lambda_i - \theta_i \overline{x_i}}{\lambda_i} \overline{x_i} = (\lambda_i - \theta_i \overline{x_i}) T_i. \tag{7}$$

Using Eq. (5), we can obtain the average downloading time of file i as:

$$T_i = \frac{1}{\nu_i(1+\frac{\theta_i}{\nu_i})}(1-\frac{S_i\nu_i}{\lambda_i\mu\eta_i}). \quad (8)$$

Again, based on Little's Law, the system-wide average downloading rate in steady state can be derived as:

$$d = \frac{\sum\limits_{i\in\mathcal{F}}\overline{x_i}d_i}{\sum\limits_{i\in\mathcal{F}}\overline{x_i}} = \frac{\sum\limits_{i\in\mathcal{F}}\overline{x_i}(\frac{f_i}{T_i})}{\sum\limits_{i\in\mathcal{F}}\lambda_i T_i} = \frac{\sum\limits_{i\in\mathcal{F}}\lambda_i f_i}{\sum\limits_{i\in\mathcal{F}}\lambda_i T_i}. \quad (9)$$

Hence, through Eq. (8), d can be expressed as Eq. (3). ■

Furthermore, we assume that no peers would abort the download (*i.e.*, $\theta = 0$), and no peers would stay in the system after finishing downloading the file (*i.e.*, $\gamma \to \infty$), as a conservative (pessimistic) approximation from the service provider's perspective. Then, Eq. (3) can be simplified to:

$$d = \sum_{i\in\mathcal{F}}\lambda_i f_i(\sum_{i\in\mathcal{F}}\frac{\lambda_i f_i}{\mu\eta_i} - R)^{-1}, \quad (10)$$

where $R = \sum_{i\in\mathcal{F}}(\frac{S_i}{\mu\eta_i})$ is referred to as a *critical factor* reflecting the system-wide server bandwidth provisioning relative to the peer uploading bandwidth contributions.

Meanwhile, the corresponding condition in Eq. (6) can be simplified to:

$$S_i \le (1-\frac{\mu\eta_i}{c})\lambda_i f_i. \quad (11)$$

Thus, the maximum amount of server bandwidth that can be assigned to file i is $S_{maxi} = (1-\mu\eta_i/c)\lambda_i f_i$.

Both Eq. (3) and Eq. (10) indicate that server involvement can help improve the downloading performance compared to a pure P2P system; however, the challenge is *how to design a near-optimal allocation strategy, that is simple enough to be implemented in practical systems.* To this end, we start from investigating the optimal server bandwidth allocation across files to achieve the upper bound of the system-wide average downloading rate.

***Theorem** 1:* Given a limited amount of server bandwidth resource S and the condition in Eq. (11), the upper bound of the system-wide average downloading rate can be achieved:

$$d_{max} = \frac{\sum\limits_{i\in\mathcal{F}}\lambda_i f_i}{\sum\limits_{i\in\mathcal{F}}\frac{\lambda_i f_i}{\mu\eta_i} - \sum\limits_{i=1}^{z-1}\frac{S_{maxi}}{\mu\eta_i} - \frac{1}{\mu\eta_z}(S-\sum\limits_{i=1}^{z-1}S_{maxi})}, \quad (12)$$

by a greedy algorithm with the optimal server bandwidth allocation as below:

$$S_i = \begin{cases} S_{maxi}, & \text{for } i = 1,\ldots,z-1; \\ 0, & \text{for } i = z+1,\ldots,|\mathcal{F}|; \\ S-\sum_{j=1}^{z-1}S_{maxj}, & \text{for } i = z, \end{cases} \quad (13)$$

where files are sorted as $\frac{1}{\mu\eta_1} \ge \frac{1}{\mu\eta_2} \ge \ldots \ge \frac{1}{\mu\eta_{|\mathcal{F}|}}$ without loss of generality, and $z = \min\{j : \sum_{i=1}^{j}S_{maxi} > S\}$.

Proof: Observed from Eq. (10), to maximize the system-wide average downloading rate d, we need to minimize the term of $\sum_{i\in\mathcal{F}}(\frac{\lambda_i f_i}{\mu\eta_i}) - R$, which in turn is equivalent to maximize the critical factor $R = \sum_{i\in\mathcal{F}}(\frac{S_i}{\mu\eta_i})$. Hence, the problem can be transform into the following optimization problem, which is essentially a classical *continuous knapsack problem* with bounded variables.

$$\text{Maximize} \qquad R = \sum_{i\in\mathcal{F}}(\frac{1}{\mu\eta_i})S_i \quad (14)$$

$$\text{Subject to:} \qquad \sum_{i\in\mathcal{F}}w_i S_i \le S,$$

$$S_i \le S_{maxi} = (1-\frac{\mu\eta_i}{c})\lambda_i f_i, i\in\mathcal{F},$$

where $w_i = 1$, $\forall i \in \mathcal{F}$, in this problem. The optimal solution of Eq. (14) can be obtained by a greedy algorithm as follows [12], [13]: sort files in non-increasing order of the ratio $(\frac{1}{\mu\eta_i})/w_i = \frac{1}{\mu\eta_i}$, so that $\frac{1}{\mu\eta_1} \ge \frac{1}{\mu\eta_2} \ge \ldots \ge \frac{1}{\mu\eta_{|\mathcal{F}|}}$. Then, we can use the critical index $z = \min\{j : \sum_{i=1}^{j}S_{maxi} > S\}$ to obtain the optimal server bandwidth allocation expressed as Eq. (13), and the maximum value of the critical factor R:

$$R_{max} = \sum_{i=1}^{z-1}\frac{S_{maxi}}{\mu\eta_i} + \frac{1}{\mu\eta_z}(S-\sum_{i=1}^{z-1}S_{maxi}). \quad (15)$$

Then, combining Eq. (10) and Eq. (15) gives the maximum system-wide average downloading rate d_{max} expressed as Eq. (12). ■

Remark: Theorem 1 with Eq. (12) and Eq. (13) provides a benchmark for the design of server bandwidth allocation strategy and performance evaluation. However, such an optimal strategy is hard to be implemented in reality due to the following reasons: *First*, it is hard to exactly capture η_i, $\forall i \in \mathcal{F}$, in real-world systems. *Second*, it is infeasible to directly use S_{maxi} for bandwidth allocation, as it is hard to proactively determine S_{maxi} without sufficient knowledge on η_i, μ, c in realistic scenarios.

However, being aware of an intuitive relationship $\eta_i \sim \lambda_i$ as pointed in [2], Theorem 1 still conveys important design guidelines for the service provider. The optimal strategy in Eq. (13) implies to allocate as much as possible of the bandwidth resource to less popular files with lower file sharing effectiveness η_i, while allowing popular files with higher η_i to largely rely on peer assistance rather than server. Based on this, we design a simple framework of server bandwidth allocation strategy with file popularity awareness, as described in Algorithm 2. Specifically, each file is associated with a *priority index* P_i, $\forall i \in \mathcal{F}$, which is inversely proportional to file popularity represented by λ_i, with a tunable knob l controlled by the service provider. Then, according to a *relative weighting* W_i normalized among P_i, *i.e.*, $W_i = P_i/\sum_{i\in\mathcal{F}}P_i$, server bandwidth is proactively allocated as $S_i = W_i S$, $\forall i \in \mathcal{F}$. Implicitly, if S_i cannot be fully consumed by certain files due to the downloading bandwidth constraint (*i.e.*, $S_i > S_{maxi}$), the residual server bandwidth will be uniformly consumed by the requests from other files.

By tuning the knob l, *Quota* provides a wide design spectrum for the service provider. For example, when l is customized to -1, it becomes a *request-driven* strategy with $P_i = \lambda_i$ and $S_i = S\lambda_i/\lambda$, where popular files with more requests from peers are provisioned with more server bandwidth resource. Such an intuitive strategy is typically used in traditional server-based systems without peer assistance. When l is customized to 0, it becomes a *water-leveling* strategy with $P_i = 1$ and $S_i = S/|\mathcal{F}|$, where the server bandwidth is equally allocated across all the files. Compared to the request-driven strategy, such a strategy shifts a considerable portion of server bandwidth occupied by popular files to support less popular files, while allowing popular files to rely more on peer assistance. As we further increase $l \to \infty$, *Quota* can also *mimic the optimal strategy* in Theorem 1 by primarily satisfying less popular files (while peer assistance is pervasive for popular files), so as to approach the optimal system-wide average downloading rate.

Algorithm 2 A Framework of Server Bandwidth Allocation Strategy in *Quota*

```
1: Monitoring a priority index for each file, $P_i$, $\forall i \in \mathcal{F}$,
   where $l$ is a tunable parameter controlled by the service provider:
       $P_i \leftarrow {\lambda_i}^{-l}$.
2: Compute the relative weighting for each file, $W_i$, $\forall i \in \mathcal{F}$,
   and allocate server bandwidth:
       $W_i \leftarrow P_i / \sum_{i \in \mathcal{F}} P_i$, $S_i \leftarrow W_i S$.
3: Server bandwidth usage of each file $i \in \mathcal{F}$:
       $\min\{S_{maxi}, \max\{S - \sum_{j \in \mathcal{F}, j \neq i} S_j, S_i\}\}$.
```

Remark: Our strategy framework can be easily applied in practical systems. *First*, P_i can be periodically updated to adapt to the evolution of user interests, wherein file popularity can be captured by recording the file request count over a certain period. *Second*, *Quota* does not require the computation of S_{maxi}, as such a bound is implicitly satisfied once the downloading bandwidth constraint is meet. *Third*, as illustrated above, different strategies with various degree of file popularity awareness (implying peer assistance awareness) can be implemented by adjusting l. We will evaluate these strategies in *Quota* from various perspectives in Sec. V, and compare them with the benchmark provided by Theorem 1.

V. Performance Evaluation

In this section, we carry out a series of numerical experiments using real-world data sets to evaluate our design of *Quota* from different perspectives.

A. Real-world Traces

To make our evaluation practical and comprehensive, we use a real-world data set extracted from our measurement trace of a large-scale peer-assisted online hosting system, FS2You [14], which is actively used in China at the time of this writing. The data set contains $87,848$ distinct files (corresponding to 15.5 TB) with diverse popularity and sizes.

For each file, we focus on the following statistics on a representative day: (1) *File size*. We record the size of each file which has received at least one request within the day. (2) *Peer arrival rate*. This is alternatively represented by the number of requests for each file within the day. By treating the peer arrival rate of the most popular file as 1, the peer arrival rates of other files are normalized by dividing the number of requests of each file by the number of requests of the most popular file. Though such a measure may not exactly capture the peer arrival rate, it can indeed reflect the relative popularity of each file. (3) *File sharing effectiveness*. We record the average uploading rate contributed by peers for each file within the day. According to the analysis in [2], the file sharing effectiveness is close to 1 when the file is popular enough. Hence, we treat the maximum value of this measure over all the files as an estimate of peers' upload capacity. Then, the file sharing effectiveness of each file can be roughly captured by dividing their respective average uploading rates by the estimated peers' upload capacity.

Note that these statistics reflect the inherent file characteristics and general user interests, which are independent of the detailed implementation of the hosting system.

To emulate different application scenarios, we further distill three representative file sets based on the above statistics, as summarized in Table I: (1) The *random file set* containing 1000 files that are randomly sampled from our data set. (2) The *popular file set* containing 1000 files that are randomly sampled from the top 5000 popular files in our data set. (3) The *small file set* containing 1000 files that are randomly sampled from those files with sizes below 100 MB ($<$ the mean over our data set).

Most of our numerical experiments are based on the random file set, as it represents the generic scenario. Meanwhile, we also use the popular file set and small file set to obtain complementary insights, and exercise our design of server strategies to the extent possible.

TABLE I
Statistics of Three Representative File Sets.

File Set	Average file size (MB)	Average peer arrival rate	Average file sharing effectiveness
Random	209.6	0.023	0.851
Popular	204.7	0.049	0.925
Small	39.7	0.028	0.841

B. Evaluation of Server Storage and Replacement Strategies

1) Flexible Choices of Design Knob k: As indicated in Sec. IV-B, the design knob k of our server storage and replacement strategy framework provides flexible design choices for the service provider. Here we quantitatively demonstrate how k can be customized by the service provider with regard to various degree of file availability and system-wide throughput. Specifically, we use the ratio of $\sum_{i\in\mathcal{F}} \lambda_i / \sum_{i\in\mathcal{M}} \lambda_i$ to represent file availability, as it gives an intuitive view on the percentage of file requests that can be served. We use the random file set as $\mathcal{M}$, and server storage capacity F is limited

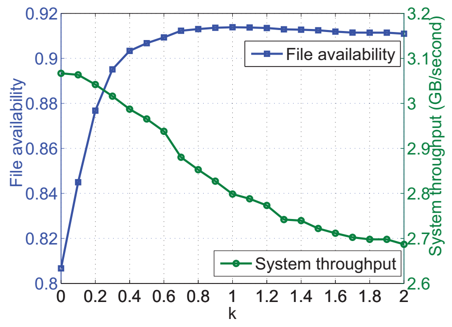

Fig. 1. File availability and system-wide throughput vs. the design knob k of server storage and replacement strategy.

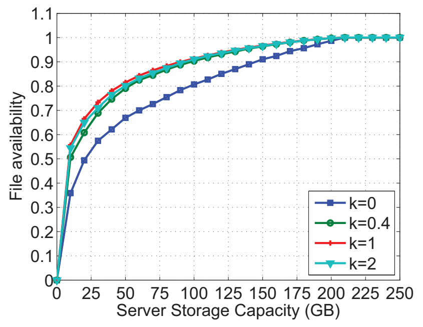

Fig. 2. File availability vs. server storage capacity F, under different settings of the design knob k of server storage and replacement strategy.

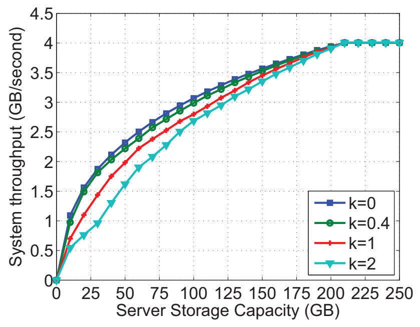

Fig. 3. System-wide throughput vs. server storage capacity F, under different settings of the design knob k of server storage and replacement strategy.

to 100 GB, which is half of the total size of the file set. Further discussion under different settings of server storage capacity will be presented in the next subsection.

Fig. 1 depicts file availability and system-wide throughput as functions of k. The curve corresponding to file availability first increases dramatically when $k < 0.4$, and then slows down when $k > 0.4$. When $k = 1$, the peak value of file availability, 91.4%, is reached, after which the curve slightly decreases. The curve corresponding to system-wide throughput goes down almost linearly as k increases. When $k > 1$, system-wide throughput decreases more slowly. The peak value of system-wide throughput is reached when $k = 0$, which is consistent with our discussion in Sec. IV-B. Combining the trends of the two curves, we find that both file availability and system-wide throughput stay at a high level when k varies between 0.3 and 0.5, which shows an opportunity to design a strategy that can achieve both high file availability and system-wide throughput.

Based on our discussion in Sec. IV-B and the above observations, apparently $k = 0$ and $k = 1$ are two interesting extreme cases that are worthy of further investigation. Besides, from the perspective of service provider, a strategy to achieve both high file availability and system-wide throughput is even more desired. Hence, the strategy with $k = 0.4$ is also chosen for further discussion in the following subsections. In addition, we also consider the case of $k = 2$, which though leads to poor system-wide throughput, will further reveal some surprising results in Sec. V-B3.

2) File Availability and System Throughput: We further compare the strategies with different choices of k that are identified previously (*i.e.*, $k = 0$, 0.4, 1 and 2), by plotting file availability and system-wide throughput when server storage capacity varies in Fig. 2 and Fig. 3, respectively.

From Fig. 2, we have made the following observations. (1) Despite the choice of k, file availability first increases dramatically when server storage capacity is less than 50 GB and then slows down as server storage capacity further increases. Specifically, when server storage capacity is 40 GB (*i.e.*, only 20% of the total size of random file set in our experiment), the strategy with $k = 1$ can already achieve file availability of nearly 80% while the strategy with $k = 0$ can achieve file availability of 60%. This gives service providers very useful implications when they are determining appropriate server storage capacity: *it is feasible to use a relatively smaller server storage capacity to satisfy most of the user interests; furthermore, using our strategy framework with fine-tuned* k*, the efficacy can be further improved.* When server storage capacity is larger than 200 GB (*i.e.*, the server can host all the files of random file set in our experiment), file availability reaches 100% for all the four strategies. (2) The gaps between different strategies first expand and then shrink when server storage capacity is larger than 50 GB. Specifically, the gaps between the strategies with $k = 0.4$, $k = 2$, and the strategy with $k = 1$ are very small. When server storage capacity is larger than 100 GB (*i.e.*, 50% of the total size of random file set in our experiment), their curves are very close. This implies that, in addition to $k = 1$, $k = 0.4$ and $k = 2$ are also two practical choices to gain high file availability.

Similar curves are observed for system-wide throughput from Fig. 3. As server storage capacity increases, system-wide throughput grows for all the four strategies. Consistent with our previous discussion, the strategy with $k = 0$ outperforms the other three strategies. However, the performance gaps between these strategies shown in Fig. 3 are not as significant as that in Fig. 2. Similar to the observations on file availability, the strategy with $k = 0.4$ achieves high system-wide throughput close to that of the strategy with $k = 0$. This observation again strengthens the argument that $k = 0.4$ is practical for server storage and replacement strategy to achieve both high file availability and system-wide throughput.

3) Downloading Performance: Finally, we examine the system-wide average downloading rate under different server storage and replacement strategies. In this experiment, the amount of server bandwidth is set to 1 GB/second, which is moderate enough for the system-wide average downloading rate to vary significantly as server storage capacity increases. In addition, we use the optimal server bandwidth allocation strategy derived in Theorem 1 (Sec. IV-C), as it can make the performance gapes between different server storage and replacement strategies more profound. The uploading and downloading bandwidth of peers are set as $\mu = 512$ Kbps and $c = 1024$ Kbps, respectively.

From Fig. 4, we have made the following observations. (1) For all the four strategies, the system-wide average download-

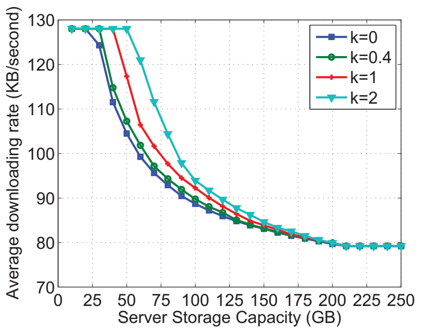

Fig. 4. System-wide average downloading rate vs. server storage capacity F, under different settings of the design knob k of server storage and replacement strategy.

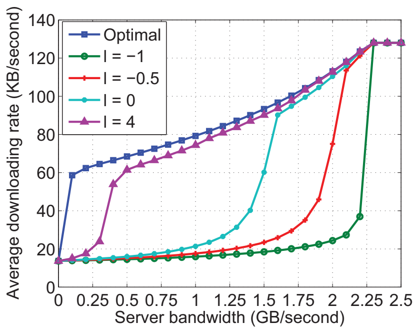

Fig. 5. System-wide average downloading rate vs. the amount of server bandwidth resource S, under different settings of the design knob l of server bandwidth allocation strategy.

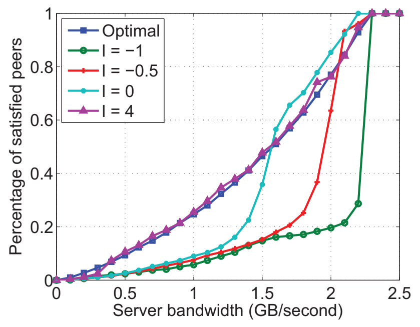

Fig. 6. Percentage of satisfied peers vs. the amount of server bandwidth resource S, under random file set and different settings of the design knob l of server bandwidth allocation strategy.

ing rate declines as server storage capacity increases. The reasons are two-folded. First, despite the setting of k, the strategies in *Quota* inherently tend to host popular files first. As a result, when server storage capacity increases, though more files can be hosted, such "later arrivals" are relatively less popular with lower file sharing effectiveness, which leads to lower system-wide uploading contributions from peers. Second, since the amount of server bandwidth is limited, the increase of number of hosted files and peers in the system along with the increase of server storage capacity, will make the server bandwidth obtained by each peer become less. (2) The system-wide average downloading rate increases as k increases, which is surprising at the first glance. Recall that in previous subsection, the strategy with $k = 2$ performs the worst in term of system-wide throughput, while it is near-optimal in term of file availability. Intuitively, the system-wide average downloading rate would be proportional to the system-wide throughput, rather than file availability. However, our result here demonstrates that such an intuition is not true. The rationale is that, the strategies with larger values of k tend to host files of smaller sizes, which leads to shorter downloading time and thus smaller number of peers in the system in steady state. Hence, with fewer peers staying in the system, such strategies can achieve higher system-wide average downloading rate despite the lower system-wide throughput.

C. Evaluation of Server Bandwidth Allocation Strategies

1) Flexible Choices of Design Knob l: As discussed in Sec. IV-C, our design framework of server bandwidth allocation strategy also provides flexible choices for the service provider. We first identify some representative settings of the design knob l to cover a wide design spectrum. Apparently, two of the most interesting choices are $l = -1$ and $l = 0$, which correspond to the request-driven strategy and water-leveling strategy, respectively. In addition, we also consider the case of $l = -0.5$, which lies in between the above two strategies. Furthermore, as discussed in Sec. IV-C, *Quota*, with $l \rightarrow \infty$, can mimic the optimal strategy derived in Theorem 1. Along this direction, we choose $l = 4$, which will be demonstrated to be able to achieve a near-optimal strategy. All the above strategies will be compared with the optimal one from different perspectives in the following.

2) Downloading Performance: First, we examine the system-wide average downloading rate under different server bandwidth allocation strategies that we identified above. We use the random file set to obtain more generic insights, and further discussion under the popular and small file sets will be presented later. Here we assume that all the files in the file set are hosted on the server, so as to focus on server bandwidth allocation strategies.

Fig. 5 depicts the system-wide average downloading rate as a function of the amount of server bandwidth, under different settings of l. In general, the system-wide average downloading rate rises as the amount of server bandwidth and l increase. The rationale is that as l increases, the strategies in *Quota* are tuned to allocate larger portion of server bandwidth to unpopular files, which is more efficient to enhance system-wide average downloading rate according to our theoretical analysis in Sec. IV-C. The performance gaps between different strategies can be remarkably large. For example, the downloading performance under the strategy with $l = 4$ outperforms that of the strategy with $l = -1$ by 90 KB/second, when the amount of server bandwidth is 2 GB/second. This evidences that, the design of server bandwidth allocation strategy plays an important role in practical peer-assisted online hosting systems; and a fine-tuned strategy can be very efficient in improving system performance. We also find that our strategy framework is successful to mimic the optimal strategy which is hard to implement in reality, when l is customized to 4 and the amount of server bandwidth is above 0.5 GB/second.

Another interesting phenomenon is that, despite the setting of l, all the curves except the optimal one follow the same pattern. As the amount of server bandwidth increases, the system-wide average downloading rate first increases slowly and then jumps up rapidly. When the downloading performance is close to optimal, the growth of the curves slows down again; and then downloading performance increases linearly until reaching the downloading bandwidth constraint of peers. As l increases, the amount of server bandwidth

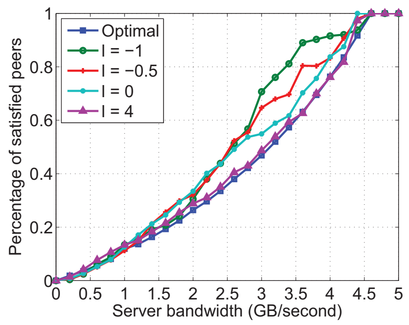

Fig. 7. Percentage of satisfied peers vs. the amount of server bandwidth resource S, under popular file set and different settings of the design knob l of server bandwidth allocation strategy.

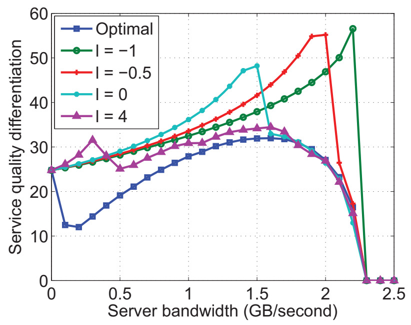

Fig. 8. Service quality differentiation vs. the amount of server bandwidth resource S, under random file set and different settings of the design knob l of server bandwidth allocation strategy.

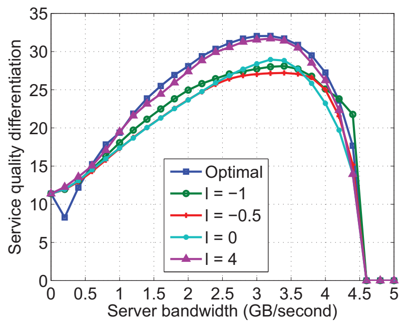

Fig. 9. Service quality differentiation vs. the amount of server bandwidth resource S, under popular file set and different settings of the design knob l of server bandwidth allocation strategy.

under which the significant jump of downloading performance occurs, is getting smaller. There are two extreme cases: (1) Under the optimal strategy, the downloading performance jumps up immediately when server bandwidth is injected into the system. (2) Under the strategy with $l = -1$ (*i.e.*, request-driven strategy), the jump of downloading performance only occurs when the amount of server bandwidth is close to the maximum volume that the system can consume (*i.e.*, the downloading bandwidth constraint of peers).

To explain the above phenomenon, we take a closer look at a single file. Based on Eq. (8), the average downloading rate of file i in steady state is $d_i = f_i/T_i = \lambda_i f_i \eta_i \mu/(\lambda_i f_i - S_i)$, where d_i can be treated as a function of S_i. Hence, the derivative of d_i is $d'_i = \lambda_i f_i \eta_i \mu/(\lambda_i f_i - S_i)^2$, which stands for the changing velocity of d_i as S_i increases. We find that d'_i is an increasing function of S_i, which implies that as S_i increases, the growth of d_i will accelerate and finally jump up rapidly. Intuitively, d_i is improved with more server bandwidth provisioned, leading to shorter downloading time and thus fewer peers staying in the system in steady state. This in turn allows each peer to obtain more server bandwidth to enhance its downloading performance.

Furthermore, given a limited amount of server bandwidth, d'_i has a bounded range of $[\frac{\eta_i \mu}{\lambda_i f_i}, \frac{c^2}{\lambda_i f_i \eta_i \mu}]$. As $\eta_i \sim \lambda_i$, the differences in the lower bound of d'_i among different files are not significant. In contrast, the upper bound of d'_i varies significantly depending on file popularity. For example, the upper bound of d'_i for a unpopular file with $\lambda_i = 0.1$ and $\eta_i = 0.5$ is twenty times of that for a popular file with $\lambda_j = 1$, $\eta_j = 1$, and the same file size. Hence, when the same amount of server bandwidth is allocated to popular files and unpopular files, respectively, the latter obtained more improvement in downloading performance than the former. Following this principle, as l increases, our strategy framework tends to allocate more server bandwidth to unpopular files, and hence the system-wide average downloading rate can jump up "earlier" with relatively lower server bandwidth cost.

3) User Satisfaction Level: From the perspective of the service provider, in addition to the objective of maximizing the downloading performance of the entire system, another important concern is to guarantee that as many users as possible can enjoy a promised level of service quality. To this end, we use the measure of *the percentage of satisfied peers*, to evaluate the system-wide satisfaction level. This is quantified as the percentage of peers whose downloading rates exceed a given threshold over all the peers staying in the system. Note that in our experiment, the maximum average downloading rate of peers without the supplement of server bandwidth is μ, the uploading bandwidth of a peer. Based on this, we use 1.2μ as a promised downloading rate (also referred to as the threshold), which though is unreachable by pure P2P, can be achieved with server bandwidth provisioning. In the following, we will compare the system-wide satisfaction level under different server bandwidth allocation strategies, using the three file sets described in Sec. V-A.

Fig. 6 plots the percentage of satisfied peers based on the random file set. We have made the following observations. (1) The system-wide satisfaction level of the strategy with $l = 4$ is close to that of the optimal strategy, which goes up almost linearly. (2) Roughly speaking, the system-wide satisfaction level rises as l increases. However, the strategies with $l = 0$ and $l = -0.5$ outperform the optimal one when server bandwidth is sufficient. The strategy with $l = -1$ always performs worse compared to other strategies. (3) When l is small (*e.g.*, $l = -1, -0.5, 0$), the percentage of satisfied peers increases at first slowly and then dramatically as the amount of server bandwidth increases. The pattern is very similar to that of the system-wide average downloading rate observed in previous subsection. This implies that unpopular files are the major cause resulting in low percentage of satisfied peers under these strategies.

We also run the experiment under the other two file sets to obtain complementary insights. Fig. 7 plots the percentage of satisfied peers based on popular file set. Since the similar pattern is observed for the small file set, we omit its figure here. The pattern is totally different from that of Fig. 6. In Fig. 7, we find that the percentage of satisfied peers increases as l decreases. When l is small (*e.g.*, $l = -1$), the percentage of satisfied peers increases fast as the amount of server bandwidth increases. The strategy with $l = -1$ can even outperform the optimal one by nearly 30% when the amount of server bandwidth is 3.6 GB/second. The rationale is that, all the files in the popular file set have high peer arrival rates and file sharing effectiveness. Leaving their file sizes out of account, these files require less and similar amount of server bandwidth to achieve the promised service quality. In such a scenario, the strategies with more emphasis on less popular files (*e.g.*, $l = 4$) dedicate too much sever bandwidth to those less popular files even if they've already achieved the promised downloading performance. In contrast, the strategies that tend to equally allocate server bandwidth across all the peers is more efficient to enhance system-wide satisfaction level under this scenario.

Combining the above discussion gives practical guidelines for the service provider: when the files hosted in the system have highly diverse popularity, a strategy which aims to enhance the system-wide average downloading rate is more efficient in improving the system-wide user satisfaction level. Otherwise, a strategy which treats each request fairly can perform better in guaranteeing a promised service quality.

4) Service Quality Differentiation: Finally, it is also essential to consider the differences in service quality across peers involved in files of different popularity. We use the standard deviation to quantify the service quality differentiation as $\sigma = \sqrt{\sum_{i\in\mathcal{F}} \overline{x_i}(d_i - d)^2 / \sum_{i\in\mathcal{F}} \overline{x_i}}$. Fig. 8 depicts the service quality differentiation as a function of the amount of server bandwidth and l, under the random file set. We observed that: (1) In general, despite the setting of l, service quality differentiation grows at first and then falls down, as the amount of server bandwidth increases. The optimal strategy shows a different phenomenon compared to others. Its resulting service quality differentiation falls down when server bandwidth starts to be injected into the system. The rationale is that this strategy allocates as much as possible of server bandwidth to unpopular files, which improves the average downloading rates of these files and thus reduces the number of peers involved in them in steady state. This results in the reduction of weights of these files in the computation of σ. (2) The strategies with small l (*e.g.*, $l = -1, -0.5, 0$) result in more profound service quality differentiation compared to the optimal one, due to their less supplement to unpopular files.

Again, we also run the experiment under the other two file sets to obtain complementary insights. Fig. 9 depicts the service quality differentiation as a function of the amount of server bandwidth, under the popular file set. Since we observed the similar pattern for the small file set, we omit its figure here. Fig. 9 shows completely different results compared to the random file set. For the random file set, the service quality differentiation is 25 when there is no server bandwidth provisioned; while that of the popular file set and the small file set is 11.4 and 16.5, respectively. Consistent with the discussion in previous subsection, the files in the popular file set have similar characteristics, and so does the small file set. Hence, a strategy which tends to serve each file and peer fairly performs better. For example, under the popular file set, the strategies with small l (*e.g.*, $l = -1, -0.5, 0$) perform similarly and better than the other two.

In summary, our strategy framework can practically work well in various scenarios, with flexible design choices for the service provider. The strategies with large l (*e.g.*, $l = 4$) are successful in approaching the optimal strategy to achieve high system-wide downloading performance, which are suitable in scenarios where files have highly diverse popularity. On the other hand, for those scenarios where files have similar characteristics, small l (*e.g.*, $l = 0$) would be a good choice to keep low differences in service quality across the files.

VI. Conclusions

In response to a number of unique challenges involved when a large number of files share scarce server resources in peer-assisted online hosting systems, this paper proposes *Quota*, which explores the design space of new protocols to allocate server resources. Based on both mathematical analysis and practical concerns, *Quota* seeks to make better use of limited server storage and bandwidth resources to guarantee adequate levels of service quality, in terms of both file availability and downloading performance. Through extensive experimental studies using real-world data sets that we collected, we demonstrate the applicability and flexibility of *Quota*. With fine-tuned design knobs controlled by the service provider, *Quota* can achieve both high file availability and system throughput with a relatively smaller server storage capacity, and the server bandwidth allocation strategy can potentially approach the optimal system-wide downloading performance. Last but not the least, our results in *Quota* have also revealed practical implications with respect to the user satisfaction level and service quality differentiation.

References

[1] X. Yang and G. de Veciana, "Service Capacity of Peer to Peer Networks," in *Proc. of IEEE INFOCOM*, Mar. 2004.

[2] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks," in *Proc. of ACM SIGCOMM*, Aug. 2004.

[3] B. Fan, D. Chiu, and J. Lui, "The Delicate Tradeoffs in Bittorrent-like File Sharing Protocol Design," in *Proc. of IEEE ICNP*, Nov. 2006.

[4] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling Peer-Peer File Sharing Systems," in *Proc. of IEEE INFOCOM*, Mar. 2003.

[5] L. Massoulie and M. Vojnovic, "Coupon Replication Systems," in *Proc. of ACM SIGMETRICS*, Jun. 2005.

[6] A. Chow, L. Golubchik, and V. Misra, "BitTorrent: An Extensible Heterogeneous Model," in *Proc. of IEEE INFOCOM*, Apr. 2009.

[7] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems," in *Proc. of ACM Internet Measurement Conference (IMC)*, Oct. 2005.

[8] Y. Tian, D. Wu, and K. W. Ng, "Modeling, Analysis and Improvement for BitTorrent-Like File Sharing Networks," in *Proc. of IEEE INFOCOM 2006*, Apr. 2006.

[9] S. Das, S. Tewari, and L. Kleinrock, "The Case for Servers in a Peer-to-Peer World," in *Proc. of IEEE ICC*, Jun. 2006.

[10] C. Wu, B. Li, and S. Zhao, "Multi-Channel Live P2P Streaming: Refocusing on Servers," in *Proc. of IEEE INFOCOM*, Apr. 2008.

[11] J. Kleinberg and E. Tardos, *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[12] S. Martello and P. Toth, *Knapsack Problems Algorithms and Computer Implementations*. John Wiley & Son Ltd, Chichester, England, 1990.

[13] V. Cerone and F. Croce, "A Continuous Knapsack Problem Formulation for The Robustness Analysis of A Polytope of Polynomials," in *American Control Conference*, 1995.

[14] [Online]. Available: http://www.rayfile.com