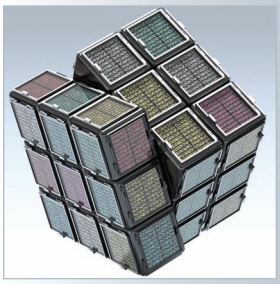

Smart Cloud Seeding for BitTorrent in Datacenters

Cloud content providers must deliver vast amounts of data to an ever-growing number of users while maintaining responsive performance, thus increasing bandwidth-provisioning expenditures. To mitigate this problem, the authors transparently integrate BitTorrent into the cloud provider infrastructure and leverage users' upstream capacity to reduce bandwidth costs. They also allocate seeder bandwidth optimally among swarms to maximize throughput. Their system delivers higher performance when dealing with large volumes of data compared to the traditional client-server paradigm.

Xavier León, Rahma Chaabouni, Marc Sánchez-Artigas, and Pedro García-López
Universitat Rovira i Virgili, Spain

Today's Internet lets an increasing number of users consume various types of content. Services such as Dropbox, SugarSync, and Google Drive deal with an unprecedented amount of data on a daily basis, storing, processing, and delivering such content in datacenters as if capacity were unlimited. To provide service to their clients in a responsive manner at a large scale, small and medium content providers must rely on third-party content delivery networks. As the number of clients grows, the cost of scaling up resources becomes problematic.

To address this issue, content providers have relied on protocols such as BitTorrent to leverage interested clients' spare upstream capacity for specific content.[1,2] In this way, the group of clients interested in certain content — also known as a *swarm* — is tasked with redistributing small parts of that content to other members. This solution not only saves precious bandwidth on the provider side but also helps reduce clients' download times compared to using a single content source, as when following the client-server model.

We foresee two challenges in the context of content distribution in datacenters. Current cloud storage services rely mainly on the client-server communication paradigm to make their content available. Therefore, our first challenge is how to transparently integrate a swarming protocol such as BitTorrent (www.bittorrent.com) into a datacenter to avoid placing the burden of managing the transition between protocols on users.

The second challenge is how to allocate the limited bandwidth on the datacenter side to different swarms, thus maximizing content download throughput and increasing the system's efficiency and responsiveness compared to current solutions. For this, we propose a smart seeding strategy that grants bandwidth according to swarm characteristics.

1089-7801/14/$31.00 © 2014 IEEE Published by the IEEE Computer Society

Related Work in Content Distribution

Various related works focus on how to efficiently distribute content to a set of users, from classical content distribution networks (CDNs) to online, multicast streaming of live content. Here, we focus on works whose main aim is to improve the responsiveness and throughput of large volumes of content given a restricted budget on server- or datacenter-side bandwidth consumption.

Classical CDNs replicate data to a set of intermediate servers to alleviate the load on the principal server, reduce download times, and handle flash crowds. The very successful Akamai, for example, gives content providers a large distributed network on which to cache content to support large volumes of client requests.[1] This business model simply results in increasing costs according to the amount of data the delivery network handles.

Instead of delivering infrastructure support to content providers, systems such as BitTorrent[2] and Avalanche[3] offer a peer-to-peer paradigm that shifts bandwidth costs to clients. The work most similar to ours is Antfarm,[4] a content distribution system that measures a swarm's response curve to seeder bandwidth to optimize its uploads among competing swarms. Such a system needs to actively measure swarm dynamics and uses an off-band protocol to incentivize users to report performance data that is later used to do the actual allocation. Although we tackle the same problem, we avoid actually measuring swarm dynamics in real time, and thus bypass a nonnegligible overhead on the content provider and measurement errors.

In contrast, our smart seeding strategy exploits some known facts about BitTorrent protocol behavior to model the response curve based on data already available from BitTorrent trackers and provided by our monitored clients. Our mathematical framework lets us solve the multiswarm bandwidth allocation in a computationally and network-wise efficient manner. As we show in the main text, our strategy provides higher throughput to swarms as well as a faster time to convergence without the fine-grained monitoring Antfarm requires.

An interesting recent work proposes a model-based allocation mechanism for client-assisted content delivery based on building up a model from offline measurements. This effectively creates a so-called cheat sheet, which provides actual response curves computed beforehand.[5] Besides their model-based mechanism, the authors propose a distinction between static mechanisms that use simple strategies to allocate bandwidth — equal and proportional sharing in our case — and dynamic mechanisms that constantly adjust bandwidth allocation according to swarm dynamics, as with Antfarm.

Our mechanism takes the best from the model-based and dynamic mechanisms. On the one hand, our mathematical model predicts swarm performance with respect to datacenter bandwidth, letting the optimization problem converge instantaneously, as in the measurement-based model.[5] On the other hand, our model is flexible enough to adapt the response curve to swarm dynamics using online measurements, which are easily gathered from our instrumented clients, as in dynamic strategies.[4] Although our model's accuracy is potentially lower than a measurement-based model, our strategy's practicality is much higher because no offline measurements are necessary.

References

1. E. Nygren, R.K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-Performance Internet Applications," *ACM SIGOPS Operating Systems Rev.*, vol. 44, no. 3, 2010, pp. 2–19.
2. B. Cohen, "Incentives Build Robustness in BitTorrent," *Proc. Workshop Economics of Peer-to-Peer Systems*, vol. 6, 2003, pp. 68–72.
3. C. Gkantsidis and P.R. Rodriguez, "Network Coding for Large-Scale Content Distribution," *Proc. 24th Ann. Joint Conf. IEEE Computer and Communications Societies*, vol. 4, 2005, pp. 2235–2245.
4. R. Peterson and E.G. Sirer, "Antfarm: Efficient Content Distribution with Managed Swarms," *Proc. 6th Usenix Symp. Networked Systems Design and Implementation*, vol. 9, 2009, pp. 107–122.
5. A.R. Abhigyan Sharma and A. Venkataramani, "Pros and Cons of Model-Based Bandwidth Control for Client-Assisted Content Delivery," *Proc. 6th Int'l Conf. Communication Systems and Networks*, 2014, pp. 1–8.

Personal Cloud Storage Scenario

The past few years have seen a rush of online storage services such as Amazon Simple Storage Service and Dropbox entering the market. From a technical viewpoint, most of these services use HTTP as a transfer protocol and fail to benefit from users' interest in the same content. To illustrate, consider a professor who wants to share a large dataset with students so that they can synchronize across different personal devices and analyze individually. In this case, the content provider would cut bandwidth costs by using a client-assisted content delivery mechanism, and students could share their upload capacity to improve download speeds.

In this context, we integrated BitTorrent,[3] a well-known, peer-to-peer (P2P) protocol that leverages users' spare upstream capacity to offload some of the burden from storage servers. The main challenge is to achieve this seamlessly and transparently without user intervention. With this approach, the storage service monitors user activity and, on detecting a certain critical mass for a specific content, transparently switches to BitTorrent.

A recent study on Dropbox client behavior[4] demonstrates that 5 percent of the service's dataflows were bigger than 10 Mbytes, accounting for a high percentage of the traffic measured.

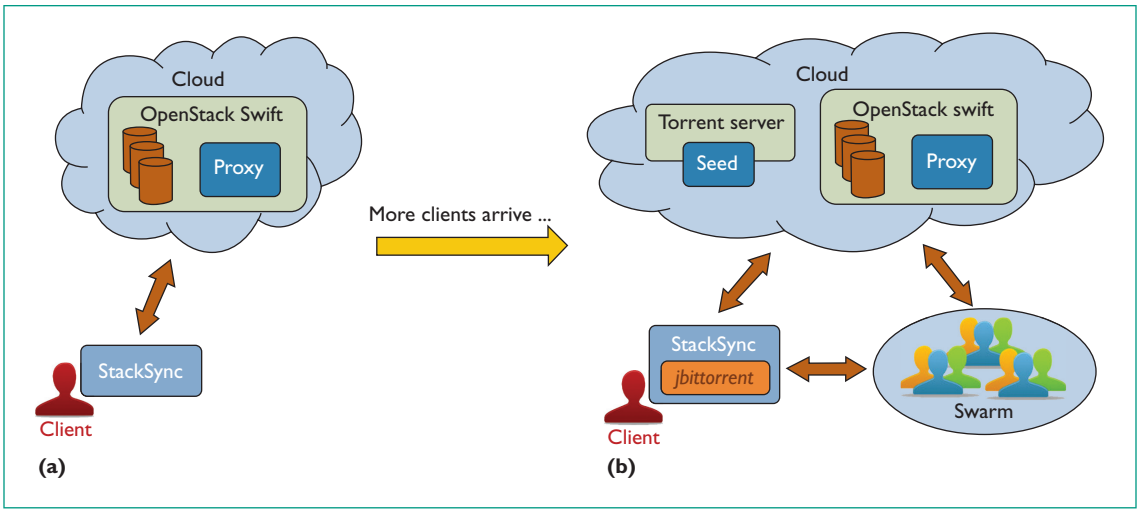

Figure 1. System architecture (a) before and (b) after the switch to the BitTorrent protocol. Clients download content using regular HTTP sessions and, when the system detects a critical mass of users interested in the same content, are migrated transparently to the BitTorrent protocol.

Considering big and moderately sized files, it makes sense to use a peer-assisted mechanism such as BitTorrent to offload storage servers.

Our system's architecture consists of a cloud storage system based on OpenStack Swift (wiki.openstack.org/wiki/Swift), which we modified to accommodate BitTorrent. We also used an open source personal cloud system extended with a BitTorrent library. Figure 1 shows the resulting system architecture, which has the following main components.

On the cloud side is OpenStack Swift, which replicates the client's files in each storage node to maintain reliability in the face of drive failures. The *proxy server* handles the requests, locating objects and routing requests accordingly. This server also monitors incoming requests and, on detecting a certain mass for specific content, decides to switch to BitTorrent. Being a logically centralized entity, the proxy server could suffer from reliability and performance issues. Although our implementation doesn't actually cover this problem, this key entity could achieve high availability using state-of-the-art replication and load-balancing techniques.

The *torrent server* is triggered when the proxy switches to the BitTorrent protocol for specific content. The server runs a seed that extracts the requested chunks from the storage nodes. It then generates a corresponding `.torrent` file and transmits it to the client.

On the personal cloud client side, we developed an open source implementation called StackSync (http://github.com/cloudspaces/stacksync) that provides storage, syncing, and sharing capabilities on top of OpenStack. We extended this prototype's implementation with the jbittorrent library (http://github.com/cloudspaces/jbittorrent) to enable a transparent switch to BitTorrent.

Our system can transparently switch from a regular client-server model that uses HTTP to a P2P model that uses the BitTorrent protocol, without interaction from users. The specific time at which the switch occurs depends on how many concurrent users there are, clients' spare upload capacity, and the file size. We're studying these tradeoffs, and an in depth analysis is left for future work.

Following our model, the datacenter can serve different popular content to different swarms concurrently. Thus, we must determine how to allocate the datacenter's limited bandwidth in a multiswarm scenario. Our main motivation is to increase content delivery's overall throughput to provide more users with a responsive system, thus minimizing content download time.

Smart Seeding

The primary goal of our smart seeding strategy is to determine the upload bandwidth w_s that the seeder (the entity providing content) should allocate to a swarm $s \in S$ – assuming an upper bound on bandwidth consumption $W = \Sigma_{s \in S} w_s$ – to maximize the aggregate download bandwidth considering all swarms $\Sigma_{s \in S} D_s$, where D_s is the aggregate download bandwidth of the peers $p \in s$ (entities downloading content) in swarm s. By maximizing swarms' download speed, we ensure minimal average download times and

Table 1. Model parameters for every single swarm.		
Parameter	**Description**	**Source of information**
$d = \sum_{p \in S} d_p$	Max download bandwidth	Instrumented client
$u = \sum_{p \in S} u_p$	Max upload bandwidth	Instrumented client
$a = \sum_{p \in S} a_p$	Self-sustained bandwidth	Instrumented client
n	Number of peers	Tracker information

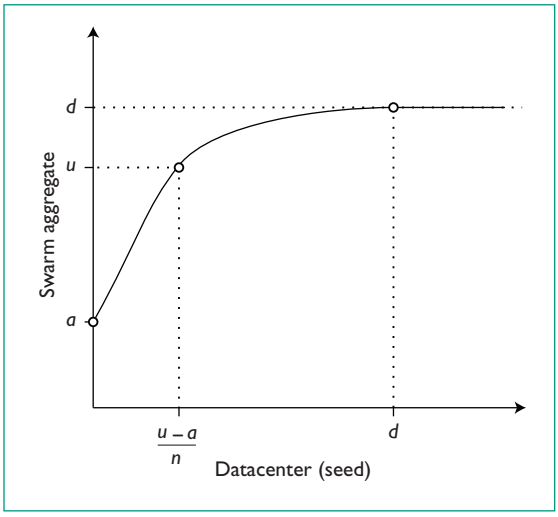

Figure 2. A swarm's response curve model. We calculate the swarm's aggregate bandwidth as a function of the datacenter's allocated bandwidth.

that the system distributes content as fast as possible within a restricted bandwidth budget.

Bandwidth Response Model

Our smart seeding strategy uses information about the swarm's state to determine the proper amount of bandwidth for allocation. This key information is the *response curve*, which represents the swarm aggregate bandwidth as a function of the allocated seeder bandwidth.[1] This function $f_s(w_s)$ embodies information about the swarm's current sustainable download bandwidth ($a = \sum_{p \in S} a_p$), as well as the swarm's aggregate upload ($u = \sum_{p \in S} u_p$) and download ($d = \sum_{p \in S} d_p$) saturation bandwidth (see Figure 2). We obtain this information from our instrumented clients and measure it every time a client uploads and downloads new content directly from OpenStack Swift (u_p and d_p) or at regular time intervals (a_p). Table 1 shows these model parameters. These response curves depend on several factors, including the number of peers (n) and seeders, their actual bandwidth contribution to the swarm, or the current distribution of unique blocks — elements that are naturally dynamic in BitTorrent swarms.

However, bandwidth swarm response curves have a characteristic shape that we can model approximately using known information. Specifically, we used a family of hyperbolic functions of the form $f_s(w_s) = a + (((d - a)w_s)/(w_s + c))$, where w_s is the seeder bandwidth allocated to swarm s, and $c = (d - u)/n$ is the parameter that shapes the function's increment rate — the higher the value, the flatter the curve. We obtain this parameter by substituting the coordinate $(x, y) = ((u - a)/n), u$ on the bandwidth response curve; it represents the point at which the datacenter provides enough bandwidth to saturate peers' upstream links.

The intuition behind this model is as follows. When the datacenter bandwidth allocated to a swarm is zero, the swarm doesn't receive any new block from the seeder, and thus the aggregate bandwidth is the current download bandwidth of the swarm sustained by any seeder other than the datacenter, which is responsible for injecting new blocks (a). If no other seeder is present, a drops to zero as soon as no new blocks are ready for exchange.

Our model's second interesting point occurs when the datacenter bandwidth is equal to peers' average uplink capacity, assuming $a = 0$ for simplicity (u/n). At this point, the datacenter can inject new blocks into the swarm at a rate sufficient enough to saturate the peers' uplink capacity, achieving an aggregate download speed equal to the peers' aggregate upload speed. From here, saturated uplinks render peers unable to redistribute blocks to other peers. Any addition to the datacenter bandwidth capacity benefits only the peer receiving this increment, and the curve starts to flatten until clients' downlinks saturate. At this point, the swarm reaches its maximum aggregate bandwidth, and any further bandwidth provisioned to the seed won't improve download performance.

Optimization Problem and Implementation

The previous function closely resembles the bandwidth response curve of a swarm comprising

heterogeneous peers. Given that our main goal is to maximize all swarms' aggregate bandwidth, we can thus state our problem as the following constrained optimization problem:

$$\begin{aligned} \max\ & F(W_1,\ldots,W_m) = \sum_{s\in S} f_s(W_s), \\ \text{where}\ & f_s(W_s) = a_s + \frac{(d_s - a_s)W_s}{W_s + C_s} \\ \text{subject to}\ & \sum_{s\in S} W_s \le W \\ & W_s \ge 0 \end{aligned} \tag{1}$$

The constraints restrict the solution space to those allocations that don't exceed the datacenter's bandwidth budget. An optimal solution $W^* = [W_1^*,\ldots,W_m^*]$ exists for the optimization problem because the objective function is continuously differentiable, strictly increasing, and concave. Such optimization problems have been studied extensively in the literature[5,6] and provide a computationally efficient solution using Lagrange multipliers, as the algorithm in Figure 3 shows. We omit the details of the proofs and the mathematical development due to space constraints.

This algorithm's computational complexity is $O(m \log m)$, dominated by the initial sorting algorithm. In practice, this algorithm would recompute bandwidth allocations every time any of our model's parameters change, which happens only when swarm membership changes — for instance, a client leaves or join the swarm, or a client becomes a seeder.

Require: W ▷ Datacenter bandwidth budget
Require: $\{(u_1, d_1, a_1, c_1), \ldots, (u_m, d_m, a_m, c_m)\}$ ▷ Parameters of the m swarms
Sort increasingly the set of swarms by its marginal value $\frac{d_i - a_i}{c_i}$
Compute largest k such that

$$\frac{\sqrt{c_k(d_k - a_k)}}{\sum_{i=1}^{k}\sqrt{c_i(d_i - a_i)}}\left(W + \sum_{i=1}^{k} c_i\right) - c_k \ge 0$$

Set $w_j = 0$ for $j > k$, and for $1 \le j \le k$, set:

$$w_j = \frac{\sqrt{c_j(d_j - a_j)}}{\sum_{i=1}^{k}\sqrt{c_j(d_j - a_j)}}\left(W + \sum_{i=1}^{k} c_i\right) - c_j$$

return $(w_1, \ldots, w_m)$

Figure 3. Datacenter bandwidth allocation algorithm. This algorithm would recompute bandwidth allocations every time any of our model's parameters change, which occurs only when swarm membership changes.

Evaluation

We successfully integrated the BitTorrent protocol into our open source personal cloud storage client, which uses OpenStack Swift as a storage back end. Moreover, we evaluated our smart seeding strategy in both a simulation and a real setting using PlanetLab nodes.

For our simulation experiments, we used reasonable values for the BitTorrent protocol (2 Gbytes per file, 64 Kbytes per chunk, 30 seconds between optimistic unchokes, and 10 seconds between regular unchokes) and selected upload and download bandwidths from a distribution used in other works involving BitTorrent.[7] Our simulation consisted of 300 swarms whose membership sizes were drawn from the distribution Zipf ($z = 2.4$), which leads to a small number of big swarms and a higher number of small swarms — a typical distribution for file popularity. We included three other strategies for comparison purposes: equal sharing, which grants the same datacenter uplink capacity to all swarms ($W_i = W/m$); proportional sharing, which allocates datacenter bandwidth in proportion to each swarm's size in terms of peers $W_i = \left(n_i \Big/ \sum_j^m n_j\right) * W$; and the Antfarm strategy,[1] the most similar system to our mechanism in terms of objectives — that is, maximizing aggregate bandwidth.

During the initial phase of operation, the Antfarm mechanism allocates a small amount of the seeder bandwidth to every swarm, and then allocates the remaining bandwidth in small increments to swarms with the highest increase in aggregate bandwidth since the last update. This way, Antfarm slowly builds response curves for each swarm and obtains an estimate of swarm performance as a function of the seeder bandwidth. This approach computes the response curve by fitting a piecewise-linear function to the set of measurements it takes periodically, producing a shape similar to our model (see Figure 2). In steady state, Antfarm uses a greedy hill-climbing algorithm to allocate bandwidth to swarms with the highest gradient.

Figure 4a shows how swarms' aggregate bandwidth evolves as we increase the datacenter bandwidth budget. Figure 4b presents our strategy's speedup gains using the equal sharing strategy as a baseline. When datacenter bandwidth is scarce, our mechanism outperforms equal and proportional sharing by a factor of 55× and 45×, respectively, in the best case. These differences are reduced as the seeder bandwidth becomes less congested. Our solution also outperforms the

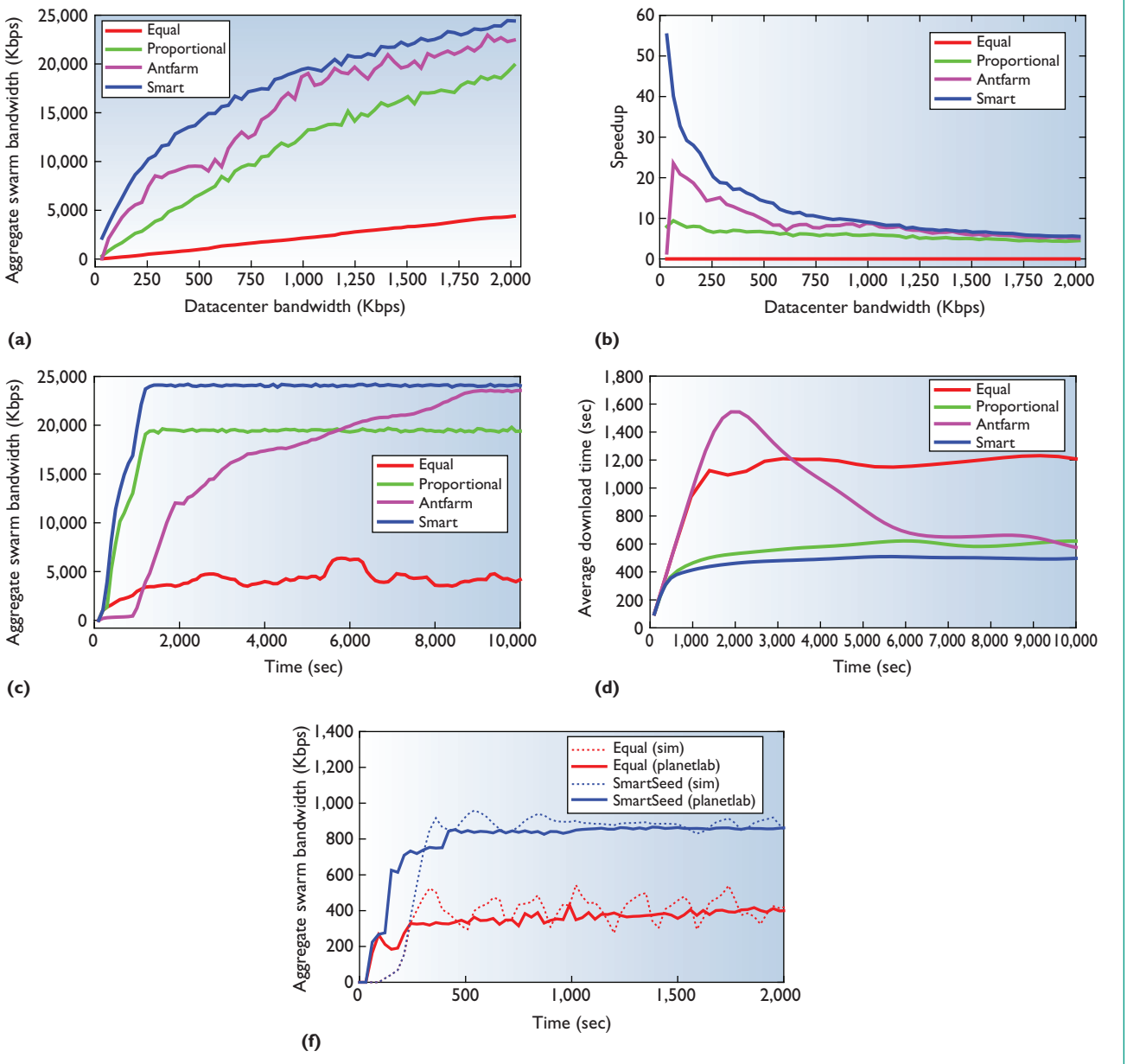

Figure 4. Experimental evaluation. Our smart seeding strategy outperforms other strategies in simulation. We measured (a) aggregate bandwidth, (b) speedup, (c) aggregate bandwidth evolution, and (d) average download time. (e) We also conducted a simulation vs. a PlanetLab experiment.

Antfarm strategy no matter what the datacenter bandwidth is, achieving a more stable aggregate throughput in steady state. The Antfarm variability comes from the BitTorrent protocol's dynamic nature, which leads to inaccurate measurements. This inaccuracy doesn't always guarantee an optimal allocation. In fact, during the initial phase, the small bandwidth increments might not go to the swarm that can obtain the higher benefit.

Another important metric we evaluated was convergence time. Figure 4c shows the evolution of the aggregate bandwidth of swarms for different strategies during a simulated period of 10,000 seconds when the datacenter bandwidth is 2,048 kilobits per second (Kbps). We can see that the smart seeding, equal, and proportional strategies converge to a steady state in a few minutes. This is because all the available

datacenter bandwidth is split among different swarms — following different strategies — at the beginning of the measurement period. In contrast, the Antfarm strategy takes a longer time to converge to a steady state — an order of magnitude six times higher — because of the slow initial phase, confirming previous findings.[2] This slow initial phase worsens if the available datacenter bandwidth is higher.

To determine the benefits from a user experience viewpoint, we evaluated the evolution of the average download time for the allocation mechanisms under study (Figure 4d), in this case, using small files (16 Mbytes) and a datacenter bandwidth equal to 400 Kbps. We compute the average using the download time of peers that completed their download, as well as the resident time for peers with downloads under progress.[2] When a peer finishes downloading a file, a new peer joins the swarm to maintain its membership size. As expected, the smart seeding, equal, and proportional strategies quickly stabilize to their steady state, whereas the Antfarm strategy takes considerably longer to achieve comparable download times because of its initial phase. Note that even in steady state, our allocation mechanism provides lower download times (20 percent reduction in the worst case) compared to other solutions.

Finally, to validate our simulations, we deployed a real prototype using the planetary-scale testbed PlanetLab. The setup comprised eight different swarms (each sharing a single file) with membership sizes of 14, 5, and 2, and 5 singleton swarms. We limited peers' upload bandwidth to 50 Kbps and the seeder bandwidth capacity to 128 Kbps. Using this setup, we compared our smart seeding strategy with the equal sharing strategy to assess the performance gains for around 30 minutes. As Figure 4e shows, our smart seeding strategy outperforms the equal sharing strategy by a factor of 3× in this specific scenario. We confirmed our simulation results by comparing them to the results obtained from an experiment with real nodes on PlanetLab using the same setup (solid lines), which faithfully matched our simulations (dashed lines).

Users expect to receive Internet content in a responsive and timely manner from providers, without perceiving unnecessary delays. However, these requirements can be hard to achieve when the provider relies on a cloud infrastructure, and costs can drive the service to be nonprofitable. We propose seamlessly and transparently integrating a well-known P2P protocol such as BitTorrent into the datacenter storage service to reduce bandwidth consumption while providing higher throughput to clients compared to traditional client-server approaches. Moreover, a simple formalization of swarm dynamics lets our algorithm allocate bandwidth among competing swarms to improve timely content delivery and overall performance compared to other approaches at Web scale.

Following this line of work, our solution could be used to provide distinct quality of service to different content depending on economic parameters or service-level agreements with clients. We believe that our integrated solution is a step forward to reducing operational costs transparently and improving the responsiveness and performance that users perceive when dealing with large-scale content distribution. ◻

Acknowledgments

This work has been partially funded by the EU in the context of the CloudSpaces project (FP7-317555).

References

1. R. Peterson and E.G. Sirer, "Antfarm: Efficient Content Distribution with Managed Swarms," *Proc. 6th Usenix Symp. Networked Systems Design and Implementation*, vol. 9, 2009, pp. 107–122.
2. A.R. Abhigyan Sharma and A. Venkataramani, "Pros and Cons of Model-Based Bandwidth Control for Client-Assisted Content Delivery," *Proc. 6th Int'l Conf. Communication Systems and Networks*, 2014, pp. 1–8.
3. B. Cohen, "Incentives Build Robustness in BitTorrent," *Proc. Workshop Economics of Peer-to-Peer Systems*, vol. 6, 2003, pp. 68–72.
4. G. Gonçalces et al., "Modeling the Dropbox Client Behavior," *Proc. IEEE Int'l Conf. Communications* (ICC 14), to appear, 2014.
5. X. León and L. Navarro, "A Stackelberg Game to Derive the Limits of Energy Savings for the Allocation of Data Center Resources," *Future Generation Computer Systems*, vol. 29, no. 1, 2013, pp. 74–83; www.sciencedirect.com/science/article/pii/S0167739X12001306.
6. M. Feldman, K. Lai, and L. Zhang, "The Proportional-Share Allocation Market for Computational Resources," *IEEE Trans. Parallel and Distributed Systems*, vol. 20, no. 8, 2009, pp. 1075–1088.
7. R. Rahman et al., "Improving Efficiency and Fairness in P2P Systems with Effort-Based Incentives," *Proc. 2010 IEEE Int'l Conf. Communications* (ICC 10), 2010, pp. 1–5.

Xavier León is a postdoctoral fellow in the Department of Computer Engineering and Mathematics at the Universitat Rovira i Virgili, Spain. His research interests include computer networks, complex systems, and self-organization of distributed and decentralized peer-to-peer systems through economics-based mechanisms. León received a PhD in computer architecture from the Universitat Politècnica de Catalunya (UPC). Contact him at xavier.leon@urv.cat.

Rahma Chaabouni is a PhD student in the Department of Computer Engineering and Mathematics at Universitat Rovira i Virgili, Spain. Her research interests include distributed systems and cloud computing. Chaabouni received a BSc in computer software engineering from the National School of Engineers of Sfax, Tunisia. Contact her at rahma.chaabouni@urv.cat.

Marc Sánchez-Artigas is a lecturer in the Department of Computer Engineering and Mathematics at the Universitat Rovira i Virgili, Spain, and helps coordinate the EC FP7 project CloudSpaces on personal cloud storage. His research interests include building massive distributed computing systems and clouds, and novel storage infrastructures for the cloud. Sánchez-Artigas received a PhD in information and communication technologies with European Mention at Universitat Pompeu Fabra, Spain. Contact him at marc.sanchez@urv.cat.

Pedro García-López is a professor in the Department of Computer Engineering and Mathematics at the Universitat Rovira i Virgili, Spain, leads the Architectures and Telematic Services (AST) research group, and coordinates the EC FP7 project CloudSpaces. His research topics are distributed systems, peer-to-peer, cloud storage, software architectures and middleware, and collaborative environments. García-López received a PhD in computing from the Universidad de Murcia. He's a member of IEEE. Contact him at pedro.garcia@urv.cat.