Leveraging P2P overlays for Large-scale and Highly Robust Content Distribution and Search

Manal El Dick

Leveraging P2P overlays for Large-scale and Highly Robust Content Distribution and Search

Manal El Dick
supervised by Esther Pacitti
ATLAS Group, INRIA and LINA, University of Nantes, France
manal.el-dick@univ-nantes.fr

ABSTRACT

In the last decade, there has been a tendency of shifting content distribution towards peer-to-peer (P2P) technology. The reason behind this is the self-scalability of P2P systems provided by the principles of communal collaboration and resource sharing in P2P systems. By building a *P2P Content Distribution Network* (CDN), peers collaborate to distribute the content of under-provisioned websites and to serve queries for large audiences on behalf of the websites. When designing a *P2P CDN*, the main challenge is to actually maintain an acceptable level of performance in terms of client-perceived latency and hit ratio while minimizing the incurred overhead. This is not a straightforward endeavor given that the P2P CDN relies on autonomous and dynamic peers rather than a dedicated infrastructure. Indeed, the distribution of duties and content over peers should take into account their interests in order to give them proper incentives to cooperate. Moreover, the P2P-CDN should adapt to increasing numbers of participants and provide robust algorithms under high levels of churn because these issues have a key impact on performance. Finally, the routing of queries should aim peers close in locality and serve content from close-by providers to achieve short latencies. This paper gives an overview of our contributions in designing and maintaining a P2P CDN that tackles the issues identified above. First, we present Flower-CDN [3], a P2P content distribution network (CDN) that tackles some of these issues. Peers store only content of websites they are interested in and serve them to others. Furthermore, peers can find close-by content providers by a locality aware P2P directory structure. Secondly, we present a highly scalable approach of Flower-CDN called PetalUp-CDN which dynamically adjusts the directory structure in order to avoid overload situations and to keep the index information any peer must maintain at an acceptable level. Thirdly, we discuss maintenance protocols for Flower-CDN and PetalUp-CDN to cope with the worst scenarios of churn. The performance evaluation wrt. scalability and churn management shows that our generic approach enhances hit ratio by 40% and reduces response time by a factor of 12, compared to a well-known P2P-CDN.

VLDB '09, August 24-28, 2009, Lyon, France

1. INTRODUCTION

In the last decade, there has been a tendency of shifting content distribution towards peer-to-peer (P2P) technology. The reason behind this is mainly the inherent self-scalability of P2P systems ensured by the communal collaboration among peers: sharing both duties and benefits. In the context of content distribution, peers collaborate to redistribute the content of their favourite and under-provisioned websites for large audiences. In other words, a P2P system can provide a low-cost *Content Distribution Network (CDN)* which handles queries of the websites' clients and relieves them from their substantial query load.

When designing a *P2P CDN*, the main challenge is to actually maintain an acceptable level of performance in terms of client-perceived latency and hit ratio while minimizing the incurred overhead. This is not a straightforward endeavor given that the P2P CDN relies on autonomous and dynamic peers rather than a dedicated infrastructure. In order to give peers proper incentives to cooperate, we believe that the distribution of duties and content should take into account their interests. Several existing works like [9, 13] force peers to store content they are not interested in, which can dramatically limit the participation and thus, the system self-scalability. Under the same matter, it is obvious that peers cannot be charged with heavy workloads and thus the system should adapt to increasing numbers of participants and accordingly balance the load. Furthermore, the robustness of the P2P system plays a major role in performance. In fact, P2P networks are characterized by high levels of churn, failures and dynamic changes, which may destabilize the algorithms and severly degrade the performance in the absence of efficient detection and recovery protocols. Finally, the client-perceived latency is short only if efficient routing algorithms redirect the query to peers that are close to the client in locality and can provide the requested content. Since a P2P network abstracts all topological information about the underlying physical network, in most P2P solutions [5, 9, 11, 13, 14], queries are routed without any locality-awareness.

This paper gives an overview of our contributions in designing and maintaining a P2P CDN that tackles the issues identified above. Our work has evolved as follows. First, we have designed a P2P-CDN called Flower-CDN [3]. Flower-CDN redistributes the content of websites with large

user base by exploiting their user communities and respecting their user interests. It relies on a two-layer architecture combining gossip-based and DHT-based overlays and thereby povides a hybrid search infrastructure for the content. Flower-CDN leverages locality-awareness to ensure fast search for nearby stored copies of the requested content. Second, we have proposed PetalUp-CDN which provides Flower-CDN with high scalability and robustness. PetalUp-CDN dynamically adapts the architecture to increasing numbers of participants in order to avoid overloading any peers. Third, we have developped maintenance protocols to cope with the worst scenarios of churn, while preserving the architecture efficiency and flexibility required to achieve the goals of both Flower-CDN and PetalUp-CDN. Through simulation, we have shown that our generic approach is extremely robust in a highly dynamic environment. Moreover, it leverages larger scales to achieve higher improvements wrt. hit ratio and lookup latencies.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 gives an overview of Flower-CDN while section 4 deals with PetalUp-CDN. The maintenance protocols that ensure our protocols' robustness under churn are depicted in Section 5. Section 6 presents our performance evaluation. Section 7 concludes.

2. RELATED WORK

In this section, we briefly review the P2P works done in the areas of web caching and CDN. The P2P CDNs that are currently available for public use mainly comprise CoralCDN [4], CoDeeN [8] and CobWeb [12]. These systems are deployed over PlanetLab which provides a relatively trusted environment consisting of nodes donated largely by the research community. Basically, they rely on a network of cooperative proxy servers that distribute the content shared by the clients and handle their queries. The only P2P characteristic exhibited by these systems is the absence of centralized administration. We beleive that such systems cannot be categorized as pure P2P solutions given that they are using dedicated servers rather than exploiting client resources.

Several P2P approaches have been designed to distribute web content over peers. Many of these approaches like Squirrel [5], PoPCache [9] and Backslash [13], rely on DHT to achieve fast lookup and propose basically two types of strategies. The first one replicates web objects at peers with ID numerically closest to the hash of the URL of the object without any locality or interest considerations. The second type of strategy stores at the peer identified by the hash of the object's URL a small directory of pointers to recent downloaders of the object. Approaches adopting this strategy may be vulnerable to high churn: the directory information is abruptly lost at the failure of its storing peer. Additionnally, there are two main drawbacks in the query routing of both strategies. First, each query has to navigate through the whole DHT, which implies high load and response times. Second, unless using a locality-aware overlay combined with proactive replication, queries are served from random physical locations.

Other approaches [14, 6] use unstructured overlays for their flexibility. Proofs [14] uses an ustructured overlay in which peers' neighborhoods are continuously changing. Peers keep their requested objects and can then provide them to other participants. To locate one of the object replicas, a query is flooded to a random subset of neighbors with a *fixed time-to-live* (TTL) i.e., the max number of hops. However, searches for not-so popular objects induce heavy traffic overheads and high latencies. Moreover, neither the overlay nor the search incorporate any information about the underlying network topology to forward queries to close results. The approach proposed in [6] organises peers in subgroups and runs a gossip-based background communication based on locality-awareness. Query search can be perfomed in one hop at the cost of aggressively replicating each object. This results in a large amount of redundant and unused replication and forces the peers to store objects they are not interested in.

Finally, some works like OLP [11] and CoopNet [7] adopt a hybrid architecture where the web-server plays the role of a super-peer and manages the directory information. However, with the web-server doing the redirection of queries, scalability is limited because the servers may get quickly overloaded.

3. FLOWER-CDN

Flower-CDN [3] is a locality and interest aware P2P CDN, that supports a set W of websites ws, each of them having its own requestable content (e.g., set of web-pages and documents). Flower-CDN relies on a hybrid architecture which consists of a set of gossip-based unstructured overlays (i.e., *petals*) linked via one DHT-based structured overlay (i.e., *D-ring*). A website ws is supported as long as there are clients of ws willing to cooperate and join Flower-CDN in behalf of ws. Flower-CDN combines efficient DHT indexing to provide fast lookup with gossip robustness for replica distribution and self-monitoring. The basic idea is to connect each peer to a *petal* which represents a cluster of peers that have the same interest and reside close to each other. Peers in a petal store content of a certain website. A peer posing a query can find a close-by petal through a special directory service structured as a DHT, called *D-ring*. Petals and their connection to D-ring are maintained via low-cost gossip techniques which are inspired of P2P membership protocols [17] proven to be highly robust in face of churn.

3.1 Petals

Participant peers are grouped into k physical localities using a landmark technique [10]. The peers belonging to the same locality loc and interested in the same website ws build together a $petal(ws, loc)$. These peers, called *content peers* and noted $c_{ws,loc}$, store and provide content of ws. They may also submit queries that request content provided by ws. To support query search within each $petal(ws, loc)$, content peers periodically exchange *contacts* (i.e., addresses of other known content peers $c_{ws,loc}$) and summaries of their stored content. For this purpose, $c_{ws,loc}$ maintains a $view(ws, loc)$, which is a set of contacts from $petal(ws, loc)$. These exchanges are performed via gossip techniques and incur low communication costs since they cover close vicinities (i.e., petals).

3.2 D-ring

One peer of each $petal(ws, loc)$ is charged with the role of a *directory peer* (noted $d_{ws,loc}$): $d_{ws,loc}$ knows about all content peers $c_{ws,loc}$ and indexes their stored content in a *directory-index*(ws, loc_i). Directory peers are also embedded in *D-ring*, the DHT-structured overlay. D-ring adopts a novel key management service that leverages interests and

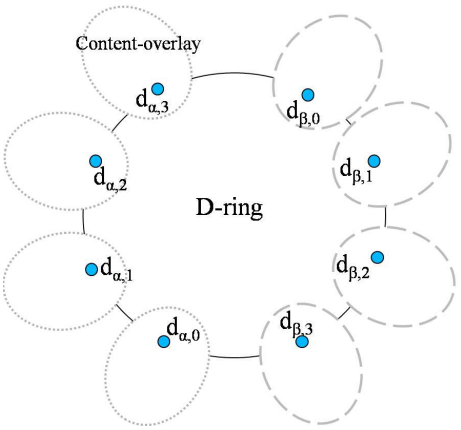

Figure 1: Flower-CDN architecture with websites α and β and four localities

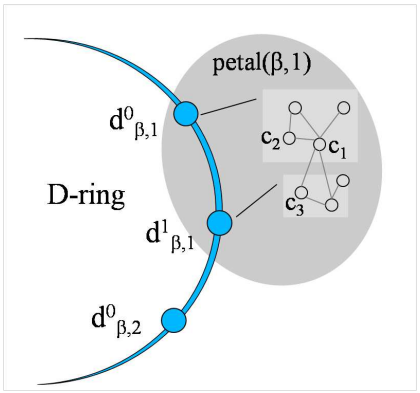

Figure 2: Example of $petal(\beta, 1)$ in PetalUp-CDN

localities of peers. Basically, we assign each directory peer $d_{ws,loc}$ a specific peer ID, based on ws and loc (i.e., one D-ring ID associated to each couple (ws, loc)). As a result, directory peers for the same website have successive peer IDs and are neighbors on D-ring.

D-ring provides two functionalities. First, it supports queries coming from new clients of W. Instead of querying server ws, a client located in loc, submits its query to D-ring and gets redirected to the directory peer in charge of ws wrt. loc. Then, $d_{ws,loc}$ tries to resolve the query based on its *directory-index*(ws, loc). The query is finally forwarded to some content peer $c_{ws,loc}$ that holds the requested content or to the original web-server. Second, D-ring serves as a fast and reliable access for new participant peers to Flower-CDN: the client can join $petal(ws, loc)$ as a content peer $c_{ws,loc}$ to resolve its own further queries and help in serving other clients' queries. To maintain the *directory-index*(ws, loc), content peers $c_{ws,loc}$ periodically send updates about their stored content to $d_{ws,loc}$ using *push messages*. Furthermore, directory peers of the same website ws may collaborate to provide content of ws. Figure 1 illustrates a simple example of Flower-CDN architecture with $W = \{\alpha, \beta\}$ and $k = 4$ localities.

4. PETALUP-CDN

Flower-CDN restricts the number of participant peers that can contribute to the system, by limiting the size of each petal. This is aimed at keeping petals at a manageable size, so that their directory peers are not overloaded with the maintenance of the overlay information. However, the P2P system may attract more participant peers than the petal capacity. To address this issue and warrant the extensive deployment of Flower-CDN to larger scale, we design *PetalUp-CDN*.

More than one directory peer for each couple (ws, loc), can consecutively join D-ring to share the management of content peers in $petal(ws, loc)$. The number of directory peers in charge of $petal(ws, loc)$ increases progressively as the number of content peers for ws in loc increases. D-ring allows up to 2^m instances of each $d_{ws,loc}$, noted $d^i_{ws,loc}$ (with $0 \leq i < 2^m$). These instances have successive D-ring IDs and are thus neighbors on D-ring. Each one manages a subset of content peers of $petal(ws, loc)$, typically by maintaining a partial directory-index and view wrt. these peers.

More precisely, queries routed over D-ring (i.e., handled by directory peers) are initiated by new clients that eventually join PetalUp-CDN as content peers. Thus, based on its query load, each directory peer $d^i_{ws,loc}$ decides when the next $d^{i+1}_{ws,loc}$ should join D-ring. A query requesting content of ws and submitted by a client in loc sequentially scans through existing directory peers $d^i_{ws,loc}$ until an underloaded directory peer is found to process the query. The load at a directory peer is evaluated in terms of the number of content peers in its view and is compared against a predefined limit. When all existing directory peers for ws in loc are overloaded, the final $d^i_{ws,loc}$ visited by the query selects from its view the content peer to join D-ring as $d^{i+1}_{ws,loc}$. The replacing content peer is then removed from the directory-index of $d^i_{ws,loc}$.

Only the directory peer $d^i_{ws,loc}$ that processes the query adds the client to its directory-index and view as a content peer $c_{ws,loc}$. An example of PetalUp-CDN configuration is illustrated in Fig. 2 which focuses on $petal(\beta, 1)$. Two directory peers $d^0_{\beta,1}$ and $d^1_{\beta,1}$ share the management of $petal(\beta, 1)$. Thus, they manage each one a subset of the content peers $c_{\beta,1}$.

Once its query satisfied, the client becomes a content peer of the petal and does not use D-ring anymore to route its queries. To enable content sharing throughout each $petal(ws, loc)$, $c_{ws,loc}$ gossips to any other $c_{ws,loc}$ of its petal. Thus, in Fig. 2, c_1 can gossip to both c_2 and c_3 and eventually benefit from their stored content to satisfy its queries. But how does c_1 get to know content peers like c_3 that are controlled by other directory peers?

Actually, a new $d^{i+1}_{ws,loc}$ uses its view and content summaries maintained while still a content peer of $d^i_{ws,loc}$, until its old view expires (see Sec. 5.1) and gets progressively replaced by a new view related to newly arrived clients. When receiving first clients, $d^{i+1}_{ws,loc}$ provides them with a subset of its old view so that they initialize their view of $petal(ws, loc)$. Thereby, these clients that will become content peers get to know content peers of $d^i_{ws,loc}$ and eventually introduce them to other content peers of $d^{i+1}_{ws,loc}$ via gossip.

5. MAINTENANCE PROTOCOLS

In this section, we discuss the maintenance protocols proposed for Flower-CDN and PetalUp-CDN to ensure their ro-

bustness in highly dynamic environments governed by churn.

5.1 Maintenance in Petals

To efficiently handle the received queries, a directory peer should maintain a fresh directory-index despite the dynamic changes frequently occuring in its petal.

This can be achieved through two features: *keepalive* and *push* messages. More precisely, $c_{ws,loc}$ regularly sends keepalive messages to $d_{ws,loc}$ which can therefore discover and remove expired pointers from its view and directory-index. Also, $c_{ws,loc}$ sends updates about its stored content to its $d_{ws,loc}$ using push messages whenever the percentage of its changes reaches a threshold.

Given that several directory peers may coexist within the same petal, each content peer of $petal(ws, loc)$ restricts its communications to the directory peer $d^i_{ws,loc}$ via which it joined the petal. For this purpose, $c_{ws,loc}$ maintains *dir-info* which holds information about $d^i_{ws,loc}$: the address and peer ID of $d^i_{ws,loc}$ as well as an *age* field. The age is a value incremented periodically by $c_{ws,loc}$ and reset to zero upon each contact with $d^i_{ws,loc}$, to detect the availability of $d^i_{ws,loc}$. Whenever two content peers gossip to each other, they also exchange their *dir-info*. If the exchanged *dir-info* share the same peer ID, then the 2 content peers belong to the same directory peer. In such a case, they both keep the *dir-info* with the smaller age, which refers to more recent information about their directory peer. Thus, whenever a directory peer leaves, some of its content peers that detect it when trying to contact it, gossip the information to the other content peers concerned with this particular directory peer so that they update their *dir-info*.

5.2 Maintenance in D-ring

Normally, DHT overlays recover from churn (i.e., failures, leaves, joins) by reorganizing the DHT and redistributing the stored data accordingly. However, our system adopts its own maintenance protocols to preserve D-ring structure.

5.2.1 Failures and Leaves

The leave or failure of a directory peer of ws wrt. loc is detected by its content peers while sending keepalive or push messages. The replacement is performed by the first peer related to ws and loc that detects the failure/leave, i.e., a content peer from $petal(ws, loc)$ or a new client.

5.2.2 Joins and Replacements

A peer p can try to join D-ring as a directory peer either in case it is initially (1) a content peer or (2) a new client. Case (1) occurs when p is replacing its failed directory peer or when it joins as $d^{i+1}_{ws,loc}$ due to its petal's growth. Case (2) only happens if p has found no directory peer available for ws in loc while routing its query over D-ring, because p is the first/only participant for $petal(ws, loc)$ or directory peers of $petal(ws, loc)$ have left D-ring and have not been replaced yet. However, p does not always succeed in joining because several peers may simultaneously target the same vacant position; the one that first integrates into D-ring, succeeds.

Similarly to the standard join in DHT-based overlays, p routes a join message with a key equal to $ID^i_{ws,loc}$ where $ID^i_{ws,loc}$ is the ID of the directory peer position targetted by p. If the targetted position is not vacant, the join message reaches the current $d^i_{ws,loc}$ and p discovers its current directory peer to update its *dir-infor*. Then, if p is a new client, it simply joins $petal(ws, loc)$ as a content peer. If the targetted position is vacant, p becomes $d^i_{ws,loc}$ and gradually contructs its view and directory-index as its content peers discover its join and send it push messages.

As introduced in Sec. 5.1, content peers discover the join of p as they try to contact their previous directory peer $d^i_{ws,loc}$ and detect its leave. Then, some of them will try to join, detect that there is already a new directory peer and update their dir-info. Subsequently, the information about the new $d^i_{ws,loc}$ spreads rapidly via gossip to content peers related to $d^i_{ws,loc}$.

If the previous $d^i_{ws,loc}$ had voluntarily left, it would have transferred a copy of its view and directory-index to p before its departure. Moreover, in case p used to be a content peer before joining D-ring, p can try to answer first received queries from its content summaries.
Note that $d^i_{ws,loc}$ eventually constructs its routing table by exchanging messages with its neighbors as normally done in DHT overlays.

6. PERFORMANCE EVALUATION

In this section, we present an evaluation of scalability and robustness through simulation. For this purpose, we use 3 metrics: (1) **Hit ratio** is the fraction of queries successfully served from the P2P system; (2) **Lookup latency** is the latency taken to resolve a query and reach the destination that will provide the requested object; (3) **Transfer distance** is the network distance, in latency, from the querying peer to the peer that will provide the requested object. Because of the simulator limitations in terms of memory constraints, we could only simulate up to 5000 peers, which does not lead to petals of large size. Thus, we evaluated the management of churn and scalability in Flower-CDN. We believe that the results can be safely generalized for PetalUp-CDN. In the following, we first argue the choice of simulation parameters, then we discuss the simulation results.

6.1 Simulation Setup

Our simulation relies on PeerSim [1]. The event-driven framework of PeerSim enables us to model the latency of each individual link but does not support simulation of bandwidth and CPU resources. Thus, we generate an underlying topology of peers connected with links of variable latencies between 10 and 500 ms. Also, we model $k = 6$ localities using a landmark-based technique [10]. The simulation parameters are summarized in Table 1.

We choose Chord [15] as our DHT-based overlay and we simulate its routing and churn stabilization protocols. On top of Chord, we implement the key management service of D-ring. We compare Flower-CDN with Squirrel [5] which shares some similarities with Flower-CDN wrt. the directory structure.

For our query workload, we use synthetically generated data because available web traces reflect object accesses while we are interested in website accesses. Each website provides 500 objects which are requestable and cacheable[1]. We apply Zipf distribution for object requests submitted to each website [2].

[1]we do not deal here with cache issues such as cache expiration and replacement policies

Table 1: Simulation Parameters

Parameter	Values
Latency (ms)	10-500
Nb of localities (k)	6
Nb of websites ($\|W\|$)	100
Mean population size (P)	2000; 3000; 4000; 5000
Total network size	$P * 1.3$
Mean uptime of a peer (m)	60 min
Nb of objects/website	500
Query rate at a peer	1 query every 6 min
Push threshold	0.5
Gossip/keepalive period	1 hour

For a realistic simulation environment, we simulate churn based on a study [16] where P2P population converges to a desired size, P. For this purpose, the arrival rate of peers must be equal to the mean departure rate, $\frac{P}{m}$, where m denotes the mean uptime of a peer. We model the uptime of a peer as an exponential distribution with $m = 60$ minutes, resulting in a high churn rate. We assume that a peer always fails (i.e., when its lifetime expires) and never leaves normally, to test Flower-CDN in highly unstable scenarios. Moreover, a peer might re-join multiple times during an experiment, each time with a different uptime.

Each experiment is run for 24 hours, which are mapped to simulation time units. Initially, each peer is randomly assigned a website from $|W|$ to which it has interest throughout the experiment. We start with a population of $k*|W| = 600$ directory peers which have limited uptimes and form the initial D-ring (i.e., one directory peer per couple (website, locality). After a small warm-up period, the population stabilizes around P as new clients keep on arriving and existing peers on failing. In order to keep the load at bay, we restrict the query generation to 6 *active* websites of W. For all other *non-active* websites, peers are only involved with churn because it affects D-ring routing. More precisely, a peer with interest for an active website submits queries on a regular basis, as soon as it arrives until it fails (see query rate in Table 1). In opposition, a peer belonging to a non-active website, is simply added to its petal upon its arrival; it is only involved in the failure management of its petal's directory peer.

We assume that a content peer has enough storage potential to avoid replacing its content through the experiment's duration. As a peer only stores content it has requested, this is a reasonable assumption given the usual browsing activity of individual users. A peer only poses queries for objects unavailable in its local storage (i.e., it never issues the same query more than once). In Table 1, *push threshold* refers to the percentage of new changes beyond which a content peer launches a push exchange with its directory peer. We do not limit the *view size* of a content peer and allow it to grow with the size of its petal which never surpasses 30 in the current configuration; also, when a peer selects a contact for gossip and finds it unavailable, the peer removes the contact from its view, which naturally bounds the view size. Finally, *gossip/keepalive period* which refers to the periodicity of gossip and keepalive messages sent by a content peer is calibrated at 1 hour based on Flower-CDN requirements.

6.2 Simulation Results

We want to assess the robustness of Flower-CDN under churn and evaluate its scalability wrt. the population size. For this purpose, we conduct three simulations, each one targetting a different population size (i.e., $P = 2000$, 3000, 4000, 5000) in the context of a highly dynamic environment. We compare the performance of Flower-CDN to that of Squirrel.

6.2.1 Robustness to churn

In this first set of experiments, we set $P = 3000$ and study our evaluation metrics under this setting.

First, we analyse the evolution of hit ratio with time (Fig. 3). At the beginning, Squirrel surpasses Flower-CDN wrt. hit ratio. This is because Flower-CDN needs a warm up period to build up and enable its petals to get populated, given that query search space involves specific petals to achieve locality-awareness. In contrast, Squirrel searches the whole overlay for queries and its hit ratio increases faster than that of Flower-CDN. However, as the impact of churn becomes more significative, Squirrel fails to preserve an increasing hit ratio while Flower-CDN keeps on improving despite of failures: the improvement reaches 40% after 24 simulation hours. In fact, in Squirrel, the information about previous downloaders (i.e, the location of content copies spread in the P2P system) which is held in a directory, is abruptly lost with the failure of the directory peer in charge of it. In contrast, Flower-CDN efficiently manages this problem because periodic updates are disseminated throughout a petal via gossip and push exchanges. Thus, a new directory peer d can progressively reconstruct its directory-index as it receives updates from content peers. Meanwhile, d can resolve first queries using content summaries previously received during gossip exchanges, given that a failed directory is replaced by a content peer.

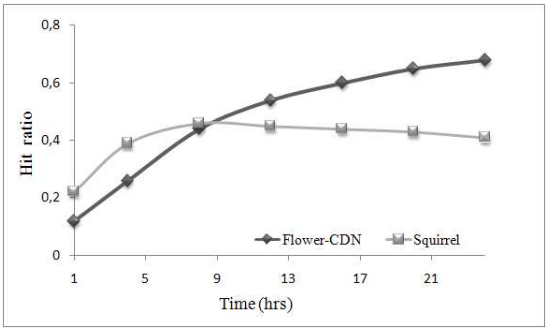

Figure 3: Comparing hit ratio between Flower-CDN and Squirrel

Second, we look at the distribution of queries wrt. lookup latency and transfer distance for $P = 3000$. Figure 4 shows that 66% of our queries are resolved within 150 ms while 75% of Squirrel's queries take more than 1200 ms. Figure 5 shows that the percentage of queries served from a distance within 100 ms is 62% for Flower-CDN and 22% for Squirrel. Thus, Flower-CDN preserves its highly significant locality-aware gains under the worst scenarios of failures, given that the directories lost with Squirrel can be fastly recovered with Flower-CDN.

6.2.2 Scalability

We study the behavior of Flower-CDN wrt. scalability and we summarize the results in Table 2 for lack of space.

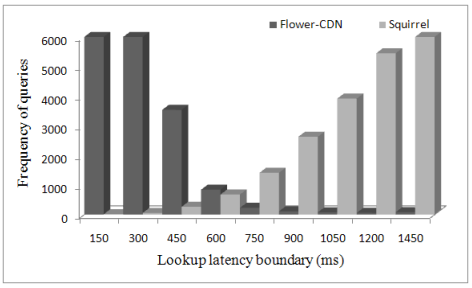

Figure 4: Lookup latency distribution

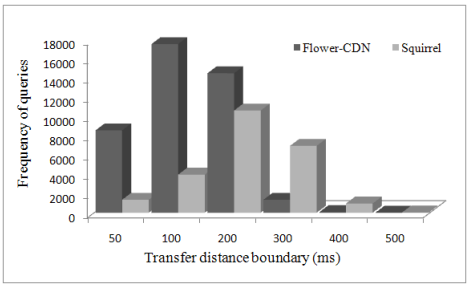

Figure 5: Transfer distance distribution

We can see that Flower-CDN leverages larger scales to achieve higher improvements. Actually, a larger population size enables Flower-CDN to build up faster and converge faster to a maximum hit ratio, going beyond 0.7 after 24 simulation hours.

Table 2: Scalability in Flower-CDN and Squirrel

P	approach	hit ratio	lookup	transfer
2000	Squirrel	0.35	1503 ms	163 ms
	Flower-CDN	0.63	167 ms	120 ms
3000	Squirrel	0.41	1544 ms	166 ms
	Flower-CDN	0.68	152 ms	92 ms
4000	Squirrel	0.45	1596 ms	169 ms
	Flower-CDN	0.7	138 ms	88 ms
5000	Squirrel	0.52	1596 ms	165 ms
	Flower-CDN	0.72	127 ms	81 ms

Furthermore, when a petal has more content peers submitting queries and becoming providers of the requested content, searches in this petal will have larger scopes and thus are more likely to be resolved within this petal. That is why large scales are also advantageous for search speed and localization of close results in Flower-CDN. Compared with Squirrel, the improvement factor increases with scale and can reach 12.6 for the average lookup latency and 2 for the average transfer distance. Therefore, Flower-CDN can drastically reduce lookup latency and transfer distance in comparison with Squirrel.

7. CONCLUSION

This paper gives an overview of our contributions in the context of P2P-CDN. Through our work, we have proposed two protocols: Flower-CDN and PetalUp-CDN. On the one hand, Flower-CDN effectively leverages interests and localities to build a hybrid P2P infrastructure that supports content distribution and search. On the other hand, PetalUp-CDN provides adaptive scalability and high robustness under churn and dynamic changes. Our protocols are combined whith maintenance protocols that preserve the efficiency and the performance at a high level despite the worst scenarios of churn. Simulation results showed that our generic approach resists successfully to churn and leverages higher scales to achieve higher improvements; in summary, hit ratio is ameliorated by 40% and response times reduced by a factor of 12, in comparison with an existing P2P-CDN. We are currently deepening the analytical and empirical analysis of our protocols. In the future, we plan to explore sophisticated search functionalities wrt. semantic and personalized search.

8. REFERENCES

[1] Peersim: a P2P simulator. http://www.peersim.sourceforge.net.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. In *IEEE INFOCOM*, 1999.

[3] M. E. Dick, E. Pacitti, and B. Kemme. Flower-CDN: a hybrid P2P overlay for efficient query processing in CDN. In *EDBT*, 2009.

[4] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *USENIX/ACM NSDI*.

[5] S. Iyer, A. I. T. Rowstron, and P. Druschel. Squirrel: a decentralized P2P web cache. In *PODC*, 2002.

[6] P. Linga, I. Gupta, and K. Birman. A churn-resistant P2P web caching system. In *ACM SSRS*, 2003.

[7] V. N. Padmanabhan and K. Sripanidkulchai. The case for cooperative networking. In *IPTPS*, 2002.

[8] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson. The dark side of the Web: an open proxy's view. *ACM SIGCOMM Comput. Commun. Rev.*, 34(1), 2004.

[9] W. Rao, L. C. 0002, A. W.-C. Fu, and Y. Bu. Optimal proactive caching in P2P network: analysis and application. In *CIKM*, 2007.

[10] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *IEEE INFOCOM*, 2002.

[11] Y.-S. Ryu and S.-B. Yang. An effective P2P web caching system under dynamic participation of peers. *IEICE Transactions*, 88-B(4), 2005.

[12] Y. J. Song, V. Ramasubramanian, and E. G. Sirer. Optimal resource utilization in content distribution networks. Technical report, Cornell University, 2005.

[13] T. Stading, P. Maniatis, and M. Baker. P2P caching schemes to address flash crowds. In *IPTPS*, 2002.

[14] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust P2P system to handle flash crowds. In *IEEE ICNP*, 2002.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a scalable P2P lookup service for internet applications. In *ACM SIGCOMM*, 2001.

[16] D. Stutzbach and R. Rejaie. Characterizing churn in P2P networks. Technical report, University of Oregon, 2005.

[17] S. Voulgaris, D. Gavidia, and M. Steen. Cyclon: Inexpensive membership management for unstructured P2P overlays. *J. Network Syst. Manage.*, 13(2), 2005.