A Highly Robust P2P-CDN Under Large-Scale and Dynamic Participation

Manal El Dick, Esther Pacitti, Bettina Kemme

A Highly Robust P2P-CDN Under Large-Scale and Dynamic Participation

Manal El Dick, Esther Pacitti
Atlas Group, Inria and Lina
University of Nantes, France
{*manal.el-dick, esther.pacitti*}*@univ-nantes.fr*

Bettina Kemme
School of Computer Science
McGill University, Montreal, Canada
kemme@cs.mcgill.ca

***Abstract*—By building a P2P Content Distribution Network (CDN), peers collaborate to distribute the content of underprovisionned websites and to serve queries for larger audiences on behalf of the websites. This can reveal very challenging, given the highly dynamic and autonomous participation of peers. Indeed, the P2P-CDN should adapt to increasing numbers of participants and provide robust algorithms under churn because these issues have a key impact on performance. Also, the distribution of tasks and content over peers should take into account their interests in order to give them proper incentives to cooperate. Finally, the routing of queries should aim peers close in locality and serve content from close-by providers to reduce network overload and achieve scalability. We have previously proposed a locality and interest-aware P2P-CDN, *Flower-CDN*, that lacks efficient management of robustness and scalability. In this paper, we focus on these crucial shortcomings and propose *PetalUp-CDN*. The performance evaluation wrt. scalability and churn shows highly significant gains.**

I. 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 *scalability* provided by P2P systems at low costs. It is commonly believed that P2P is naturally suited for handling large-scale applications, due to its inherent *self-scalability*: as more peers join the system, they contribute to the aggregate resources of the P2P network. In the context of content distribution, peers collaborate to redistribute the content of some websites for large audiences, basically by storing content copies. This means that peers build a *P2P Content Distribution Network* (CDN) to which queries are redirected in order to relieve the web-servers from their substantial query load.

However, building a P2P-CDN that makes efficient use of scalability can reveal extremely challenging, given the autonomous and dynamic partcipation of peers. We investigate this issue under three angles. First, we argue that the distribution of tasks and content over peers should take into account their interests in order to give them proper incentives to cooperate. Several works like [7], [10] 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. Second, since a P2P network abstracts all topological information about the underlying physical network, the P2P-CDN must incorporate locality-awareness to provide fast lookup to nearby stored copies of the requested content. This feature contributes significantly in achieving scalability, as the consumption of network resources is kept at bay. In most existing approaches [4], [7], [9], [10], [11], queries are routed without considering whether the requested content is available in a peer close to the requestor in locality. Finally, we are concerned with the *robustness* of the P2P system under failures and dynamic changes that massively occur in P2P networks. In fact, the participation of peers is highly dynamic, implying thousands of continuous joins and leaves, which creates the effect of churn. It is crucial that every P2P-CDN should be able to maintain an acceptable level of performance despite churn.

In [3], we presented Flower-CDN, a locality and interest-aware P2P-CDN. In the context of a large-scale application, the main limitation of Flower-CDN is that peers providing access to the system for new participants may become a bottleneck, which severely threatens the scalability of Flower-CDN. Another crucial limitation that was not extensively studied in [3] consists of churn management which dramatically affects hit ratio and response times of queries. Aiming at high scalability and robustness, this paper brings three main contributions: (1) a highly scalable P2P-CDN called PetalUp-CDN which dynamically adapts to increasing numbers of participants in order to avoid overload situations; (2) a maintenance protocol for PetalUp-CDN to cope with the worst scenarios of churn, while preserving the architecture efficiency and flexibility; (3) an empirical analysis of scalability and robustness under churn.

II. Related Work

We briefly review the main P2P-CDNs and discuss their contributions wrt. robustness and scalability. In [9], [6], the web-server provides a centralized directory service by redirecting each query to previous downloaders of the requested object. With the web-server doing all the redirection, the system scalability is dramatically limited. [6] does not address churn management but tries to incorporate location-awareness in the web-server directory. [9] proposes a strategy that models the object lifetime to guide the web-server's redirection under churn, but it does not consider localities of clients. [11] uses an unstructured and dynamically constructed overlay where peers keep their requested objects to provide them to other participants. Query search is performed via flooding, which induce heavy traffic and affects scalability. The approach in [5] organises peers in groups and runs gossip communication based on locality-awareness. Query search can be perfomed in one hop at the cost of aggressive and redundant replication, without any consideration for interests of peers. Several approaches [4], [7], [10] rely on DHT-structured overlays and store for each requested object either a copy or a directory of pointers to recent downloaders of the object. The storing peer is identified by the hash of the object's identifier without any locality or interest considerations. A query navigates through the DHT and then receives a pointer. Such approaches may be vulnerable to high churn because the directory information is abruptly lost at the failure of its storing peer.

III. PetalUp-CDN

This section introduces the design and construction of PetalUp-CDN which rely on Flower-CDN architecture.

A. Flower-CDN architecture

In Flower-CDN, each client participates on behalf of a website with content it likes; it selects one of k predefined physical localities based on some latency measurements. The group of peers in the same locality loc and interested in the same website ws forms $petal(ws, loc)$. These peers, called *content peers* ($c_{ws,loc}$), store and provide content of ws. One peer of each $petal(ws, loc)$ is charged with the role of a *directory peer* ($d_{ws,loc}$): $d_{ws,loc}$ stores addresses of all content peers $c_{ws,loc}$ in $view(ws, loc)$ and indexes their stored content in *directory-index*(ws, loc). Directory peers are also embedded in the DHT-based overlay, *D-ring*.

D-ring fulfills two roles. First, it handles queries of new clients. Instead of querying ws, a client in loc submits its query to D-ring and gets redirected to $d_{ws,loc}$ which resolves the query based on its directory-index. Second, D-ring serves as a reliable access for new participants: the client joins $petal(ws, loc)$ as a content peer $c_{ws,loc}$ where it can resolve its further queries and help in serving other clients.

Each $petal(ws, loc)$ provides a search infrastructure for queries of content peers $c_{ws,loc}$. For this purpose, within $petal(ws, loc)$, content peers gossip to exchange *contacts* (i.e., addresses of other known content peers $c_{ws,loc}$) and summaries of their stored content. Thus, each $c_{ws,loc}$ maintains a partial $view(ws, loc)$ of its $petal(ws, loc)$, initially obtained from either its directory peer $d_{ws,loc}$ or another content peer $c_{ws,loc}$. $petal(ws, loc)$ expands progressively as more clients of ws in loc join the P2P system.

B. Design of PetalUp-CDN

PetalUp-CDN is designed in a way that enables the extensive deployment of a petal while keeping the load on its directory peer at bay. More than one directory peer for each couple (ws, loc), can consecutively join D-ring.

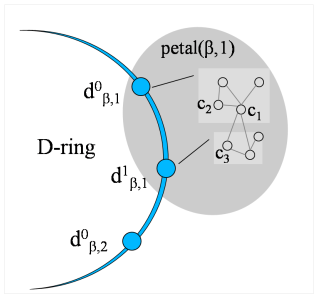

Figure 1. Example of $petal(\beta, 1)$ in PetalUp-CDN

D-ring adopts a novel key management service that leverages interests and localities. In Flower-CDN, D-ring assigns to $d_{ws,loc}$ a peer ID that concatenates the ID of ws and the ID of loc. To manage scalability in PetalUp-CDN, another ID of m additional bits (called *scalable ID*) is suffixed to the peer ID. We obtain 2^m consecutive peer IDs for each couple (ws, loc) instead of only one. Thus, we may have up to 2^m instances of each $d_{ws,loc}$, noted $d^i_{ws,loc}$ $(0 \leq i < 2^m)$. All directory peers for same website and locality have successive peer IDs and are neighbors on D-ring.

Each directory peer $d^i_{ws,loc}$ manages a partial view noted $view(ws, loc)^i$ and thereby a partial *directory-index*$(ws, loc)^i$ of $petal(ws, loc)$. More formally, for each website ws and locality loc, we have two properties:

Property 1: $\forall i, j : view(ws, loc)^i \cap view(ws, loc)^j = \emptyset$

Property 2: $petal(ws, loc) = \bigcup_{0 \leq i < 2^m} view(ws, loc)^i$

By having multiple directory peers in charge of a petal, the failure of one or more of these directory peers will not lead to a complete loss of directory information, and will allow the system to continue in a slightly-reduced capacity. An example of PetalUp-CDN configuration is illustrated in Fig. 1 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 $c_{\beta,1}$.

C. Construction of PetalUp-CDN

The construction of PetalUp-CDN involves progressive expansions of D-ring and its petals.

1) D-ring expansion: A new directory peer is created for $petal(ws, loc)$ when the number of content peers $c_{ws,loc}$ can no more be manageable by the existing directory peers $d^i_{ws,loc}$. In other words, directory peers of $petal(ws, loc)$ are created sequentially, starting from $d^0_{ws,loc}$. Recall that queries routed over D-ring are initiated by new clients that eventually join the petals. Thus, in PetalUp-CDN, a query targeting $petal(ws, loc)$ scans through the existing directory peers $d^i_{ws,loc}$ in search for an underloaded directory peer that can resolve the query and take in charge the client as a new content peer. If no such directory peer is found, the latest created $d^i_{ws,loc}$ initiates the join of a new $d^{i+1}_{ws,loc}$.

A query routed over D-ring uses a key in which the website and locality IDs reflect the client's information. To determine the scalable ID, we consider the optimal route that the query should follow while scanning the directory peers of its targeted petal. First, the number of query redirections required to reach an underloaded directory peer should be minimized in order to limit query response time. Second, no directory peer should be overloaded with query redirections. Indeed, if contacted by every new client of its petal, $d^i_{ws,loc}$ can become overloaded even if its is just redirecting queries to other directory peers. Thus, as directory peers share the management of directory information, they should also share the handling of new queries. Therefore, an optimal route can be achieved if each client can discover the number of directory peers that have been created so far for its petal and randomly choose one of them to contact it. When no such discovery scheme is available, we use a safe alternative by picking for the scalable ID a random value between 0 and its middle value. For instance, if the scalable ID is formed of 2^3 bits, the scalable ID takes a value between 0 and 4. Consider a query with $ID^4_{ws,loc}$. If $d^4_{ws,loc}$ does not exist, the DHT routing protocol delivers the query to the first preceding directory peer (i.e., $d^i_{ws,loc}$ with $0 \leq i < 4$) because the latter has the closest ID to $ID^4_{ws,loc}$. In such a case, the query would have reached the latest created directory peer which can locally process the query or create a new directory peer for $petal(ws, loc)$ if overloaded. If $d^4_{ws,loc}$ does exist, the query gets to $d^4_{ws,loc}$ which keeps on redirecting the query to further directory peers of $petal(ws, loc)$ until an underloaded directory peer is found or created. This redirection approach shortens the route of the query and distributes load rather evenly accross directory peers.

Whenever the query reaches a directory peer $d^i_{ws,loc}$ of the targeted petal, d_{ws,loc_i} processes the query based on its view size. If the view size has reached a predefined limit called $maxDirectory$, $d^i_{ws,loc}$ verifies if $d^{i+1}_{ws,loc}$ is in D-ring. In case $d^{i+1}_{ws,loc}$ exists , $d^i_{ws,loc}$ redirects the query to $d^{i+1}_{ws,loc}$ which in its turn processes it. In case $d^{i+1}_{ws,loc}$ does not exist, $d^i_{ws,loc}$ selects from its view a content peer to join D-ring as $d^{i+1}_{ws,loc}$. The content peer is then removed from the view and directory-index of $d^i_{ws,loc}$. Afterwards, in order to avoid waiting for $d^{i+1}_{ws,loc}$ to join, $d^i_{ws,loc}$ processes the query, in its stead, based on its directory-index and eventually forwards the query to some content peer that holds o_{ws}. Consequently, $d^i_{ws,loc}$ adds the client to its directory-index as a provider of o_{ws} and to its view as a content peer $c_{ws,loc}$. If the view size has not reached $maxDirectory$ yet, $d^i_{ws,loc}$ performs the same steps to resolve the query and add the new client.

In consequence of the above, a new client is only added to the view and directory-index of one specific directory peer, which achieves Properties 1 and 2.

2) Petal expansion: 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. 1, 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 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.

IV. Robustness Under Churn

A. Maintenance of Connection

Flower-CDN mechanisms are achieved via the connection between D-ring and the petals. Thus, a primary concern is to maintain this connection despite the highly dynamic environment. Given that several directory peers coexist within the same petal, we have to maintain a connection between each $d^i_{ws,loc}$ and the subset of content peers in its $view(ws, loc)^i$.

More precisely, each content peer of $petal(ws, loc)$ restricts its communications to the directory peer $d^i_{ws,loc}$ via which it joined the petal. $c_{ws,loc}$ 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 $d^i_{ws,loc}$.

To keep the directory peer connected to its content peers, we exploit a feature inherent to P2P systems, *keepalive messages*, which are periodically sent to check links between peers. $c_{ws,loc}$ regularly sends keepalive messages to $d^i_{ws,loc}$ which can therefore discover and remove expired pointers from its view and directory-index. Moreover, given that a content peer may request and access new content, $c_{ws,loc}$ sends updates about its newly stored objects to $d^i_{ws,loc}$, using *push messages*. $c_{ws,loc}$ monitors the changes (i.e., the newly stored objects) and sends them in a push message whenever they reaches a predefined threshold. In Fig. 1, c_1 which is linked to $d^0_{\beta,1}$, only sends push and keepalive messages to $d^0_{\beta,1}$. Additionally, a content peer may need to evict some locally stored objects via a cache replacement policy (e.g., LRU) because the content is generally stored in a local cache with a limited storage size. Such object evictions are also reported to $d^i_{ws,loc}$ as new changes via push messages.

Two content peers that gossip to each other 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 other concerned content peers that can thus update their *dir-info*.

B. Maintenance of 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.

1) Failures and Leaves: A directory peer may leave D-ring at any moment. The leave of $d^i_{ws,loc}$ is detected by its content peers, i.e., contained in its $view(ws, loc)^i$, while sending keepalive or push messages. PetalUp-CDN replaces $d^i_{ws,loc}$ by a peer that shares the interest in the same website's content and belongs to the same locality, i.e., a content peer from $view(ws, loc)^i$ or a new client. If $d^i_{ws,loc}$ leaves voluntarily, it selects from its view a content peer to replace it. Otherwise, any content peer of *view*$(ws, loc)^i$ can perform the replacement as soon as it detects the failure.

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 participant of $petal(ws, loc)$ or directory peers of $petal(ws, loc)$ have left D-ring and have not been replaced yet. In all cases, p does not always succeed in joining because several peers may simultaneously target the 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 $ID^i_{ws,loc}$, the ID of the directory peer position targeted by p ($i = 0$ for new peers). If the targeted 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-info*. Then, if p is a new client, it simply joins $petal(ws, loc)$ as a content peer. If the targeted 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. IV-A, 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 to content peers via gossip.

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.

When $petal(ws, loc)$ shrinks because ws has lost its popularity, directory peers of $petal(ws, loc)$ merge their views, then they withdraw from D-ring, leaving only one directory peer to manage $petal(ws, loc)$. For lack of space, we do not detail this algorithm and leave it for future work.

V. Performance Evaluation

For our performance evaluation, 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. Additionnally, we analyse the overhead of our approach by measuring the average traffic in bps experienced by a content or directory peer due to its received and sent messages of gossip or push.

A. Preliminary Discussion

This evaluation focuses on robustness and scalability provided by the maintenance protocols. It aims at quantifying the performance in serving queries under dynamic participation. On this issue, it sounds reasonable to state that the performance of having one or multiple directory peers per petal will be similar (due to the lack of space we cannot deepen this discussion). This is because the maintenance protocols manage dynamicity in order to ensure that the query search infrastructure common to both approaches is not disrupted by bottlenecks or failures. Regarding the overhead, the gossip behavior of content peers is unaffected whether one or more directory peers manage the petal. Due to memory constraints, we could simulate up to 11000 peers, which lead to small petals and prevented the creation of multiple directory peers per petal. Thus, we can generalize that one or multi-directory performance is equivalent.

B. Simulation Setup

Our simulation relies on PeerSim [1]. 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 [8]. We choose Chord [12] as our DHT-based overlay. We compare PetalUp-CDN with Squirrel [4] that relies on one DHT-structured overlay. For each requested object, a peer identified by the DHT without locality or interest considerations stores a directory of pointers to recent downloaders of the object. A query always navigates through the DHT and then receives a pointer. 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 whith Zipf popularity distribution [2]. We do not deal with cache expiration policies.

For a realistic simulation environment, we simulate churn based on a study [13] 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 PetalUp-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 mapped to simulation time. 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 (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 *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 (i.e., 1 query every 6 min), as soon as it arrives until it fails. A peer of a non-active website, is simply added to its petal upon its arrival and involved in the failure management of its 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). 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 50 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.

C. Simulation Results

1) Robustness to churn: We conduct for both Squirrel and PetalUp-CDN the same experiment which targets a mean population size of 3000. First, we analyse the evolution of hit ratio with time (Fig. 2). At the beginning, Squirrel surpasses PetalUp-CDN wrt. hit ratio. This is because PetalUp-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 PetalUp-CDN. However, as the impact of churn becomes more significant, Squirrel fails

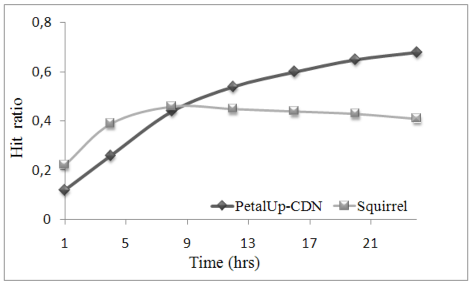

Figure 2. Comparing hit ratio evolution

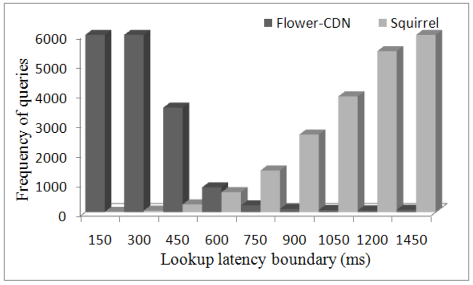

Figure 3. Lookup latency distribution

to preserve an increasing hit ratio while PetalUp-CDN keeps on improving: the improvement eventually reaches 40%. In fact, in Squirrel, the directory information is abruptly lost with the failure of the directory peer in charge of it. PetalUp-CDN efficiently manages this problem because periodic updates are disseminated in a petal via gossip and push. Thus, a new directory peer d can progressively reconstruct its directory-index as it receives updates from content peers. Meanwhile, d resolves first queries using content summaries previously received during gossip exchanges, given that a failed directory is replaced by a content peer.

Second, we compare the lookup latency and transfer distance between Squirrel and PetalUp-CDN. Figure 3 shows the distribution of queries wrt. lookup latency: 66% of our queries are resolved within 150 ms while 75% of Squirrel's queries take more than 1200 ms. A major reason behind these significant gains is the query routing algorithms of PetalUp-CDN that rely on shortcuts provided by the petals. In contrast, Squirrel has to route every single query through the whole DHT, leading to high lookup latencies.
Figure 4 shows the distribution of queries wrt. transfer distance: the percentage of queries served from a distance within 100 ms is 62% for PetalUp-CDN and 22% for Squirrel. This is because PetalUp-CDN focuses its query search within the same petal of the querying peer (or client) in order to localize a close-by copy of the queried object.

2) Scalability: For each approach (i.e., PetalUp-CDN and Squirrel), we conduct 5 experiments, each one targeting a different population size P in the context of a highly dynamic environment. For each experiment, we collect the

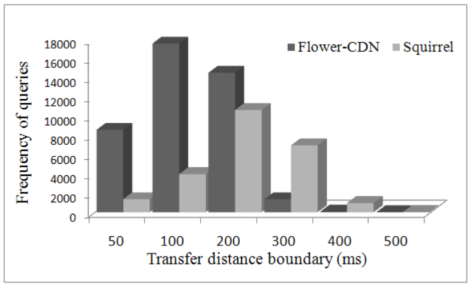

Figure 4. Transfer distance distribution

P		HIT RATIO	AVG LOOKUP	AVG TRANSFER
3000	Squirrel	0.41	1544	166
	PetalUp-CDN	0.7	178	107
5000	Squirrel	0.52	1596	165
	PetalUp-CDN	0.72	141	89
7000	Squirrel	0.58	1618	167
	PetalUp-CDN	0.78	160	91
9000	Squirrel	0.59	1692	165
	PetalUp-CDN	0.79	156	87
11000	Squirrel	0.62	1743	164
	PetalUp-CDN	0.83	143	84

Table I
SCALABILITY COMPARISON

hit ratio obtained after 24 simulation hours, and the average lookup latency and transfer distance for a query. To avoid over-fitted results, we run each experiment 3 times and compute the average hit ratio, lookup latency and transfer time for this experiment. We also measure for PetalUp-CDN the gossip & push overhead per peer. The results of the 5 experiments are summarized in Table I.

We can see that the hit ratio of PetalUp-CDN increases from 0.7 to 0.82 when increasing P from 3000 to 11000. PetalUp-CDN leverages larger scales to achieve higher gains. Actually, a larger population size enables PetalUp-CDN to build up faster and converge faster to a maximum hit ratio. Also, PetalUp-CDN maintains its improvement over Squirrel through different population sizes.

By comparing results of lookup latency and tranfer distance between PetalUp-CDN and Squirrel, we observe that the improvement factor increases with scale and reaches 12 for lookup latency and 2 for transfer distance. 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 locally. That is why large scales are advantageous for search speed and localization of close results in PetalUp-CDN.

Finally, the results of gossip & push overhead (not represented for lack of space) show that a peer experiences around 90 bps due to its exchanges. This is very low bandwidth that could be sustained even by modem connections, which proves that PetalUp-CDN incurs very acceptable overhead via its highly effective gossip protocols.

VI. CONCLUSION

In this paper, we proposed PetalUp-CDN that aims at providing a P2P-CDN with effective scalability and high robustness under dynamic participation of peers. PetalUp-CDN relies on a P2P directory service that ensures a locality-aware redirection of clients towards the content of their interest. Aiming at large-scale audiences, the P2P directory service dynamically adapts to avoid overload situations and warrant the extensive deployment of PetalUp-CDN. PetalUp-CDN is combined with a maintenance protocol that preserves the efficiency and the performance at a high level despite the worst scenarios of churn. Simulation results showed that our generic approach successfully resists to churn and leverages higher scales to achieve higher improvements.

REFERENCES

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

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

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

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

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

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

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

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

[12] 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.

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