2012 11th International Symposium on Parallel and Distributed Computing

Stabilizing peer-to-peer systems using public cloud: A case study of peer-to-peer search

Janakiram Dharanipragada
DOS lab, Dept. of CSE
Indian Institute of Technology(IIT), Madras
Chennai, India
Email: djram@iitm.ac.in

Harisankar Haridas
DOS lab, Dept. of CSE
Indian Institute of Technology(IIT), Madras
Chennai, India
Email: harisankarh@gmail.com

***Abstract*—Co-operative peer-to-peer systems have lot of relevance due to their desirable properties like lack of centralized control and transparency. But in certain applications, the stability of peer-to-peer systems is affected when concentrated load spikes occur. In this work, we explore the case for cloud-assisted peer-to-peer systems to handle spikes using peer-to-peer search as a case study. We identify the issues involved in realizing cloud-assisted peer-to-peer systems and propose, implement and evaluate Cloud-Assisted Peer-to-Peer Search(CAPS) architecture which fits in with the co-operative nature of peer-to-peer systems. The experimental results show that CAPS provides stability to peer-to-peer search service during query spikes without affecting user experience adversely.**

***Keywords*-peer-to-peer computing; search engines;**

I. Introduction

Co-operative peer-to-peer systems is a class of distributed systems where the different participating nodes(*peers*) share symmetric responsibilities. Thus the peers who are consumers of a service collaborate in realizing the service also. In popular peer-to-peer applications like Faroo[1], the application is distributed across millions of autonomous peers over the internet. This is in contrast to centralized systems where a server(or a set of servers), mostly managed by a single company, provides a service while users only consume the service. The distributed and autonomous nature of the peers realizing a peer-to-peer service enables it to have several desirable properties over centralized systems *by design*. Since the application and associated data are distributed across all the peers participating in the service, it provides transparency and avoidance of centralized control. As the peers share the idle computing power on their already "on" machines, there is negligible cost to realize the service and potential power savings compared to a centralized system.

The peers participating in a peer-to-peer system share only their idle resources and join/leave independently. Moreover, they are connected to each other through low bandwidth internet. Due to these reasons, the stability of certain applications is affected when a concentrated load spike occurs. In case of search, popular news events lead to spikes in the queries related to the event. For example, rate of queries related to "Bin laden" increased by 10,000 times within one hour in Google on May 1, 2011 after Operation Geronimo[1]. The high update nature of web content and low bandwidth available at individual peers make it tough to handle such spikes in a peer-to-peer search engine leading to instability(the details are discussed in Section III).

[1]http://www.faroo.com, accessed: February, 2012

Public clouds like Amazon Web Services offer on-demand access to elastic cloud resources in a pay-as-you-go model. In this work, we explore the case for cloud-assisted peer-to-peer systems for handling spikes, using peer-to-peer search as a case study. In cloud-assisted peer-to-peer systems, elastic cloud resources are used transiently to alleviate concentrated load spikes in peer-to-peer systems. However, this involves satisfying several important considerations as explained below.

Before part of the load in the system can be offloaded to the cloud, the relevant portions of the application state have to be transferred to the cloud. This becomes an important issue in data intensive applications like search. The state transfer has to be done fast as the load spike will soon increase to a level which cannot be handled by the peers. However, it also has to be ensured that the transfer process itself does not deplete the already scarce resources like bandwidth in the peers. Cloud-assistance should be enabled seamlessly without affecting the end user experience adversely. A payment model for the cloud resources which fits in with the cooperative nature of peer-to-peer systems has to be worked out. Finally, it has to be ensured that the use of centrally managed cloud resources will not undermine the desirable properties of peer-to-peer systems like transparency, distributed control, etc.

In this paper, we study the above considerations associated with cloud-assisted peer-to-peer systems in the context of peer-to-peer search and propose Cloud-Assisted Peer-to-peer Search(CAPS). Background on peer-to-peer search is described in Section II. The effect of query(load) spikes on the stability of peer-to-peer search systems is described in Section III. The technical challenges involved in realizing CAPS and the proposed architecture are explained in Section IV. A payment model for the cloud resource consumption in CAPS is proposed in Section V. Concerns on the effect of centralized cloud usage on the desirable properties of peer-to-peer search are addressed in Section VI. Experimental evaluation in a realistic setting is described in Section VII. It demonstrates the potential of CAPS to alleviate transient load spikes in peer-to-peer search. We survey related works in Section VIII and conclude in Section IX.

II. Background on peer-to-peer search

The desirable properties of peer-to-peer architecture have led to lot of research interest in peer-to-peer search techniques(see [2] for a survey) and development of real-world peer-to-peer web search engines(Yacy[2],Faroo). Faroo claims to have more than 2 million participating peers[3] and provides realtime search functionality.

Search engines crawl the content available in the web which includes web pages, tweets, etc. The stored content is processed which involves several operations like extracting the terms and computing frequency of terms within documents. Other information like hyperlink structure are also captured. An index(*inverted index*) is created which consists of a mapping between each term in the content collection and the list of identifiers of the documents having the term(*posting list*). Information required for ranking of results like term frequency is also stored along with each document identifier in the posting lists. For a single term query, the top-k results in the posting list associated with the query term can be returned, where k could be the number of results which could be displayed in a page. For a multi-term query, the posting lists associated with each of the query terms are intersected to get the list of documents containing all the query terms. These documents are ranked and the top-k results are returned to the user. Peer-to-peer search engines perform the above processes in a distributed manner by partitioning the index among the peers.

In term-partitioned peer-to-peer search(used in popular peer-to-peer search engines like Yacy and Faroo), the search index is partitioned term-wise and the responsibility of the portion of the index associated with each term or term-combination is assigned to individual peers. The term-to-peer mapping is realized in a distributed manner by organizing the peers using a Distributed Hash Table(DHT)(e.g., Chord[3]). DHTs provide mapping from keys to unique peers in a fully distributed manner. The unique peer responsible for a key can be identified by sending a lookup message in the DHT. The responsibility of keys change when peers join and leave, but the mapping from a key to a unique peer is always maintained. The lookup message returns the address of the responsible peer after traversing through at most $O(logN)$ peers, with high probability, where N is the total number of peers. Further details on DHT are available in [3].

In DHT-based term-partitioned peer-to-peer search, each term is hashed to generate a key. The key associated with a term is used to assign the responsibility of the term to a unique peer in the DHT. The processing of documents is performed collaboratively by all the peers in the network. A peer processing a document, extracts the terms from the documents after pre-processing. For each term, an update message is then sent to the peer responsible for the term, containing the document id and additional metadata required like score(measure of relevance), document creation time, etc. The peer responsible for the term accepts the update messages and maintains the posting list of the term. The queries on the term are sent to the responsible peer which sends the results back, based on the term posting list.

In basic term-partitioned peer-to-peer search, for a query containing multiple terms, the posting lists of all terms in the query need to be intersected. Since, these posting lists are present in different peers, the intersection has to be performed over the network leading to high bandwidth costs. Though several optimizations like using bloom filters have been suggested to address this problem, the bandwidth usage for intersection is still high[4]. But for recurring queries, the results for the queries can be precomputed and maintained separately to reduce the bandwidth overhead as in [4]. Thus, a unique peer is responsible for each recurring query. The peer responsible for a query is identified through DHT by using the hash of the query as key. The update messages related to newly created documents relevant to the query need to be forwarded to the peer to keep the result list fresh. Each update message contains the document id, score of the document for the query and additional metadata like document creation time depending on the ranking scheme. In the rest of the paper, we assume that in the basic peer-to-peer search scheme, a single peer is responsible for a spiking query.

The DHT-based peer-to-peer search model described above is illustrated in Figure 1. Peers A to H are some of the peers participating in a DHT-based peer-to-peer search network. Peer A is responsible for query q, based on the hash value of q. Peer A receives and processes updates and queries on q from other peers. The query on q sent from peer H reaches peer A after getting forwarded through peers F and C. Similarly, updates on q from peers E and G reach peer A after passing through peers D,B and F,C respectively.

III. Effect of query spikes in peer-to-peer search

In this section, we consider a 10,000 times spike[1] in query load and show its impact on the stability of a DHT-based peer-to-peer search system.

[2]http://yacy.net, accessed: February, 2012

[3]http://www.faroo.com/hp/p2p/p2p.html, accessed: February, 2012

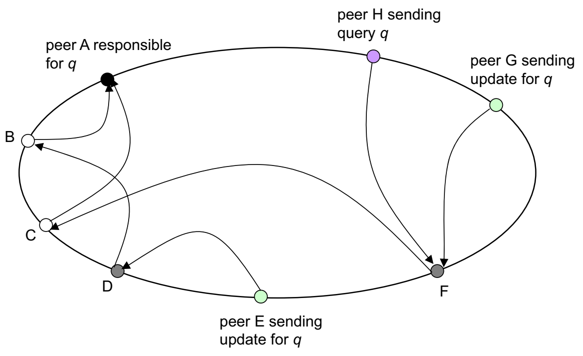
peer H sending query q
peer A responsible for q
peer G sending update for q
B
C
D
F
peer E sending update for q

Figure 1: DHT-based peer-to-peer search model.

In term-partitioned peer-to-peer search, each peer is responsible for a set of queries/terms which are assigned to it based on the DHT mapping scheme. At steady state, the peers will be processing the query/update messages related to all the queries/terms, it is responsible for. Consider a peer P responsible for query Q, which started spiking, resulting in a 10,000 times increase in its query rate within 1 hour. Thus, if the steady state query rate of Q is r, after the spike the query rate becomes $10000r$ in 1 hour. If the query rate of Q at P is not maintained close to the original rate(rate before spike, r), the processing of other queries as well as the queries related to Q will get affected, resulting in poor user experience. Next, we consider the adequacy of traditional methods like caching and replication to handle the spike.

Caching: Caching of the results of Q could be performed on other peers traversed in the query path to reduce the load at P(for example, in Figure 1, the results for queries on q could be cached on peers B and C for reducing query load on peer A). But, if excessive caching is performed for a query with an associated high update rate(content creation rate), the result freshness will be adversely affected. The freshness of results is becoming more important currently with the search engines indexing realtime contents from social media like twitter which have a high update rate(e.g., Bing social search[4], Faroo real-time peer-to-peer search).

Spikes in query rate are accompanied by a corresponding spike in the content creation rate due to the increased user interest in the popular event. If the update rate is 1% of the query rate(a conservative estimate as the overall update to query ratio in twitter is above 8%[5]), it can be observed that, at most 99% of the queries can be served from the caches without affecting the result freshness. This could reduce the query rate of Q incident on P to $(1/100)^{th}$ of the actual rate. Thus, the incident query rate of Q on P reduces to $100r$. But, there is still an excess query load of $99r$ on peer P. Thus, caching alone cannot handle a popularity spike without affecting user experience. Next, we consider whether index replication on other peers could handle the excess $99r$ query load at peer P.

Caching + replication: In index replication, multiple peers are made responsible for the same query. Hence, we need to replicate the result list and update messages of the spiking query on the additional peers. But, the bandwidth available at each peer is limited(shared with other applications) and each peer may be already handling other queries for which it is responsible for. Thus, each peer will be able to handle only a limited amount of excess query load. Hence, a large number of peers will be required to handle the excess query load fully. For example, we need at least 99 peers which can handle an excess query load of r or 990 peers which can handle an excess query load of $(r/10)$. Copies of the result list and each update message has to be sent to all the peers handling the spike, incurring bandwidth overhead. Also, discovering large number of peers which currently have excess bandwidth, within a short span of time, incurring reasonable overhead is a challenge. Thus, even caching combined with replication using other peers will be insufficient to handle huge concentrated spikes in query load.

IV. Cloud-assisted peer-to-peer search(CAPS)

In this section we first describe the technical challenges involved in the transient use of public clouds to stabilize peer-to-peer search. Next we describe the CAPS architecture and its components which address the challenges involved.

In CAPS, the responsibility of a spiking query is switched temporarily from the responsible peer to cloud for maintaining stability. The decision making on when to switch as well as the switching of responsibility need to be done fast as the peer responsible for a spiking query will not be able to handle an increasing query spike for long. Switching of query responsibility from a peer to cloud involves transfer of the result list to cloud as well as updating the peer-to-peer routing structure for routing the queries to peer or cloud appropriately. An important challenge involved in implementing switching is to ensure that the user experience is not adversely affected during switching. Specifically, the response time for queries as well as the quality of results should not be affected while the index transfer and routing changes are being carried out. Only a small portion of the bandwidth available at the peer should be used for performing switching because excessive usage of bandwidth for switching will affect the query processing at the peer. Further, it is desirable that the traditional methods for load reduction(caching and replication) can work simultaneously with cloud assistance.

Architecture: In CAPS, public cloud temporarily processes the queries and updates related to a spiking query and provides the results for the queries. The CAPS architecture

[4]http://www.bing.com/social, accessed: February, 2012

[5]http://engineering.twitter.com/2010/10/twitters-new-search-architecture.html, accessed: February 2012

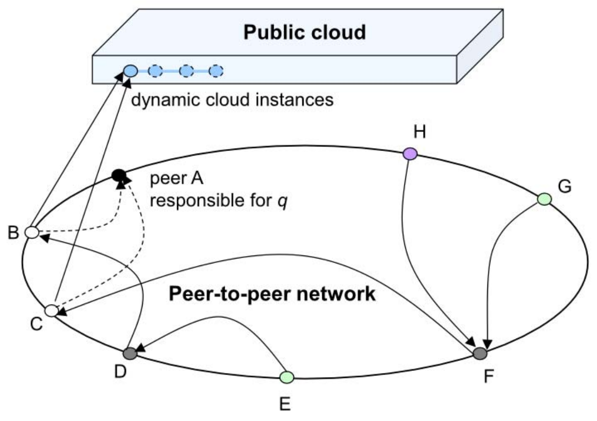

Figure 2: Cloud-assisted peer-to-peer search architecture

is depicted in Figure 2. In this work, we have considered “Infrastructure as a Service”(IaaS) offered by many of the public cloud providers(e.g., Amazon EC2) as the underlying cloud service used. IaaS allows to create and turn off dynamic cloud instances when required with a pay-as-you-go pricing model. But, in future, the public clouds could offer search as a service specialized for on-demand query processing with appropriate abstractions for indexing and querying. The DHT-based routing in peer-to-peer search is modified to route queries to peers or cloud appropriately. CAPS can be realized in an existing term-partitioned peer-to-peer search system by adding two components: switching decision maker and switching implementor. The details of the two additional components are explained below.

Switching decision maker: The switching decision maker component at the peer responsible for a query makes the decision on when to switch the responsibility of the query to cloud, and back. The decision to switch needs to consider various factors like query/update rate, cloud status, cloud payment model, required QoS, etc. The relevant parameters involved in making the switching decision are monitored by the component. Since, the switching criteria could be different for different peer-to-peer search applications, the application-specific switching policy based on these parameters is included in the search application. Examples for simple switching criteria could be: “switch when the query rate exceeds threshold X”, “switch when query rate increased by X% in the last Y seconds”. After the decision to switch a query is made, the switching implementor component switches query responsibility to a public cloud instance. The cloud instances for a spiking query can be created on public cloud availability zones which are geographically closer to the area with maximum interest in the spiking query. This results in faster response times and lower network load. More elaborate switching criteria, considering cloud payment model, required QoS, etc. are the subject of future work. The decision to switch the responsibility of query back from cloud to peer is made by the switching decision maker component, based on the switching policy, after monitoring the cloud instance remotely.

Switching implementor: The task of the switching implementor component is to switch the responsibility of a query to the public cloud and back, after the respective decisions are made. The switching implementor component needs to transfer the relevant index associated with the switched query to the cloud, and modify the query/update routing to reflect the same. The switching of query responsibility has to be performed fast, without depleting the available bandwidth at the peer, and without affecting the result freshness and response times.

In DHT, each query/update message reaches the peer responsible for it after traversing through $O(logN)$ other peers. The routing paths of messages are defined based on the peer and message identifiers. At any time, each peer has $O(logN)$ preceding peers which forward DHT messages to it directly. Henceforth, these peers are referred as preceding peers of a peer(For example in Figure 2, peers B and C are the preceding peers of peer A responsible for query q). The query/update messages reach a peer only through its preceding peers. The switching algorithm to be executed at the peer responsible for the query to be switched is shown in Algorithm 1. The query to be switched($query$), handle(IP address or hostname) of the dynamic cloud instance(handle of $cloud\ instance$) and the maximum bandwidth to be used for result list transfer($bandwidth\ limit$) are supplied as parameters.

Algorithm 1 Switch query to cloud($query$, handle of $cloud\ instance$, $bandwidth\ limit$)

```
Send messages to all the preceding peers, to send further updates on $query$, to $cloud\ instance$ also.
Wait for ACKs from all the preceding peers.
Send the current version of result list of $query$, to $cloud\ instance$, using a maximum bandwidth of $bandwidth\ limit$.
Wait for ACK from $cloud\ instance$.
Send messages to all the preceding peers to send further queries on $query$ to $cloud\ instance$ alone.
```

On getting the result list and updates, the cloud instance updates its local index for the query. Thus, after performing the first three steps in the algorithm, the cloud instance will have an updated result list, exactly same as the one in the responsible peer. Thus, it can be observed that the results freshness and hence, the quality of results is not affected by the switching process. The response times of queries are also not affected as queries are not sent through additional hops during switching. Switching the responsibility of query back from cloud to the responsible peer can be done by sending messages to all the preceding peers instructing to send further updates and queries to the responsible peer alone.

The above algorithm assumes that a single instance is available at the public cloud for handling the query load. However, as the query load increases further due to the spike, more instances have to be created and the query load need to be shared among the instances through index partitioning or replication. As the internal bandwidth between the different instances within a cloud is very high, index partitioning and replication can be performed easily to enable the cloud to handle increasing query loads. Also, though multiple cloud instances may be created, only a single copy of updates and the posting list need to be sent to the cloud as they can be internally shared across the instances easily using the high bandwidth internal network. Further details on the dynamic creation, placement and management of cloud instances considering various factors like load, pricing and geographic locality are the subject of future work.

Caching of query results at preceding peers in query routing path can be performed in CAPS as the query is routed to the cloud only after passing through the preceding peers. Replication of the index on other peers can also be performed along with CAPS as some of the queries could be forwarded to the replicas from the preceding peers. Thus, existing methods for load reduction in peer-to-peer search can still be used along with CAPS. CAPS reuses the inherent fault tolerance mechanisms in DHT along with straightforward notification and forwarding mechanisms to handle peer/cloud failure and peer joining. Full details on the fault tolerance mechanisms are available in the technical report[6].

V. Cloud payment model

As the earlier works on peer-to-peer search relied on a fully collaborative model, to the best of our knowledge, there has not been any work on monetization of peer-to-peer search, which fits in with its co-operative nature. However, for CAPS, monetization is required for payment of cloud resource usage. The current centralized search engines use query-specific advertisements to generate revenue from the search service. We believe that query-specific advertisements can be extended to monetize peer-to-peer search. A basic outline for realizing this is explained below.

An ad service similar to Google adsense[7] which generates advertisements relevant for user queries can be used by the peer-to-peer search application to show advertisements relevant to the user queries. The ad service accepts a query and returns a list of advertisements relevant to the query. For each user query, the application sends the query to the ad service, in addition to the peer responsible for the query. The relevant advertisements are obtained from the ad service and displayed along with the search results obtained from the responsible peer. The user details could be anonymized from the ad service through peer-to-peer anonymization schemes. For example, the ad service can be contacted by forwarding query through multiple peers using encrypted connections as in Tor[8].

The revenue generated from the ad service can be distributed among the peers and cloud nodes which contribute to realize the search service. The relative contributions of the resources towards peer-to-peer search service by the different peers and cloud instances can be determined through peer-to-peer reputation schemes like [5]. The relative contribution information can be considered for dividing the obtained revenue among the contributing peers and clouds.

VI. Addressing concerns due to centralized cloud

Since CAPS uses centrally managed cloud resources, the associated issues need to be discussed and addressed. The search monetization model required for payment of usage of cloud resources is discussed in Section V. Additional issues are discussed below.

Centralized control: The cloud provider to use for each query can be decided dynamically before switching. The responsibility of queries can also be reassigned to other clouds or peers if required. Moreover, the public cloud does not form a core component of the search service and most of the search traffic will be handled by the peers alone. The above features prevent centralized control on the search service.

Transparency: The search application and index are distributed across the peers and clouds. During spikes, copies of the result lists available at the cloud are available at the peers responsible for the corresponding queries. Thus, the transparency of peer-to-peer search remains unaffected.

Use of dedicated resources: The use of public cloud results in additional cost and could reduce power savings from using idle computing power. But, the public cloud instances are used for handling spiking queries only and are paid in a pay-as-you-go model. Thus, the usage of dedicated resources is only a tiny fraction compared to that by centralized search engines.

VII. Evaluation

As explained in Section IV, existing methods for load reduction, caching and replication, can be used along with CAPS to maintain stability during query spikes. Further, caching and replication alone are insufficient to handle the query load during a large query spike as illustrated in Section III. Due to the above reasons, no experimental comparison was made with the existing methods for load reduction. The

[6]http://dos.iitm.ac.in/publications/2012/02.pdf
[7]www.google.com/adsense/, accessed: February 2012
[8]www.torproject.org/about/overview.html.en, accessed: February 2012

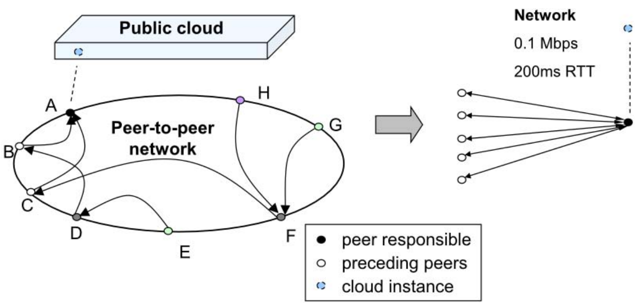

Figure 3: Experimental setup

purpose of the evaluation is to demonstrate that switching of query responsibility to the cloud can be performed to maintain stability under query spikes without affecting user experience. For this, we perform two sets of experiments. First set of experiments is to demonstrate the instability at the responsible peer during a query spike and shows how CAPS helps in maintaining stability. In the second set of experiments, we evaluate the performance of CAPS under a range of query/update rates and bandwidth conditions. In this section, we first describe the general experimental setup, then the details of both the sets of experiments performed along with the corresponding results.

Experimental setup: Since a peer-to-peer search system could consist of millions of peers, only the relevant portions of the system were considered for evaluating the switching scheme in a realistic scenario. The primary components involved in the switching scheme for a query are the peer responsible for the query, the preceding peers in the routing paths to the peer and the cloud instance available for offloading the query responsibility as shown in Figure 3. These primary components were used to demonstrate the switching of responsibility of a single query to cloud. Each of these components were implemented on separate workstations. Heterogeneous workstations were used, with configurations varying from 2-4 core processors, 8-16 GB RAM, 7500-15k HDD, etc. 6 workstations were used, where 4 were used as preceding peers in routing path, one as peer responsible for the query and one as cloud instance. To realize a realistic internet-like network, the bandwidth available(for peer-to-peer search application) on each node was limited to 0.1Mbps and an RTT of 200ms was set between all pairs of nodes using Linux kernel-level utilities. To ensure that the switching process does not deplete the bandwidth at the responsible peer, the available bandwidth for result list transfer was limited within the application(0.01Mbps in the first set of experiments). In-memory list data structures supporting concurrent access and asynchronous networking libraries were used for handling the query and update loads.

Scores assigned to documents for a query(as measure of relevance) need to consider recency along with other features like pagerank. In general, the final score of a document is calculated by combining various features using machine learning techniques. But, datasets having real document creation times and corresponding pagerank, term frequencies, etc. along with corresponding queries are not available in the public domain. Hence, for this experiment, we considered the creation times of the newly created documents alone for ranking. Thus, the list of most recent k documents indexed at the peer was returned as the result for each query. This applies directly to the ”sort by date” option available for ranking in search engines.

Due to the unavailability of required datasets in the public domain, synthetic query and update messages related to the query considered were generated, and sent from the preceding peers to the peer responsible at different rates. The corresponding results for each query were sent back to the peer which forwarded the query. Query message consisted of the query and the handle of the (preceding)peer which forwarded the query. Each update message related to a spiking query consisted of a unique document id, creation time of the document and the query. The number of results in the result list($kmax$) to be transferred to cloud was varied from 50 to 1000 in different experiments. The number of top results returned per query(k) was limited to 10. Note that, in this case, a query will return the top k relevant results first. But, additional results can be obtained by resending the query to get the next k relevant results upto a maximum of total $kmax$ results. Result freshness of the result of a query was calculated as the overlap between the observed result for each query and the ideal top-k result. Ideal top-k result for a query consists of the most recently created k documents relevant to the query assuming zero network and query processing delays.

Handling query spikes: In the first set of experiments, we demonstrate the need for cloud-assistance during a query spike. We sent increasing query and update rates to the responsible peer in two separate experiments. In the first experiment(exp 1), cloud assistance was not used(pure peer-to-peer), and in the second experiment(exp 2) using CAPS, switching of responsibility to cloud was initiated 45 seconds after starting the experiment. The aggregate rates of the queries and updates sent from the preceding peers are shown in Figure 4(a). Note that the same query and update rates were sent in both the experiments. In the second experiment, switching got completed after 82 seconds(shown in vertical dashed line). The query/update rate received at the responsible peer is shown in Figure 4(b). It can be observed that, in exp 2, query rate received at the peer dropped to zero after 82 seconds as the query responsibility got switched to cloud, while it continues to increase in case of exp 1. Figure 4(c) shows the query receiving rate at the cloud instance in exp 2. The queries start arriving at the cloud after the result list transfer is completed. Figure 4(d) shows the query drop rates in experiments 1 and 2. It can be observed that query

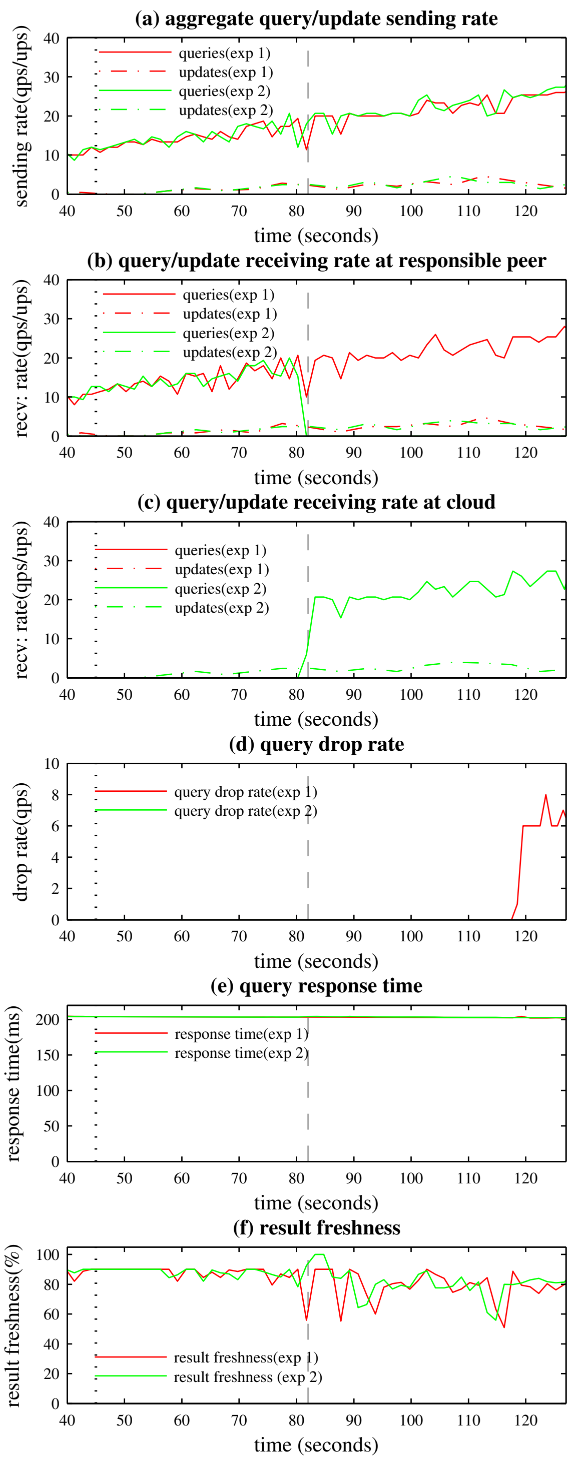

Figure 4: Effect of query spike on stability with and without cloud-assistance. CAPS is used to maintain stability in exp 2, while no cloud-assistance is used in exp 1. Vertical dotted lines show switching initiation time and vertical dashed lines show switching completion time. qps/ups denotes queries per second/updates per second.

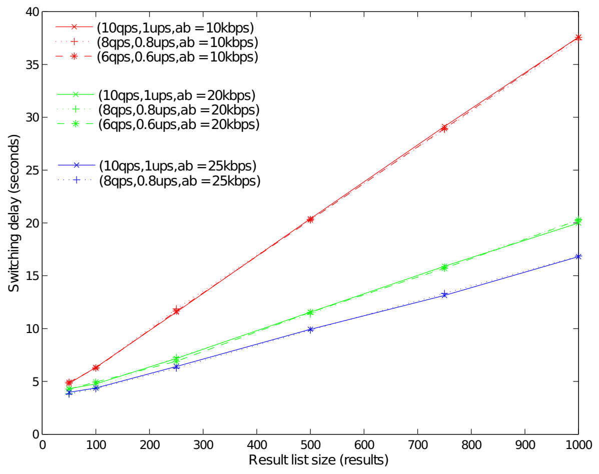

Figure 5: Variation of Switching delay with result size, query/update rate(queries/sec(qps),updates/sec(ups)) and available bandwidth for switching(ab))

dropping never occurred in exp 2(cloud-assisted) while the query starts to drop when the query rate increases beyond a point in exp 1. Figures 4(e) and 4(f) show the query response time and average result freshness respectively in both the experiments. Note that, both response time and average result freshness are unaffected by the switching process in case of exp 2. This shows that the user experience is unaffected during the switching process. Response time remains almost constant throughout while the result quality shows a slow decline with increasing query and update rates which could be an artifact of the index data structure used.

Performance of CAPS: The purpose of the second set of experiments is to study the performance of CAPS under different parameters. We study the time taken for switching and the effect of switching on user experience under a range of query/update rates and bandwidth usage. Constant query/update rates ranging from (6 queries/sec, 0.6 updates/sec) to (10 queries/sec, 1 update/sec) were sent in separate experiments from the preceding peers to the responsible peer. Switching was initiated at the responsible peer after starting the query and update streams. Switching delay, result freshness and query response times throughout each experiment duration were observed. Switching delay was calculated as the difference between the time when switching was initiated and the time when the last query reached the responsible peer. The experiments were repeated for different combinations of query/update rates, result list sizes and available bandwidth for switching at responsible peer. The available bandwidth for switching was varied from 10%(10kbps) to 25%(25kbps) of the total bandwidth available for search application.

The variation of the switching delay with size of the result list, query/update rates and available bandwidth for

Result freshness	**Before switching**	**After switching**
Avg	86.1%	86.2%
Max	100.0%	100.0%
Min	80.0%	80.0%
SD	8.1	7.9

Table I: Result freshness(percentage overlap with ideal result) before and after switching initiation for (8 qps, 0.8 ups) with result size 100

Response time	**Peer 1**	**Peer 2**	**Peer 3**	**Peer 4**
Min	202.0 ms	202.0 ms	202.0 ms	202.0 ms
Max	208.0 ms	206.0 ms	206.0 ms	206.0 ms
Avg	202.6 ms	203.5 ms	203.5 ms	203.5 ms
SD	0.64	0.68	0.68	0.72

Table II: Response time statistics at different peers for (8 qps, 0.8 ups) with result size 100

switching is shown in Figure 5. The results show that, the switching can be performed within seconds after the decision is made. In the results obtained, the switching delay mainly depends on the result list size and available bandwidth.

Freshness of the search results observed at a preceding peer, before and after switching is initiated, for one of the experiments is shown in Table I. The variation of the observed query response times throughout an experiment(includes before, during and after switching) at different preceding peers is shown in Table II. The bandwidth available for switching was 10kbps. The low variation in the results show that query response times and quality of search results are unaffected by the switching process. Thus, offloading of responsibilities to cloud can be performed seamlessly within seconds incurring minimal bandwidth overhead without affecting the query response times and result quality.

VIII. Related works

Hybrid approaches combining centralized and peer-to-peer systems have been proposed for improving the availability and performance of peer-to-peer systems. In Cloudcast[6], public cloud resources are used to improve availability and for overlay management for peer-to-peer content distribution. Peer-to-peer video streaming systems like [7] use a peer-assisted approach to centralized video streaming to reduce the bandwidth costs of the provider. Online backup solutions like [8] combine peer and centralized resources to improve the performance of data backup applications. But, none of the above applications have a high content update rate as in peer-to-peer search which limits the scalability during spikes. To the best of our knowledge, CAPS is the first work proposing transient use of public cloud to handle popularity spikes in peer-to-peer search. Skobeltsyn etal.[4] propose indexing of popular term-combinations in DHTs for peer-to-peer search, but do not consider popularity spikes leading to bottleneck at the peer responsible for a spiking query. Tigelaar etal.[9] study the effectiveness of passive caching of search results in a hybrid centralized-p2p search system, but do not consider content creation resulting in cache invalidation and the associated bottlenecks during spikes.

IX. Conclusion and future work

In this work, we study the case for cloud-assisted peer-to-peer systems to handle concentrated load spikes in the context of peer-to-peer search. Transient use of public cloud is a promising solution to alleviate the effect of query spikes on peer-to-peer search as it maintains stability without affecting the desirable properties of peer-to-peer search. We proposed, implemented and evaluated Cloud-Assisted Peer-to-peer Search(CAPS) architecture to demonstrate the same. The payment for cloud resources can be realized by using targeted keyword-based advertisements. Future works include extending the idea of handling spikes using public cloud to small centralized search engines with limited resources and elaborate study on cloud resource provisioning, switching policies and advertisement-based peer-to-peer search monetization scheme.

Acknowledgment

The authors would like to thank Department of Science and Technology, Government of India for supporting this work.

References

[1] “Google status message on query spike,” http://twitter.com/#!/google/status/65502190315376640, Accessed: February 2012.

[2] J. Risson and T. Moors, “Survey of research towards robust peer-to-peer networks: Search methods,” *Computer Networks*, vol. 50, pp. 3485–3521, 2006.

[3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: a scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, February 2003.

[4] G. Skobeltsyn, T. Luu, I. P. Zarko, M. Rajman, and K. Aberer, “Web text retrieval with a p2p query-driven index,” in *SIGIR*, 2007.

[5] M. Gupta, P. Judge, and M. Ammar, “A reputation system for peer-to-peer networks,” in *NOSSDAV*, 2003.

[6] A. Montresor and L. Abeni, “Cloudy weather for p2p, with a chance of gossip,” in *P2P*, 2011.

[7] C. Huang, J. Li, and K. W. Ross, “Peer-Assisted VoD: Making Internet Video Distribution Cheap,” in *IPTPS*, 2007.

[8] L. Toka, M. Dell’Amico, and P. Michiardi, “Online data backup: A peer-assisted approach,” in *P2P*, 2010.

[9] A. S. Tigelaar and D. Hiemstra, “Query load balancing by caching search results in peer-to-peer information retrieval networks,” in *DIR*, 2011.