2005－DBS－137（Ｉ）（28）
2005／7／13

CDNにおける近傍ノードのコンテンツによる検索木の分散構成に基づく効率的コンテンツ検索手法

スレスタ　サンブ†　　小林　亜樹†　　山岡　克式†　　酒井　善則†

† 東京工業大学大学院理工学研究科集積システム専攻
〒 152-8552 東京都目黒区大岡山 2-12-1-S3-67
E-mail: †shambhu@net.ss.titech.ac.jp, ††{koba,yamaoka,ys}@ss.titech.ac.jp

あらまし　コンテンツ配信ネットワーク (CDN) における重要な問題の一つとしてコンテンツ検索の問題があげられる。しかし, 従来の分散型コンテンツ検索手法はコストの高いフラッディングに基づいているかコンテンツ配置アルゴリズムに制約をおいた上で効率的探索を可能とするものが多い。そこで, 本稿では近傍ノードのコンテンツによる検索木の分散構成に基づく効率的コンテンツ検索手法を提案する。提案手法の重要な特徴はコンテンツ配置アルゴリズムに依存しないという点である。本稿では提案手法を説明し, 従来のコンテンツ検索アルゴリズムと性能の比較を行うことによって提案手法は検索時に生じるトラフィックを少なく保ちながらもコンテンツ発見確率の向上に効果的であることを示す。

An Efficient Content Location Algorithm for CDN based on Distributed Construction of Search Tree from Contents of Proximal Nodes

Shambhu SHRESTHA†, Aki KOBAYASHI†, Katsunori YAMAOKA†, and Yoshinori SAKAI†

† Dept. of Communications and Integrated Systems, Graduate School of Science and Engineering, Tokyo Institute of Technology
2-12-1-S3-67, Ookayama , Meguro-ku, Tokyo 152-8552, Japan
E-mail: †shambhu@net.ss.titech.ac.jp, ††{koba,yamaoka,ys}@ss.titech.ac.jp

Abstract　One of the most important issues in Content Distribution Networks (CDN) is the algorithm for searching the contents. The existing distributed content location systems for CDNs are either dependent upon inefficient flooding techniques or on methods imposing a restriction on content placement algorithms. In this paper, we propose an efficient content location algorithm for CDNs based on distributed construction of search tree using the contents of proximal nodes. One of the important characteristics of our algorithm is its independence on the content placement algorithm. We describe our algorithm, compare it with the existing content location algorithms and show its effectiveness in increasing the success rate of queries while maintaing the traffic generated very low.

1. Introduction

Recent years have seen a huge raise in the size of contents transmitted over the Internet because of the increasing popularity of audio/video contents. Content Distribution Networks (CDNs) are being considered as a platform for distributing such contents. CDNs are content distribution platforms which aim to achieve efficient network use, fault tolerancy, load distribution, among others, by dynamically placing and replicating contents at op-

timal locations within the network. A detailed discussion about CDNs and the issues related to it are given in [1].

In this paper, we consider distributed CDNs, where there are no central or hierarchical group of servers to manage the CDN and everything is done in an entirely distributed fashion. Distributed CDNs resemble Peer-to-Peer(P2P) networks a lot since both are internetworks of nodes with equal roles and capacities and provide a platform for storing, sharing and retrieving data in a distributed manner. CDNs are distinguished from P2P networks in the following way: the number of nodes joining and leaving the network per unit time is negligible in CDNs whereas it cannot be ignored in P2P networks.

Two of the most important issues in CDN are :

1. Content placement and replication algorithm.
2. Content location algorithm.

The former deals with the placement and replication of contents in the CDN so as to optimize metrics such as average latency of content retrieval, network traffic or load distribution among servers under certain constraints (e.g. capacity of a server) . The latter, on the other hand, deals with the problem of searching a content within the CDN. The scalability and performance of a distributed CDN depends highly on the content location algorithm deployed. It should be noted that searching a content in CDN is not a trivial problem since the location and number of replicas of a content is not fixed, which is in sharp contrast to the WWW where a content is always bound to a particular server.

Content placement algorithms and content location algorithms are not necessarily independent as there are certain efficient content location algorithms, discussed in Section 2., which require contents to be distributed in a particular way. The main goal of our research is to develop efficient content location algorithm that do not impose any restriction on the content placement algorithm deployed in the CDN.

2. Previous Works

Gnutella [2] uses breadth first search (BFS) of network for content location. This method, also known as flooding (or broadcasting), searches for a content in all the nodes within HTL hops from the node initiating the query. While very simple, flooding generates a large number of messages per query, most of which end up in failure and as a result, inflict a huge load upon the network.

Local Indices [3], Probabilistic Search Protocol (PSP) [4], Adaptive Probabilistic Search (APS) [5] improve upon flooding by introducing certain kind of index on each node where information or hints about content location is stored. This makes it possible to selectively forward a query at each node or find a content faster, which helps to decrease the traffic produced by a query. Local Indices and PSP increase the success rate of a query but the traffic generated by a query, although smaller, is still comparable with flooding. Besides, each of them require a node to maintain a huge index or search history. APS deploys random walkers to search the network, which helps to decrease the traffic per query considerably but at the cost of decreased success rate. Quantitative analysis about the performance of these algorithms will be presented in section 6.

Chord [6], CAN [7], Tapestry [8] are searching algorithms based upon *Distributed Hash Tables(DHT)*. They provide infrastructures for efficient content location by maintaining a structured overlay network over the physical network. A major disadvantage of these methods, however, is that each of them requires a content to be stored in a particular node (analogous to hash tables where keys with same hash value are stored in same bucket). This makes them inappropriate for CDNs, where the location of contents changes dynamically according to some content placement algorithm. By storing pointer to a content instead of a content itself at designated nodes, it might be possible to adapt these methods for CDNs but the request for a particular content will always have to be routed to the node responsible to store

the pointer for the content. This leaves the various problems with WWW, such as fault-tolerancy, unresolved. In addition to that, there remains issues about locality of a search too — i.e. a search for a content might be routed to any node within the CDN, even though the content resides in the neigborhood of the node initiating the query.

3. Our Model

We consider a CDN of N interconnected nodes (or servers). The distance between 2 nodes $\mathcal{A}$ and $\mathcal{B}$, $\mathrm{Dist}_p(\mathcal{A}, \mathcal{B})$, is the number of links (or edges) on the shortest path between them. Each content served by the CDN has $\bar{r}$ replicas in the average and an unique content ID (abbr. *CID*), which is a bit string of m bits . Throughout this paper, the CID of a content will be treated as a positive integer equal to the decimal value of the CID. A content will be represented by a pair (c, X), where c is the CID of the content and X is the node that stores this content. We define, $\mathrm{Cid}((c, X)) = c$ and $\mathrm{Node}((c, X)) = X$.

There are $M(= 2^m)$ unique contents in the CDN — i.e. there exists at least one replica of a content with a given CID. Thus, the total number of contents in the CDN is $M\bar{r}$ and we suppose that these are distributed among the N nodes. Let, $\{c_0, c_1, \ldots, c_{t-1}\}(\forall i < j, 0 \leqq c_i < c_j \leqq M-1)$ be the CIDs of the contents in any node, $\mathcal{A}$ (this set will be denoted by $C(\mathcal{A})$). Then, the *Region of Responsibility* of $(c_i, \mathcal{A})$, denoted by $\mathrm{RoR}((c_i, \mathcal{A}))$, is defined as: $\mathrm{RoR}((c_i, \mathcal{A})) \equiv [c_i, (c_k - 1 + M) \bmod M]_I$, where $k = (i+1) \bmod t$ and, for $a, b \in Z_M$ [1],

$$[a, b]_I \equiv \begin{cases} \text{Set of integers in } [a, b] & a \leqq b \\ [b, M-1]_I \cup [0, a]_I & a > b \end{cases}$$

Furthurmore, for $a, b \in Z_M$, we define,

$\mathrm{Dist}_l(a, b) \equiv |[a, b]_I| - 1$

to be the *distance of b from a*. It should be noted that this definition of distance does not obey the commutative law – i.e. $\mathrm{Dist}_l(a, b)$ and $\mathrm{Dist}_l(b, a)$ are not necessarily equal.

To clarify these definitions, we will consider an example where $m = 4$ or $M = 16$, $C(\mathcal{A}) = \{5, 9, 10, 13\}$. In this case, $\mathrm{RoR}((9, \mathcal{A})) = [9, 9]_I = \{9\}$, $\mathrm{RoR}((13, \mathcal{A})) = [13, 4]_I = \{13, 14, 15, 0, 1, 2, 3, 4\}$, $\mathrm{Dist}_l(13, 4) = 7$ and $\mathrm{Dist}_l(4, 13) = 9$.

4. Proposed Method

In this section, we will explain the proposed algorithm in detail. Our method operates in the following 4 stages:

4.1 Learning Phase

In this phase, each content, $(c, \mathcal{A})$, gathers *information* about the contents whose CID lies in $\mathrm{RoR}((c, \mathcal{A}))$. This *information*, which will be denoted by $R_{all}((c, \mathcal{A}))$, is the set of contents whose CID lie in $\mathrm{RoR}((c, \mathcal{A}))$ and have replica(s) in a node within h hops of $\mathcal{A}$ — i.e. $R_{all}((c, \mathcal{A})) = \{(c_i, \mathcal{A}_i) | c_i \in \mathrm{RoR}((c, \mathcal{A})), \mathrm{Dist}_l(\mathcal{A}, \mathcal{A}_i) \leqq h\}$ This information can be collected, for example, by querying every node within h hops. Each element of $R_{all}((c, \mathcal{A}))$ will be called a *reference*.

4.2 Selection Phase

If t_{max} is the maximum number of contents that can be stored in a node, then the average size of $\mathrm{RoR}((c, \mathcal{A}))$ is $\frac{M}{t_{max}} = O(M)$ [2]. Thus, the size of $R_{all}((c, \mathcal{A}))$ can be of $O(M)$. In real CDNs, where M can be very large, having a content maintain a table of this size might pose a threat to the scalability of the system.

To ensure scalability, we select only $n_r(c, \mathcal{A})$ references out of $R_{all}((c, \mathcal{A}))$ according to an algorithm, described below, and require $(c, \mathcal{A})$ to store them. The set of the selected references will be denoted by $R_{sel}((c, \mathcal{A}))$. In our algorithm, $n_r(c, \mathcal{A}) \approx \log_2(|\mathrm{RoR}((c, \mathcal{A}))|)$. Hence, the average number of references to be maintained by a content becomes $\log_2(\frac{M}{t_{max}}) = O(\log(M))$.

Now, we explain our algorithm for selecting $n_r(c, \mathcal{A})$ references from $R_{all}((c, \mathcal{A}))$. For the sake of clarity, ror will be used to denote $|\mathrm{RoR}((c, \mathcal{A}))|$. We partition $\mathrm{RoR}((c, \mathcal{A}))$ into s subsets : $\{P_i | i \in Z_s\}$, where $s = \lceil \log_{\frac{1}{\alpha}}(\mathrm{ror}) \rceil$ for some $\alpha \in (0, 1)$, in the following way:

(1) : $Z_M = \{0, 1, \ldots, M-1\}$.

(2) : When t_{max} has an upper bound.

$P_0 = [c, \lfloor d(1) - 1 \rfloor \bmod M]_I$

$P_j = [\lfloor d(j) + 1 \rfloor \bmod M, \lfloor d(j+1) - 1 \rfloor \bmod M]_I$
$(1 \leqq j < s)$

Here,

$d(x) = c + \alpha^{s-x}\mathrm{ror}$

As an example, we will consider the case when $M = 128$, $C(\mathcal{A}) = \{2, 34, 51, 76, 125\}$. Here, $\mathrm{RoR}((2, \mathcal{A})) = [2, 33]_I$ and thus, ror = 32. If we take $\alpha = 0.5$, then $s = 5$. Hence, the partition of $\mathrm{RoR}((2, \mathcal{A}))$ will be $\{[2, 3]_I, [4, 5]_I, [6, 9]_I, [10, 17]_I, [18, 33]_I\}$.

Each reference in $R_{all}((c, \mathcal{A}))$ has its CID in one of the subsets P_i. For each subset P_i, we choose at most $\min(|P_i|, N_r(P_i))$ references whose CIDs lie in this subset and add it to $R_{sel}((c, \mathcal{A}))$. Here,

$$N_r(P_i) = \lceil (\frac{\theta_i}{\mathrm{Sum}}) \log_2(\mathrm{ror}) \rceil$$

$$\theta_i = ((\frac{\beta}{\alpha})^{s-i} - 1)|P_i|$$

$$\beta = 1 - \frac{\log_2(\mathrm{ror})}{\mathrm{ror}}$$

$$\mathrm{Sum} = \sum_{i=0}^{s-1} \theta_i$$

The basic idea behind this algorithm will be explained in section 5. If we think of a content $(c, \mathcal{A})$ and the references held by it in $R_{sel}((c, \mathcal{A}))$ as being connected, then we get a logical network of the contents, after this phase.

4.3 Dissemination Phase

In the *Selection Phase*, each content$(c, \mathcal{A})$ selected certain number of references from $R_{all}((c, \mathcal{A}))$ and added it to $R_{sel}((c, \mathcal{A}))$. In this phase, the remaining references – i.e. those in $R_{all}((c, \mathcal{A})) - R_{sel}((c, \mathcal{A}))$, are forwarded to references in $R_{sel}((c, \mathcal{A}))$ and then discarded by $(c, \mathcal{A})$. The basic idea is that the references which are not *important* for one content might be important to other contents to which the references are *closer*(3).

For the sake of convenience, we suppose that $\mathrm{RoR}((c, \mathcal{A})) = [c, d]_I$, where $c \leqq d$ (4). Suppose,

$R_{all}((c, \mathcal{A})) = \{(c_0, A_0), (c_1, A_1), \cdots, (c_{l-1}, A_{l-1})\}$,
$(\forall i \in Z_l,\ c_i \in [c, d]_I$ and $\forall i \in Z_{l-1},\ c_i < c_{i+1})$

$R_{sel}((c, \mathcal{A})) = \{(c_{x_0}, A_{x_0}), (c_{x_1}, A_{x_1}), \cdots, (c_{x_{k-1}}, A_{x_{k-1}})\}$,
$(\forall i \in Z_k,\ c_{x_i} \in R_{all}((c, \mathcal{A}))$ and $\forall i \in Z_{k-1},\ c_{x_i} < c_{x_{i+1}})$

$R_i((c, \mathcal{A})) = \{c_j | c_j \in R_{all}((c, \mathcal{A})), c_{x_i} < c_j < c_{x_{i+1}}\}$,
$(i \in Z_{k-1})$

$R_{k-1}((c, \mathcal{A})) = \{c_j | c_j \in R_{all}((c, \mathcal{A})), c_{x_{k-1}} < c_j\}$

Now, content $(c, \mathcal{A})$ forwards $R_i (i \in Z_k)$ to reference (c_{x_i}, A_{x_i}), and discards $R_{all}((c, \mathcal{A}))$. Now, let's say that a content (c_r, A_r) receives a set of references R forwarded by other content. If $R_{sel}((c_r, A_r)) = \{c_{r_0}, c_{r_1}, \cdots, c_{r_{u-1}}\}$ and $n_r(c_r, A_r)$ is the maximum number of references that can be stored by (c_r, A_r), we do the following for all $r \in R$:

1. If $\mathrm{Cid}(r) \notin \mathrm{RoR}((c_r, A_r))$, forward $\{r\}$ to the content, c_f in $C(A_r)$ for which $\mathrm{Cid}(r) \in \mathrm{RoR}((c_f, A_r))$.
2. If $r \in R_{sel}((c_r, A_r))$, do nothing.
3. If $\mathrm{Cid}(r) \in \mathrm{RoR}((c_r, A_r))$ and $u < n_r(c_r, A_r)$, $R_{sel}((c_r, A_r)) = R_{sel}((c_r, A_r)) \cup \{r\}$.
4. If $\mathrm{Cid}(r) \in \mathrm{RoR}((c_r, A_r))$ and $u \geqq n_r(c_r, A_r)$, find the content r_f in $R_{sel}((c_r, A_r))$ for which $\mathrm{Dist}_l(\mathrm{Cid}(r_f), \mathrm{Cid}(r))$ is the minimum. If $\mathrm{Cid}(r_f) = c_r$, do nothing, otherwise forward $\{r\}$ to r_f. Since, $\mathrm{Dist}_l(\mathrm{Cid}(r_f), \mathrm{Cid}(r)) < \mathrm{Dist}_l(c_r, \mathrm{Cid}(r))$, this will end after a finite number of forwardings.

4.4 Searching Phase

Once, the previous 3 phases are completed, the actual searching can be done. If q is the CID of the content to be searched, our searching algorithm works as follows:

1. When a node, $\mathcal{A}$ receives a query, it forwards it to the content, $(c, \mathcal{A}) \in C(\mathcal{A})$ for which $q \in \mathrm{RoR}((c, \mathcal{A}))$.
2. When a content, $(c, \mathcal{A})$ receives the query: If $q \notin \mathrm{RoR}((c, \mathcal{A}))$, the query is forwarded to $\mathcal{A}$ else, if $c = q$, the query ends in success otherwise $(c, \mathcal{A})$ searches for the reference r_f in $R_{sel}((c, \mathcal{A}))$ for which $\mathrm{Dist}_l(\mathrm{Cid}(r_f), q)$ is the minimum. If $c = \mathrm{Cid}(r_f)$, the query ends in failure otherwise the query is forwarded to r_f.

(3) : In the sense of $\mathrm{Dist}_l(\cdot, \cdot)$.

(4) : The algorithm is exactly the same when $d < c$; only the notations become complicated.

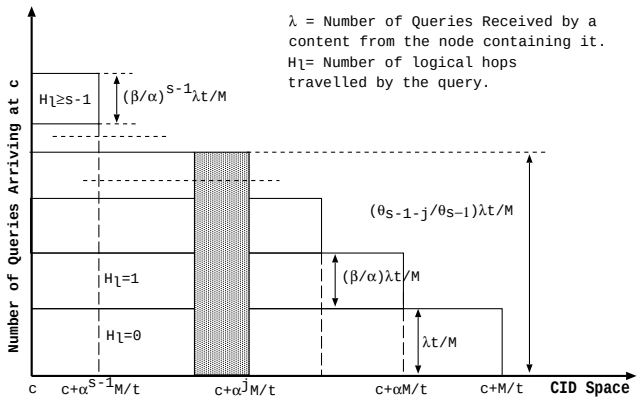

Fig. 1 Estimation of number of queries for a content in $\mathrm{RoR}((c, \mathcal{A}))$

5. Discussions

As described in the section 4. 4, in our searching algorithm, each content forwards a query, q to a reference which is closer to it. When a query is passed from one content to another, we say that the query travels one *Logical Hop*(H_l). Suppose, each node, $\mathcal{A}$ holds t contents so that the average size of $\mathrm{RoR}((c, \mathcal{A}))$ is $\frac{M}{t}$. If each content is equally likely to be queried, then the λ queries that are received by $(c, \mathcal{A})$ from the node containing it (i.e. when $H_l = 0$) will be distributed among the contents in $[c, c + M/t]_I$ equally, as shown in Fig. 1. Since, $(c, \mathcal{A})$ contains $\log_2(M/t)$ (5) references, a fraction of these queries are resolved and the remaining fraction, $\beta(= 1 - \frac{\log_2(M/t)}{M/t})$ of queries is forwarded to other contents.

When a query is forwarded by $(c_i, \mathcal{A}_i)$ to $(c_j, \mathcal{A}_j)$, if we can ensure that $\mathrm{Dist}_l(c_j, q) < \alpha \mathrm{Dist}_l(c_i, q)$ where $\alpha \in (0, 1)$, the query will converge to a content closest to it exponentially fast. In other words, the query, which has travelled $H_l = j$ logical hops, should lie within $[c, c + \alpha^j M/t]_I$, as shown in Fig. 1, so that it converges to the closest content in about $\log_{\frac{1}{\alpha}}(M/t)$ logical hops. When the queries follow this kind of convergence, contents in various subsets of $\mathrm{RoR}((c, \mathcal{A}))$ receive various number of queries with contents closer to c receiving more queries than those far from it. This is the reason behind partitioning $\mathrm{RoR}((c, \mathcal{A}))$ and selecting various number of references from each subset depending upon the *importance* of each subset, as explained in section 4. 2.

To summarize, in our method, each content takes part in the process of creating a distributed search network, based upon the contents of neighboring nodes. Each content stores only a part of that network and decides locally where to route a query. With the addition of an initial cost of building the distributed index, a huge decrease in the the number of useless traffic generated by a query can be achieved. In the next section, we present the results of our simulation to measure and compare the performance of our method with existing methods.

6. Simulation Results

To determine the effectiveness of the proposed method and to compare its performance with existing methods, we carried out a simulation in Java VM.

We performed the simulation in a CDN with 256 nodes arranged in a (16×16) $2D$-Mesh. The capacity of each node was set to 10 contents. Each content has a 10 bits long CID with a total of 1024 unique contents in the CDN. The number of replicas of contents were determined with a Zipfian distribution (parameter = 0.6). We generated a query set of 2^{20} queries. For each query, a node to initiate it was chosen uniformly and a CID of the content to search, was chosen randomly (Zipfian distribution of parameter 0.6) (6). The following algorithms were used to process the queries at each node:

1. Probabilistic Search Protocol(PSP). [4]
2. Adaptive Probabilistic Search(APS). [5]
3. Flooding.
4. Proposed Method.
5. Local Indices. [3]

Table 1 summarizes the important parameters of the simulation.

A search was carried out with all the queries

(5) : Because, ror = M/t in this case.

(6) : The number of replicas of a content and the number of queries for it are proportional.

Table 1 Parameters of the simulation.

Network Topology	16×16 2D-Mesh
N	256
Contents per Node	10
Replica Distribution	Zipf (0.6)
Query Distribution	Zipf (0.6)
m	10
M	1024
Directory Cache Size (PSP)	$0.1M$
Broadcast Probability (PSP)	1
Neighbor Index Size (APS)	$0.05M$
α (Proposed Method)	0.6
Radius of Index (R) (Local Indices)	1 to 15

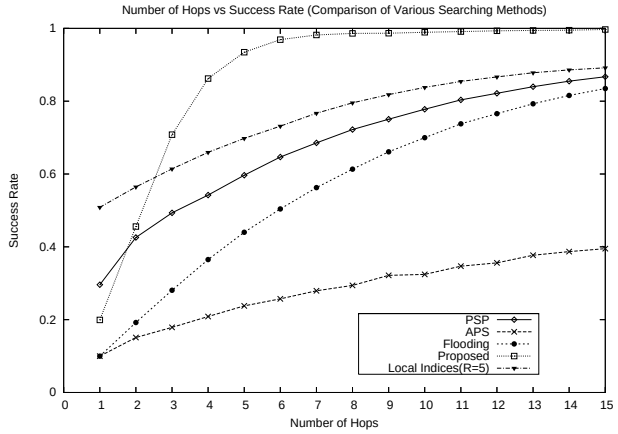

Fig. 2 Comparison of success rate of queries.

in our query set with varying HTL(7) values of queries for methods (1), (2), (3), (5) and varying h values (Ref. 4. 1) for the proposed method. For every 2^{13} queries, an average of success rate and traffic generated per query was calculated for each method. This was done to detect the steady state of PSP and APS, since both of them require certain time to arrive at a steady state.

A comparison of the success rates of queries for various searching algorithms is depicted in figure 2. The horizontal axis represents the h value for the proposed method and HTL value of queries for other methods. For the sake of fairness, results for $R = 5$ is used for Local Indices since the index to be maintained by a node becomes larger for higher value of R. The values of the the steady state are used for PSP and APS while an overall average of averages are used for other methods.

(7) : Hops to Live

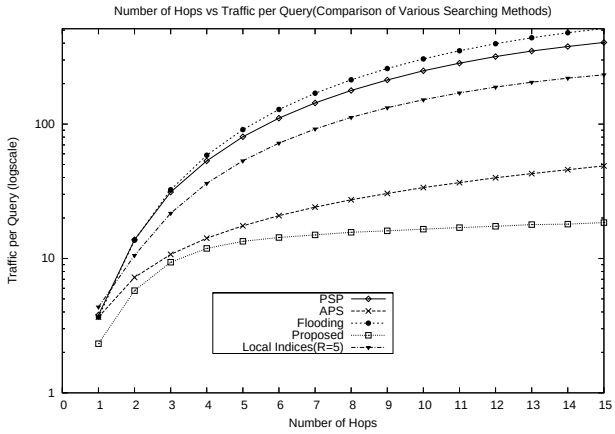

Fig. 3 Comparison of traffic per query.

It can be seen that the success rate of a query can be significantly increased using the proposed method. For example, if a success rate above 0.8 is required, we can apply the proposed method with $h = 4$ while a query with HTL $\geqq 10$ is required by most of the other searching algorithms, which results in a huge increase in the generated traffic, as shown later on. This is mainly because of the *Dissemination Phase* of our algorithm where we disseminate the information gathered by each content from its neighborhood to various other contents, over the logical network of contents, formed after the *Selection Phase*. This makes it possible to transmit the local information of a content to a content farther away without relying on costly flooding techniques.

Figure 3 shows a comparison of the traffic generated by each query. The traffic generated when a query is transmitted through one physical link is considered as unity. The traffic generated by PSP, APS, Flooding and Local Indices all increase considerably with the HTL value of the query. APS deploys random walkers instead of flooding messages. So, the traffic generated by it is relatively less but the success rate is also smaller. In the proposed method, each query produces only one message which follows a single path on the logical network of contents. This helps to largely reduce the number of wasteful traffic as can be verified from the figure.

In general, there is a tradeoff between the success rate and the the traffic generated by a query.

To evaluate them aggregately, we introduce a metric, the *Efficiency* of a searching algorithm, which is defined as

$$\text{Efficiency} = \frac{\text{Success Rate of Query}}{\text{Traffic Generated per Query}}$$

Figure 4 shows a comparison of the efficiency of the various searching algorithms. It should be noted that apart from the traffic generated per query, there are various other factors that affect the success rate. For example, in the proposed method and Local Indices, each node produces a huge traffic to build the initial index. Since, this is an one-time process, it should not be such a big problem for the CDN. Similarly, APS and PSP require certain time to reach to the stable state. Again, since this happens only once initially, it is not analyzed here in detail.

Apart from that, by requiring each node to maintain a larger index, the success rate can be improved because each node can hold extra information which can be helpful for searching a content. Since this index might have to be kept on the main memory of a node and its size also affects the processing time of a query, it is necessary to evaluate the size of the index to be maintained by a node for these various searching methods.

Flooding doesn't require a node to maintain any extra index. PSP requires each node to maintain a directory cache of the size equal to a fraction of total number of unique contents. Similary, APS also requires each node to maintain an index for each of the neighbor. In Local Indices, the total size of the index is equal to the total number of contents in nodes within R hops from a node. In the proposed method, using the selection algorithm explained in 4. 2, the number of references to be stored by each content is reduced from $O(M)$ to $O(\log(M))$.

The comparison of the index size of a node for our simulation is given in Figure 5. The horizontal axis represents R for Local Indices and h value for the proposed method. For other methods which are shown only in *points*, the index size is dependent upon M and the horizontal axis doesn't have a meaning. Even though the index size of a node in the proposed method depends only on M

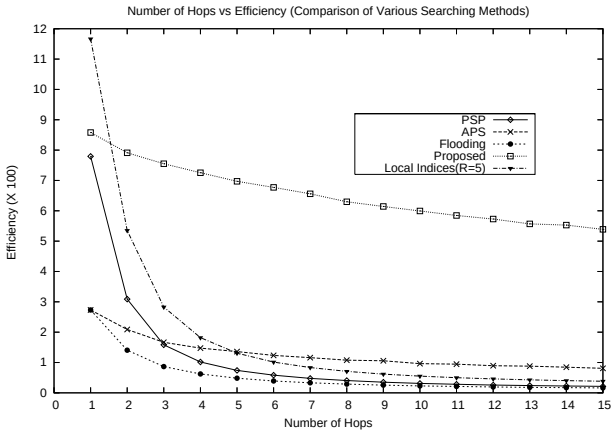

Fig. 4 Comparison of efficiency of searching methods.

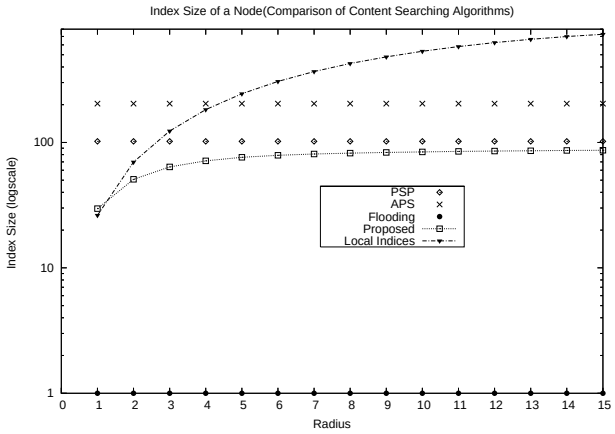

Fig. 5 Comparison of index size of a node.

and the number of contents in a node, the figure shows slightly varying values of index size because at smaller values of h each content, doesn't have enough references to store.

7. Conclusion and Future Works

In this paper, we proposed an efficient method for content location in CDNs. Our method basically constructs a search tree from the contents in the neighborhood in a distributed manner. We evaluated our method in terms of success rate, traffic generated by queries and index size of a node. We compared our method with the existing content location algorithms and showed that it is significantly more efficient than other methods in *static* conditions.

However, there are certain drawbacks in our method. Our method generates a huge traffic in the initial phase for creating an index. Besides,

our method also makes an assumption that for every bitstring of m bits, there exists a content having it as the CID, which may not be a realistic supposition.

We are working on extending our algorithm for the cases when contents are dynamically added to, removed from or moved within the CDN. The integrity of the logical network of contents needs to be maintained when these events occur. We are also planning to extend our algorithm to more realistic cases e.g. without the previously mentioned assumption about the existence of contents with every possible CID.

References

[1] Gang Peng. *CDN: Content Distribution Network*. Research Proficiency Exam report, Computer Science Department,SUNY at Stony Brook, NY, USA, 2003.

[2] Gnutella. *The Gnutella Protocol Specification*. www9.limewire.com/developer/gnutella_protocol_0.4.pdf, 2004.

[3] B.Yang and H.Garcia-Molina. *Improving Search in Peer-to-Peer Networks*. ICDCS, 2002.

[4] D.Menascé and L.Kanchanapalli. *Probabilistic Scalable P2P Resource Location Services*. ACM SIGMETRICS Performance Evaluation Review, 2002.

[5] S.Tsoumakos and N.Roussopoulus. *Adaptive Probabilistic Search for Peer-to-Peer Networks*. Technical Report CS-TR-4451, University of Maryland, 2003.

[6] I.Stoica et al. *Chord: A scalable Peer-to-peer Lookup Service for Internet Applications*. Proc. of SIGCOMM, 2001.

[7] S.Ratnasamy et al. *A Scalable Content Addressable Network*. Proc. of SIGCOMM, 2001.

[8] B.Zhao et al. *Tapestry: An Infrastructure for Fault-tolerant Wide Area Location and Routing*. Technical Report UCB/CSD-01-1141, Computer Science Division, Univ. of California, Berkeley, 2001.