Optimal Policy of Data Dissemination in CDNs

Gadiraju Mahesh,
Department of C.S.E.,
S. R. K. R. Engineering College,
Bhimavaram, India

Vatsavayi Valli Kumari,
Department of CS&SE.,
Andhra University College of Engineering,
Visakhpatnam, India

ABSTRACT

A dynamic data dissemination network is a content delivery network (CDN) implemented with a hierarchical network of data aggregators (repositories) for disseminating dynamic data like stock quotes, number of votes polled for a political party in an election in different regions and environmental parameters. Continuous aggregate query is a query with aggregation operations and is repeatedly requested by the user. Executing continuous aggregate queries in dynamic data dissemination networks/ CDNs is the essence of our work. There are two major tasks of data dissemination networks. First one is effectively providing data to clients from sources through the network of data aggregators by assigning optimal data aggregators to clients. The second one is propagating the updates of dynamic data to clients. There are different algorithms like enhanced greedy algorithm with withdrawals and primal dual parallel algorithm for accomplishing the first task. The second task can be performed using policies like push, pull, push-or-pull, and push-and-pull. The existing algorithms for dissemination of data and policies for distributing the updates of data are explored in this paper. Then a policy for consistently propagating the updates of dynamic data and an algorithm for optimally assigning data aggregators to clients for disseminating data in CDNs are extracted.

General Terms

Content distribution networks or content delivery networks (CDNs), set cover problem, vertex cover problem, continuous aggregate queries, greedy algorithm.

Keywords

Dynamic data dissemination networks, primal-dual parallel algorithm for continuous aggregate query dissemination (PDPA), enhanced greedy algorithm with withdrawals (EGAWW), dynamic data dissemination graph, data aggregator (DA), data incoherency bound.

1. INTRODUCTION

Web sites in the internet provide content according to needs of the users and previously most of the clients used to request static content. But, nowadays users demand not only static content but most of the time they are demanding dynamic content like videos with live streaming, live stock quotes, and live election results. In applications like live election results, clients issue queries like total number of votes polled and maximum number of votes polled to a party. These queries are issued again and again repeatedly and contain aggregation operators like sum and maximum. Such queries are known as continuous aggregate queries. As these queries are repetitive in nature and generally include dynamic data, caching is best technique for disposing the data from sources to clients.

By using the caches, server overload of executing the query again and again can be avoided. The result of the query can be cached when the query is requested for the first time and the cached result can be used for subsequent requests. Not only result of the query but also most frequently used data is cached for further use. When the values of data change then the copies of query results and data are updated.

As discussed in [1], there are two techniques of caching namely backend caching and proxy-based caching. In backend caching caches are maintained at the servers. In proxy-based caching, replicated servers called proxies are maintained nearer to the clients. Backend caching technique has the advantage that cached data is consistent with original copy of data as the caches are maintained at the server itself. Proxy based caching has the advantage of high availability of data and reduces the load on the servers. It also reduces frequency of communication between clients and servers. The main problem with proxy caching is that it is difficult to maintain agreement of copies of data at proxies with that at source of the data.

Content distribution network is an overlay network of proxy servers for disseminating the data to the clients with high fidelity and throughput. Content distribution networks use caches at the edge nodes of the networks which are closer to the clients than the data sources. Content distributed networks (CDNs) were initially designed to handle static content. Nowadays CDNs are used to serve dynamic content. Content delivery service providers (CDSP) maintain CDNs for effectively providing data to their clients. If the CDSP is commercial, the clients have to pay for the services requested. There are many CDNs available in the market both commercial and free. Free CDNs include Corel content distribution network and Incapsula. Commercial CDNs include Akamai technologies, Limelight, CloudFlare and EdgeCast networks.

Akamai Technologies maintain the caches at the edge of the network nearer to the clients. The clients use the edge caches for getting most frequently used data instead of getting the data from data sources. Here the main advantage is that it reduces the user access time to data. According to [2] the standard DNS resolution process includes the steps shown in Fig. 1. In steps 1 and 2 the server’s name is resolved. In step 3, HTTP request to the edge server is made by the client. If the requested data is cached at the edge server, it returns data to the client and stores the request completion status. If needed, in the step 4, edge server retrieves the data from Akamai server or content provider's server

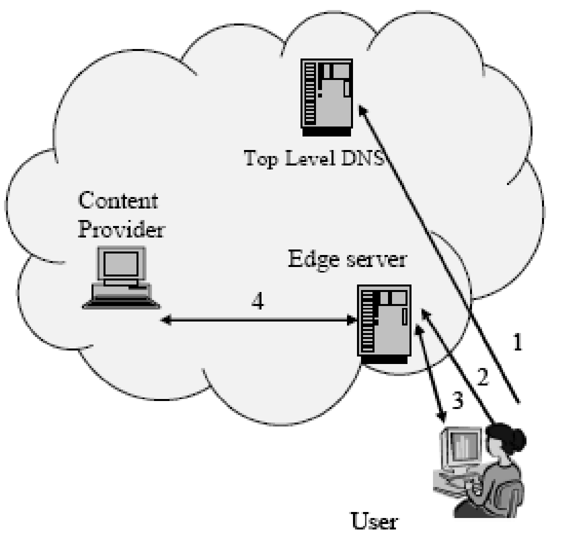

Fig. 1: DNS resolution process in CDNs

An architecture for content distribution networks, called dynamic data dissemination network, is proposed in [3, 4]. Dynamic data dissemination network is a content distribution network implemented with a hierarchy of proxy servers called data aggregators. In dynamic data dissemination networks the main aim is dissemination of dynamic data to users satisfying the coherence requirements of the users. As the number of users increase the sources may not support all the users and if the same data is requested by clients several times, the clients have to obtain the data again and again from the server. So, repositories generally called data aggregators (DAs) which duplicate the data are used for disseminating the data in these networks. DAs may cache many data items and the same data item can be severed to clients from different DAs.

The data aggregators nearer to users serve the data to users much efficiently within no time. By maintaining such repositories which duplicate the data create new problems like maintaining consistency of the duplicated data items on different data aggregators. For an ideal condition the values of a data item at all the data aggregators and at the data source should be equal. But practically it is impossible to satisfy such an ideal condition and all the clients who need the data item may not require such an ideal condition. Most of the users require data items at some specified maximum limit of tolerance value called data incoherency bound. Here the main issue is how to disseminate data to clients satisfying the coherency requirements of the data requested by them.

2. EXECUTION OF CONTINUOUS AGGREGATE QUERIES

As the name indicates continuous aggregate queries are the queries requested by the clients repeatedly with continually changing data items that use aggregate operations like sum, average, minimum and maximum on the data items. The examples of continuous aggregate queries include queries related to atmospheric parameters like query for obtaining average temperature in a region. Another example is the query obtaining total number of votes polled for a party in all the regions. The general form of the query for obtaining total number of votes polled is as follows.

Select sum (votes) from election where party = ‘My party’;

Here votes is a column name in the election table and it stores number of votes in different regions for different parties. Execution of continuous queries is discussed in [3, 5].The execution of a query requires the retrieval of data items from data aggregators. There exist many data aggregators that disseminate the same data item i.e., there are many DAs that can disseminate data items required by the client query. A single DA can disseminate several data items. A client query can be executed effectively by a single DA. But it may not be always possible to have a DA that holds all the data items of the query. Another option of obtaining each data item of the query from different DAs is not efficient. So, for accomplishing the task of query execution, the client queries can be divided into sub-queries and optimal DAs can be selected for executing the sub-queries. The problem of dividing the queries into sub-queries and optimal selection of DAs for executing these sub-queries is minimum weighted set cover problem. The different algorithms that can be applied to set cover problem are given in [6].

2.1. Greedy Algorithm

In the minimum weighted set cover problem, a set of elements V, a collection T of sub-sets of elements belonging to set V such that the sub-sets cover all elements of the set V and costs C of sub-sets are given. The problem is to find optimal sub-sets R of elements which cover all the elements of V and total cost of sub-sets is minimal. The problem of optimally dividing query into sub-queries and selecting the sub-queries from optimal DAs can be compared to set cover problem. Here $V=\{v_1,v_2,v_3,...\}$ is set of all data items in the client issued query, $T=\{q_1, q_2, q_3, \ldots\}$ is all options of sub-queries that can be executed by different DAs and $C=\{c(q_1),c(q_2),c(q_3), ..\}$ is costs of sub-queries. Greedy algorithm is used in [3] for solving this problem and is given below.

In the following algorithm, D is sum-difference, B is incoherency bound of the query and n(q) is the number of data items involved in q. The calculation of D is given in [3]. In this algorithm the set of optimal sub-queries R is first initialized to NULL and after execution of the algorithm, R contains all the optimal sub-queries. The average cost p(q) of different optional sub-queries in T are calculated. The sub-query which has the least value of p(q) is selected as optimal sub-query in each iteration of the algorithm. The sub-query is added to the result R and it is removed from T. Then all the data items contained in the selected sub-query are removed from other sub-queries. Finally the sub-queries which become empty after this step are also removed from T. This is continued until all the data items are covered or indirectly there is no sub-query in T.

2.1.1. Greedy algorithm for continuous aggregate query execution

```
1.  R:=NULL
2.  Repeat steps 2.1 to 2.5 until T=NULL
    2.1. for each q in T
         2.1.1. calculate $c(q):=D/B^2$
         2.1.2. compute cost per data item p(q):=c(q)/n(q)
    2.2. select $q_m$ in T with $p(q_m)=min(p(q))$
    2.3. R:=R union $q_m$
    2.4. T:=T minus $q_m$
    2.5. for each v in $q_m$
         2.5.1. for each q in T
                2.5.1.1. q :=q minus {v}
                2.5.1.2. if q=NULL then T:=T minus q
3.  return R
```

.

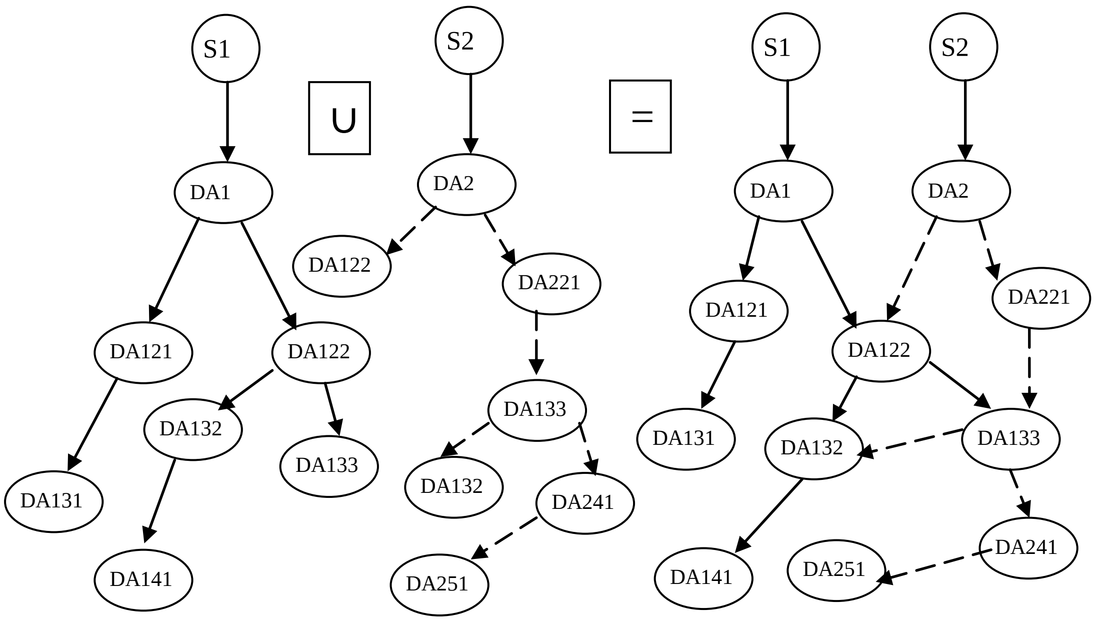

Fig. 2: Data dissemination graph (union of data dissemination trees)

3. CONSISTENCY OF DATA IN DATA DISSEMINATION NETWORKS

In dynamic data dissemination networks the main issue is to construct a network of data aggregators and to provide the data items to users satisfying the coherence requirements. For maintaining the coherence of copies of dynamic data at different data aggregators, the updates to the data items at source are propagated to data aggregators only if the incoherency of data item at a DA exceeds the incoherency bound. These updates are propagated from source of the data item to the DAs through a network of data aggregators. A hierarchical network of data aggregators is used in [3]. A dynamic data dissemination tree is constructed for each data item with the DAs containing the data item as nodes and source as root node of the tree. The whole network of data aggregators disseminating different data items is the union of dissemination trees for all the data items which is a dynamic data dissemination graph as shown in Fig. 2.

The different policy for the propagation of data updates to data aggregators are discussed and compared in [7, 8]. They are listed below.

i) Push: Pushing the data from top to bottom of the dynamic data dissemination tree from source to DAs and higher level DAs to lower level DAs.
ii) Pull: DAs pull the updates from source or higher level DA in the data dissemination tree.
iii) Push or pull: Push connection for some clients and pull connection for some clients according to requirements.
iv) Push and pull: Default pull connection but may be switched to push in the event of frequent updates.
v) Leases: Clients lease the sources for a span of time.

The first policy is discussed in section 4 and the other policies are discussed in this section. As mentioned earlier the hierarchical network of DAs is used as a scheme of data dissemination and each DA serves data items at some guaranteed coherence. For preserving consistency of specified incoherency bound, a DA gets data updates from the data source or some higher level DA only when data incoherency is greater than or equal to data incoherency bound based on different policies as discussed in [7].

3.1. Pull Policy of Data Refreshing

In this policy of dissemination of data updates, every data aggregator/node pulls the data whenever there is a need of updating its copy of data. A parameter called time to refresh (TTR) is maintained by the node. The TTR determines the time at which node should query for update of data item at source. If the TTR value is low the data updates are disseminated more frequently ensuring data coherence. If the value is high then some of the updates at the server may be missed by the node but reduces number of polls of clients to sources for data updates. So, deciding the optimal value of TTR is main issue in pull based dissemination of data. An adaptive TTR is used in [7, 9] and the formula for estimating the TTR is as given below.

Adaptive TTR = maximum(t_m, minimum (t_h, $(a*t_s+(1-a)*t_d)$))

Where t_d is time to refresh estimate based on learning which is computed using the following formula.

$t_d = p*t_e + (1-p)*t_l$

t_m is lower limit of TTR

t_h is upper limit of TTR

a is adjustment factor for adjusting the fidelity required.

t_s is smallest TTR value used upto the present estimate of TTR

t_e is TTR estimate based on recent change in data and is given by the formula,

$t_e = t_l * b/(d_l - d_u)$

t_l is latest change in the value of TTR

b is incoherency bound

d_l is latest value of the data item

d_u is penultimate value of data item

p is the relative weight given to recent and old changes whose value is such that .5< =p<1.

The pull based technique of propagating the updates is preferred to push based dissemination when high resilience to failures is required. But it has less fidelity than the push technique.

3.2. Push-or-Pull Policy

As described in [7], the concept of this approach is that the server decides whether a DA is served with push connection or pull connection depending on the requirements of the DA. By default push connection is granted if it has enough resources. When server does not have enough connections but client fidelity requirements are high, push connection is granted to the new node by dropping some of the already granted push connections to DAs with low fidelity requirements or with high temporal coherence requirement and for which data served is small in quantity. The push-or-pull approach also facilitates to detect failures at the servers. So that push connection may be converted to pull connections during failures. For this purpose a TTR (time to refresh) parameter is maintained at DAs. TTR is an estimate of time intervals at which server pushes the data updates. In ordinary operation, the server pushes the data changes. If a client does not receive an update even after time elapsed from previous push is greater than TTR limit, the node recognizes that a server failure is there. Then the client can request the server for its present state or can start pulling the data. The performance of push-or-pull depends on whether connection granted is push or pull [7].

3.3. Push-and-Pull Policy

As discussed in [7], the disadvantage of pull approach of data refresh is that it mainly depends on the estimation of time to refresh (TTR) parameter. The generally used method of TTR estimate is adaptive TTR and its disadvantage is that it is a slow learning algorithm and in the event of faster changes to data at the server this estimate may not give correct value and some of the updates to data at the server may not be propagated to nodes. To avoid these disadvantages push-and-pull technique can be used. In this technique pull or push is used depending on the rate at which data is changing.

As in the pull policy the data is pulled from the clients at regular time interval called time to refresh. But whenever there is a case of a client not pulling the data update required by the user, the server pushes such changes. To know such cases of clients missing the updates, the server also has to run the adaptive push algorithm and has to know when the node will pull the data. If a data change which violates the coherence requirements occurs before the next expected pull according to adaptive TTR estimate then the server knows that this update is missed by the node and pushes the data.

3.4. Lease Based Policy

The CDNs can use different policies for disseminating data updates like pull, push, push-or-pull and push-and-pull. Scalability is the main problem in all the above approaches. The cooperative leases approach proposed in [8] overcomes this problem. This approach is based on leases for distributed file cache consistency proposed in [10].

The meaning of leases in CDNs is that there is an agreement or lease of a proxy with the server for the server to notify the proxy all the updates to a data item during an agreed period of time. After the lease time the proxy could renew the lease for further updates. The lease record contains the following attributes

i) The object for which lease is requested(d)

ii) The proxy which takes the lease(N)

iii) The time period for which lease is agreed (t)

The ordinary leases policy of update propagation in CDN have disadvantage that all updates are to be notified to all the nodes which take the lease. But all the updates disseminated may not be of interest to many repositories and another disadvantage is scalability of leasing many repositories. To avoid these problems, cooperative leases approach of disseminating the updates is proposed in [8].

The two notions in this approach are as follows

i) Cooperative consistency

ii) Delta consistency

The objective of cooperative consistency is, for maintaining consistency, DAs cooperate with each other using cooperative leases technique. The cooperative consistency maintenance is different from cooperative caching. In cooperative consistency, maintaining the cooperation between different nodes is limited to consistency maintenance and the separate overlay networks can be used for data dissemination and for consistency maintenance. This policy uses application level multicast for disseminating data updates reducing overheads on the servers. The delta consistency ensures that the data at proxies is refreshed for every delta time units within lease period ensuring copy of data at the client is never inconsistent by more than delta time from its server version of the data. Here delta is notification rate. In this technique, instead of taking lease for all the updates in a period of time, lease is taken only for notifying the updates at a given frequency within the lease period. Here frequency of notifications is the inverse of notification rate.

In cooperative leases approach, instead of granting lease to each and every repository, groups of nodes are formed and lease is given to a single node of a group on behalf of the group and call the designated node as the leader of the group. The server interacts only with the leaders of the groups and transfers the updates only to leaders. It is the responsibility of the leader to propagate the notification to other nodes in the group. So, the leader shares the over head of transferring the notifications reducing the over head of the server. The cooperative leases tuple (d, g, N, t, r) contains the following attributes.

i) d- Data item of interest

ii) g- Group which requests the lease

iii) N- Group leader node

iv) t- Lease time period

v) r- Notification rate

4. OPTIMAL POLICY FOR PROPAGATION OF UPDATES

Already different policies for propagating data updates were discussed in section 3. An optimal policy which has a higher degree of fidelity than the other policies is discussed here. Higher degree of fidelity means disseminating the data with coherence requirements at a higher degree of accuracy.

4.1. Push Based Data Dissemination

In this policy, data updates at source are pushed to DAs whenever source recognizes data incoherence at the DAs. As discussed in [11], the hierarchical dynamic data dissemination network is maintained such that the higher level aggregators have a lower incoherency bound than the lower level aggregators in the hierarchical network. This is due to the fact that the updated copies of data are pushed from the data source to higher level data aggregators and consequently to the lower level data aggregators. Let us discuss how updates to a data item d which is disseminated by a source S are forwarded to a data aggregator Q through a higher level data aggregator P .

Let d_t^s , d_{t+1}^s, d_{t+2}^s,... be the values of a data item at Source S at consecutive time instances, d_u^p , d_{u+1}^p, d_{u+2}^p, ... be the consecutive values of data item at the data aggregator P and d_v^q, d_{v+1}^q, d_{v+2}^q, ... be the consecutive values of data items at the data aggregator Q. Let d_u^p corresponds to update d_t^s and Let d_{u+1}^p corresponds to update d_{t+k}^s where k>=1.Then for all m, $1 \leq m \leq k-1$, for maintaining coherency at P the following condition must be satisfied.

$$| d_{t+m}^s - d_t^s | < b^p$$

If this condition is satisfied, the present value of data item at source is not propagated to proxy P. Else the update must be propagated to P. At each P, if $| d_u^p - d_v^q | >= b^q$, the update received by P from source is forwarded to its child proxy Q. Here b^p and b^q are the incoherency bounds at P and Q .

Table1. Propagation of updates

	b^p= 3	**b^q= 5**
Source	**D A**	**D A**
S	**P**	**Q**
No. of votes at source	**No. of votes cached at P**	**No. of votes cached at Q**
1 →	1 →	1
2	1	1
3	1	1
4 →	4	1
5	4	1
6	4	1
7 →	7 →	7
8	7	7
9	7	7
10 →	10	7

But in some situations shown in the table, these conditions are not sufficient. The table shows propagation of number of votes polled to a candidate in an election. In the table all the updates of number of votes polled to the candidate are correctly propagated. But the number of votes polled "6" is not propagated to Q. To avoid missing such updates the following additional condition is used. If the following condition is satisfied then also the update received to P should be propagated to Q.

$$| d_u^p - d_v^q | >= b^p - b^q$$

The above discussed approach includes node based computations. In centralized approach the source should know and store list of all incoherency bounds of all data items at different repositories. Whenever there is a new update of data item at the source, the source checks incoherency bounds (b) of the data item and last update for that data item. The source checks all the b values violated by the update and such updates are transmitted to the corresponding repositories through the dissemination tree. This update and the maximum b value that is violated by the update are stored at the source and this record can be used for propagating future updates.

The disadvantage of this approach is source has the overheads of maintenance of c values which results in computational and space overheads. So, node based push approach is better than centralized approach. The push based policy has high fidelity than other policies. Hence, in this paper, push based policy is selected as optimal policy for propagating the updates. Another important issue in dynamic data dissemination networks is construction of data dissemination graph. The insertion of a new node in data dissemination graph is discussed in the following sub-section.

4.2. Construction of Dynamic Data Dissemination Graph

Dynamic data dissemination graph is the union of dynamic data dissemination trees of all data items of interest. For inserting a new node in data dissemination graph, there are two techniques proposed in [11]. The first technique is level at a time algorithm and another one is data at a time algorithm.

4.2.1. Level at a time algorithm

In data dissemination graph, the sources of data are at level zero and all immediate child nodes of sources are at level one and child nodes of level one nodes are at level two and so on. According to [11], for inserting a data aggregator or node or repository, the level at a time algorithm checks at each level from level zero for the suitability of the node to be dependent of any parent node. For facilitating this decision, at each level there is a load controller node. The load controller decides whether a node Q can become parent of R. A node Q can become potential parent of R for a data item d, if both R and Q are interested in d, the incoherency bound for the data item at the node Q is less than or equal to that at the node R and preference factor is within some percentage of smallest preference factor at that level.

The preference factor of a repository is calculated using the following formula.

$$P_f = c_d * n_d / s_{qr}$$

Where

P_f is preference factor

c_d is communication delay factor

n_d is number of dependents of Q

s_{qr} is number of data items Q can serve to R

A repository is considered as a candidate only if the number of dependents currently served is less than the degree of cooperation. Degree of cooperation is calculated as given below.

Degree of cooperation = $\min(1/c{*}d_c/d_t, p_c)$

Where d_c is average communication delay and d_t is average computational delay

p_c is offered degree of cooperation

c is average number of dependents of the node which are interested in an update.

4.2.2. Data at a time algorithm

In this algorithm the main principle of inserting a new node in the data dissemination graph is that a node which requires more precise data is placed nearer to the source than the node which requires less precise data. For each data item a dynamic dissemination tree is constructed and the nodes in different tress cooperate with each other. In this algorithm it is assumed that a node requesting n number of data items has n number of resources. The following conditions are used for deciding the position of a new node R requiring a data item X.[11]

i) If the source of the data item has no child or if it has enough resources then the source itself is made the parent of R.

   Else a suitable sub-tree starting at child of the source is selected for inserting R, so that the level of R is the least and the communication delays between R and Q the parent of R is less. These requirements are recursively applied to select proper sub-tree of sub-trees until a node Q is found.

ii) If Q has data with less incoherence bound than incoherence bound for the data item at R and Q has enough resources to serve R, the Q is selected as parent of R.

   Else if Q has data with more incoherence bound than incoherence bound for the data item at R, then parent of Q is made the parent of R and R is made the parent of Q.

5. OPTIMAL ALGORITHM FOR EXECUTION OF QUERIES

The algorithms other than greedy algorithm that can be used for optimally dividing the query into sub-queries and assigning them to proper DAs are EGAWW proposed in [12] and primal-dual parallel algorithm for continuous aggregate query dissemination proposed in [13]. The two algorithms are discussed in this section and the second one is selected for optimal query execution as it is faster than the greedy algorithms. As mentioned in section 2, the sub-sets in set cover are equivalent to sub-queries and elements in the set cover are equivalent to data items in the query.

5.1. Enhanced Greedy Algorithm with Withdrawals

Greedy algorithm with withdrawals for set cover problem is proposed in [14]. An enhanced greedy algorithm with withdrawals for continuous aggregate query execution is proposed in [12]. Enhanced greedy algorithm has the main steps of greedy algorithm with an extended set of steps. In each iteration of the greedy algorithm the sub-sets with minimum average cost are selected. This step is continued in enhanced greedy algorithm with withdrawals. But there are two extra steps which gives the EGAWW algorithm an improved performance from greedy algorithm. First one is adding all the possible sub-sets or sub-queries $\{q_1^1, q_2^1, q_3^1, ..\}$ of given collection of sub-sets $T\{q_1, q_2, q_3, ..\}$ to the collection. This step does not change the cover. The cost of each sub-set q_i^1 is minimum of all qi costs for which q_i^1 is sub-set. Another step is withdrawal step. In the withdrawal step some of the already selected sub-sets are withdrawn from the solution. In this step some of non optimal sets are withdrawn improving the approximation ratio of the algorithm. In the algorithm V $\{v_1, v_2, v_3, ..\}$ is set of all data items and c is cost of a sub-query. It is calculated by $c=D/B^2$. Here D is sum difference and B is incoherency bound of the sub-query.

Approximation ratio of greedy algorithm is H_k and approximation ratio of enhanced greedy algorithm with withdrawals is $H_k-(k-1)/(8{*}k^9)$. The approximation ratio of enhanced greedy algorithm is better than that of greedy algorithm but takes more time than greedy algorithm. So, a better algorithm namely primal-dual parallel algorithm for continuous aggregate query dissemination (PDPA) is selected as optimal algorithm and discussed in the next section.

5.1.1. EGAWW Algorithm

```
1.  Initialize the solution collection of sub-sets or sub-queries (R) to NULL and the set of uncovered data items (U) to V and let $\alpha=1-1/k^3$
2.  For every sub-set $q\in T$ and every $q^1 \subseteq q$, add $q^1$ to T with cost of q as cost of $q^1$ .If $q^1 \subseteq q_1, q_2$ and $q_1, q_2 \in T$, then minimum cost of $q_1$ and $q_2$ is the cost of $q^1$. Let T: = $\{q_1, q_2, .., q_n\}$ be the resulting extended collection and cost of every $q_j$ is denoted by $c_j$
3.  While $U \neq NULL$ repeat steps 3.1 to 3.5
    3.1. For every j, let $w_j := |q_j \cap U|$. If $w_j \neq 0$ , $rj:= c_j / w_j$
    3.2. For every $q_j \in R$ and every sub collection $D \subseteq T$ of at most k subsets such that $q_j \subseteq \cup_{q\in D} (q)$, Let $w(D) := |\cup_{q\in D} (q) \cap U|$ be the still uncovered elements that belong to the subsets in D. If $w(D)\neq 0$, let $r(q_j,D) := (\sum_{i:q_i \in D} (p_i - p_j))/w(D)$
    3.3. Let j* be an index such that $r_{j*}$ is minimized, and let $j^{\sim}$, $D^{\sim}$ be such that $r(q_{j^\sim}, D^{\sim})$ is minimized.
    3.4. Greedy step: If $r_{j*} \leq r(q_{j^\sim}, D^{\sim})/\alpha$ then add $q_{j*}$ to the solution and define the price of the newly covered items as $r_{j*}$. i.e., do the following steps
        3.4.1. For every $v\in q_{j*} \cap U$
            3.4.1.1. $price(v) := r_{j*}$
        3.4.2. $U := U \setminus q_{j*}$
        3.4.3. $R := R \cup \{q_{j*}\}$
    3.5. Withdrawal step: If $r(q_{j^\sim}, D^{\sim}) < \alpha\, r_{j*}$ then replace $q_{j\sim}$ by the subsets in $D^{\sim}$ and define the price of the newly covered items as $r(q_{j^\sim}, D^{\sim})$. i.e., do the following steps
        3.5.1. For every $v \in \cup_{q\in D^\sim}(q) \cap U$
            3.5.1.1. $price(v) := r(q_{j^\sim}, D^{\sim})$
        3.5.2. $U := U \setminus \cup_{q \in D^\sim}(q)$
        3.5.3. $R := (R \setminus \{q_{j*}\}) \cup D^{\sim}$
4.  Return R
```

5.2. Primal -Dual Parallel Algorithm for Continuous Aggregate Query Execution

Since the application under consideration is a distributed application, a parallel algorithm named primal-dual parallel algorithm for continuous aggregate query dissemination proposed in [13] is selected as optimal algorithm and is discussed below. The main principle of framing this algorithm is as follows.

The primal-dual technique can be used to solve minimum weighted set cover problem. The dual for set cover problem is packing problem. It is a well known fact that minimization or maximization of a linear function with inequality conditions is a linear programming and the optimized function is objective function. Primal linear programming deals with minimization of objective function and dual LP deals with maximization of objective function. A primal-dual parallel approximation technique applied to weighted set and vertex cover is proposed in [15].

The set cover problem and vertex cover problem in hyper graph are equivalent. The dual of vertex cover problem is edge packing. As given in [15], an edge packing is said to be e-maximal packing if the approximate slackness condition is satisfied. The condition is given below. In this algorithm the cost of the edges are increased until approximate slackness condition is satisfied.

$\sum_{v \in Va}(q)\ \delta_v \geq c(q) - e*c(q)$

i.e., $e*c(q) \geq c(q) - \sum_{v \in Va}(q)\ \delta_v$

i.e., $e*c(q) \geq c_a(q)$

Where c(q) is cost of the sub-query or sub-sets (q) and $c_a(q)$ is residual cost of q.

The primal-dual parallel algorithm for continuous aggregate query dissemination is given below. In the algorithm V(q) is the set of data items in q and $V_a(q)$ is the set of uncovered data items in q, δ_v is cost of the edge(data item v) in the corresponding dual edge packing. The algorithm is implemented by pdpa (Primal-dual parallel algorithm for continuous aggregate query dissemination) function. In the algorithm T is set of different combinations of sub- queries , e is a small quantity such that the result is at most r/(1-e) times optimal value of result, $n_a(q)$ is number of uncovered data items in q and r is maximum number of duplicates of data in different sets. D and b are sum-difference and incoherency bound of the sub-queries. In each round of the algorithm the cost of the edges or data item is increased until the packing is e-maximal. The vertices or sub-queries satisfying the approximate slackness condition are selected as optimal sub-queries. Then the data items of the selected sub-queries are deleted from the other sub-queries and $n_a(q)$ is updated. This process is continued until all the data items are covered. At the end the optimal sub-queries are returned.

The time complexity of primal-dual algorithm is $O(r \log^2(m) \log(1/e))$ and the result is at most r/(1-e) the minimum value. For the problem of optimal execution of continuous aggregate queries prima-dual parallel algorithm is best suited as the algorithm obtains the results in least time and the application is dynamic and users need immediate results. So primal-dual parallel algorithm for query execution along with the push technique of data updates is selected in this paper as the optimal policy for data dissemination in CDNs. The section 6 compares primal-dual parallel algorithm with greedy algorithm.

5.2.1. PDPA Function

```
pdpa(V, T, c, e)
1.  for q∈T par-do
    1.1.  $c(q)=D/b^2$
    1.2.  $c_a(q)=c(q)$
    1.3.  $V_a(q)=V(q)$
    1.4.  $n_a(q)= |V(q)|$
2.  While there is an uncovered v do
    2.1.  for each uncovered v par-do
          2.1.1.  $\delta_v = \min_{q \in v} (c_a(q) / n_a(q))$
    2.2.  for each unselected q par-do
          2.2.1.  $c_a(q) = c_a(q) - \sum_{v \in Va(q)} \delta_v$
          2.2.2.  if $c_a(q) \leq e * c(q)$ then
                  2.2.2.1.  mark q as selected
                  2.2.2.2.  mark related v as covered
                  2.2.2.3.  update Va(q) and $n_a(q)$
3.  return the selected sub-queries
```

6. RESULT ANALYSIS

To compare the primal-dual parallel algorithm with greedy algorithm, a data dissemination network with different number of data aggregators ranging from 5 DAs to 40 DAs with random data items is taken. The greedy algorithm and primal-dual algorithm are implemented using c code. Data with random costs is given as input to the programs and got the number of iterations of the two algorithms as output. The graph is plotted by taking number of nodes on the x-axis and number of iterations of the algorithms on the y-axis. The two algorithms are compared for 2,3 and4 number of duplicates of the data items on different data aggregators/proxy servers.

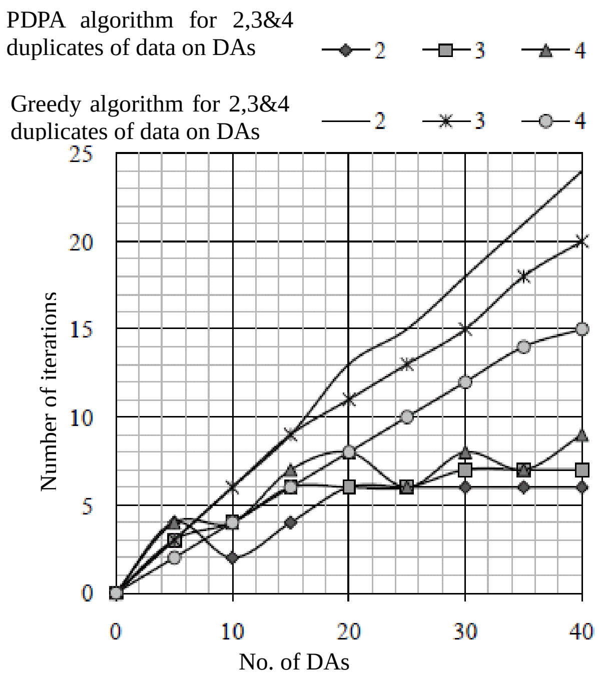

Fig. 3: The graph showing comparison between Greedy and Primal-dual parallel algorithm

The performance of greedy algorithm is satisfactory for small number of data aggregators. As the size of the network increases, the number of iterations of the greedy algorithm increases proportionately showing poor performance. But for more repetitions of the data items on DAs, the performance of the greedy algorithm is improved. Enhanced greedy algorithm with withdrawals has better approximation ratio than greedy algorithm but takes more time than greedy algorithm. So, greedy algorithm and primal-dual algorithm are compared here. The primal-dual parallel algorithm for continuous aggregate query execution outperforms greedy algorithms for all the cases of number of nodes and number of duplicates of the data items on different nodes. The graph in the Fig.3 confirms the fact that primal-dual algorithm is better than greedy algorithm for optimally disseminating continuous queries in data dissemination networks. Another fact is that for the primal-dual algorithm the graph is horizontal after 20 nodes. That means it is giving consistent results even when the number of nodes increase.

7. CONCLUSION

Optimal dissemination of data in CDNs is the main issue of this paper. In this paper an optimal policy for disseminating the data is obtained. That is an optimal algorithm namely primal-dual parallel algorithm for continuous aggregate query dissemination in combination with push policy of propagating the updates of data from sources to DAs is selected as optimal policy of data dissemination in CDNs. The advantages of our policy is that push approach provides data to clients with high fidelity and the primal-dual parallel algorithm facilitates optimal assignment of continuous queries to DAs with better time complexity than the other algorithms and the result analysis confirms the fact. This policy is well suited for the applications where clients need data fidelity and faster execution of continuous aggregate queries like Election results. The construction of dynamic data dissemination graph is also discussed in this paper. Other policies like pull and push-and-pull can be used if resilience to failures is important than fidelity.

8. ACKNOWLEDGEMENTS

We convey our sincere gratitude to Dr. K. V. S. V. N. Raju for giving us continuous support to complete this work. We thank students of M. Tech. who helped us a lot in doing this work. We use this opportunity to express our thanks to A.U. Engineering College Principal and staff for giving us infrastructural facilities. We thank krithi Ramamritham of IIT, Bombay whose papers guided us to do this work. We dedicate this work to the Principal of S.R.K.R. Engineering College for giving us financial support.

9. REFERENCES

[1] Datta, A., Dutta, K., Thomas, H., VanderMeer, D., & Ramamritham, K. 2004. Proxy-based acceleration of dynamically generated content on the world wide web: An approach and implementation. ACM Transactions on Database Systems (TODS), 29(2).

[2] Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., & Weihl, B. 2002. Globally distributed content delivery. IEEE Internet Computing, 6(5).

[3] Gupta, R., & Ramamritham, K. 2007. Optimized query planning of continuous aggregation queries in dynamic data dissemination networks. In Proceedings of the 16th international conference on World Wide Web. ACM.

[4] Shah, S., Ramamritham, K., & Shenoy, P. 2002. Maintaining coherency of dynamic data in cooperating repositories. In Proceedings of the 28th international conference on Very Large Data Bases. VLDB Endowment.

[5] Ramamritham, K. 2010. Maintaining coherent views over dynamic distributed data. In Proceedings of the 6th international conference on Distributed Computing and Internet Technology. Springer Berlin Heidelberg.

[6] Hochbaum, D. S. 1996. Approximation algorithms for NP-hard problems. PWS Publishing Co.

[7] Deolasee, P., Katkar, A., Panchbudhe, A., Ramamritham, K., & Shenoy, P. 2001. Adaptive push-pull: disseminating dynamic web data. In Proceedings of the 10th international conference on World Wide Web. ACM.

[8] Ninan, A. G., Kulkarni, P., Shenoy, P., Ramamritham, K., & Tewari, R. 2003. Scalable consistency maintenance in content distribution networks using cooperative leases. IEEE Transactions on Knowledge and Data Engineering, 15(4).

[9] Srinivasan, R., Liang, C., & Ramamritham, K. 1998. Maintaining temporal coherency of virtual data warehouses. In Proceedings of Real-Time Systems Symposium. IEEE.

[10] Gray, C., & Cheriton, D. 1989. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. ACM, 23(5).

[11] Shah, S., Ramamritham, K., & Shenoy, P. 2004. Resilient and coherence preserving dissemination of dynamic data using cooperating peers. IEEE Transactions on Knowledge and Data Engineering, 16(7).

[12] Gadiraju, M., & Kumari, V. V. 2010. Distribution of continuous queries over data aggregators in dynamic data dissemination networks. In Information and Communication Technologies. Springer Berlin Heidelberg.

[13] Mahesh Gadiraju, V. Valli Kumari. “Primal-dual parallel algorithm for continuous aggregate query dissemination”, Submitted to ICACCI2013, Mysore, India.

[14] Hassin, R., & Levin, A. 2005. A better-than-greedy approximation algorithm for the minimum set cover problem. SIAM Journal on Computing, 35(1).

[15] Khuller, S., Vishkin, U., & Young, N. 1994. A primal-dual parallel approximation technique applied to weighted set and vertex covers. Journal of Algorithms, 17(2).