A Distributed Hybrid Load Management Model for Anycast CDNs

Jing'an Xue*, Haibo Wang**, Jilong Wang**, Zhe Chen*, Chuang Wang* and Tong Li*

*Huawei, Email: {xuejingan, chenzhe17, wangchuang, li.tong}@huawei.com
**Tsinghua University, Email: hbwang1994@gmail.com, wjl@cernet.edu.cn

***Abstract*—Anycast content delivery networks rely on the underlying routing to schedule clients to their nearby service nodes, which however is not natively aware of server load or path latency. Requests burst from some regions may cause overload and hurt user experience. This scenario demands of quickly adjusting clients to other nearby servers with available capacity. However, state-of-the-art solutions do not work well. On one hand, native routing-based scheduling is not flexible and precise enough, which may cause cascading damage and interrupt ongoing sessions. On the other hand, centralized algorithm is vulnerable and not responsive due to high complexity. We propose a practical distributed hybrid load management model to solve load burst problem. First, the hybrid mechanism leverages flexible DNS-based redirection, which can schedule at per-request granularity without interrupting ongoing sessions. Second, the distributed model is responsive by reducing computation overhead and theoretically guarantees to converge to the optimal solution. Based on the model, we further propose an cooperative and two heuristic distributed algorithms. At last, using a measurement dataset, we demonstrate their effectiveness and scalability, and illustrate how to adapt them to different scenarios.**

***Index Terms*—anycast, content delivery network, scheduling, load management**

I. Introduction

Content delivery networks (CDNs) have been being widely employed nowadays for better performance by mapping clients to their nearby service nodes. Client mapping mechanism basically includes traditional DNS-based one and recently emerging anycast routing-based one. In anycast CDNs, service nodes in different regions announce the same IP prefixes through BGP, the de-facto inter-domain routing protocol. Clients will be mapped to the *closest node in terms of routing metrics, e.g., economic cost*. Anycast-based way do not need infrastructure to measure performance metric for scheduling, it can avoid the well-known inaccurate client localization problem in DNS-based way [1–3] and has shorter fail-over time.

However, there exist some native drawbacks in anycast CDNs that simply schedule clients based on routing metrics. This is beacuse routing-based scheduling is not aware of path latency or server load [4–7], it cannot react timely if load burst occurs. In other words, load burst remains the main challenge for anycast CDNs. Generally, the scenarios of load burst roughly fall into three categories: 1) Load fluctuation because of normal periodicity. It happens frequently and can be solved by proper capacity provision. 2) Load surge caused by network attacks, *e.g.,* DDoS, which is malicious and extremely heavy that needs to be blocked immediately. Anycast mechanism itself can inherently prevent distributed traffic from aggregating, *e.g.,* AS112 Project. 3) Sudden increase of loads in some regions, *e.g.,* when important news occur. This scenario happens frequently with a bit higher loads, which is not cost-effective to handle by provision. Clients need to be quickly adjusted to other **nearby servers with available capacity** timely and flexibly.

Traditional load management methods are not specifically designed for anycast service [8–10]. In terms of implementation mechanism, some traffic engineering works [11, 12] investigate routing-based scheduling as fine-grained routing technology advances. For example, by employing intelligent controller, traffic congestion on hot links could be avoided. However, it is not sophisticated enough since ongoing TCP sessions will be interrupted and cascading overload could happen on other healthy nodes [13]. In terms of formalized model, previous work [14] designs a centralized scheduling model for anycast CDNs to minimize network cost such as client-server distance, as well as the number of interrupted TCP sessions. However, centralized scheduling model has several drawbacks. First, there exists the single point failure problem. Second, computation complexity on the central controller is very high. When load suddenly increases, re-scheduling may not be timely enough.

To mitigate the problem of load burst in anycast CDNs, we first formulate scheduling as an optimization problem to minimize network cost under the constraints of server capacity. We then propose a distributed hybrid scheduling architecture. Specifically, the logical central scheduler is equivalently replaced by a set of distributed sub-controllers, *i.e.,* authoritative DNS (ADNS) colocated with each ayncasted service node, which also share common anycast prefixes. They can naturally monitor local load in each node since all service starts with DNS. They can also execute scheduling policy at per-request granularity without interrupting ongoing TCP sessions.

We further design a sophisticated distributed algorithm which can converge to the optimal solution with few communication between sub-controllers. Specifically, we first decouple the global problem to equivalent distributed local problems at each sub-controller, thereby avoiding problems caused by centralized management. Based on theoretical analysis, we put forward a cooperative distributed algorithm **ACCO**. The global optimal mapping is realized by each sub-controller cooperatively optimizing its local client mapping problem. To further improve scalability and timeliness, we propose two non-communication heuristic algorithms **Dprob** and **Drand**.

Using a large-scale measurement dataset, we demonstrate the effectiveness and scalability of the proposed load management model. Basically, as service nodes increase, the convergence time of ACCO grows much slower than the centralized method, which shows good scalability. Besides, all three algorithms can achieve satisfying proximity. On one hand, we test how well clients are mapped to nearby available nodes against different extent of load burst. Results show all three algorithms work well. The average ranks of client mappings are close to 1-3 and gradually converge to 1 as the burst extent decrease. On the other hand, heuristic algorithms can achieve better scalability and responsiveness by sacrificing accuracy. Particularly, computation time of Drand and Dprob almost does not change with the increase of service nodes and burst load, which performs better than cooperative ACCO, since they are loosely coupled. However, they may map a few clients to non-optimal nodes and result in a few service nodes overloaded while ACCO does not. The administrators could choose proper algorithm tools for specific scenario requirements by trading off between accuracy and responsiveness.

Our main contributions can be summarized as follows:

- We formulate the optimization problem of global load scheduling and further propose a distributed load-aware model, which overcomes the drawbacks of the centralized management model and theoretically converges to the same optimal solution.
- We design a practical hybrid load management mechanism by leveraging a set of distributed ADNS as sub-controllers. It enables flexible scheduling at per-request granularity without interrupting ongoing sessions.
- Based on the distributed model, we propose three scheduling algorithms, namely ACCO, Dprob and Drand, to satisfy different demands of scheduling effectiveness and timeliness. Experiment results show that compared with the centralized method, distributed algorithms has better scalability. Moreover, the three algorithms have different benefits in terms of accuracy and timeliness.

This paper is organized as follows. Load management problem and constraints are formulated in Section II. Section III presents the architecture of hybrid scheduling mechanism the distributed scheduling model. We propose three scheduling algorithms in Section IV and evaluate their scalability and performance in Section V. Finally, we summarize related works in Section VI and conclude the work in Section VII.

II. Load Management Problem for Anycast CDNs

In this section, we formulate the basic problem of load burst in anycast CDNs. Note that we use the terms "load scheduling/management" and "client mapping" interchangeably. We summarize key notations in Table I.

Basically, user experience is crucially important for CDNs. Load scheduling for CDNs should minimize network cost such as global client-server proximity and make the best use of inherent proximity provided by anycast. Meanwhile, scheduler should also reasonably map client requests to service node with available capacity, *i.e.,* the rate of requests should note exceed node capacity threshold. As stated in Section I, load burst in regions cannot be solved by pre-provision and require quick re-scheduling by directing newly arrived requests to other nearby available nodes. If overload happens, insufficient resources may cause long service queues and slow response.

TABLE I
Key notations used in the paper

Notation	Description
$Node$	Set of service nodes (colocated with control nodes, $\lvert Node\rvert = N$).
R	Set of client regions ($\lvert R\rvert = M$).
$R^{(n)}$	Client regions scheduled by control node n ($R^{(n)} \subseteq R$).
T_i	Capacity of service node i.
r_j	Requests originated from region j.
$x_{i,j}$	Proportion of requests from region j allocated to node i.
B_i	Total requests handled by service node i.
P_i	Proportion of requests handled by service node i.
λ_i	Penalty factor for node i when overloading.
μ_j	Constraint coefficient for region j.

Assume that there are totally N service nodes in an anycast CDN, consisting of set $Node$ ($|Node| = N$). The available capacity (number of requests) of each service node i is T_i, hereafter noted as threshold. All clients are divided into M client regions, consisting of the set R ($|R| = M$). The total requests generated per unit time in each region j is r_j. Without loss of generality and expressiveness, network cost regarding node i serving client region j is referred as funcation $cost(i, j)$. There exists many ways to express cost. For example, when bandwidth and energy consumption cost is optimized to minimize operation costs, the cost function can be a complex nonlinear function with regard to the traffic volume[15]. $cost(i, j)$ for proximity is generally proportional to the traffic volume and distance $d_{i,j}$ between service node i and client region j, *i.e.,* $cost(i, j) = \alpha \cdot r_j \cdot d_{i,j}$ (α is a constant). In practice, there are many methods that can measure and estimate the complete network cost matrix [16–18]. Although helpful to improve the accuracy of our model, they are beyond the concern of this study.

Briefly, load management problem of load burst in anycast CDNs is to re-schedule quickly to minimize the network cost while meeting the available capacity of each service node. The global problem $Q^{(g)}$ can be formulated as follows:

$$\begin{aligned} minimize \quad & \sum_{i=1}^{N}\sum_{j=1}^{M} cost\,(i,j)\, x_{i,j} \\ s.t. \quad & \sum_{j=1}^{M} r_j x_{i,j} \le T_i, \quad \forall i \in Node \\ & \sum_{i=1}^{N} x_{i,j} = 1, \quad \forall j \in R \end{aligned} \tag{1}$$

where $x_{i,j} \in [0, 1]$ represents the proportion of requests from region j mapped to service node i. The first constraint means that all service nodes are not overloaded and the second means that all client requests will be served. Assuming that the total requests from all regions is B ($B = \sum_{j=1}^{M} r_j$), the total fraction of requests handled by each service node

i can be represented as $P_i = \frac{\sum_{j=1}^{M} r_j x_{i,j}}{B}$. For brevity, the first constraint can be transformed into the following form:

$$B \bullet P_i \leq T_i, \quad \forall i \in Node \tag{2}$$

It usually needs to make a trade-off between the requirements of proximity and server capacity. For example, in order to keep the load of a service node below threshold, we need to schedule some clients to nodes with longer distance.

III. Load-aware anycast CDN architecture

In this section, we propose a distributed hybrid load-aware architecture to solve the load management problem formalized in Section II. Next we illustrate the implementation and theoretical scheduling model in detail.

A. Hybrid scheduling architecture for Anycast CDNs

We first introduce the implementation architecture of load management for anycast CDNs. Generally, Anycast CDNs choose to interconnect with a single large upstream network providers [4, 19] with worldwide coverage at many physical points of presence (PoPs). *e.g.,* Cloudflare (AS13335) and the telecom operator Cogent (AS174) interconnect at about 40 PoPs. Then all service nodes connect to the closest PoPs and announce the same IP prefixes from the same Anonymous System (AS) through Border Gateway Protocol (BGP). Consequently, all nodes share the same prefixes and client requests from different edge eyeball networks will be inherently routed to their nearest service nodes in terms of routing metrics.

Fig. 1 briefly illustrates the architecture of a basic load-aware anycast CDN d with three widely distributed nodes. To achieve precise load scheduling control of anycast CDNs, a global scheduler need to *1) monitor global system status, 2) compute optimal client-server mapping in a reasonable time and 3) carry out the scheduling plan*. We now discuss the implementation details of the three functions.

First, there are two candidate mechanism to *execute the scheduling plan*: **routing-based** and **DNS-based**. Since client mapping of anycast CDNs is natively based on routing, the former mechanism is realized by sending control signals to each node to change their route announcement,thereby realizing load re-mapping [4, 20]. For instance, for the root DNS server, network administrators achieve load balance by using *no-export* attribute or *AS Path Prepending* method. Precise control of inter-domain routing is very difficult [10] and routing-based scheduling may cause cascading damage [13]. The latter one is a hybrid mechanism by using an ADNS as the scheduler to redirect requests. Specifically, when the available service capacity is sufficient, requests are directed to the nearest node by default anycast routing. When loads burst, the central scheduler replies DNS request with the unique unicast address of a specific service node, so that subsequent content request can be routed to the designated service node rather than the one chose by anycast routing. As shown in Fig. 1, clients in $Region2$ will be directed to the service node T_2 by anycast under normal circumstances. When T_2 is overloaded, the central scheduler ADNS will return the unicast address of T_3 based on global load state, and subsequent content requests will be scheduled to T_3. It enables flexible scheduling at per-request granularity without interrupting ongoing sessions.

Second, the scheduler needs to monitor two aspects of status timely, *i.e., the remaining available capacity of each nodes* and *the amount of requests from each regions*. The former status can be sent to the scheduler through routing session or separate channels[14]. The latter can be obtained through ADNS, *i.e.,* DNS request rate reflects approximate content request rate.

Last but not least, computing client mapping efficiently is our main focus. Theoretically, a central scheduler with a complete view of the overall system state can compute the optimal re-mapping matrix X that satisfies performance and capacity requirements. However, it is not practical to take a centralized manner. First, there exists single point failure problem. Besides, it is not scalable since the computational complexity is too high to get re-mapping timely when load suddenly increases. To overcome these limitations, a distributed load management model is needed, where the logical global scheduler is equivalently replaced by several distributed **sub-controllers**.

In fact, DNS itself is a distributed system, *i.e.,* there is a module of ADNS colocated with each service node in the data center [21, 22]. They share the same anycast prefixes so that associated DNS requests and content requests from the same client will be directed to the same node. All regions are partitioned and associated with a service node based on anycast routing. This implementation makes the ADNS in each data center a good **sub-controller**, since it can locally perceive the remaining available capacity of the colocated service node and the corresponding requests number. It can also control the client mapping of its associated regions by returning anycast address or unicast address. The distributed load management architecture for anycast CDNs is shown in Fig. 1(c). The number of sub-controller is also N and we also use n to index sub-controller due to their colocation.

Next, we formally show how to distributedly solve the load burst problem formalized in Section II.

B. Distributed Load Management Model

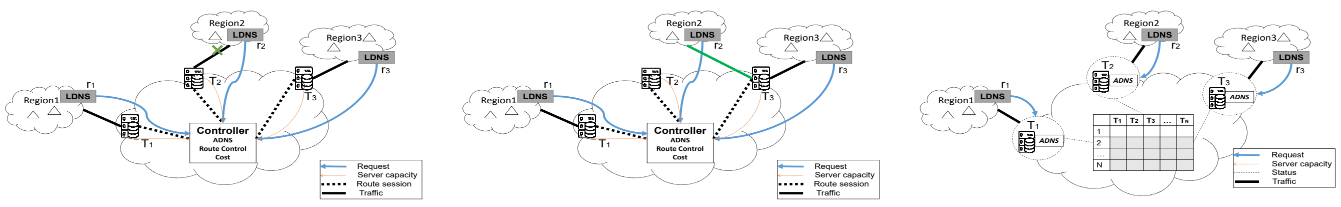
The global optimization problem $Q^{(g)}$ in (1) can be solved by *Lagrangian* method. We first slack the inequality constraint in (1) into the following form:

$$(B \bullet P_i)^2 - {T_i}^2 \leq 0 \ , \quad \forall i \in Node \tag{3}$$

After that, the Lagrangian dual of $Q^{(g)}$ can be expressed as:

$$min_X \ max_\lambda \ \ L(X, \lambda, \mu) = \ Q^{(g)} + \sum_{i \in Node} \lambda_i \left((B \bullet P_i)^2 - {T_i}^2 \right) + \sum_{j \in R} \mu_j \left(\sum_{i=1}^{N} x_{i,j} - 1 \right) \tag{4}$$

where λ_i and μ_j are dual coefficients of constraints. λ_i can also been explained as the penalty cost when the service node i are overloaded. X represents the mapping matrix, which is the variable to be solved and optimized.



(a) before scheduling (b) after scheduling (c) the distributed architecture

Fig. 1. A basic load management architecture for anycast CDNs (a)(b) and a distributed one (c)

Suppose we have N' sub-controllers. The global optimization goal $Q^{(g)}$ can be transformed as the sum of all sub-controllers' local optimization $Q^{(l)}$ as follows:

$$Q^{(g)} = \sum_{n'\in N'} Q_{n'}^{(l)} = \sum_{n'\in N'} \sum_{i=1}^{N} \sum_{j\in R^{(n')}} cost\,(i,j)\, x_{i,j} \tag{5}$$

For the global optimization, we need to search the optimal solution in the global solution space ($i.e.$, mappings between all client regions in R and all service nodes in $Node$). Instead, in this distributed model, each sub-controller n is only responsible for clients, noted as $R^{(n')}$ ($R^{(n')} \subseteq R$). The expectation of the size of $R^{(n')}$ is M/N' (*i.e.,* $E(R^{(n')}) = M/N'$). Therefore, the complexity of solving the local optimization problems on each sub-controller is greatly reduced.

The form of the local optimization problem for each sub-controller is consist with the global optimization problem. For a given sub-controller n', it can be expressed as follows:

$$\begin{aligned} minimize \quad & \sum_{i=1}^{N} \sum_{j\in R^{(n')}} cost\,(i,j)\, x_{i,j} \\ s.j. \sum_{n'\in N'} \sum_{j\in R^{(n')}} & r_j x_{i,j} \le T_i, \forall i \in Node \\ & \sum_{i=1}^{N} x_{i,j} = 1, \quad \forall j \in R^{(n')} \end{aligned} \tag{6}$$

Although the optimization goal is decoupled for each sub-controller, the constraints are not, because each sub-controller still need global status, *e.g.,* available capacity of all service nodes, to decide its local client mappings. Meanwhile, each service node receives requests scheduled by all sub-controllers. Therefore, sub-controllers need to communicate for synchronization to ensure the optimization can converge to the globally optimal one.

We next show how to decompose the problem for synchronization communication. We still use *Lagrangian* method as shown in (7):

$$\begin{aligned} min_X max_\lambda \;\; L'\,(X_{n'}, \lambda, \mu_{n'}) = Q_{n'}^{(l)} + \sum_{i\in Node} \lambda_i \left(B_i^2 - {T_i}^2\right) \\ + \sum_{j\in R^{(n')}} \mu_j \left(\sum_{i=1}^{N} x_{i,j} - 1\right) \end{aligned} \tag{7}$$

where B_i represents the total load of service node i and can be expressed as follows:

$$\begin{aligned} B_i &= \sum_{n'\in N'} \sum_{j\in R^{(n')}} r_j x_{i,j} \\ &= \sum_{j\in R^{(n')}} r_j x_{i,j} + \sum_{y\in N'\setminus\{n'\}} \sum_{j\in R^{(y)}} r_j x_{i,j} = B_{n'i} + B_{-n'i} \end{aligned} \tag{8}$$

where $B_{n'i}$ represents the load contributed by sub-controller n' itself to service node i, and $B_{-n'i}$ represents the total load scheduled by other $N-1$ sub-controllers except n' itself.

Each sub-controller only need to solve its own local optimization problem. For the sub-controller n', the first part of the optimization equation (7) ($i.e.$, $Q_{n'}^{(l)}$) is only related to itself, while the second part ($i.e.$, $\sum_{i\in Node} \lambda_i (B_i^2 - {T_i}^2)$) is related to the penalty coefficients λ_i ($i \in Node$) for all service nodes and the amount of requests contributed by the other $N'-1$ sub-controllers to every service node.

To solve its local problem under the hybrid architecture, sub-controller n needs two aspects of states. States obtained **locally** includes the cost matrix ($\{cost(i,j) | \forall i,j \in Node\}$) and the amount of the received requests ($\{r_j | j \in R^{(n)}\}$). Remote states obtained **by communication** includes λ_i and B_{-ni} ($i \in Node$).

IV. Distributed Scheduling Algorithm

Based on the above theoretical model, We first design a cooperative distributed algorithm, which converge to the same optimal solution as the central optimization problem. After that, we further propose two heuristic algorithms, which can reduce commutation by sacrificing a little accuracy.

A. Cooperative Distributed Algorithm (ACCO)

According to the previous theoretical analysis, we propose a cooperative distributed load management algorithm, namely ACCO (**A**ny**C**ast **CO**operative). Briefly, the algorithm contains iterative rounds and will stop when the iterative difference is less than the minimum tolerance μ in two successive rounds. In each round, every sub-controller execute three steps: collect the latest remote status as analyzed in Section III-B, optimize its local client mapping, update related status after re-mapping. Algorithm 1 describes the iterative solution process of the entire system, which includes detailed execution process of each sub-controller.

At initialization, λ_i and X_n can be randomized or clients are mapped to the nearest nodes. sub-controllers are updated in turn (*line 5*) and needs the latest state through communication before solving local optimization problems. In terms of B_{-ni}, the latest version of B_{*i} of the precedent $(n-1)$ nodes is updated in the current round while that of the rest nodes is updated in the last round (*line 6*). In terms of constraint coefficients λ, it can only obtain the last round version (*line 7*) since they are updated only at the end of each iteration (*line 13*). Next, we theoretically prove that ACCO can converge to the optimal solution.

Algorithm 1 ACCO

```
Input:  Node,$R$,$R^{(n)}$,$\{r_j|j \in R\}$,$\{T_i|i \in Node\}$
Output:  Mapping matrix $\mathbf{X}$
 1: $\lambda$ and $\mathbf{X}$ set random
 2: $k = 1$ ($k$ represents the number of iterations)
 3: while $\left|B_{-ni}^{(k)} - B_{-ni}^{(k-1)}\right| > \mu$ $(\forall\, i \in Node)$ do
 4:     for each sub-controller $n$ do
 5:         get the latest status $B_{-ni}$ $(\forall\, i \in Node)$ of other
            nodes: $\{B_{1i}^{(k)}, ..., B_{(n-1)i}^{(k)}, B_{(n+1)i}^{(k-1)}, ..., B_{Ni}^{(k-1)}\}$
 6:         get the latest status $\lambda_i$ $(\forall\, i \in Node)$ of other nodes:
            $\{\lambda_1^{(k-1)}, \lambda_2^{(k-1)}, ..., \lambda_N^{(k-1)}\}$
 7:         solve $Q_n^{(l)}$and get $\mathbf{X_n}^{(k)}$ and $\mu_n^{(k)}$
 8:         compute $\left\{B_{ni}^{(k)}|\forall i \in Node, B_{ni}^{(k)} = \sum_{j \in R^{(n)}} r_j x_{i,j}^{(k)}\right\}$
            and update
 9:         if $n = N$ then
10:             for each service node $i$ do
11:                 compute $B_i^{(k)} = \sum_{n \in Node} B_{ni}^{(k)}$
12:                 compute $\lambda_i^{(k)} = max\{0, \lambda_i^{(k-1)} +$
                    $\theta\left(\left(B_i^{(k)}\right)^2 - {T_i}^2\right)\}$
13:             end for
14:         end if
15:     end for
16:     $k = k + 1$
17: end while
```

Theorem 1. *The distributed load management algorithm ACCO described in Algorithm 1 will converge to the optimal solution of the global optimization problem $Q^{(g)}$ if the following two conditions are satisfied: (1) Each sub-controller iteratively solves the local optimization problem $Q^{(l)}$; (2) The constraint coefficients λ for all service nodes are updated after all sub-controllers finish the update of the mapping matrix $\mathbf{X}$ in the current round.*

Proof: Combined with (4) and (5), the global optimization problem can be expressed as follows:

$$min_x max_\lambda \;\; L\left(X, \lambda, \mu\right) = \sum_{n \in Node} Q_n^{(l)} + \sum_{i \in Node} \lambda_i \left(\left(B \bullet P_i\right)^2 - {T_i}^2\right) + \sum_{j \in R} \mu_j \left(\sum_{i=1}^{N} x_{i,j} - 1\right) \tag{9}$$

To effectively solve the problem, we adopt the dual form of the original problem ($i.e.$, $max_\lambda min_x\; L(X, \lambda, \mu)$). We omit the update of μ for brevity because it does not affect the proof.

First, given a fixed set $\{\lambda_i|i \in Node\}$, we need to minimize $L(X, \lambda, \mu)$. When sub-controllers iteratively optimizes local problems, $i.e.$, satisfying Condition (1) of Theorem 1, the overall process is the classical *Guass-Seidel* algorithm [23] and the local solution is equivalent to *Guass-Seidel* algorithm:

$$\mathbf{X_n^{(k+1)}} = argmin \;\; Q^{(g)}(\cdots, \mathbf{X_{n-1}^{(k+1)}}, \mathbf{X_n}, \mathbf{X_{n+1}^{(k)}}, \cdots) \tag{10}$$

where $\mathbf{X}_n$ represents client mappings of sub-controller n.

According to *Proposition3.9, Ch.3*[23], *Guass-Seidel* algorithm is convergent if $F : R^n \to R$ is a continuous and differentiable convex function on set $\mathbf{X}$ and it is also convex on any X_i when other sub-vectors are fixed, the solution $\mathbf{X}_n^{(k)}$ obtained by iterations will eventually converge. In our scenario, the objective function $L\left(X, \lambda, \mu\right)$ is continuous and differentiable, and also a convex function on $\mathbf{X}$ (its Hessian matrix can be proved semi-definitive). Besides, solving the local problem $Q_n^{(l)}$ is equivalent to minimizing the global problem $Q^{(g)}$ against variable X_i when other variables are fixed. Third, the function is quadratic so that the optimal solution for each local problem can be uniquely obtained. Therefore, all conditions in the *proposition* [23] are satisfied, and the solution for each sub-controller ($i.e.$, $\mathbf{X}_n^k$ $(n \in Node)$) will be convergent and uniquely obtained. Note that μ is also updated when solving the local problem in each sub-controller.

At the end of a round, based on the dual form, we need to solve the following problem:

$$\begin{aligned} maximize \quad & f(\lambda) = min_X \;\; L\left(X, \lambda, \mu\right) \\ & s.j. \quad \lambda \geq 0 \end{aligned} \tag{11}$$

Because $f(\lambda)$ is continuous and differentiable, we can solve the problem (11) by gradient descent, which is implemented in *line 13* of Algorithm 1 (the condition (2) of Theorem 1).

With regards to global optimization problem, the solution of the dual form is consistent with that of the primary form. Because each local problem $Q_n^{(l)}$ has an unique solution, *the distributed algorithm ACCO will converge to the optimal solution of the global optimization problem $Q^{(g)}$*.

Communication complexity for each node: sub-controller can adopt two ways to collect the state information: (1) Directly request state variables from other sub-controllers that generate the state. For instance, sub-controller n requires the amount of requests scheduled by other $N-1$ sub-controllers to all service nodes, which needs $N-1$ communications. (2) Set up a central state storage *CStore*, where columns are servers and rows are sub-controllers, as shown in Fig. 1(c). When sub-controllers need to update (its own row) or fetch (other rows) information, they only communicate with CStore, thereby reducing communication cost to $\mathcal{O}(1)$.

B. Non-communication Distributed Algorithm

We start with the ideally steady case where all service nodes have sufficient capacity, *i.e.,* capacity constraints does not exist. Basically, for clients in $R^{(n)}$, *the service node colocated with the sub-controller* n *is the closest node chosen by anycast* and the sub-controller should direct them to the colocated service node. Therefore, there is no need to communicate to obtain the remaining capacity of other service nodes and amount of requests contributed by other sub-controller.

In practice, CDN providers usually try to obtain the actual load B_i of each service node under normal scenario by pre-estimation or trial operation, and accordingly provision the capacity of each node, such as 10% higher than the actual load. Therefore, the above non-communication scheduling method is feasible if no load burst happens. However, the load management needs to handle unconventional scenarios of frequent load burst, cooperation is needed to ensure no overload. By trading off between the two cases, the constraints of the sub-controllers in ACCO can *degenerate* to only consider the available capacity of local service nodes. In other word, in most normal cases sub-controllers will greedily direct its catchment to the colocated service node by returning anycast addresses. When load burst happens, some unicast address will be returned to direct clients to other nodes. It can eliminate the communication cost and achieve better scalability by computing client mapping in parallel.

Based on the above rationale, we further propose two heuristic methods that can be executed by all sub-controllers simultaneously without communication. When the amount of requests in $R^{(n)}$ exceeds the available capacity of the local service node, the strategies of the two algorithms to schedule the overloaded requests are described as follows:

- **Drand:** Randomly schedule the overloaded requests to one of the other $N-1$ nodes.
- **Dprob:** Schedule the overloaded requests to one of the other $N-1$ nodes with a certain probability. The scheduling principle is that the closer the service node is to the client, the more likely it is to be selected. Specifically, assuming the geographical distance between the client region j and the service node i is $d_{i,j}$, the priority of node i serving region j is $prio_{i,j} = \frac{1}{d_{i,j}}$ and the probability to be selected is $p_{i,j} = \frac{prio_{i,j}}{\sum_{i \in Node} prio_{i,j}}$.

V. Evaluation

In this section, we compare and evaluate the ACCO, Drand, and Dprob algorithms, and mainly focus on the trade-off between the scheduling effectiveness ($i.e.$, proximity) and responsiveness ($i.e.$, convergence time). We first describe the experiment settings and dataset. We then compare the scheduling scalability between ACCO and the centralized algorithm. Finally, we evaluate the effectiveness of algorithms under different extent of burst.

A. Experiment Settings

We first describe parameter initialization. We use the geographic location of clients and service nodes from a real measurement data. Specifically, we use a measurement platform [24] with $\mathcal{O}(1M)$ probes, *i.e.,* clients, to measure anycast CDN Cloudflare with 96 nodes. The normal client mapping relations

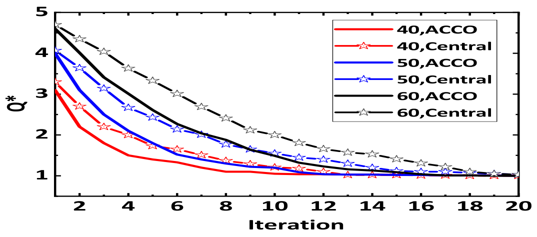


Fig. 2. The iterative convergence process of ACCO and Central for different numbers of service nodes

can also be measured as described in [19]. In our measurement, all clients are distributed in 186 time zones, 146 countries. With regard to service nodes, we select top-k popular nodes from the 96 ones in different experiment. Node popularity rank is computed by sorting the number of associated routable client prefixes. Specifically, we set a medium scale (35-60 service nodes) similar to Microsoft and Level 3 [7] when evaluating scalability in Section V-B. When we evaluate the scheduling effectiveness in Section V-C, it is fixed to 50 since CDN scale is not the main variable.

Besides, it is necessary to have an estimate for the amount of requests r_j in each client region. We assume that it follows *Poisson distribution* with a mean of 5Gbps [15]. The cost function $cost(i,j)$ in this problem is set to the product of geographical distance rank and the proportion of requests from region j.

In addition, we still need to set the capacity threshold of each service node T_i. Setting T_i equally across all nodes is not reasonable since the demand from each region is not balanced, which will lead to under or over utilization. As described in [20], anycast CDN providers usually take trial operation to get a proper estimation of request load on each service node and the proportion from each region. Therefore, we calculate T_i as $C * P_i$, where C is provisioned total system capacity and P_i is the request load proportion of each node under normal circumstances, *i.e.,* all clients are mapped to the service nodes with the lowest network cost ($i.e.$, the highest rank). Note that the total system capacity C should be just sufficient to serve all demands and cost-effective, similar as in general traffic engineering problems [14, 20]. We use *capacity coefficient* to adjust system capacity for evaluation, noted as $\rho = C/R$, where $R = \sum_j r_j$ and is the total requests.

B. Scalability

In this part, we compare the distributed algorithm ACCO with the centralized algorithm. For the centralized algorithm (noted as *Central* hereafter), the implementation is to solve the global problem immediately after obtaining all state information. For ACCO, the implementation follows Algorithm 1, *i.e.,* each sub-controller is strictly synchronous and executes serially during the iterations. In this experiment, the *capacity coefficient* ρ is set to 1.1 and the available capacity for each service node is set accordingly. We assume that the amount of requests from radomly selected 10% client regions increases by 15% suddenly.

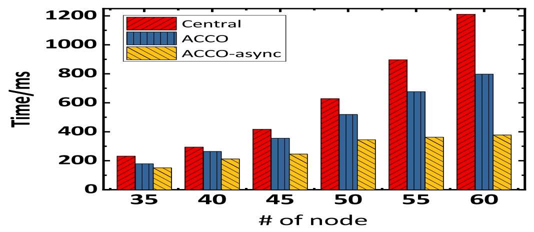


Fig. 3. The convergence time against the increase of service nodes

Fig. 2 shows the iterative convergence process of ACCO and Central for different numbers of service nodes. X-axis represents the number of iterations and Y-axis represents the optimization goal. We observe that both algorithms will converge after a certain number of iterations, and the goal values at convergence are close to 1. This is because most of the requests (except the overloaded ones) are still directed to the optimal node (*i.e.*, distance rank is 1), since the overload problem in the experiment settings is not serious. Therefore, we can conclude that ACCO and Central will eventually converge to the optimal value, which verifies the correctness of Theorem 1 by experiment. Besides, for the same number of service nodes, ACCO converges faster than Central.

Sub-controllers in ACCO solve the local problems in a sequential and circular manner, whose parallelism can be further improved by sacrificing a little accuracy. Specifically, sub-controllers do not need strict synchronization and performs iterations independently. When one round ends, regardless of whether the state information for all nodes is the latest, a new round of iteration directly performs. The asynchronous distributed algorithm is called *ACCO-async*. Given that the division of client regions usually does not change while the CDN scale may expand, we compare the convergence time against the increase of the service nodes for the three algorithms Central, ACCO and ACCO-async in Fig. 3.

We can see that Central has the worst scalability. The convergence time increases dramatically as the CDN scale expands. For ACCO, although the sub-controllers need to solve the local problems serially, the running time grows approximately linear with the number of nodes. This is because the solution space for local problems is greatly reduced. The scalability of ACCO-async outperforms the other two algorithms due to the sufficient parallelism across nodes. However, it is easy to overload other service nodes, which needs to be used according to the actual situation.

C. Effectiveness against the Extent of Load Burst

We next evaluate all the algorithms under different extent of load burst. We assume that the requests from a certain proportion (α) of client regions increases sharply. α follows *uniform distribution* with the mean $\bar{\alpha}$ and the corresponding number of client regions are randomly selected. β means the increased proportion of the load in each selected client region and takes three different values, *i.e.,* 0.15, 0.2, and 0.25.

Fig. 4 shows the final average rank of service nodes for all client regions under the circumstances of load burst ($\bar{\alpha} = 0.15$ for the upper plots and $\bar{\alpha} = 0.2$ for the lower plots). X-axis represents the *capacity coefficient* ρ. Obviously, with the increase of α or β, the overall optimization capability for all three algorithms is reduced, and clients are mapped to lower ranked nodes. In terms of proximity, ACCO can schedule the requests to higher ranked service nodes compared with Dprob and Drand, which is because ACCO strictly solves the global problem and get the optimal solution. Because Dprob also assigns a higher weight to the closer node, we can observe that Dprob works better than Drand. Due to the proximity of clients in anycast CDNs, the gap between them is not large.

Note that Drand and Dprob do not consider the global state (only local state) and some other service nodes may be overloaded. Fig. 5 shows the average number of overloaded nodes and deviation when $\bar{\alpha} = 0.15$. We can find that with the increase of β, the average number of overloaded nodes increases, but the total number is relatively small. Furthermore, in our measurement, a large number of clients are located densely in Europe. The sudden increased requests for these clients may be scheduled to other service nodes whose capacity is limited, leading to the overload. It means that we may overestimate the number of the overloaded nodes in the experiment settings. As a result, the two heuristic algorithms are applicable in practice.

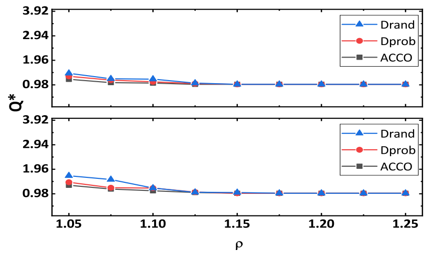
Besides, Drand and Dprob are fairly loosely coupled so that the running time almost does not change with the number of service nodes and the amount of the load, while ACCO needs longer convergence time as described in Section V-B. The advantage of ACCO is that the overload problem can be eliminated when the total system capacity is sufficient although the scalability is not as good as the two completely distributed algorithms. Network administrators can flexibly adopt these three tools based on the scenario requirements.

VI. Related Work

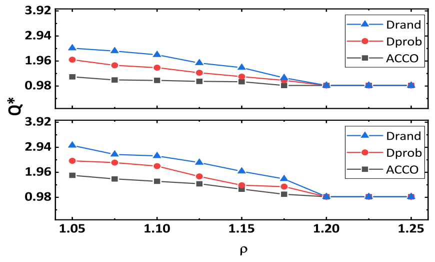
Previous works about anycast mainly focus on two aspects. One is to identify anycast nodes using various advanced methods [25–28]. The other one is to evaluate the performance of anycast services, such as client proximity [4–7, 29], stability [4, 7], deployment scheme [5, 6], reliability [30] and load management [4, 11, 20].

In this paper, we mainly focus on the problem of the load burst management in anycast CDNs. There exist many traditional load management models [8–10]. Nevertheless, few of them are designed for anycast CDNs, and it is difficult to make them compatible with the anycast mechanism. Some works achieve the request scheduling by the routing configuration [4, 11, 12, 20, 31]. For example, Hashim *et al.* propose "active anycast" to integrate the network latency and congestion state into routers for scheduling [11]. Merwe *et al.* design an intelligent routing control mechanism (exchanging BGP MED (Multi-Exit Discriminators) with appropriate peering AS) to avoid congested links [12]. However, the routing-based scheduling method is not flexible enough and may cause session interruption.

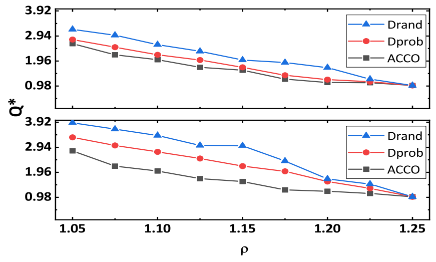
Alzoubi *et al* propose a centralized anycast CDN load management model to minimize the network cost as well as the number of interrupted TCP sessions [14]. Unfortunately, The computational overhead of the centralized algorithm is



(a) $\beta = 0.15$

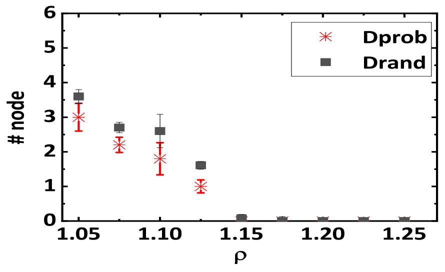


(b) $\beta = 0.2$

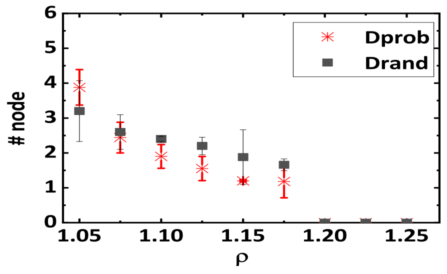


(c) $\beta = 0.25$

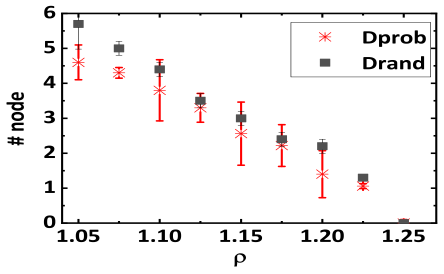
Fig. 4. The average client mapping rank for different values of β (increased proportion of requests). Upper plots: $\bar{\alpha} = 0.15$; Lower plots: $\bar{\alpha} = 0.2$



(a) $\beta = 0.15$



(b) $\beta = 0.2$



(c) $\beta = 0.25$

Fig. 5. The average number of overloaded nodes and deviation when applying Dprob and Drand. ($\bar{\alpha} = 0.15$ for the three plots)

expensive and the central scheduler is easy to be an attack target. To fill the gaps, in this paper, we propose a distributed hybrid scheduling model, which overcomes the drawbacks of the routing-based scheduling mechanism and the centralized scheduling model.

VII. Conclusion

In this paper, we propose a distributed hybrid scheduling model for anycast CDNs to alleviate load burst. On implementation, the hybrid scheduling mechanism overcomes drawbacks of the traditional routing-based scheduling mechanism. On the other hand, the distributed scheduling model decouples the global load management problem and improves the scalability, which can theoretically guarantee to converge to the global optimal solution. The characteristics of anycast CDNs provide the basis for the implementation of the model. Based on the model, we further propose three distributed scheduling algorithms. The experiments demonstrate that they can meet the different demands of the scheduling effectiveness and timeliness. We believe that our work is a meaningful step towards exploring the load-aware anycast CDN architecture.

References

[1] Z. M. Mao, C. Cranor, F. Douglis, and M. Rabinovich, "A precise and efficient evaluation of the proximity between web clients and their local dns servers," in *Proc. of USENIX ATC*. USENIX, 2002.

[2] J. S. Otto and e. Sánchez, "Content delivery and the natural evolution of dns," in *IMC*. ACM, 2012, pp. 523–536.

[3] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan, "On the responsiveness of dns-based network control," in *IMC,Sicily, Italy,*. ACM, 2004.

[4] H. Ballani, P. Francis, and S. Ratnasamy, "A measurement-based deployment proposal for IP anycast," in *IMC, Brazil, October 25-27*. ACM, 2006, pp. 231–244.

[5] L. Colitti., "Evaluating the effect of anycast on dns root." in *ID RIPE-393.*, 2006.

[6] R. de Oliveira Schmidt, J. S. Heidemann, and J. H. Kuipers, "Anycast latency: How many sites are enough?" in *PAM, Australia, March*. Springer, 2017.

[7] M. Calder and e. Ashley Flavel, "Analyzing the performance of an anycast CDN," in *IMC, Tokyo, Japan, October*. ACM, 2015.

[8] L. Wang, V. S. Pai, and L. L. Peterson, "The effectiveness of request redirection on CDN robustness," in *OSDI, Boston, USA, Dec.* USENIX, 2002.

[9] M. Colajanni, P. S. Yu, and D. M. Dias, "Scheduling algorithms for distributed web servers," in *ICDCS, Baltimore, MD, USA, May*. IEEE, 1997.

[10] P. Verkaik and e. Dan Pei, "Wresting control from BGP: scalable fine-grained route control," in *ATC, Santa Clara, CA, USA, June*, 2007, pp. 295–308.

[11] H. B. Hashim and J. A. Manan, "An active anycast rtt-based server selection technique," in *Proc. 13th IEEE Int. Conf. Networks*, vol. 1, 2005.

[12] J. Van der Merwe and e. Cepleanu, "Dynamic connectivity management with an intelligent route service control point," in *IMW*, 2006, pp. 29–34.

[13] A. Flavel, P. Mani, D. A. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev, "Fastroute: A scalable load-aware anycast routing architecture for modern cdns," in *NSDI, Oakland, CA, USA, May*, 2015, pp. 381–394.

[14] H. A. Alzoubi, S. Lee, and *et al.*, "Anycast CDNs revisited," in *WWW, Beijing, China, April*. ACM, 2008, pp. 277–286.

[15] S. Hasan, S. Gorinsky, C. Dovrolis, and R. K. Sitaraman, "Trade-offs in optimizing the cache deployments of cdns," in *INFOCOM,Toronto, Canada, April 27*. IEEE, 2014, pp. 460–468.

[16] P. K. Gummadi, S. Saroiu, and S. D. Gribble, "King: estimating latency between arbitrary internet end hosts," in *IMW, Marseille, France, November*. ACM, 2002, pp. 5–18.

[17] F. Dabek, R. Cox, M. F. Kaashoek, and R. T. Morris, "Vivaldi: a decentralized network coordinate system," in *SIGCOMM, Portland, Oregon, USA*. ACM, 2004, pp. 15–26.

[18] Caida ark project. [Online]. Available: www.caida.org/projects/ark/

[19] J. Xue, W. Dang, H. Wang, J. Wang, and H. Wang, "Evaluating performance and inefficient routing of an anycast CDN," in *IWQoS 2019, Phoenix, USA, June*.

[20] W. B. de Vries, R. de Oliveira Schmidt, and *et al.*, "Broad and load-aware anycast mapping with verfploeter," in *IMC, London, UK, November*. ACM, 2017, pp. 477–488.

[21] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: a platform for high-performance Internet applications," vol. 44, no. 3, 2010.

[22] Cloudflare. [Online]. Available: www.cloudflare.com

[23] B. D. P, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[24] Luminati proxy. [Online]. Available: luminati.io

[25] X. Fan, J. S. Heidemann, and R. Govindan, "Evaluating anycast in the domain name system," in *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*. IEEE, 2013, pp. 1681–1689.

[26] D. Madory and C. Cook, "Who are the anycasters?" in *NANOG59*, 2014.

[27] e. Drossi, "where are anycasters." in *RIPE2014.*, 2014.

[28] D. Cicalese, J. Augé, D. Joumblatt, T. Friedman, and D. Rossi, "Characterizing ipv4 anycast adoption and deployment," in *CoNEXT 2015, Heidelberg, Germany, December*. ACM, 2015, pp. 16:1–16:13.

[29] Z. Liu and e. Bradley Huffaker, "Two days in the life of the DNS anycast root servers," in *PAM 2007, Louvain-la-neuve, Belgium, April 5-6*, ser. Lecture Notes in Computer Science, vol. 4427. Springer, 2007, pp. 125–134.

[30] G. C. M. Moura, R. de Oliveira Schmidt, and *et al.*, "Anycast vs. ddos: Evaluating the november 2015 root DNS event," in *IMC, Santa Monica, CA, USA, November*. ACM, 2016, pp. 255–270.

[31] N. G. Duffield and e. Kartik Gopalan, "Measurement informed route selection," in *PAM 2007, Louvain-la-neuve, Belgium, April 5-6*, vol. 4427. Springer, 2007, pp. 250–254.