

On Dominating Set Allocation Policies in Real-Time Wide-Area Distributed Systems*

Chengdu Huang Tarek Abdelzaher
Department of Computer Science
University of Illinois at Urbana-Champaign
{chuang30, zaher}@cs.uiuc.edu

Xue Liu
School of Computer Science
McGill University
xueliu@cs.mcgill.ca

Abstract

This paper investigates resource allocation policies for achieving real-time content distribution with subsecond delay bounds on the current Internet. Resource allocation in real-time systems has been concerned primarily with meeting time constraints on single processors and multiprocessors. On a single processor, the main degree of freedom available for the real-time designer is the scheduling policy. On a (partitioned) multiprocessor, a second degree of freedom is the partitioning policy. This paper explores a third degree of freedom unique to large-scale (i.e., wide-area) distributed systems with non-negligible communication delays among individual nodes. We call it the *dominating set allocation policy*. This policy is a primary determinant of schedulability in such systems. We present some initial steps towards understanding the properties of different dominating set allocation policies in terms of resulting task schedulability. We describe an optimal dominating set allocation algorithm (subject to certain design decisions), propose a number of simple heuristics, and evaluate them using realistic Internet measurements and HTTP workload. The key contribution of this paper lies in the practical applicability of the proposed heuristics in achieving delay guarantees (with a high probability) over best-effort wide-area networks.

1. Introduction

In real-time scheduling theory, it is generally desired that resources be assigned to tasks in a manner that meets deadlines. There are primarily two types of policies that assign resources to tasks, depending on whether resources are assigned in time (which we call scheduling) or in space (which we call allocation). It may be argued that as systems grow larger (i.e., as load and resource multiplicity increase), the role of allocation dominates that of scheduling. Scheduling is inherently a consequence of multitasking, where more than one task or class of tasks share the same resource such as a CPU (for the sake of this discussion, a class of tasks refers to those tasks with the same deadline). In contrast, in systems where the load imposed by a class of tasks is much larger than the capacity of a single processor, it is possible to treat each processor as an indivisible unit that is not shared among different classes of tasks. The

relevant problem therefore becomes one of resource (e.g., processor) allocation. This formulation is especially true of environments such as Internet server farms where a finite small number of client classes are served by hundreds of servers. Different subsets of servers in the farm can thus be allocated to different classes. In large-scale distributed systems that run critical applications (e.g., content distribution services delivering latency sensitive content), physical performance isolation between different classes of content is commonly desired for privacy and security reasons. Hence, when allocating, any server is dedicated to serving only one class. A server allocation policy that meets real-time performance requirement is therefore needed.

In this paper, we consider the problem of resource allocation in wide-area distributed systems such that timing constraints are met. Servers in a wide-area network are separated by delays that often dominate the end-to-end client-perceived latency. Content should be allocated to servers such that there exists a path from each client's network entry point to each server that satisfies the delay bound of the client's class. Consider a graph where servers and other network entry points are the vertices. Given a delay bound, an edge is drawn between a pair of vertices if there exists a path between them that meets the delay bound. To ensure on-time delivery to any network entry point, servers must form a dominating set in that graph. The question of resource allocation reduces to that of dominating set allocation, where different non-overlapping dominating sets are allocated to different content classes such that schedulability is maximized. In this problem formulation, *schedulability* is defined as the fraction of client requests (for all classes) that can be served within their delay bounds.

To address the above problem, we compose a centralized optimal algorithm (subject to certain design assumptions) and explore different decentralized dominating allocation policies using Internet delay measurements from Planet-Lab [19] and synthetic HTTP traffic workload to evaluate their performance. Our evaluation results demonstrate that the simple heuristic allocation policies we propose perform very close to the optimal allocation.

The rest of the paper is organized as follows. Section 2 presents the problem formulation and background on wide-area content distribution. Section 3 presents different dominating set allocation policies. Section 4 evaluates their performance. Section 5 presents a brief discussion on the allo-

*This work was funded in part by NSF grants CNS 06-15301 and CNS 05-53420.

cation policies. The paper concludes with Section 6.

2. Problem Formulation

Consider a wide-area network with N edge nodes, N_i , $i = 1, \dots, N$, that represent access routers or servers connected to this network. The network path from some node N_i to another node N_j and back incurs an end-to-end network round-trip delay that is bounded (with a specified high probability) by d_{ij} . In this work, we focus on small content objects (such as typical web objects), as opposed to large files or streaming data. Note that, for small data transfers (large data objects and streaming data are out of the scope of this paper) this delay is fairly independent of the requested data size as latency becomes dominated by queueing delays in the network as opposed to the transfer time of any one request/reply. This assumption is consistent with the fluid data model, which is representative of high-performance resources (servers, backbone links, etc.). Delays on such resources are the cumulative result of waiting for a very large number of service items each of which is served by the resource very quickly. In previous publications [9, 10], the authors have shown using a large set of empirical measurements that Internet round-trip delays tend to be relatively stable over long periods of time (barring rare short-term events such as flash crowds).

Tasks are executed by sending requests to remote servers. For example, in Web browsing, a request is sent to a server which performs the task of generating a reply. The reply is then sent back. A similar interaction occurs in multiplayer interactive Internet games (hosted by centralized servers) or in data-center transactions (such as Google search requests). We assume a task execution model where K classes of tasks exist in the system. A request for a task of class k , $k = 1, \dots, K$ has an end-to-end latency constraint, L_k . Such a request, introduced into the network at time t , must generate a response that exits the network by time $t + L_k$.

As servers are loaded with requests (of the same class), the server's queueing delay grows with the request rate. Often a sharp knee exists in the delay curve at the threshold of server overload, after which delays grow much more steeply [25]. We call that threshold server capacity, and denote the corresponding delay by D_i , that depends on the server, i . Our resource allocation scheme ensures that server capacity is not exceeded and hence bounds the server delay at D_i (with high probability). Whenever site i sends a request of class k to server j , it is desired to ensure that $d_{ij} + D_j \leq L_k$. The allocation problem addressed in this paper is to assign non-overlapping sets of servers to classes such that the fraction of total requests that are served within the end-to-end latency bounds is maximized.

As a specific application of the above model, in the following we focus on the case of a content distribution service. The service (depicted in Figure 1) deploys CDN servers spanning multiple ISPs. Edge nodes are the "entry points" of client requests: requests are sent to the closest

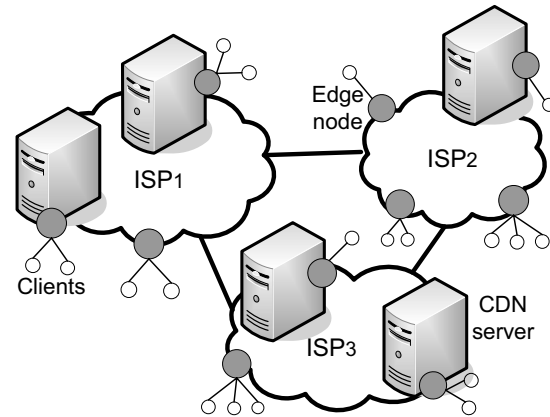


Figure 1: System model of a wide-area content distribution service providing bounded latency for content access. Edge nodes forward client requests to CDN servers that have the requested content and are within the associated access latency bound. CDN servers co-locate with some edge nodes.

edge node, which forwards the requests to one of the CDN servers that have the requested content objects. In this application, to enforce the latency bounds on retrieving content objects, the content distribution service needs to deploy replicas of the content objects to the servers. The replicas should be placed in a strategic way such that when an edge node issues a request, one can always retrieve a replica of the requested object from a server that is within the latency bound. To achieve that, we first construct a graph for each class, k , where vertices represent edge nodes, and arcs connect vertex i to vertex j if $d_{ij} + D_j < L_k$. To ensure that content retrieval latency is always bounded, it is enough to replicate the content at a subset of vertices (associated with CDN servers) that constitute a dominating set for this graph. By definition of the dominating set, any edge node is connected to one of the content replica servers and thus can retrieve the content within the bound.

Since servers can not be shared by classes, it is important to keep the dominating set for each class small so that other classes may find available servers to serve their workload. Moreover, to provide latency bounds on content retrieval, one must make sure that server workload does not exceed capacity thresholds. Hence, the algorithm must find a small dominating set of the graph associated with each latency bound, subject to the limits on server capacity. The aforementioned dominating set allocation problem becomes the fundamental problem in ensuring latency constraints in wide-area distributed systems. Any distributed system with multiple partitioned classes of tasks and a per-class end-to-end latency bound must solve this problem in a way that maximizes deadlines met.

With the basic system model, we now present the formal optimization model of the multi-class server allocation problem as a mixed integer programming (MIP) problem. Consider a multi-class content distribution network with S CDN servers (or servers for short) and C edge nodes that are not associated with servers (see Figure 1). Each edge

node may issue requests for content objects of all K content classes. The content distribution service need to replicate content to serve these requests within delay bounds associated with the content classes. We achieve this by constructing an overlay network for each of these K content classes based on its delay bound. In one such overlay network for content class k , an edge node i is connected to a server node j if content class k requests from i to j can be serviced within the delay bound L_k .

We want to find an optimal allocation which assigns the servers to the content classes such that content objects are replicated on the servers nodes, such that edge nodes get their requests serviced by these selected content servers. As discussed in Section 1, each server is dedicated to one content class. The optimality is defined as maximizing the total amount of requests (of all classes) that can be served within their delay bounds.

We denote individual content class as class k , $k \in \{1, \dots, K\}$, individual edge nodes in the network as i , $i \in \{1, \dots, C\}$, and individual server node in the network as j , $j \in \{1, \dots, S\}$. Let y_j^k ($j \in \{1, \dots, S\}$, $k \in \{1, \dots, K\}$) be a 1 or 0 to denote in an allocation scheme, whether or not server j is dedicated for serving content class k , respectively.

For each edge node i , let r_i^k denote the request rate from node i for content class k , and $x_{i,j}^k$ denote for content class k , the fraction of r_i^k that is requested from server j .

For each server node j , let H_j be the capacity threshold of server j . For each edge node i , let $N^k(i)$ denote the neighborhood node set of node i in the overlay network constructed for content class k . Here, a node can be either an edge node or a server.

The decision variables are:

- y_j^k : which server is dedicated for servicing which content class;
- $x_{i,j}^k$: for each edge node i , each content class k , what is the fraction of its requests for the class (r_i^k) should be served by server j .

Using these notations, for each edge node i , each content class k , its requests that meet their deadlines and are served by server node j can be expressed as $x_{i,j}^k \cdot y_j^k \cdot r_i^k$. Since only those servers within the neighborhood set $N^k(i)$ can service those requests, our goal of maximizing the total serviced requests that meet their deadlines can be expressed as

$$\max \sum_{k=1}^K \sum_{i=1}^S \sum_{j \in N^k(i)} x_{i,j}^k y_j^k r_i^k. \quad (1)$$

We now formulate the constraints for the optimization problem. First, for each edge node i , for each content class k , its requests served by all the servers should be less than or equal to its total request rate for class k :

$$\sum_{j \in N^k(i)} x_{i,j}^k y_j^k \leq 1$$

$$\forall i \in \{1, \dots, S\}, \forall k \in \{1, \dots, K\}. \quad (2)$$

Second, for each server node j , its total capacity H_j should not be exceeded by serving requests of all K content classes:

$$\sum_{k=1}^K \sum_{i, s.t. j \in N^k(i)} x_{i,j}^k y_j^k \leq H_j, \quad \forall j \in \{1, \dots, S\}. \quad (3)$$

Third, each server is dedicated for only one class; y_j^k is an integer which can be only 0 or 1:

$$y_j^k y_j^m = 0, \quad \forall j \in \{1, \dots, S\}, \forall k \neq m, 1 \leq k, m \leq K. \quad (4)$$

$$y_j^k \in \{0, 1\}, \quad \forall j \in \{1, \dots, S\}, \forall k \in \{1, \dots, K\}. \quad (5)$$

Fourth, for any edge node i and content class k , only nodes in the neighborhood of the class k overlay network can be selected as a content server to serve requests:

$$x_{i,j}^k = 0, \text{ if } j \notin N^k(i), \quad \forall i \in \{1, \dots, C\}, \forall k \in \{1, \dots, K\}. \quad (6)$$

Finally, we know $x_{i,j}^k$ is a real number between 0 and 1:

$$0 \leq x_{i,j}^k \leq 1, \quad \forall i \in \{1, \dots, C\}, \forall j \in \{1, \dots, S\}, \forall k \in \{1, \dots, K\}. \quad (7)$$

Since the decision variables y_j^k are required to be integers in $\{0, 1\}$, the above problem is an MIP problem. For general network topology, it is NP-hard.

The paper presents an optimal algorithm to solve this problem and a comparative study of several dominating set allocation policies, building insights into their performance.

3. Dominating Set Allocation

In this section, we investigate the problem of dominating set allocation among multiple content classes. We begin (in Section 3.1) by describing a branch-and-bound algorithm to search for an optimal server allocation in the sense of maximizing the number of deadlines met. This algorithm is centralized and quickly becomes computationally intractable when the problem scale grows. We therefore further propose a set of heuristic dominating set allocation policies. Their performance is compared to the optimal algorithm.

The basic algorithm that serves as the building block for the heuristic allocation policies is described in Section 3.2. It allocates one server to one class. The different heuristics differ only in the order in which they apply the basic algorithm to different classes. Observe that since different classes have different deadlines, and since servers allocated to one class cannot be reused by another, the allocation or-

der makes a difference in that classes considered later in the order must choose from a progressively smaller number of remaining servers. This is not unlike the effect of scheduling policies where tasks of lower priority can only be allocated resources not already used by higher-priority tasks.

3.1. Optimal Allocation Search

As mentioned above, the problem of allocating S servers to K classes to maximize the requests served within their delay bounds can be formulated as a mixed integer programming (MIP) problem. We use a branch-and-bound algorithm to search for the optimal allocation. Each node \mathcal{T} at depth k in the search-tree is a partial solution $ps(\mathcal{T})$ to the allocation problem, representing the allocation of the first k servers to classes. The solution is quantified by an optimistic upper bound $ub(\mathcal{T})$ on the percentage of deadlines met in all solutions descending from the node. The leaves of the search-tree are complete allocations of the servers to classes. The actual percentage of deadlines met can be computed for each leaf node. When such a node is reached, all nodes with a lower $ub(\mathcal{T})$ are pruned. The branching and bounding algorithms are briefly described below.

Branching strategy The root of the search-tree \mathcal{R} is the node with $ps(\mathcal{R}) = \emptyset$ and $ub(\mathcal{R}) = 0$. At each iteration of the search, the node with the highest bound $ub(\mathcal{T}_m)$ (after pruning) is expanded. Namely, K new search-tree nodes are created, each representing the allocation of the next server on the list to one of the K classes. This process stops when \mathcal{T}_m is actually a leaf; the corresponding solution is the optimal allocation.

Deciding upper bound To decide the upper bound of a search-tree node \mathcal{T} , we release the constraint that servers are dedicated to only one class (for the servers that have not been allocated in the partial solution $ps(\mathcal{T})$). The MIP problem mentioned above then reduces to a linear programming problem, whose solution is tractable and gives an upper bound on met deadlines for the subtree rooted at \mathcal{T} .

Note that the efficiency of algorithm depends on the topology of the network, workload distribution, and capacity of the servers. In the worst case, for K classes and S servers, we need to examine all the K^S leaves in the search-tree. Hence, searching for the optimal allocation is feasible only for relatively small network. This algorithm is used to assess the performance of heuristics below.

3.2. The Basic Dominating Set Algorithm

Our heuristics rely on one basic building block which is the algorithm described in this section. Our basic replica selection algorithm runs in a decentralized fashion, incrementally adding servers to form a dominating set for a class of tasks. Edge nodes periodically access each other to estimate round-trip delays. If delays fall below the bound for a particular class, a link is established in a virtual overlay graph for that class. Each edge node maintains a local view of that graph, where it knows only its neighbors on the

overlay and its degree. Besides, each edge node maintains a “forbidden server list” of its own, which is initialized to be empty (those are the previously allocated servers). For a given content class, the incremental algorithm runs as follows:

1. All servers broadcast their degree information to their neighbors on the overlay. The degree of a server is defined as the number of neighbors (including itself).
2. After the edge nodes collect the degree information of their neighboring servers, each individual edge node independently nominates a server to be a replica.
 - a. If the edge node has already received a COVER message (described later) from some server, then stop.
 - b. Otherwise, it nominates the server that has the highest degree among all the neighboring servers (including itself) that are not on the forbidden list.
3. Among all the servers that have been nominated, the server with the highest degree becomes the new replica.
 - a. The new replica orders all the edge nodes it received nominations from by $\frac{\mathcal{L}_i^c}{|N(i,c)|}$ in a descending order, where \mathcal{L}_i^c is the workload of class c from edge node i , and $N(i,c)$ is the set of neighboring servers of edge node i for class c .
 - b. Then the replica traverses this sorted list, sending COVER messages to the edge nodes. When sending a COVER message, it updates its workload by adding the workload of the edge node. This process continues until the workload of the replica reaches its capacity threshold. For the rest of the edge nodes in the list, it sends DUMMY_COVER messages.
4. When an edge node receives a COVER or DUMMY_COVER message, it adds the sender of the message to its forbidden server list.

Every invocation of this basic algorithm selects at most one replica for a given content class. The selected replica is the server that has highest degree among all the servers that have not been selected before. Note that it is possible that an invocation of the algorithm does not generate a new replica because if all edge nodes have no valid candidate to nominate, or they have all received COVER messages in previous invocations, then no new replica will be selected.

Note that in Step 2b, the algorithm selects the server that has the highest degree among all the available servers to be a new replica. The rationale is that using nodes with high degrees to dominate the graph helps minimize the size of the dominating set. In Step 3a, the selected replica attempts to cover those edge nodes that have higher workload and fewer neighboring servers, for the reason that those edge nodes are more likely to fail to meet the latency bound requirement. Observe that this is only one way of incrementally building dominating sets in a distributed system. In general, other algorithms can be borrowed from graph theory for that purpose. The contribution of this paper lies

in investigating the *order* in which dominating set construction should be interleaved to maximize schedulability for multiple classes, as opposed to contributing to basic graph theory another algorithm for building dominating sets. It should be noted, though, that good heuristics (such as [13, 8]) building minimal dominating sets typically share in common the fact that servers with a higher degree are chosen for the dominating set first. This property is shared by the above algorithm as well (see Step 2b) and will have important implications on the order in which classes should be considered for allocation. The property means that servers allocated earlier tend to have better (more centralized) locations with a higher degree of connectivity.

3.3. Multi-Class Algorithms

Using the basic replica selection algorithm described above, we now investigate the problem of multi-class dominating set allocation. As mentioned earlier, the allocation process is sensitive to the *order* of running the algorithm on the classes because no server can be shared by multiple content classes. Obviously, there are servers that have favorable network locations and could cover more edge nodes than others. On the other hand, different content classes have different latency bound requirements. A server that can cover only a small number of edge nodes for a class with a short latency bound may be able to cover many more nodes for another class with a longer bound. This makes the assignment algorithm challenging. The main knob we manipulate is the order of applying the basic replica selection algorithm (Section 3.2) to the content classes. We have a few natural options stated as follows:

Tightest First This algorithm finds replicas for the class with the most stringent latency bound first. Only when all the edge nodes have their workload of that class covered (or no more replicas can be generated), will the algorithm run on the class with the next less stringent latency bound. This is the spatial equivalent of deadline monotonic scheduling where tasks of one class are assigned processor time before assigning any processor time to tasks of the next less urgent class. The algorithm utilizes the fact that the basic replica selection algorithm described above picks the most centrally located servers first. Hence, it makes sense to assign them to the highest priority class.

Loosest First Conversely, this algorithm starts with the class that has the loosest latency bound, and runs until all the edge nodes are covered for this class. Then, the algorithm proceeds to the class with a tighter latency bound, and so forth. This is the reverse deadline monotonic.

Round robin Both tightest first and loosest first algorithms greedily select replicas to satisfy one class at the cost of sacrificing the other classes. When one class consumes too many “good” servers, the other classes can suffer. Hence, balancing the resource allocation among classes may be helpful. A round robin algorithm is a natural candidate. The round robin algorithm avoids the above two

extremes by selecting one replica for each class in a round robin fashion, essentially spreading servers with good connectivity more evenly among different classes.

Weighted Round Robin This basic round robin algorithm provides roughly equal chances of acquiring “good” replicas to all the classes. This might not always be ideal due to the fact that classes with shorter latency bounds may be harder to satisfy (e.g., they probably need more replicas). Hence, we propose a variant of the basic round robin algorithm, called weighted round robin, which is the basic round robin algorithm enhanced with a vector parameter $W = \{w_0, w_1, \dots, w_{K-1}\}$, where w_i is the “weight” of class i . At each round, weighted round robin selects w_0 replicas for class 0 before it moves to class 1, for which it selects w_1 replicas, and so forth.

Bidding The server selection order in the algorithms above is based on latency only. However, there is another dimension of concern in deciding this order; namely, workload. Since the ultimate goal is to maximize the deadline hit ratio, we want to give preference to those servers that can serve more workload in time. Based on this principle, we propose a bidding algorithm that takes both the latency bound constraints and workload of the edge nodes into consideration. At each round, the bidding algorithm generates a replica candidate for each content class using the algorithm described in Section 3.2. The candidates then “bid” for becoming a replica by announcing their *contributions*. The contribution of a server s for a given class c is the amount of traffic it is expected to serve in time if allocated to that class. It is defined as $\sum_{i \in N^*(s,c)} \mathcal{L}_i^c$, where $N^*(s,c)$ is the set of neighboring edge nodes of server s for class c that do not have replicas for class c yet, and \mathcal{L}_i^c is the workload of class c from edge node i . The candidate that has the largest contribution will be selected as a new replica. The goal is to maximize schedulable traffic. If multiple candidates have the same contribution, the candidate for the shortest latency bound class will be chosen. Note that as more replicas are selected for a class, the contribution of an additional server for that class gradually decreases as more of the traffic for that class gets covered. This observation actually reveals a problem with the tightest first algorithm. As more and more replicas are selected for the most stringent bound class, the contributions of the newly selected replicas become smaller resulting in diminishing returns. Newly allocated servers will not be loaded to capacity resulting in wasted computing power and reduced schedulability. Leaving those servers to other classes that can better utilize their capacities may be more beneficial for reducing the overall latency bound miss ratio. This is exactly what the bidding algorithm does.

Scaled Bidding There is a potential problem with the contribution-based bidding algorithm above. In this algorithm, at every step, the class that claims the largest contribution gets the next best server regardless of its latency bound. This, in effect, is a greedy decision. It does not

take into account the effects of the current selection on the ability of remaining servers to satisfy latency bound requirements of other classes. Classes with shorter latency bounds generally need higher-degree servers. Hence, since high-degree servers have different utility to different classes, the comparison of contributions should be “biased”. Similar to the weighted round robin algorithm, we can enhance the bidding algorithm with a preference vector $P = \{p_0, p_1, \dots, p_{K-1}\}$, p_i being the preference of class i . In the bidding process, the algorithm compares the contributions of the candidates scaled up by their preference values p_i . To bias allocation to higher priority classes, we want to make $p_0 \geq p_1 \geq \dots \geq p_{K-1}$. We call this enhanced algorithm, scaled bidding. In fact, the basic bidding algorithm introduced above is a special case of the scaled bidding algorithm where $p_0 = p_1 = \dots = p_{K-1}$; the tightest algorithm is $p_0 \gg p_1 \gg \dots \gg p_{K-1}$; the loosest algorithm is $p_0 \ll p_1 \ll \dots \ll p_{K-1}$.

Random In addition to the algorithms above, we also include a random algorithm as a trivial baseline. This algorithm randomly selects a certain number of replicas for each class, based on the workload breakdown of the classes. If the total workload of the system is distributed among the classes by a ratio of $r_0 : r_1 : \dots : r_{K-1}$, $\sum_{i=0}^{K-1} r_i = 1$, then every server simply claims to be a replica of class c with a probability of r_c . Note that every server will be a replica of some class, but never multiple classes. The numbers of replicas of the classes are therefore probabilistically proportional to their workload ratio.

These algorithms are evaluated in the next section.

4. Evaluation

In this section, we present an extensive performance evaluation on the dominating set allocation policies presented in Section 3. We study their performance, as well as compare with the optimal allocation, using a wide spectrum of system configurations.

We built a prototype of our bounded-latency content distribution service, and implemented the algorithms. We tested and deployed the algorithms on PlanetLab [19], a real-world WAN platform. PlanetLab is a shared platform with a non-real-time operating system. Hence, we do not have control on resource allocation and timing, which makes it hard to run repeatable real-time experiments. Besides, for obvious reasons, we were explicitly discouraged from running overload experiments on PlanetLab. Therefore, we conducted our experiments using a hybrid approach. We first deployed network latency measurement daemons on PlanetLab servers. The measured fluctuating network delay time sequences were then fed to our simulator. Besides the topologies generated by PlanetLab, we also used randomly generated transit-stub networks using GT-ITM [26]. All the experimental data reported in this paper are average values of running the algorithms on all the topologies.

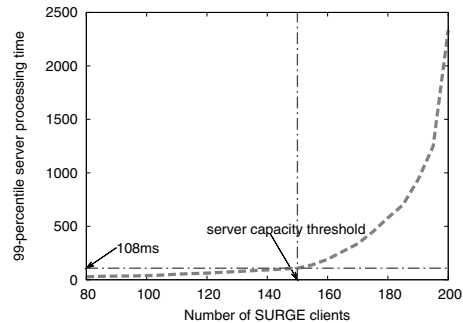


Figure 2: Deciding server capacity threshold using 99-percentile server processing delay.

As discussed in Section 2, the response-load curve of a server has a well-defined knee. We define the server load at that knee as the capacity threshold, or capacity for short, of the server. To find out the knee, we used an array of client machines running SURGE [4], a synthetic Web traffic generator, to generate Web requests to an Apache Web Server running in the same LAN. In Figure 2, we plotted the 99th-percentile of request processing time of a server versus offered workload. As depicted in the figure, there exists a point beyond which the server response time increases nonlinearly and significantly more steeply with workload. We chose that point as the server capacity threshold. It is worth pointing out that our definition of server capacity is rather conservative. We chose this definition to ensure that random workload fluctuations do not cause significant deadline misses. Observe that in the region below the capacity threshold, the response time of the server is quite insensitive to load and hence is not affected significantly by load fluctuations. This makes the timing properties of the system stable even in the face of less predictable server load.

To control the offered workload offered, we used a parameter called *workload factor*, defined as the total workload of all the content classes experienced by all the edge nodes over the total capacity of all servers. The other parameter we introduced is the *server ratio*, defined as the total number of servers over the total number of edge nodes (remember that network edge nodes include both servers and access nodes, shown in Figure 1). In our experiments, for each topology, we randomly selected a fraction of server edge nodes (determined by the server ratio parameter) as servers. Different edge nodes would originate a different amount of input traffic, chosen uniformly from a range that spans one order of magnitude. For all experiments, we had three content classes, with latency bounds 300ms, 600ms, and 1000ms respectively. These numbers were intentionally chosen to be well separated to cover a wide range of latencies.

4.1. Comparison with Optimal Allocation

We start with comparing the performance of the heuristics presented in Section 3.3 with the optimal results obtained by the branch-and-bound searching algorithm described in Section 3.1 in terms of meeting deadlines. As the search for

optimal allocation is very computationally expensive, it is feasible only for small-scale networks. We used a configuration of 40 nodes with a server factor of 0.3 (i.e., 12 CDN servers), and the 3 content classes described above. For the scaled bidding algorithm, we used a preference parameter of $\{4, 2, 1\}$.

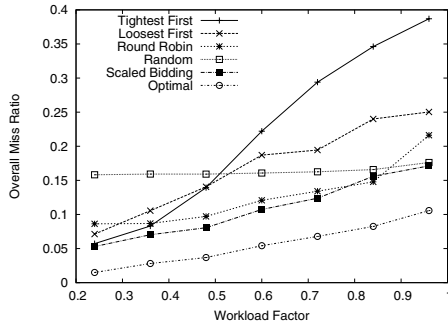


Figure 3: Comparison with optimal allocation.

We study the overall *latency bound miss ratio*, which is the most important performance metric of the allocation policies, of all the allocation policies. Overall latency bound miss ratio is defined as the total number of misses divided by the total number of requests across all classes. The results of this study are summarized in Figure 3. As shown in the figure, an obvious general trend is that the miss ratios increase with the workload for all the algorithms. Note that when the workload factor is very close to zero, the miss ratios are not zero. This is because there is a possibility that some edge nodes do not have sufficient neighboring servers regardless of the overall system workload, especially when the server ratio is very small.

The main observation we make here is that the scaled bidding algorithm performs best. Also, the performance difference between it and the optimal is largely insensitive to the workload factor. When the system is heavily loaded (a workload factor of 0.96), the scaled bidding policy has a miss ratio that only 7% higher than that of the optimal (which then has a miss ratio of 11%).

In the rest of the evaluation study, we focus on the relative performance of the heuristic allocation policies at larger network sizes to determine if the same trends persist.

4.2. Overall Latency Bound Miss Ratio

We study the overall miss ratio of all the algorithms when the total workload of the three classes is the same. Figure 4 gives the overall miss ratio (with 95% confidence interval) for different workload factors, with server ratio ranges from 0.2 to 0.8. For the weighted round robin algorithm, we set the weight vector of the three classes to be $\{2, 1, 1\}$.

A general trend can be observed is that increasing server ratio cuts down the miss ratios of all the algorithms. This translates to the fact that a wider deployment of servers can help achieve a higher hit ratio, which is understandable.

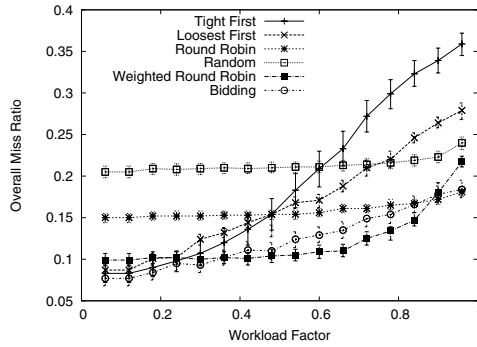
When the server ratio is 0.2 (Figure 4(a)), both the tightest first and loosest first algorithms perform well when the system is underloaded. As the workload increases, their miss ratios, however, go up quickly. In fact, when the system is heavily loaded, the tightest first algorithm is the worst among the collection of algorithms; the loosest first comes next. The reason for the poor performance of the tightest first algorithm under heavy load lies in that when the workload is high and servers are not abundant in the system, the strictly-prioritized attempt to cover the tightest latency class first drains the server supply causing a high miss ratio for other (lower-priority) classes. Comparatively, the weighted round robin and bidding algorithms are better in this case.

When the server ratio is more generous (0.4-0.8), as depicted in Figure 4(b) 4(c) 4(d), the tightest first algorithm manages to keep the miss ratio very low (around 1%-2%) until the workload gets very high (80%-90%, depending on server ratio). Increasing the workload further causes a steep jump in the miss ratio (e.g., see case when server ratio is 0.4). This is similar to what can be observed in the case where server ratio is 0.2, except that the “turning point” is pushed further by having more servers. The loosest first algorithm performs well when the workload factor is small (< 0.4). However, the miss ratio grows with the workload quickly. The random algorithm is relatively insensitive to workload. This is because the allocation of replicas for each class is oblivious to workload information. Even under very light workload, due to improper allocation of replicas, many edge nodes can not find replicas that are within the required latency bounds. Those edge nodes are hence unable to serve their requests in a timely manner.

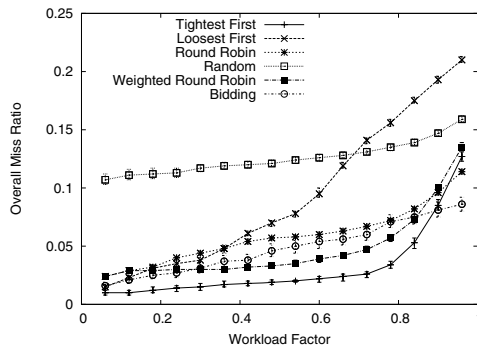
The round robin algorithm exhibits a similar trend as the tightest first algorithm. With a server ratio of 0.4, there is a performance gap between the two algorithms when the workload is mild (0.3-0.8). The difference diminishes when the workload becomes very high. When the workload factor is 0.96, round robin even narrowly beats the tightest first. For the cases of higher server ratios (0.6 and 0.8), the tightest first algorithm wins for all workload conditions.

The performance of the weighted round robin falls short our expectation. When the workload is not very high, it performs slightly better than the basic round robin algorithm. When the workload becomes very heavy, it can not even outperform the basic round robin. Compared to the tightest first algorithm, its performance is consistently inferior. Its problem is indeed similar to that of the tightest first algorithm. Although giving class 0 a larger weight can reduce the miss ratio of that class, the other classes may suffer.

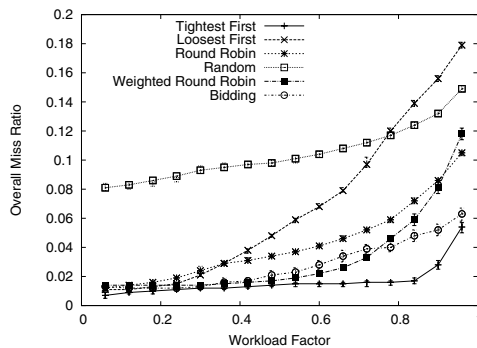
Figure 5 investigates the scaled bidding algorithm with different preference parameters. For comparison purpose, we also included the tightest first algorithm and the basic bidding algorithm in the figure. For the case of server ratio equals 0.2, we also included the weighted round robin algorithm. The point is to compare the best algorithms in Figure 4 with the scaled bidding algorithm. As can



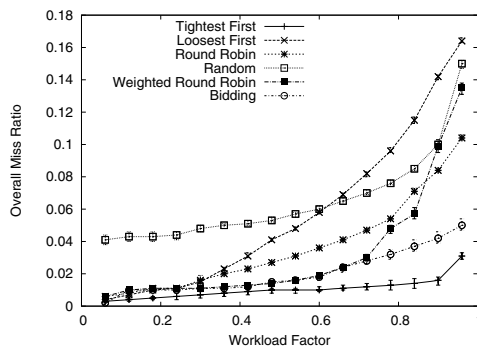
(a) Server ratio = 0.2



(b) Server ratio = 0.4



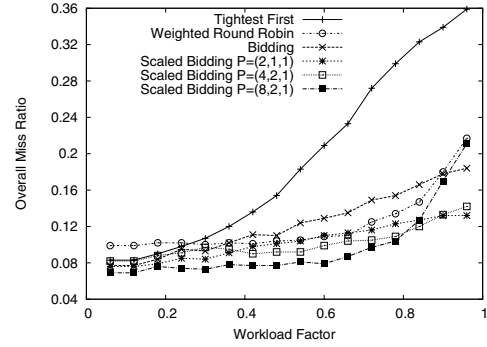
(c) Server ratio = 0.6



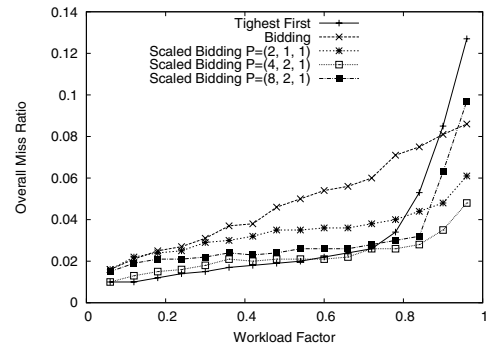
(d) Server ratio = 0.8

Figure 4: Overall latency bound miss ratio of different algorithms

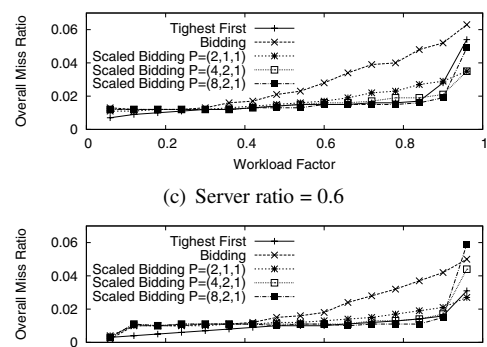
be observed in figure, when the preference parameter is $\{4, 2, 1\}$, the performance of the scaled bidding algorithm is very close to that of the tightest first algorithm when the



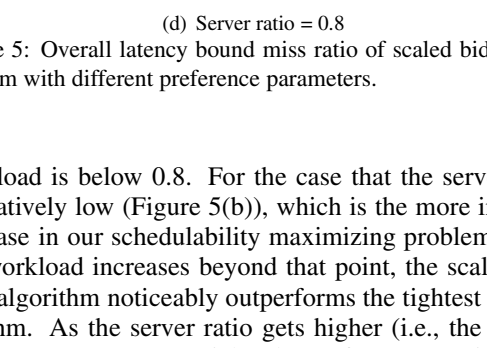
(a) Server ratio = 0.2



(b) Server ratio = 0.4



(c) Server ratio = 0.6



(d) Server ratio = 0.8

Figure 5: Overall latency bound miss ratio of scaled bidding algorithm with different preference parameters.

workload is below 0.8. For the case that the server ratio is relatively low (Figure 5(b)), which is the more interesting case in our schedulability maximizing problem, when the workload increases beyond that point, the scaled bidding algorithm noticeably outperforms the tightest first algorithm. As the server ratio gets higher (i.e., the system becomes more resource rich), the performance gain of the scaled bidding algorithm (with a preference parameter of $\{4, 2, 1\}$) shrinks somewhat, as depicted in Figure 5(c) and 5(d). However, in those cases, the absolute values of miss ratios of both algorithms are very small. A preference parameter of $\{4, 2, 1\}$ works the best in our experiments. In the rest of the paper, we used these values when we refer to

the scaled bidding algorithm.

4.3. Miss Ratio for Non-Uniform Workload

In this section, we make the workload of the three classes not uniform. Intuitively, the impact of varying the workload of the most stringent bound class and the most generous bound class are the most interesting cases to investigate. Figure 6 summarizes the experimental data. In Figure 6(a) plots the overall miss ratios when the workload factor of class 0 varies from 0.1 to 0.5, while those of class 1 and 2 are fixed at 0.2. The weight vector of the weighted round robin algorithm was set to be $\{2, 1, 1\}$. Note that the x-axis shows the total workload factor of the system. In this experiment, the miss ratios of the tightest first, loosest first, random, weighted round robin, and scaled bidding algorithms are all fairly close to those of the experiment reported in Figure 4(b) with the same overall system workload. However, the round robin algorithm exhibits a noticeable performance degradation when the workload is high. Note that the random algorithm does not suffer much from unevenly distributed workload among classes because the number of servers reserved for the classes are proportional to their contribution to the total system workload.

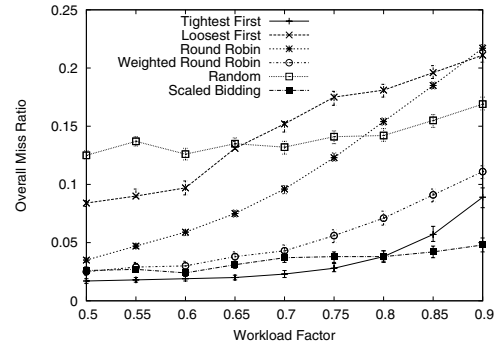
Figure 6(b) reports the data of a related experiment in which the workload of class 0 and 1 are fixed at 0.2 and that of class 2 varies from 0.1 to 0.5. Since in this experiment, the workload of class 2 can be significantly higher than those of class 0 and 1, it makes sense to give class 2 a higher weight in the weighted round robin algorithm. However, the concern that high-degree servers are essential to cover workload of class 0 remains. It is unclear which factor would dominate. Hence, we included two weighted round robin algorithms, with weight vectors of $\{2, 1, 1\}$ and $\{1, 1, 2\}$, respectively.

When the workload is low, giving priority to class 0 by using $\{2, 1, 1\}$ is clearly better than the other way around. However, as the portion of class 2 workload increases, the benefit of allocating more servers for class 2 (by using $\{1, 1, 2\}$) starts to out-weigh the disadvantage of sacrificing class 0 workload. In fact, the overall miss ratio when using $\{1, 1, 2\}$ even decreases as the total workload increases. The reason is that the increment of total workload is solely contributed by class 2. Hence, the impact of a high miss ratio for class 0 diminishes as the total workload increases. Although carefully tuning the weight vector according to global workload information may help, performance of a weighted round robin algorithm will presumably only approximate the better of the two we presented here.

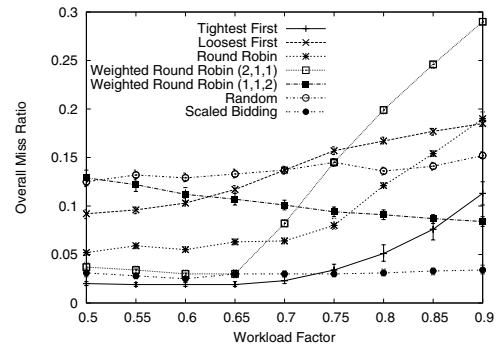
Overall, for all scenarios, the scaled bidding algorithm, with a fixed preference parameter, achieves very low miss ratios in a stable manner.

4.4. Miss Ratios of the Content Classes

To better understand the internals of the algorithms, we further investigated the miss ratio of each of the content classes. In Figure 7, we plotted the miss ratios of the three



(a) Varying workload of class 0



(b) Varying workload of class 2

Figure 6: Overall latency bound miss ratio using non-uniform workload among classes. The server ratio is 0.4.

content classes for different algorithms. For conciseness, we only presented the data when server ratio is 0.4, and the three classes have the same workload. For other cases, we observed similar results.

From the figure, we can see that the tightest first, weighted round robin and scaled bidding algorithm all manage to keep the miss ratio of class 0 relatively low, as they are supposed to. For the other three algorithms, the miss ratios of class 0 are mostly very high. This is because the stringent latency bound of class 0 makes it very hard to find servers for it that are reachable within the bound. Hence, even if the three classes have roughly the same number of servers, class 0 still suffers a higher miss ratio than the other two classes.

The tightest first algorithm strives to satisfy class 0 at the cost of sacrificing classes with longer latency bounds. Hence, when the workload is high, the miss ratios of class 2 are very high. The weighted round robin alleviates the problem to some extent, but also brings a higher miss ratio for class 0. The scaled bidding algorithm achieves a good balance among the classes. Miss ratios of all three classes climb slowly as the workload increases.

4.5. Latency Bound Conformity

Besides average miss ratio, we are also interested in some statistical properties of the miss ratio over different topologies and workload settings. Specifically, we want to know the likelihood that all the latency bound requirements of all

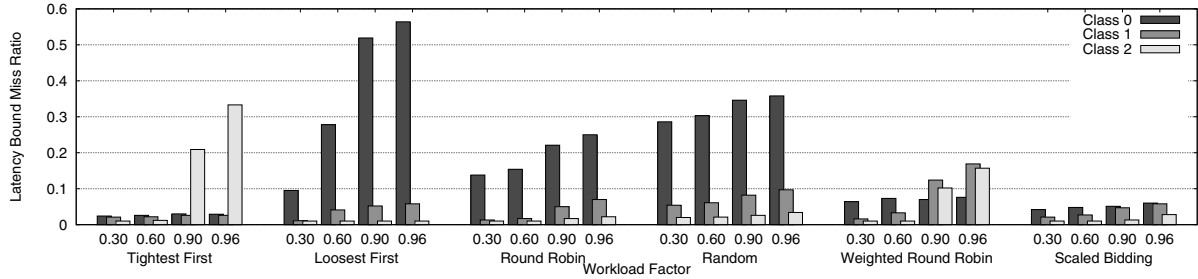


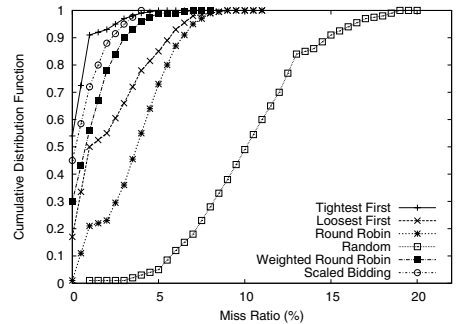
Figure 7: Miss ratio of content classes

the classes are met (i.e., how often can a resource scheduling algorithm achieve a zero or near-zero miss ratio).

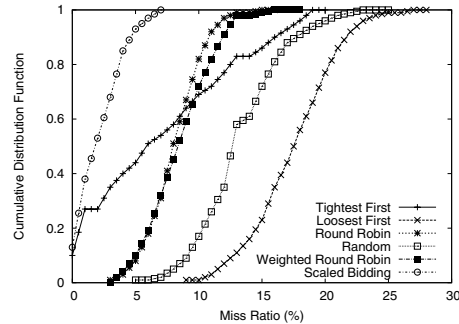
Figure 8 plots the empirical cumulative distribution function (CDF) of overall miss ratio for the algorithms under different workload factors. In this experiment, all classes had the same workload. This workload is input to multiple allocation algorithms. An interesting metric is the percentage of input workload sets for which each algorithm managed to find a zero miss ratio allocation. This metric is given by the first point on the CDF curve. More generally, the CDF shows the percentage of workloads for which each algorithm was able to secure a given miss ratio bound. As can be observed from the figure, when the workload factor is benign (Figure 8(a)), over 50% of the workloads under the tightest first algorithm are scheduled with a zero miss ratio. Moreover, in over 90% of the workloads the miss ratio is bounded by 1%. For the scaled bidding algorithm, 45% of the workloads have zero miss ratio, and over 70% of the workloads have less than 1% miss ratio. In contrast, the percentage of workloads with zero miss ratio for the random algorithm is virtually none. In the case of heavy workload (Figure 8(b)), the scaled bidding and tightest first algorithms still achieve zero miss ratio in 10% of the cases. Over 50% of the cases under the scaled bidding algorithm are below a 2% miss ratio. The other algorithms essentially never manage to perfectly meet the latency bound requirements of all the classes. Note that since the allocation problem is NP hard, we do not compare the above numbers to an optimal algorithm. In other words, we do not know what percentage of workloads in each case is feasible (in the sense that a zero miss ratio is achievable under some allocation).

4.6. Number of Replicas

Given a fixed number of servers, a scheduling algorithm would ideally generate only a small number of replicas for each class when the system is underloaded. However, when the system is heavily loaded, it is more desirable to utilize all the servers as replicas such that the miss ratio can be minimized. Figure 9 investigates this property of the scheduling algorithms. We excluded the random algorithm because it always allocates all the servers based on the workload breakdown of the classes. In this experiment, we made the workload of all classes equal which is more illustrative for our purpose; the server ratio is 0.4.



(a) Workload factor = 0.3



(b) Workload factor = 0.9

Figure 8: Empirical cumulative distribution function of overall latency bound miss ratios. Server ratio is 0.4.

The weight vector of the weighted round robin algorithm is $\{2, 1, 1\}$. Figure 9 presents the breakdown of servers used as replicas in each content class with different workload factors.

To better appreciate the performance of the algorithms in terms generating small dominating sets, we implemented a centralized minimal dominating set (MDS) algorithm [13, 8]. The algorithm is actually the best approximation algorithm known so far for MDS problem, which is NP-Hard. Note that this centralized algorithm has no knowledge of workload and server capacity threshold. Essentially, this is the case in which the servers have infinite capacity, or the system workload is infinitely small. This algorithm yielded an average replica ratio (total number of replicas over total number of servers) of 41%. Compared to the numbers depicted in Figure 9 when the workload factor is 0.1, the number of replicas of the centralized MDS algorithm is only slightly smaller. This serves as evidence of

the efficacy of the algorithms in selecting small dominating sets.

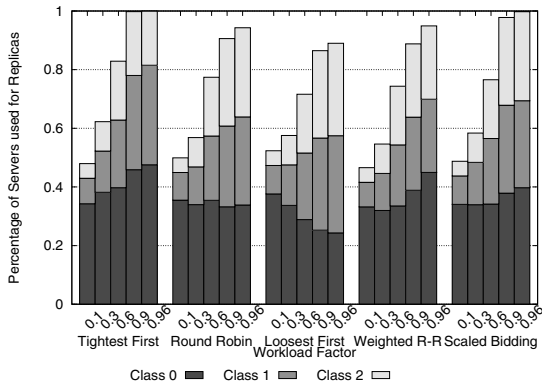


Figure 9: Breakdown of number of replicas among the classes

Clearly, for all the algorithms, the total number of servers used as replicas increases with the system workload, which is intuitively correct. For the tightest first algorithm, the number of replicas used for class 0, the class with the most stringent latency bound, consistently increases with the workload. This, however, is accomplished at the cost of class 2 suffering a shortage of servers under heavy load. When the workload is very high, all servers are used as replicas. Similar behavior can be observed in the scaled bidding algorithm. The only difference is that under heavy load, the servers are distributed more evenly among the classes, which is the reason why the scaled bidding algorithm achieves a lower overall miss ratio.

The round robin algorithm keeps a fair share of servers among the classes when the system workload is high. However, it can not utilize all the servers. The reason lies in the fact that each edge node has only a small number of servers that can serve as replicas of class 0 due to its tight latency bound requirement. Hence, if those servers were allocated to serve other classes, some edge nodes may not be able to find a server to forward their class 0 requests. Even if there are servers that are not allocated to other classes, they can not be used to serve class 0. The weighted round robin behaves similarly, and the share of the servers among the classes is approximately its weight parameter: $\{2, 1, 1\}$.

The issue of round robin is exacerbated when the loosest first algorithm is applied, because those servers that have good connectivities to the edge nodes are consumed by class 2. When the workload is high, class 2 needs to use more servers, leaving class 0 with even fewer choices. In fact, the number of servers serving class 0 even *decreases* when the workload grows. The total number of servers used by class 2 is also smaller than those of the other algorithms.

5. Discussion

Performance of content distribution service has been extensively studied in literature [2, 24, 14, 12, 20, 23, 17, 22]. The problem investigated in this problem differ from most

of the related work in that it focuses on issues on providing *bounded latencies* on content access. The existence of multiple content classes (with different latency bounds) further gives rise to the problem of allocating the servers to the classes.

There has been a considerable amount of research in real-time literature on multiprocessor task allocation, on both homogeneous [18, 16, 3, 7, 6] and heterogeneous platforms [5, 1, 15, 11, 21]. Our model is different in that network latencies between the processing nodes (servers) are a significant fraction of the end-to-end delay of task execution. The network latencies can not be controlled by the system. In that model, the resource allocation algorithm reduces to a problem of allocating dominating sets for different classes.

In uniprocessor scheduling, deadline-based scheduling is optimal in terms of maximizing schedulability. In our system model, through detailed simulations, we identified that the scaled bidding algorithm is the best from a schedulability perspective. The tightest first algorithm, which is the “counterpart” of deadline monotonic scheduling on uniprocessor systems, performs very well only when resources are plentiful (i.e., when the system is underloaded). As resource constraints tighten, its performance degrades substantially compared to the scaled bidding algorithm.

One take-away lesson is that optimality of dominating set allocation policies (from a schedulability perspective), depends on two conflicting factors. The first is the timing constraints (or deadlines) associated with classes of clients. Well-connected servers should be preferably given to classes with tighter constraints. This calls for policies of the tightest first flavor, which allocate better servers to more urgent classes. The second factor is the contribution of servers measured in terms of the demand they can cover. As servers get allocated to a class, the remaining unmet demand of that class drops. Hence, subsequent servers allocated to that class are more likely to be underutilized. It is thus a waste to allocate well-connected servers to this class at that point. This calls for contribution-based policies that allocate the next server to the class that can utilize it the most. The two factors are at odds because the class that can utilize the next server the most is not always the next in priority. A combination of the two policies, therefore, yields the best results. Observe further that the conflict between urgency and contribution arises only when load is high. If load is low, server capacity constraints do not come into play, leaving timing constraints as the prime consideration, hence making tightest first optimal.

Note that, the above trade-off between urgency and contribution has no equivalent on uniprocessors. This is because processor time can be utilized equally well by tasks of any class. This explains why optimality results from uniprocessor scheduling do not carry over to our problem. An interesting open question remains to formally find the best way to combine urgency and contribution considerations in server allocation. The contribution of this paper

has been to verify the basic intuitions on factors that affect schedulability in wide-area distributed systems, and quantify the advantages (in terms of schedulability) of making the right tradeoff between these factors in the allocation policy.

6. Conclusions

In this paper, we explored the dimension of dominating set allocation policies in distributed systems. This can be thought of as an extension of partitioned multiprocessor scheduling policies to distributed systems. Three major differences exist. One is that task invocation on remote resources in the distributed system is dominated by the network delays of sending the request and receiving a response. This is consistent, for example, with Web content retrieval delays. Second, since physical resource isolation between different content classes is commonly desired and the number of machines in the network typically exceeds by far the number of classes, it makes sense to segregate the classes to improve the efficiency of scheduling. Hence, only one class of tasks executes on each server. Finally, allocation refers to server (dominating) sets, as opposed to individual processors. We explored multiple dominating set allocation policies and outlined promising candidates. We also compare these allocation policies with the optimal allocation which requires expensive computation to obtain. These are important to provide insights into the problem for subsequent analysis. Our evaluation results are drawn from simulations using realistic Internet measurements and HTTP workload, which testify the practical applicability of the proposed heuristic in real-world large-scale distributed systems. In future work, the authors intend to take a more analytic approach to the problem to determine provably near-optimal dominating set allocation policies in a partitioned-class wide-area network.

References

- [1] T. F. Abdelzaher and K. G. Shin. Period-based load partitioning and assignment for large real-time applications. *IEEE Transaction on Computers*, 49(1), 2000.
- [2] Akamai. <http://www.akamai.com>.
- [3] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*, 2003.
- [4] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, 1998.
- [5] S. Baruah. Task partitioning upon heterogeneous multiprocessor platforms. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)*, 2004.
- [6] S. Baruah and N. Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*, 2005.
- [7] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of edf on multiprocessor platforms. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, 2005.
- [8] V. Chvátal. A greedy heuristic for the set-covering problem. In *Mathematics of Operations Research*, 1979.
- [9] C. Huang and T. F. Abdelzaher. Towards content distribution service with latency bound guarantees. In *The Twelfth IEEE International Workshop on Quality of Service (IWQoS 2004)*, June 2004.
- [10] C. Huang and T. F. Abdelzaher. Bounded-latency content distribution: Feasibility and evaluation. *IEEE Transaction on Computers*, November 2005.
- [11] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of ACM*, 24(2), 1977.
- [12] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proceedings of IEEE INFOCOM'01*, April 2001.
- [13] D. S. Johnson. Approximation algorithms for combinatorial problems. In *Journal of Computer and System Sciences*, 1974.
- [14] J. Kangasharju, J. Roberts, and K. W. Ross. Object replication strategies in content distribution networks. In *Proceedings of The 6th International Web Caching Workshop and Content Delivery Workshop (WCW'01)*, Boston, MA, June 2001.
- [15] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli. Dynamic mapping in a heterogeneous environment with tasks having priorities and multiple deadlines. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.
- [16] J. Lopez, M. Garcia, J. Diaz, and D. Garcia. Worst-case utilization bound for EDF scheduling on real-time multiprocessor systems. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS 2000)*, 2000.
- [17] T. V. Nguyen, C. T. Chou, and P. Boustead. Provisioning content distribution networks over shared infrastructure. In *The 11th IEEE International Conference on Networks (ICON2003)*, September 2003.
- [18] D.-I. Oh and T. P. Bakker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment. *Real-Time Systems*, 15(2), 1998.
- [19] PlanetLab. <http://www.planet-lab.org>, 2005.
- [20] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *Proceedings of IEEE INFOCOM'01*, April 2001.
- [21] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transaction on Computers*, 38(8):1110–1123, 1989.
- [22] X. Tang and J. Xu. On replica placement for qos-aware content distribution. In *Proceedings of IEEE INFOCOM '04*, March 2004.
- [23] A. Venkataramani, P. Weidmann, and M. Dahlin. Bandwidth constrained placement in a WAN. In *Proceedings of ACM Principles of Distributed Computing (PODC'01)*, August 2001.
- [24] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on CDN robustness. In *Proceedings of The Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, December 2002.
- [25] M. Welsh, D. Culler, and E. Brewer. SEDA: an architecture for well-conditioned, scalable internet services. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 230–243, New York, NY, USA, 2001. ACM Press.
- [26] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM '96*, 1996.