Practical Load Balancing for Content Requests in Peer-to-Peer Networks

Mema Roussopoulos Mary Baker
Department of Computer Science
Stanford University, Stanford, California, 94305
{mema, mgbaker}@cs.stanford.edu

***Abstract*—**

This paper studies the problem of load-balancing the demand for content in a peer-to-peer network across heterogeneous peer nodes that hold replicas of the content. Previous decentralized load balancing techniques in distributed systems base their decisions on periodic updates containing information about load or available capacity observed at the serving entities. We show that these techniques do not work well in the peer-to-peer context; either they do not address peer node heterogeneity, or they suffer from significant load oscillations which result in unutilized capacity. We propose a new decentralized algorithm, Max-Cap, based on the maximum inherent capacities of the replica nodes. We show that unlike previous algorithms, it is not tied to the timeliness or frequency of updates, and consequently requires significantly less update overhead. Yet, Max-Cap can handle the heterogeneity of a peer-to-peer environment without suffering from load oscillations.

[Non-student status; Regular presentation]

I. Introduction

Peer-to-peer networks are becoming a popular architecture for content distribution. The basic premise in such networks is that any one of a set of "replica" nodes can provide the requested content, increasing the availability of interesting content without requiring the presence of any particular serving node.

Many peer-to-peer networks push index entries throughout the overlay peer network in response to lookup queries for specific content [gnu], [RFH$^+$01], [RD01], [SMK$^+$01]. These index entries point to the locations of replica nodes where the particular content can be served, and are typically cached for a finite amount of time, after which they are considered stale. Until now, however, there has been little focus on how an individual peer node should choose among the returned index entries to forward client requests.

One reason for considering this choice is load balancing. Some replica nodes may have more capacity to answer queries for content than others, and the system can serve content in a more timely manner by directing queries to more capable replica nodes.

In this paper we explore the problem of load-balancing the demand for content in a peer-to-peer network. This problem is challenging for several reasons. First, in the peer-to-peer case there is no centralized dispatcher that performs the load-balancing of requests; each peer node individually makes its own decision on how to assign incoming requests to replicas. Second, nodes do not typically know the identities of all other peer nodes in the network, and therefore they cannot coordinate this decision with those other nodes. Finally, replica nodes in peer-to-peer networks are not necessarily homogeneous. Some replica nodes may be very powerful with great connectivity, whereas others may have limited inherent capacity to handle content requests.

Previous load-balancing techniques base their decisions on periodic or continuous updates containing information on *load* or *available capacity*. We refer to this information as load-balancing information (LBI). These techniques have not been designed with peer-to-peer networks in mind and thus

- do not take into account the heterogeneity of peer nodes (e.g., [GC00], [Mit97], or
- use techniques such as migration or handoff of tasks (e.g., [LL96]) that require close coordination amongst serving entities that cannot be achieved in a peer-to-peer environment, or
- suffer from significant load oscillations, or "herd behavior" [Mit97], where peer nodes simultaneously forward an unpredictable number of requests to replicas with low reported load or high reported available capacity causing them to become overloaded. This herd behavior defeats the attempt to provide load-balancing.

Most of these techniques also depend on the timeliness of LBI updates. The wide-area nature of peer-to-peer networks and the variation in transfer delays among peer nodes makes guaranteeing the timeliness of updates difficult. Peer nodes will experience varying degrees of staleness in the LBI updates they receive depending on their distance from the source of updates. Moreover, maintaining the timeliness of LBI updates is also costly, since all updates must travel across the Internet to reach interested peer nodes. The smaller the inter-update period and the larger the overlay peer network, the greater the network traffic overhead incurred by LBI updates. Therefore, in a peer-to-peer environment, an effective load-balancing algorithm should not be critically dependent on the timeliness of updates.

In this paper we propose a new and practical load-balancing algorithm, Max-Cap, that makes decisions based on the inherent maximum capacities of the replica nodes. We define maximum capacity as the maximum number of content requests per time unit that a replica claims it can handle. Alternative measures such as maximum (allowed) connections can be used. The maximum capacity is like a contract by which the replica agrees to abide. If the replica cannot sustain its advertised rate, then it may choose to advertise a new maximum capacity to avoid overload. Max-Cap is not tied to the timeliness or frequency of LBI updates, and as a result, when applied in a peer-to-peer environment, outperforms algorithms based on load or available capacity, whose benefits are heavily dependent on the timeliness of the updates.

We show that Max-Cap takes peer node heterogeneity into account unlike algorithms based on load. While algorithms based on available capacity take heterogeneity into account, we show that surprisingly, they can suffer from significant load oscillations in a peer-to-peer network in the presence of small fluctuations in the workload, even when the workload request rate is well below (as low as 60%) the total maximum capacities of the replicas. On the other hand, Max-Cap avoids overloading replicas in such cases and is more resilient to very large fluctuations in workload. This is because a key advantage of Max-Cap is that it uses information that is not affected by changes in the workload.

In a peer-to-peer environment the expectation is that the set of participating nodes changes constantly. Since replica arrivals to and departures from the peer network can affect the information carried in LBI updates, we also compare Max-Cap against availability-based algorithms when the set of replicas continuously changes. We show that Max-Cap is less affected by changes in the replica set than the availability-based algorithms.

We evaluate load-based and availability-based algorithms and compare them with Max-Cap. We use the Controlled Update Propagation (CUP) protocol [RB03a] to propagate the LBI updates required by these algorithms. LBI updates are propagated from replica nodes serving particular content down a conceptual tree, similar to an application-level multicast tree. The vertices of this tree are peer nodes receiving requests for that content. The peer nodes use the LBI updates when choosing to which replica to forward a client request. Since the specific propagation semantics are not essential to our central focus of comparing the load-balancing algorithms themselves, we defer discussion of the details of the propagation mechanism to Appendix I.

The rest of this paper is organized as follows. Section II introduces the algorithms compared. Section III presents experimental results showing that in a peer-to-peer environment, Max-Cap outperforms the other algorithms and does so with much less or no overhead. Section IV describes related work, and Section V concludes the paper.

II. The Algorithms

We evaluate two different algorithms, Inv-Load and Avail-Cap. Each represents a different class of algorithms that has been proposed in the distributed systems literature. We study how these algorithms perform when applied in a peer-to-peer context and compare them with our proposed algorithm, Max-Cap. These three algorithms depend on different LBI being propagated, but their overall goal is the same: to balance the demand for a specific piece of content fairly across the set of replicas providing that content. In particular, the algorithm should avoid overloading some replicas while underloading others, especially when the aggregate capacity of all replicas is sufficient to handle the content request workload. Moreover, the algorithm should prevent individual replicas from oscillating between being overloaded and underloaded.

Oscillation is undesirable for two reasons. First, many applications limit the number of requests a host can have outstanding. This means that when a replica node is overloaded, it will drop requests it receives. This forces the requesting client (or user) to resend its request with additional network delay which has a negative impact on response time. Even for applications that allow requests to be queued while a replica node is overloaded, the queueing delay incurred will also increase the average response time. Second, in a peer-to-peer network, the issue of fairness is sensitive. The owners of replica nodes are likely not to want their nodes to be overloaded while other nodes in the network are underloaded. An algorithm that can fairly distribute the request workload without causing replicas to oscillate between being overloaded and underloaded is preferable.

We describe each of the algorithms we evaluate in turn:

Inv-Load: Allocation Proportional to Inverse Load. There are many load-balancing algorithms that base the allocation decision on the load observed at and reported by each of the serving entities (see Related Work Section IV). The representative load-based algorithm we examine is Inv-Load, based on the algorithm presented by Genova et al. [GC00]. In a homogeneous environment, this algorithm has been shown to perform as well as or better than other load-based algorithms. In this algorithm, each peer node in the network chooses to forward a request to a replica with probability inversely proportional to the load reported by the replica. This means that the replica with the smallest reported load (as of the last report received) will receive the most requests from the node. Load is defined as the number of request arrivals at the replica per time unit. Other possible load metrics include the number of request connections open at the replica at reporting time [AB00] or the request queue length at the replica [Dah99].

When applied in a heterogeneous environment such as a peer-to-peer network, Inv-Load fails. This is intuitive because Inv-Load does not distinguish between replicas observing the same load but having different maximum capacities. For completeness only, we verify this intuition with an experiment in Appendix II.

We have considered varying Inv-Load to take heterogeneity into account with a weighting scheme based on maximum capacity. We find that the results for this variation are identical to those of Avail-Cap which is the algorithm we consider next. For this reason, we do not consider Inv-Load nor weighted Inv-Load in the remainder of the paper.

Avail-Cap: Allocation Proportional to Available Capacity. In this algorithm, each peer node chooses to forward a request to a replica with probability proportional to the available capacity reported by the replica. Available capacity is the maximum request rate a replica can handle minus the load (actual request rate) experienced at the replica. This algorithm is based on the algorithm proposed by Zhu et al. [ZYZ+98] for load sharing in a cluster of heterogeneous servers. Avail-Cap takes into account heterogeneity because it distinguishes between nodes that experience the same load but have different maximum capacities.

Intuitively, Avail-Cap seems like it should work; it handles heterogeneity by sending more requests to the replicas that are currently more capable. Replicas that are overloaded report an available capacity of zero and are excluded from the allocation decision until they once more report a positive available capacity. Surprisingly, this exclusion causes Avail-Cap to suffer from wild load oscillations (Section III-A).

Both Inv-Load and Avail-Cap implicitly assume that the load or available capacity reported by a replica remains roughly constant until the next report. Since both these metrics are directly affected by changes in the request workload, both algorithms require that replicas periodically update their LBI. (We assume replicas are not synchronized in when they send reports.) Decreasing the period between two consecutive LBI updates increases the freshness of the LBI at a cost of higher overhead, measured in number of updates pushed through the peer-to-peer network. This overhead is exacerbated with increasing network size. In large peer-to-peer networks, there may be several hops over which updates will have to travel, and the time to do so could be on the order of seconds.

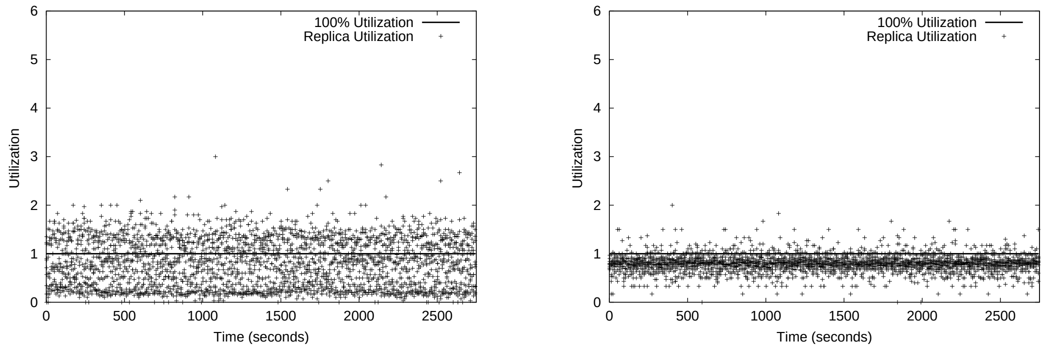
Max-Cap: Allocation Proportional to Maximum Capacity. This is the algorithm we propose. In this algorithm, each peer node chooses to forward a request to a replica with probability proportional to the maximum capacity of the replica. The maximum capacity is a contract each replica advertises indicating the number of requests the replica claims to handle per time unit. Unlike load and available capacity, the maximum capacity of a replica is not affected by changes in the request workload. Therefore, Max-Cap does not depend on the timeliness of LBI updates. In fact, replicas only push updates when they choose to advertise a new maximum capacity. This choice depends on extraneous factors that are unrelated to and independent of the workload (see Section III-F). If replicas rarely choose to change contracts, Max-Cap incurs near-zero overhead. We show that this independence of the timeliness and frequency of LBI updates makes Max-Cap practical and elegant for use in peer-to-peer networks.

III. Experiments

In this section we describe experiments that measure the ability of the Avail-Cap and Max-Cap algorithms to balance requests for content fairly across the replicas holding the content. We simulate a content-addressable network (CAN) [RFH+01] using the Stanford Narses simulator [MGB01]. In each of these experiments, requests for a single piece of content are posted at nodes throughout the CAN network for 3000 seconds. Using the CUP protocol [RB03a] briefly summarized in Appendix I, a peer node that receives a content request from a local client retrieves a set of index entries pointing to replica nodes in the network that serve the content. The peer node applies a load-balancing algorithm to choose one of the replica nodes. It then points the client at the chosen replica.

The simulation input parameters include: the number of nodes in the overlay peer-to-peer network, the number of replica nodes holding the content of interest, the maximum capacities of the replica nodes, the distribution of content request inter-arrival times, and the LBI update period, which is the amount of time each replica waits before sending the next LBI update for the Avail-Cap algorithm.

We assign maximum capacities to replica nodes by applying results from recent work that measured the upload capabilities of nodes in Gnutella networks [SGG02]. This work has found that for the Gnutella network measured, around 10% of nodes are connected through dial-up modems, 60% are connected through broadband connections such as cable modem or DSL where the upload speed is about ten times that of dial-up modems, and the remaining 30% have high-end connections with upload speed at least 100 times that



(a) (b)

Fig. 1. Replica Utilization versus Time for (a) Avail-Cap, (b) Max-Cap.

of dial-up modems. Therefore we assign maximum capacities of 1, 10, and 100 requests per second to nodes with probability of 0.1, 0.6, and 0.3, respectively.

In all the experiments we present in this paper, the number of nodes in the network is 1024, each of which individually decides how to assign incoming content requests to the replica nodes. We use both Poisson and Pareto request inter-arrival distributions, both of which have been found to hold in peer-to-peer networks [Cao02], [Mar02].

First, we compare Avail-Cap with Max-Cap for Poisson arrivals and show that while Avail-Cap takes replica heterogeneity into account, it can suffer from significant load oscillations caused by even small fluctuations in the workload. Second, we compare Max-Cap with Avail-Cap for bursty Pareto arrivals. We then explain why Avail-Cap suffers. We also compare the effect on the performances of Avail-Cap and Max-Cap when replicas continuously enter and leave the system. Finally, we consider the effect on Max-Cap when replicas cannot always honor their advertised maximum capacities because of significant extraneous load.

A. Poisson Request Arrivals

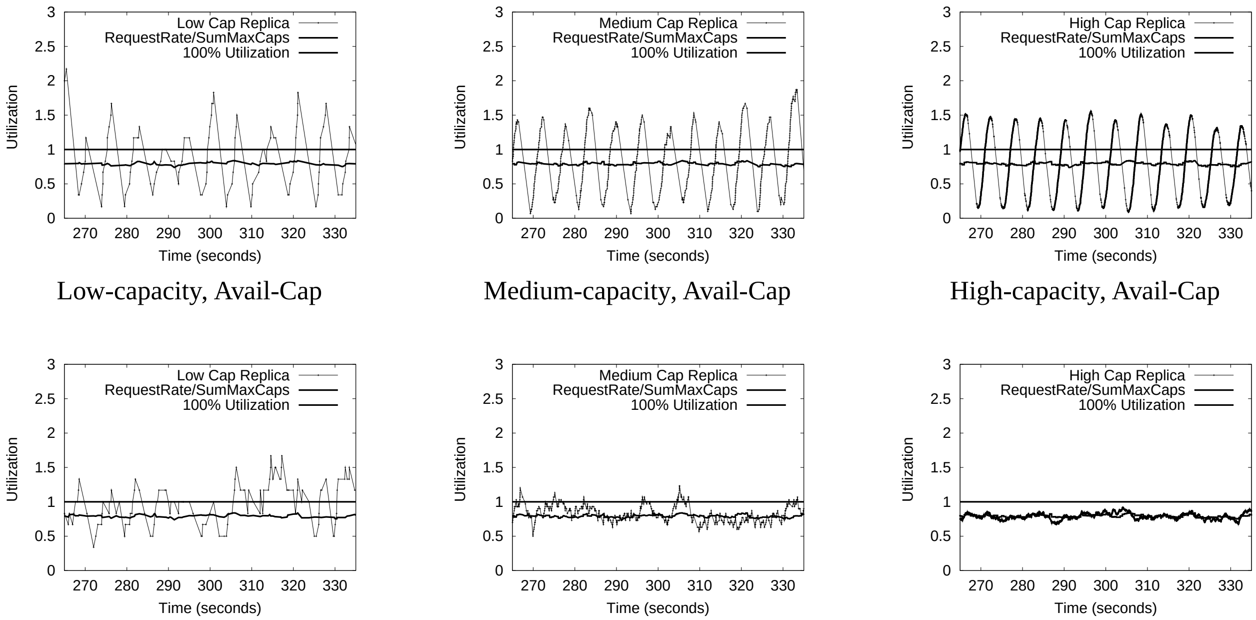
We first compare Avail-Cap with Max-Cap for an experiment with ten replicas with a Poisson request arrival rate of 80% the total maximum capacities of the replicas. Under such a workload, a good load-balancing algorithm should be able to avoid overloading some replicas while underloading others. For Avail-Cap, we use an inter-update period of one second, which is quite aggressive in a large peer-to-peer network. For Max-Cap, this parameter is inapplicable since replica nodes do not send updates unless they experience extraneous load (see Section III-F).

Figure 1 shows a scatterplot of how the utilization of each replica proceeds with time for Avail-Cap and Max-Cap. We define utilization as the request arrival rate observed by a replica divided by the maximum capacity of the replica. In this graph, we do not distinguish among points of different replicas. We see that Avail-Cap consistently overloads some replicas while underloading others. In contrast, Max-Cap tends to cluster replica utilization at around 80%. We ran this experiment with a range of Poisson arrival rates and found similar results for rates that were 60-100% the total maximum capacities of the replicas. Avail-Cap consistently overloads some replicas while underloading others whereas Max-Cap clusters replica utilization at around X% utilization, where X is the average overall request rate divided by the total maximum capacities of the replicas.

It turns out that in Avail-Cap, unlike Inv-Load, it is not the same replicas that are consistently overloaded or underloaded throughout the experiment. Instead, from one instant to the next, individual replicas oscillate between being overloaded and severely underloaded. We can see a sampling of this oscillation by looking at the utilizations of some individual replicas over time. In Figure 2, we plot the utilization over a one minute period in the experiment for a representative replica from each of the replica classes (low, medium, and high maximum capacity). We also plot the ratio of the overall request rate to the total maximum capacities of the replicas and the line $y = 1$ showing 100% utilization. We see that for all replica classes, Avail-Cap suffers from significant oscillation when compared with Max-Cap which causes little or no oscillation above the 100% utilization line. This behavior occurs throughout the experiment.

Figure 3 shows for each replica, the percentage of received requests that arrive while the replica is overloaded for a series of ten experiments, each with ten replicas, for Avail-Cap and Max-Cap respectively. On the x-axis we order replicas according to maximum capacity, with the low-capacity replicas plotted first (replica IDs 1 through 10), followed by the medium-capacity replicas (replica IDs 11-70), followed by the high-capacity replicas (replica IDs 71-100).

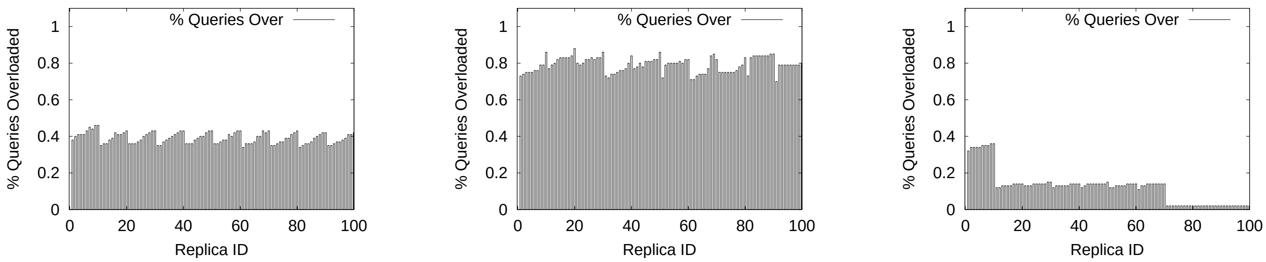
Avail-Cap with an inter-update period of one second (Figure 3a) causes much higher percentages than Max-Cap (Figure 3c) Avail-Cap also causes fairly even percentages at around 40%. This can be explained by looking at the os-



Low-capacity, Avail-Cap — Medium-capacity, Avail-Cap — High-capacity, Avail-Cap

Low-capacity, Max-Cap — Medium-capacity, Max-Cap — High-capacity, Max-Cap

Fig. 2. Replica Utilization versus Time, for representative low, medium, and high capacity replicas. Top graphs show Avail-Cap, bottom show Max-Cap.



(a) Avail-Cap, period = 1 — (b) Avail-Cap, period = 10 — (c) Max-Cap

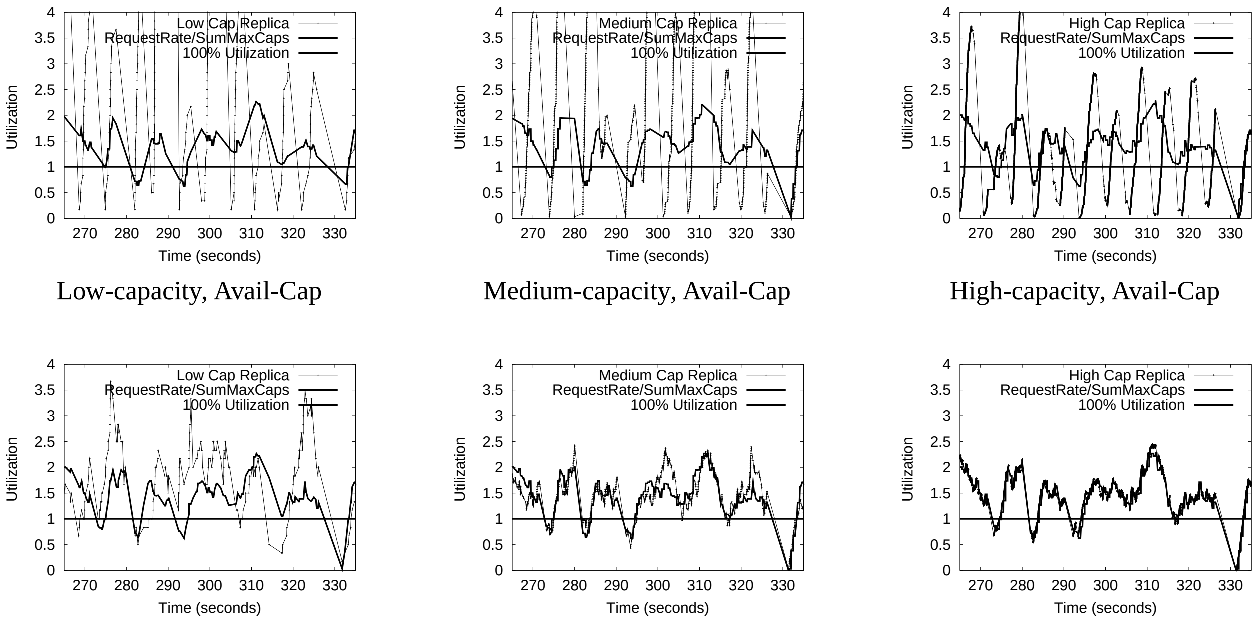
Fig. 3. Percentage Overloaded Requests versus Replica ID, Ten experiments. We show Avail-Cap with an inter-update period of 1 and 10 seconds, and Max-Cap which has no inter-update period.

cillations observed by replicas in Figure 2. In Avail-Cap, each replica is overloaded for roughly the same amount of time regardless of whether it is a low, medium or high-capacity replica. This means that while each replica is getting the correct proportion of requests, it is receiving them at the wrong time and as a result all the replicas experience roughly the same overloaded percentages.

The performance of Avail-Cap is highly dependent on the inter-update period used. As we increase the period and available capacity updates grow more stale, the performance of Avail-Cap suffers more. As an example, in Figure 3b, we show the overloaded request percentages in the same series of ten experiments for Avail-Cap with an inter-update period of ten seconds. The overloaded percentages jump up to about 80% across the replicas.

Max-Cap (Figure 3c) exhibits a step-like behavior with the low-capacity replicas having the highest overloaded percentages, followed by the medium capacity replicas, and then the high-capacity replicas which are never overloaded. This step behavior occurs because the lower-capacity replicas have less tolerance for noise in the random coin tosses the nodes perform while assigning requests. They also have less tolerance for small fluctuations in the request rate. As a result, lower-capacity replicas are overloaded more easily than higher-capacity replicas. We also see this in Figure 2 where for Max-Cap, replicas with lower maximum capacity are overloaded for more time than replicas with higher maximum capacity.

In a peer-to-peer environment, we argue that Max-Cap is a more practical choice than Avail-Cap. First, Max-Cap typically incurs no overhead. Second, Max-Cap can better handle request rates that are below 100% the total maximum capacities of the replicas and can handle small fluctuations in the workload as are typical in Poisson arrivals.



Low-capacity, Avail-Cap | Medium-capacity, Avail-Cap | High-capacity, Avail-Cap

Low-capacity, Max-Cap | Medium-capacity, Max-Cap | High-capacity, Max-Cap

Fig. 4. Representative Replica Utilization versus Time, Pareto arrivals. Top graphs show Avail-Cap, bottom show Max-Cap.

A question remaining is how do Avail-Cap and Max-Cap compare when workload rates fluctuate beyond the total maximum capacities of the replicas? Such a scenario can occur for example when requests are bursty, as when inter-request arrival times follow a Pareto distribution. We examine Pareto arrivals next.

B. *Pareto Request Arrivals*

Recent work has observed that in some peer-to-peer networks, request inter-arrivals exhibit burstiness on several time scales [Mar02], making the Pareto distribution a good candidate for modeling these inter-arrival times. Pareto request arrivals are characterized by frequent and intense bursts of requests followed by idle periods of varying lengths [PF95]. During the bursts, the average request arrival rate can be many times the total maximum capacities of the replicas. We present a representative experiment in which the Pareto shape parameter α and scale parameter κ are 1.1 and 0.000346 respectively. These particular settings cause bursts of up to 230% the total maximum capacities of the replicas. With such intense bursts, no load-balancing algorithm can be expected to keep replicas underloaded. Instead, the best an algorithm can do is to avoid underloading some of the replicas and leaving unutilized capacity.

In Figure 4, we plot the same representative replica utilizations over a one minute period in the experiment for this Pareto experiment. We also plot the ratio of the overall request rate to the total maximum capacities as well as the $y = 100\%$ utilization line. From the figure we see that Avail-Cap suffers from much wilder oscillation than Max-Cap, causing much higher peaks and lower valleys in replica utilization than Max-Cap. Even when the overall request rate is above 100% of the total maximum capacities, there are times when the replicas in Avail-Cap are underloaded. In contrast, Max-Cap generally avoids having unutilized capacity when the overall request rate is above 100%. Max-Cap never underutilizes the medium and high capacity replicas and causes little under-utilization of the low capacity replica.

C. *Why Avail-Cap Can Suffer*

From the experiments above we see that Avail-Cap can suffer from severe oscillation even when the overall request rate is well below (e.g., 80%) the total maximum capacities of the replicas. The reason why Avail-Cap does not balance load well here is that a vicious cycle is created where the available capacity update of one replica affects a subsequent update of another replica. This in turn affects later allocation decisions made by nodes which in turn affect later replica updates. This description becomes more concrete if we consider what happens when a replica is overloaded.

In Avail-Cap, if a replica becomes overloaded, it reports an available capacity of zero. This report eventually reaches all peer nodes, causing them to stop redirecting requests to the replica. The exclusion of the overloaded replica from the allocation decision shifts the entire burden of the workload to the other replicas. This can cause other replicas to overload and report zero available capacity while the excluded replica experiences a sharp decrease in its utilization. This sharp decrease causes the replica to begin reporting positive available capacity which begins to attract requests again. Since in the meantime other replicas have become overloaded and excluded from the allocation decision, the replica receives a flock of requests which cause it to become overloaded again. As we observed, a replica can experience severe and periodic oscillation where its utilization continuously rises above its maximum capacity and falls sharply.

In Max-Cap, if a replica becomes overloaded, the overload condition is confined to that replica. The same is true in the case of underloaded replicas. Since the overload/underload situations of the replicas are not reported, they do not influence follow-up LBI updates of other replicas. It is this key property that allows Max-Cap to avoid herd behavior.

There are situations however where Avail-Cap performs well without suffering from oscillation. We next describe the factors that affect the performance of Avail-Cap to get a clearer picture of when the reactive nature of Avail-Cap is beneficial (or at least not harmful) and when it causes oscillation.

D. Factors Affecting Avail-Cap

There are four factors that affect the performance of Avail-Cap: the inter-update period U, the inter-request period R, the amount of time T it takes for all nodes in the network to receive the latest update from a replica, and the ratio of the overall request rate to the total maximum capacities of the replicas. We examine these factors by considering three cases:

Case 1: U is much smaller than R ($U << R$), and T is sufficiently small so that when a replica pushes an update, all nodes in the CUP tree receive the update before the next request arrival in the network. In this case, Avail-Cap performs well since all nodes have the latest load-balancing information whenever they receive a request.

Case 2: U is long relative to R ($U > R$) and the overall request rate is less than about 60% the total maximum capacities of the replicas. (This 60% threshold is specific to the particular configuration of replicas we use: 10% low, 60% medium, 30% high. Other configurations have different threshold percentages that are typically well below the total maximum capacities of the replicas.) In this case, when a particular replica overloads, the remaining replicas are able to cover the proportion of requests intended for the overloaded replica because there is a lot of extra capacity in the system. As a result, Avail-Cap avoids oscillations. We see experimental evidence for this in Section III-E. However, over-provisioning to have enough extra capacity in the system so that Avail-Cap can avoid oscillation in this particular case seems a high price to pay for load stability.

Case 3: U is long relative to R ($U > R$) and the overall request rate is more than about 60% the total maximum capacities of the replicas. In this case, as we observe in the experiments above, Avail-Cap can suffer from oscillation. This is because every request that arrives directly affects the available capacity of one of the replicas. Since the request rate is greater than the update rate, an update becomes stale shortly after a replica has pushed it out. However, the replica does not inform the nodes of its changing available capacity until the end of its current update period. By that point many requests have arrived and have been assigned using the previous, stale available capacity information.

In Case 3, Avail-Cap can suffer even if $T = 0$ and updates were to arrive at all nodes immediately after being issued. This is because all nodes would simultaneously exclude an overloaded replica from the allocation decision until the next update is issued. As T increases, the staleness of the report only exacerbates the performance of Avail-Cap.

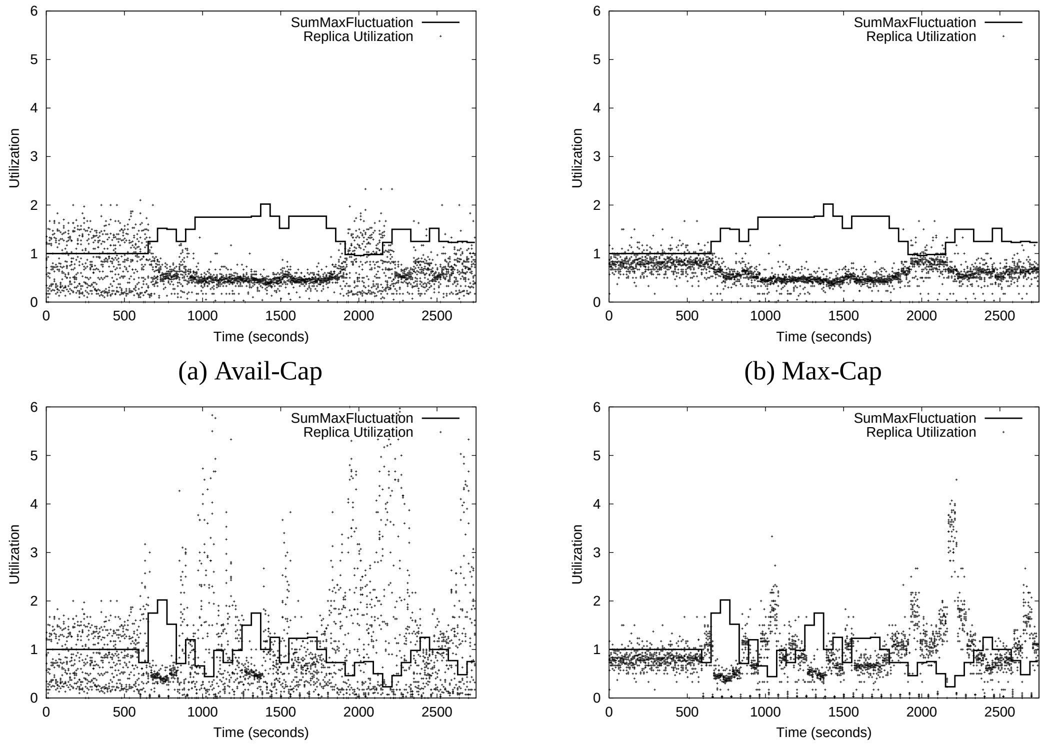
In a large peer-to-peer network (more than 1000 nodes) we expect that T will be on the order of seconds since current peer-to-peer networks with more than 1000 nodes have diameters ranging from a handful to several hops [RF02]. We consider U = 1 second to be as small (and aggressive) an inter-update period as is practical in a peer-to-peer network. In fact even one second may be too aggressive due to the overhead it generates. This means that when particular content experiences high popularity, we expect that typically $U + T >> R$. Under such circumstances Avail-Cap is not a good load-balancing choice. For less popular content, where $U + T < R$, Avail-Cap is a feasible choice, although it is unclear whether load-balancing across the replicas is as urgent here, since the request rate is low.

The performance of Max-Cap is independent of the values of U, R, and T. More importantly, Max-Cap does not require continuous updates; replicas issue updates only if they choose to re-issue new contracts to report changes in their maximum capacities. Therefore, we believe that Max-Cap is a more practical choice in a peer-to-peer context than Avail-Cap.

E. Dynamic Replica Set

A key characteristic of peer-to-peer networks is that they are subject to constant change; peer nodes continuously enter and leave the system. In this experiment we compare Max-Cap with Avail-Cap when replicas enter and leave the system. We present results here for a Poisson request arrival rate that is 80% the total maximum capacities of the replicas.

We present two dynamic experiments. In both experiments, the network starts with ten replicas and after a period of 600 seconds, movement into and out of the network begins. In the first experiment, one replica leaves and one replica enters the network every 60 seconds. In the second



(a) Avail-Cap

(b) Max-Cap

(c) Avail-Cap

(d) Max-Cap

Fig. 5. Replica Utilization versus Time. Top graphs show one switch every 60 seconds, bottom show 5 switches every 60 seconds.

and much more dynamic experiment, five replicas leave and five replicas enter the network every 60 seconds. The replicas that leave are randomly chosen. The replicas that enter the network enter with maximum capacities of 1, 10, and 100 with probability of 0.10, 0.60, and 0.30 respectively as in the initial configuration. This means that the total maximum capacities of the active replicas in the network varies throughout the experiment, depending on the capacities of the entering replicas.

Figures 5a and 5b show for the first dynamic experiment the utilization of active replicas throughout time as observed for Avail-Cap and Max-Cap. Note that points with zero utilization indicate newly entering replicas. The jagged line plots the ratio of the current sum of maximum capacities in the network, S_{curr}, to the original sum of maximum capacities, S_{orig}. With each change in the replica set, the replica utilizations for both Avail-Cap and Max-Cap change. Replica utilizations rise when S_{curr} falls and vice versa.

From the figure we see that between times 1000 and 1820, S_{curr} is between 1.75 and 2 times S_{orig}, and is more than double the overall workload request rate. During this time period, Avail-Cap performs well because the workload is not very demanding and there is plenty of extra capacity in the system (Case 2 above). However, when at time 1940 S_{curr} falls back to S_{orig}, we see that both algorithms exhibit the same behavior as they do at the start. Max-Cap adjusts nicely and clusters replica utilization at around 80%, while Avail-Cap starts to suffer again.

Figures 5c and 5d show the utilization scatterplot for the second dynamic experiment. We see that changing half the replicas every 60 seconds can dramatically affect S_{curr}. For example, when S_{curr} drops to $0.2S_{orig}$ at time 2161, we see the utilizations rise dramatically for both Avail-Cap and Max-Cap. This is because during this period the workload request rate is four times that of S_{curr}. However by time 2401, S_{curr} has risen to $1.2S_{orig}$ which allows for both Avail-Cap and Max-Cap to adjust and decrease replica overload. At the next replica set change at time 2461, S_{curr} equals S_{orig}. During the next minute we see that Max-Cap overloads very few replicas whereas Avail-Cap does not adjust well.

These dynamic experiments show two things; first, when the workload is not very demanding and there is plenty of extra capacity, the behavior of Avail-Cap comes close to that of Max-Cap. However, Avail-Cap suffers more as overall capacity decreases. Second, Avail-Cap is affected more by short-lived decreases in total maximum capacity than Max-Cap. This is because the reactive nature of Avail-Cap causes it to adapt abruptly to any changes in capacities. From the experiments, we conclude that in a dynamic environment such as a peer-to-peer network, Max-Cap is the better choice.

F. Extraneous Load

As we have shown above, when replicas can honor their maximum capacities, Max-Cap avoids the oscillation that Avail-Cap can suffer, and does so with no update overhead. Occasionally, some replicas may not be able to honor their maximum capacities because of *extraneous load* caused by other applications running on the replicas or network conditions unrelated to the content request workload.

To deal with the possibility of extraneous load, we modify the Max-Cap algorithm slightly to work with honored maximum capacity, which is maximum capacity minus the extraneous load a replica is experiencing. A peer node chooses a replica to forward a content request to with probability proportional to the honored maximum capacity.

We view the honored maximum capacity reported by a replica A as a contract. If A cannot adhere to its contract or has extra capacity to give, and does not report the deficit or surplus, then A alone will be affected and may be overloaded or underloaded since it will be receiving a request share that is proportional to its previous advertised honored maximum capacity.

If, on the other hand, replica A chooses to issue a new contract with the new honored maximum capacity, then this can cause a portion of A's workload to shift to the other replicas. This shift however does not affect the contracts of the other replicas. The contract of another replica B is only affected by the extraneous load experienced by B. In contrast, in Avail-Cap the available capacity reported by one replica directly affects the available capacities reported by the others.

In experiments where we inject extraneous load into the replicas, we have found that the performances of Max-Cap and Avail-Cap are similar to those seen in the dynamic replicas experiments [RB03b]. This is because when a replica advertises a new honored maximum capacity, it behaves as if that replica were leaving and being replaced by a new replica with a different maximum capacity.

IV. Related Work

Load-balancing has been the focus of many studies described in the distributed systems literature. In the interest of space we describe previous techniques that could be applied in a peer-to-peer context. Other techniques that cannot be directly applied in a peer-to-peer context such as task handoff through redirection (e.g., [CCY99], [AYI96], [AB00]) or process migration (e.g., [LL96]) from heavily-loaded to lightly-loaded servers in a cluster are described in the extended version of this paper [RB03b].

Of the algorithms based on load, a very common approach to performing load-balancing is to choose the server with the least reported load from among a set of servers. This approach performs well in a homogeneous system where the task allocation is performed using complete up-to-date load information [Web78], [Win77]. In a system where multiple dispatchers are independently performing the allocation of tasks, this approach however has been shown to behave badly, especially if load information used is stale [ELZ86], [MTS89], [Mit97], [SKS92].

Dahlin [Dah99] proposes *load interpretation* algorithms which take into account the age (staleness) of the load information reported by each of a set of distributed homogeneous servers as well as an estimate of the rate at which new requests arrive at the whole system to determine to which server to allocate a request.

Many studies have focused on the strategy of using a subset of the load information available. This involves first randomly choosing a small number, k, of homogeneous servers and then choosing the least loaded server from within that set [Mit], [ELZ86], [VDK96], [ABKU94], [KLH92], [GC00]. In particular, for homogeneous systems, Mitzenmacher [Mit] studies the tradeoffs of various choices of k and various degrees of staleness of load information reported. As the degree of staleness increases, smaller values of k are preferable.

Of the algorithms based on available capacity, one common approach has been to choose amongst a set of servers based on the available capacity of each server [ZYZ$^+$98] or the available bandwidth in the network to each server [CC97]. The server with the highest available capacity/bandwidth is chosen by a client with a request. The assumption here is that the reported available capacity/bandwidth will continue to be valid until the chosen server has finished servicing the client's request.

Another approach is to exclude servers that fail some utilization threshold and to choose from the remaining servers. Mirchandaney et al. [MTS90] and Shivaratri et al. [SKS92] classify machines as lightly-utilized or heavily-utilized and then choose randomly from the lightly-utilized servers. This work focuses on local-area distributed systems. Colajanni et al. use this approach to enhance round-robin DNS load-balancing across a set of widely distributed heterogeneous web servers [CYC98]. The maximum capacities of the most capable servers are at most a factor of three that of the least capable servers. As we see in Section III-A, when applied in the context of a peer-to-peer network where the maximum capacities of the replicas can differ by two orders of magnitude, excluding a serving node temporarily from the allocation decision can result in wild load oscillation.

V. Conclusions

In this paper we examine the problem of load-balancing in a peer-to-peer network where the goal is to distribute the demand for a particular content fairly across the set of replica nodes that serve that content. Existing load-balancing algorithms proposed in the distributed systems literature are not appropriate for a peer-to-peer network. Algorithms based purely on load do not handle peer heterogeneity and algorithms based on available capacity can surprisingly suffer from wild load oscillations even when the workload request rate is as low as 60% of the total maximum capacities of replicas. These load oscillations result in unutilized capacity.

We propose and evaluate Max-Cap, a practical algorithm for load-balancing. Max-Cap handles heterogeneity, yet does not suffer from oscillations even as the workload rate approaches 100% of the total maximum capacities of the replicas. It adjusts better to very large fluctuations in the workload and constantly changing replica sets. Moreover, Max-Cap incurs much less overhead, since the contract issued by one replica is independent of the contracts issued by others. In fact, if replicas rarely choose to change contracts, Max-Cap incurs near-zero overhead regardless of the network size. We believe this makes Max-Cap a practical and elegant algorithm for load-balancing in peer-to-peer networks.

References

[AB00] L. Aversa and A. Bestavros. Load Balancing a Cluster of Web Servers Using Distributed Packet Rewriting. In *IEEE Intl Performance, Computing, and Communications Conference*, February 2000.

[ABKU94] Y. Azar, A. Broder, A. Karlin, and E. Upfal. Balanced Allocations. In *ACM Symp. on Theory of Computing*, 1994.

[AYI96] D. Andresen, T. Yang, and O.H. Ibarra. Towards a Scalable Distributed WWW Server on Networked Workstations. *Journ. of Parallel and Distributed Computing*, 42:91–100, 1996.

[Cao02] P. Cao. Search and Replication in Unstructured Peer-to-Peer Networks, February 2002. Talk at http://netseminar.stanford.edu/.

[CC97] R. Carter and M. Crovella. Server Selection Using Dynamic Path Characterization in Wide-Area Networks. In *Infocom*, 1997.

[CCY99] V. Cardellini, M. Colajanni, and P.S. Yu. Redirection Algorithms for Load Sharing in Distributed Web Server Systems. In *ICDCS*, June 1999.

[CYC98] M. Colajanni, P. S. Yu, and V. Cardellini. Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers. In *ICDCS*, 1998.

[Dah99] M. Dahlin. Interpreting Stale Load Information. In *ICDCS*, 1999.

[ELZ86] D. Eager, E. Lazowska, and J. Zahorjan. Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Trans. on Software Engineering*, 12(5):662–675, 1986.

[GC00] Z. Genova and K. J. Christensen. Challenges in URL Switching for Implementing Globally Distributed Web Sites. In *Workshop on Scalable Web Services*, 2000.

[gnu] The Gnutella Protocol Specification v0.4. http://gnutella.wego.com.

[KLH92] R. Karp, M. Luby, and F. M. Heide. Efficient PRAM Simulation on a Distributed Memory Machine. In *24th ACM Symposium on Theory of Computing*, 1992.

[LL96] C. Lu and S.M. Lau. An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple Task Classes. In *ICDCS*, 1996.

[Mar02] E. P. Markatos. Tracing a large-scale Peer-to-Peer System: an hour in the life of Gnutella. In *2nd IEEE/ACM Intl Symposium on Cluster Computing and the Grid*, 2002.

[MGB01] P. Maniatis, T.J. Giuli, and M. Baker. Enabling the Long-Term Archival of Signed Documents through Time Stamping. Technical Report cs.DC/0106058, Stanford University, June 2001. http://www.arxiv.org/abs/cs.DC/0106058.

[Mit] M. Mitzenmacher. The Power of Two Choices in Randomized Load Balancing. PhD thesis, UC Berkeley, Sept. 1996.

[Mit97] M. Mitzenmacher. How Useful is Old Information? In *PODC*, 1997.

[MTS89] R. Mirchandaney, D. Towsley, and J. Stankovic. Analysis of the Effects of Delays on Load Sharing. *IEEE Trans. on Computers*, 38:1513–1525, 1989.

[MTS90] R. Mirchandaney, D. Towsley, and J. Stankovic. Adaptive Load Sharing in Heterogeneous Distributed Systems. *Journ. of Parallel and Distributed Computing*, 9:331–346, 1990.

[PF95] V. Paxson and S Floyd. Wide-area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3), June 1995.

[RB03a] M. Roussopoulos and M. Baker. CUP: Controlled Update Propagation in Peer to Peer Networks. In *To appear in 2003 USENIX Annual Technical Conference*, June 2003.

[RB03b] M. Roussopoulos and M. Baker. Practical Load Balancing for Content Requests in Peer-to-Peer Networks. Technical report, Stanford University, January 2003. http://mosquitonet.stanford.edu/ mema.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *MiddleWare*, November 2001.

[RF02] Matei Ripeanu and Ian Foster. Mapping the Gnutella Network: Macroscopic Properties of Large-Scale Peer-to-Peer Systems. In *IPTPS*, 2002.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, 2001.

[SGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. of Multimedia Computing and Networking*, 2002.

[SKS92] N. Shivaratri, P. Krueger, and M. Singhal. Load Distributing for Locally Distributed Systems. *IEEE Computer*, pages 33–44, Dec 1992.

[SMK+01] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[VDK96] N. Vvedenskaya, R. Dobrushin, and F. Karpelevich. Queuing Systems with Selection of the Shortest of Two Queues: an Asymptotic Approach. *Problems of Information Transmission*, 32:15–27, 1996.

[Web78] R. Weber. On the Optimal Assignment of Customers to Parallel Servers. *Journal of Applied Probability*, 15:406–413, 1978.

[Win77] W Winston. Optimality of the Shortest Line Discipline. *Journal of Applied Probability*, 14:181–189, 1977.

[ZYZ+98] H. Zhu, T. Yang, Q. Zheng, D. Watson, O. H. Ibarra, and T. Smith. Adaptive load sharing for clustered digital library services. In *7th IEEE HPDC)*, 1998.

APPENDIX I
THE CUP PROTOCOL

In this appendix we briefly describe how we leverage the CUP protocol [RB03a] to study the load-balancing problem in a peer-to-peer context. CUP is a protocol for maintaining caches of index entries in peer-to-peer networks through *C*ontrolled *U*pdate *P*ropagation. We describe how CUP works over structured peer-to-peer networks. In such networks, lookup queries for particular content follow a well-defined path from the querying node toward an *authority node*, which is guaranteed to know the location of the content within the network [RFH+01], [RD01], [SMK+01].

In CUP every node in the peer-to-peer network maintains two logical channels per neighbor: a query channel and an update channel. The query channel is used to forward lookup queries for content of interest to the neighbor that is closest to the authority node for that content. The update channel is used to forward query responses asynchronously to a neighbor. These query responses contain sets of index entries that point to nodes holding the content in question. The update channel is also used to update the index entries that are cached at the neighbor.

Figure 6 shows a snapshot of CUP in progress in a network of seven nodes. The four logical channels are shown between each pair of nodes. The left half of each node shows the set of content items for which the node is the authority. The right half shows the set of content items for which the node has cached index entries as a result of handling lookup queries. For example, node A is the authority node for content $K3$ and nodes C,D,E,F, and G have cached index entries for content $K3$. The process of querying and updating index entries for a particular content K forms a CUP tree whose root is the authority node for content K. The branches of the tree are formed by the paths traveled by lookup queries from other nodes in the network. For example, in Figure 6, node A is the root of the CUP tree for $K3$ and branch $\{$F,D,C,A$\}$ has grown as a result of a lookup query for $K3$ at node F.

It is the authority node A for content $K3$ which is guaranteed to know the location of all nodes, called *content replica nodes* or simply *replicas*, that serve content $K3$. Replica nodes first send birth messages to authority A to indicate they are serving content $K3$. They may also send periodic refreshes or invalidation messages to A to indicate they are still serving or no longer serving the content. A then forwards on any birth, refresh or invalidation messages it receives, which are propagated down the CUP tree to all interested nodes in the network. For example, in Figure 6 any update messages for index entries associated with content $K3$ that arrive at A from replica nodes are forwarded down the $K3$ CUP tree to C at level 1, D and E at level 2, and F and G at level 3.

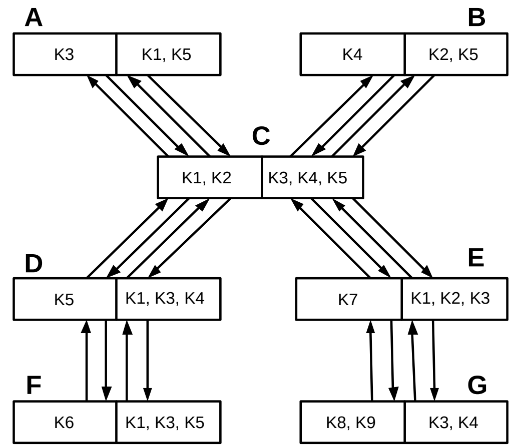


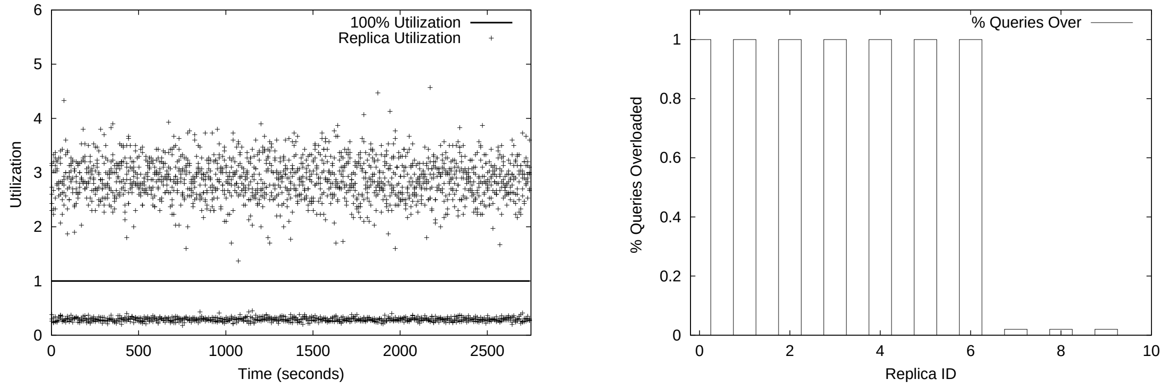
Fig. 6. CUP Trees

We can leverage CUP to propagate updates for a variety of metadata, including load balancing information such as replica load or available capacity to interested peer nodes throughout the overlay network. These peer nodes can then use this information when choosing to which replica a client request should be forwarded.

APPENDIX II
INV-LOAD AND HETEROGENEITY

In this appendix, we examine the performance of Inv-Load in a heterogeneous peer-to-peer environment. We use a fairly short inter-update period of one second, which is quite aggressive in a large peer-to-peer network. We have ten replica nodes that serve the content item of interest, and we generate request rates for that item according to a Poisson process with arrival rate that is 80% the total maximum capacities of the replicas. Under such a workload, a good load-balancing algorithm should be able to avoid overloading some replicas while underloading others. Figure 7a shows a scatterplot of how the utilization of each replica proceeds with time when using Inv-Load. We define utilization as the request arrival rate observed by a replica divided by the maximum capacity of the replica. In this graph, we do not distinguish among points of different replicas. We see that throughout the simulation, at any point in time, some replicas are severely overutilized (over 250%) while others are lightly underutilized (around 25%).

Figure 7b shows for each replica, the percentage of all received requests that arrive while the replica is overloaded. This measurement gives a true picture of how well a load-balancing algorithm works for each replica. In Figure 7b, the replicas that receive almost 100% of their requests while overloaded (i.e., replicas 0-6) are the low and medium-capacity replicas. The replicas that receive almost no requests while overloaded (i.e., replicas 7-9) are the high-



(a) (b)

Fig. 7. Inv-Load: (a) Replica Utilization versus Time, (b) Percentage Overloaded Requests versus Replica ID.

capacity replicas. We see that Inv-Load penalizes the less capable replicas while giving the high-capacity replicas an easy time.

Inv-Load is designed to perform well in a homogeneous environment. When applied in a heterogeneous environment such as a peer-to-peer network, it fails. As we see in Section III-A, Max-Cap is much better suited for heterogeneous environments. Moreover, a nice bonus is that Max-Cap has better load-balancing performance than Inv-Load even in a homogeneous environment. For more details, see the extended version of this paper [RB03b].