RYAN PETERSON,
VENUGOPALAN RAMASUBRAMANIAN,
AND EMIN GÜN SIRER

a practical approach to peer-to-peer publish-subscribe

Ryan Peterson is a graduate student in the Department of Computer Science at Cornell University. He is interested in distributed systems and networking.

ryanp@cs.cornell.edu

Venugopalan Ramasubramanian is a graduate student in the Department of Computer Science at Cornell University and is currently affiliated with Microsoft Research, Silicon Valley Lab. His current research interests lie in applying principled approaches to build sound yet practical distributed systems.

rama@microsoft.com

Emin Gün Sirer is an assistant professor of computer science at Cornell University. He received his Ph.D. in computer science from the University of Washington. His recent research includes self-organizing peer-to-peer systems, reputation systems, and a new operating system for trusted computing.

egs@cs.cornell.edu

THE WEB HAS FAILED TO FULFILL ITS promise of delivering relevant news and information in a timely fashion. In fact, it doesn't deliver anything on its own at all; instead, it requires its users to explicitly poll information sources. Checking for updates by pointing, clicking, and reloading Web sites, whether the sites are Slashdot, news, or online classifieds, is not only slow, inefficient, and cumbersome for users, but it places an unnecessary bandwidth burden on content providers. Recent attempts to automate this process, with the aid of feed readers, have created more problems than they have solved. A system that detects updates to content anywhere on the Web and delivers it to users via an asynchronous channel, such as an instant message, would do much to relieve the burden on users and content providers alike.

Recent years have seen an explosion of online services, including countless blogs and frequently updated news sites. Consequently, sifting through newly published data for useful and interesting information has become a daunting task. The Web is based on a pull-based architecture that forces users to receive new content by explicit polling, which adds to the difficulty of discovering new information. Past efforts to replace the pull-based paradigm with a push-based publish-subscribe paradigm have not seen wide-scale adoption in practice. This is primarily because publish-subscribe systems have required content providers to make significant changes to their infrastructure and workflow for publishing information.

Such inefficiencies motivate a notification system for the Web that alerts users when sites are updated without demanding any support from content providers. The increasingly popular Really Simple Syndication (RSS) standard provides one alternative. RSS is both a format for publishing content and a system for detecting published updates. Content providers publish updates to special XML-formatted documents called feeds or channels. Clients subscribe to their favorite RSS feeds using special feed-reader software, which checks for updates by periodically polling the feeds and comparing their contents with the results of the previous poll. Unfortunately, RSS places a large burden on content servers: Every client sub-

scribed to the same site must poll that site independently and repeatedly. This has forced content providers to limit client polling rates based on IP address, to save bandwidth. For instance, Slashdot allows a maximum of two polls per client each hour. Such polling-rate limitations restrict update detection time to an average of 15 minutes at best. Also, because content providers limit clients by IP address, RSS is especially impractical for clients behind NATs, since all clients behind a NAT share an IP address. For content providers, RSS traffic incurs ongoing costs as it tends to be "sticky": Once users subscribe to a site, they are unlikely to unsubscribe even as their interest in the site diminishes, resulting in wasted bandwidth. Overall, the automation that RSS provides compounds the problems inherent in a pull-based architecture.

Luckily, the recent emergence of self-organizing overlays, where nodes arrange themselves to provide a service without a centralized authority or administrator, provides a starting point for building practical systems that address the flaws of previous notification systems.

We have built and deployed a peer-to-peer publish-subscribe system for the Web called Corona [2] that uses cooperative polling and intelligent allocation of available resources to efficiently discover new information on the Web and push it to users. Corona enables clients to subscribe to Web sites called *channels*, monitors these information sources, and quickly disseminates any detected updates to clients interested in those channels. The central tradeoff in any such system revolves around the most limited resource: bandwidth limits imposed by the physical link capacities on the polling client side, the physical bandwidth limits on the server side, and the bandwidth limits stemming from the polling limits set forth by the content providers. Corona uses mathematical optimization to achieve the best possible update performance given the bandwidth available to the system. It does this by setting up a constrained optimization problem of maximizing a performance function while a cost function does not exceed a given limit. In one incarnation, used in our current deployment, it can minimize average update detection time for all subscriptions while guaranteeing that content providers never see any more requests than they would allow if the clients polled the servers directly. Thus, Corona does not require content providers to make any changes to their systems.

Corona can be accessed conveniently through instant messengers. Clients subscribe to channels by simply registering a screen name with one of several popular instant messaging systems we support [1]. When Corona discovers new updates for a subscribed channel, it pushes the updates to users as concise instant messages showing the updated portions of the Web site.

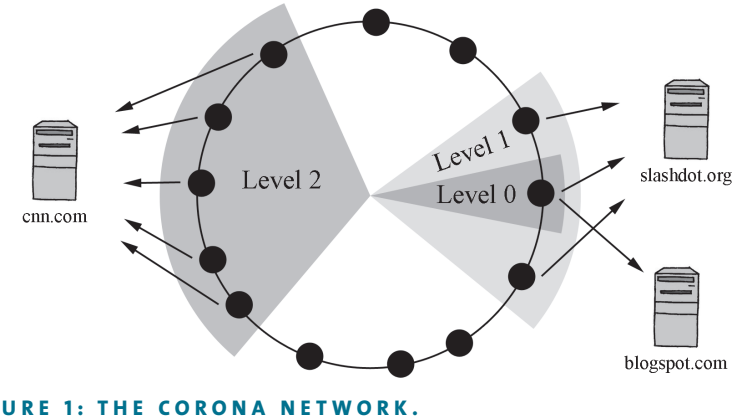

FIGURE 1: THE CORONA NETWORK. ARROWS DENOTE PERIODIC POLLING.

Corona is structured as a logical ring of nodes in an overlay network. We built Corona on top of Pastry [3], but any structured overlay that has a uniform distribution of nodes is sufficient. Corona sits between users, who submit subscriptions into the system and await updates, and content providers, which post the updates that Corona detects. Figure 1 depicts Corona's internal structure and how it polls content servers. The solid black dots represent the system's nodes, each of which is mapped to a unique location on the ring. Each channel has a designated home node, derived from a hash of the channel's URL. The home node for a channel is responsible for periodically polling the channel for updates and delegating neighbors to poll the channel as well if doing so would improve performance according to the system's performance goals. The notion of a group of adjacent nodes polling the same channel is captured formally by using polling levels, shown as shaded regions in the figure. We say that a channel has a polling level of 0 if only its home node polls the channel. A polling level of 1 means that all nodes within a one-hop neighborhood of the channel's home node poll the channel. Therefore, the number of nodes polling a channel increases exponentially as the channel's level increases. Nodes polling the same channel share updates with each other and instant-message new updates to subscribers.

The challenge here is to determine the best polling levels for the channels, each of which has a different number of subscribers, feed size, and update rate. Corona accomplishes this using a novel optimization framework, which assigns polling levels based on informed tradeoffs between network bandwidth and update detection time. Corona uses this to achieve specific performance goals, subject to given resource constraints. If network bandwidth and server load were not issues at all, the optimal solution would simply be to assign each channel the maximum polling level, so that every node would poll for every subscription. However, network bandwidth is a concern in the Internet, and, as we have discussed, content providers limit polling to prevent clients from inundating them with requests. When assigning polling levels to channels, intuitively it seems that one should give more popular channels higher polling levels. That is, if there are many subscriptions for a particular channel, there should be more nodes polling it, so that updates are detected more quickly. A channel with only one or two subscribers, in contrast, should only be polled by a couple of nodes, since there are fewer clients that benefit from updates to that channel. Similar tradeoffs hold for the channel size and update rates, which complicate the process of determining the optimal solution and render it NP-hard. Corona sets up this tradeoff as an optimization problem that describes the performance goals and constraints, then solves for the polling level of each channel using an approximate search algorithm. The algorithm runs in a matter of seconds to find a solution that is within one channel per node of the optimal solution [2].

Since there is no central authority in Corona, the system uses a completely distributed approach to assigning polling levels to channels. Each node solves the optimization problem locally and uses the results to decide the polling level of each channel it is currently polling. If the polling level of a channel changes, it notifies the nodes at the next higher polling level. Since each node uses only local information in its optimizing computation, Corona employs a mechanism that occasionally aggregates information from multiple nodes so that each node has an approximate system-wide perspective.

By assigning different polling levels to each channel, Corona can achieve a variety of different performance goals. We have already described one performance goal for Corona, that is, to minimize average update detection time across all subscriptions while ensuring that content servers do not see an increase in bandwidth consumption as clients opt to use Corona instead of legacy RSS. We call this performance goal Corona Lite. Our experiments show that this strategy improves detection time by a factor of 20, finding updates in an average of only 45 seconds after they have been published. Suppose, however, that it is important to receive prompt updates (say, 30 seconds on average) when a blogger posts a new message. Another performance goal, called Corona Fast, is well suited for this scenario: It guarantees that, on average, updates are detected within a specified amount of time, while minimizing bandwidth consumption.

The traditional approach for allocating resources in a publish-subscribe notification system is to use heuristics tailored to specific performance metrics. One such system is FeedTree [4]. Like Corona, FeedTree is a completely decentralized system for detecting Web updates, taking advantage of cooperative polling among nodes to reduce update detection time. However, FeedTree uses heuristics instead of analytical models to assign each channel to the optimal number of polling nodes, so it cannot guarantee that it gets updates to subscribers as quickly as possible.

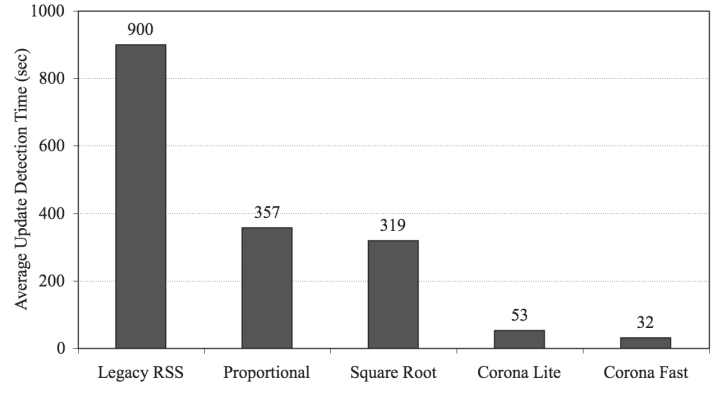

FIGURE 2: CORONA VERSUS HEURISTICS.

An intuitive heuristic-based strategy for polling nodes for updates is to poll each channel with a number of nodes proportional to the popularity of the channel, that is, to the number of subscriptions. This strategy represents the scenario where all the clients interested in a channel cooperate and share updates. An alternative heuristic, which gives more weight to the more unpopular channels, is to set the number of nodes per channel proportional to the square root of channel popularity. We compared these heuristics to legacy RSS systems and Corona to determine their effectiveness in detecting updates. The results are summarized in Figure 2. Using a typical polling period of 30 minutes, legacy RSS discovers new updates an average of 15 minutes after they have been published. We find that both heuristics improve detection time almost threefold compared to legacy RSS. In contrast, treating the problem formally as a mathematical optimization problem enables Corona Lite to provide more than an order of magnitude increase in performance, using the same amount of bandwidth as naive RSS and heuristic methods. (Corona Fast uses slightly more bandwidth for even better update detection latency, as its optimization goal is to guarantee a targeted update latency without limiting its bandwidth consumption.) Overall, heuristic approaches, although easy to devise, tend to be tailored to specific scenarios or benchmarks, do not perform well for nontrivial problems, and ultimately do not provide any guarantees.

Corona is a practical publish-subscribe system for the Web that solves the problems posed by RSS and current publish-subscribe systems. It polls on the user's behalf, intelligently allocating resources to achieve performance goals, instead of requiring clients to repeatedly poll content servers directly. Corona provides strong performance guarantees, ensuring that subscribers receive up-to-the-minute information from their favorite Web sites without introducing yet more unnecessary traffic onto the Internet.

More importantly, Corona exemplifies a new method for building decentralized Internet services. We have shown that a rigorous approach to distributed system design where performance goals and cost metrics are expressed formally can yield practical high-performance systems that vastly outperform heuristics commonly encountered in system design today. We believe that as systems become more complex and difficult to reason about, heuristics are unlikely to be successful in substantially improving system performance, and using mathematical optimization can yield more efficient systems with better resource utilization.

REFERENCES

[1] Corona: http://www.cs.cornell.edu/people/egs/beehive/corona/, May 2006.

[2] V. Ramasubramanian, R. Peterson, and E. G. Sirer, "Corona: A High Performance Publish-Subscribe System for the World Wide Web," in *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)* (Berkeley, CA: USENIX, 2006).

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms* (Heidelberg, Germany, 2001).

[4] D. Sandler, A. Mislove, A. Post, and P. Druschel, "FeedTree: Sharing Web Micronews with Peer-to-Peer Event Notification," in *Proceedings of the 4th International Workshop on Peer-to-Peer Systems* (Ithaca, NY, 2005).