DHT-Based Distributed Crawler
CS294-4 Peer-to-Peer Systems

Owen Cooper, Sailesh Krishnamurthy, Boon Thau Loo
{owenc,sailesh,boonloo}@cs.berkeley.edu

1.0 Motivation

A search engine, like Google, is built using two pieces of infrastructure - a crawler that indexes the web and a searcher that uses the index to answer user queries. While Google's crawler has worked well, there is the issue of timeliness and the lack of control given to end-users to direct the crawl according to their interests. The interface presented by such search engines is hence very limited. Since the underlying index built out of the crawl is not exposed, it is difficult to build other applications, such as focused crawlers (eg Bingo! [1]) that try to do more than just return a set of URLs.

The goal of our project is to build a peer-to-peer (p2p) decentralized crawler deployable over the wide-area. This decentralized crawler will serve as a base infrastructure for clients to build their own personalized crawlers. Different clients can choose to build their own different and personalized searching tools on top of this common crawling infrastructure. On top of allowing for user control over the crawl, a distributed crawler can give the advantage of having better aggregate bandwidth usage (as compared to a centralized crawler) where the crawlers are chosen based on geographic proximity to the web sites that they are downloading from. As more nodes join the system, the aggregate bandwidth increases and the crawl will scale organically. Based on recent proposals on using p2p networks to index the web [13], it is conceivable that such an infrastructure would require the use of the p2p networks themselves to actively crawl the web sites for indexing.

2.0 Focus of the Project

The focus of the project is to build a distributed crawler that displays good scaling properties. We will also explore different tradeoffs in the design space. We plan to deploy the crawler on Planetlab [11].

The distributed crawler will be built using Distributed Hash tables (DHTs) [10]. DHTs provide good load balancing and also graceful handling of node joins and leaves. This is important both from a crawl-load distribution perspective, and also allows the crawl to proceed despite churn in the network. To drive the actual crawl, we will utilize PIER, a P2P relational query engine [4] over DHTs. In particular, we will utilize PIER's recursive query facilities (repeated web crawling fits the recursive query model quite well). As for the actual downloading of the web pages, we will use the TeSS (Telegraph Screen Scraper) infrastructure [3] which handles both ordinary and deep web pages.

2.1 Load Distribution Policies

The key to achieving a scalable "crawling service" is in distributing the crawling load appropriately. The data-flow shown above depicts only the load distribution _mechanism_ in its essence. To partition the crawling load, there are many policies we can choose from. One of the important contributions of this project is to study different partitioning policies. Some of these policy choices include:

- Partitioning the crawling by URL
- Partitioning the crawling by hostname
- Hierarchical partitioning by hostname that provides more crawling nodes for bigger domains
- Select geographically close crawlers for each target web site. Explore the use of triangle inequality or the use of the "King" tool [2] from the University of Washington.

Picking different partitioning policies will have an impact on load balancing of both crawler and web sites. It will also have an impact on bandwidth consumption, which may come be an important factor in keeping our distributed crawler efficient.

2.2 Avoiding a DOS effect

It is important for the distributed crawler not to launch a DOS attack on the sites being crawled. An important task will be to explore ways in which we can throttle back the number of total connections to a given hostname. This is especially important if URLs display locality within certain pages. E.g. a Yahoo page would contain several Yahoo links that would require crawling the same page within a short time frame. Of course, different domains may be able to withstand a different number of connections. There are trade-offs between the load balancing policy of choice and the efficacy of our throttling techniques. For instance, it is easy to throttle load to a given web site when we have partitioned the crawling by the domain hostname, but that might not be a good way to evenly balance the load. Coordinating different crawlers to throttle their access to a web site would require distributed state maintenance. A naive solution is to require nodes to periodically publish information on their download rates on rate-limited sites into the DHT. A continuous query executed in PIER is used to aggregate the crawl statistics and sent to the participating crawler. This scheme exposes tradeoffs in timeliness vs bandwidth consumptions.

2.3 Crawl Reordering

Individual crawler nodes may choose to shuffle the order in which the crawl is performed. This reordering can be encapsulated within a reordering PIER operator, and the reordering policies will be based on user-defined functions. This allows crawl policies that can be prioritized by how important page is or catered to user interests. This gives more flexible crawl orderings apart from the standard randomized, breadth-first or depth-first crawls.

2.4 Metrics

We plan to use some of the following metrics in evaluating the tradeoffs discussed:

- Throughput (total pages crawled per unit of wall clock time)
- Message overheads (bandwidth per page crawled)
- Accesses per unit time (for each hostname) => this serves as a metric for how effective our anti-DOS policies are

2.5 Other Research Issues

Our primary goal in building this crawler is to make it feasible and efficient. We have identified several other important and interesting issues worth exploring as part of future work beyond the class project.

- Provide a keyword search facility for the crawler. This is orthogonal to the design of the crawler.
- Build-in crawl personalization facilities for end-users. We will provide the base infrastructure to do personalized crawls, but leave the personalization to higher-level applications.
- Persistent storage and indexing of web pages. This is orthogonal to the design of the crawler. For better persistence, storing the web pages in Oceanstore [12] would be ideal.
- In the P2P environments, we recognize that fault tolerance is an important issue. When a crawler node fails, some sites may fail to get crawled. Fault tolerance can be addressed with running redundant queries at the Query Processor layer. Better techniques may be available to us that provide similar reliability at lower bandwidth overheads.
- We will primarily use the Bamboo DHT, but it is interesting to see how different DHTs will impact the crawler's performance.

3.0 Related Work

Most of the existing distributed crawlers [5,6,9] deal with implementation issues of parallel crawlers. In these environments, they do not have to deal with node failures, and communication overheads are less of a concern. Because they are typically deployed "in-house", their bottleneck bandwidth is usually their organization's incoming and outgoing bandwidth to the outside world. As their nodes are not distributed, they cannot leverage on geographic proximity. The MIT Chord project has a web archival system [8] proposed that uses Chord for crawling the web. However, they have only tried their system for a small number of nodes, and do not handle rate-limiting of content providers.

To our best knowledge, Grub [7] is the only widely used P2P web crawler. They are not truly a decentralized infrastructure as they use a SETI@Home architecture. Their selling point is that their system will crawl the web at a much faster rate as more nodes participate in the crawl.

References:

[1] Sergej Sizov, Martin Theobald, Stefan Siersdorfer, Gerhard Weikum, Jens Graupmann, Michael Biwer, Patrick Zimmer: "The BINGO! System for Information Portal Generation and Expert Web Search." CIDR 2003

[2] Krishna P. Gummadi, Stefan Saroiu and Steven D. Gribble: "King: Estimating Latency between Arbitrary Internet End Hosts" in the Proceedings of SIGCOMM IMW 2002, November 2002, Marseille, France.

[3] TeSS, Telegraph Screen Scraper, http://telegraph.cs.berkeley.edu/tess/

[4] Ryan Huebsch, Joseph M.Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, Ion Stoica, Query the Internet with PIER, VLDB,2003

[5] Sergey Brin and Lawrence Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine.

[6] Vladislav Shkapenyuk and Torsten Suel, Deisgn and Implementaiton of a High-Performance Distributed Web Crawler, ICDE 2002

[7] Grub's Distributed Web Crawling Project, http://www.grub.org

[8] Timo Burkard, Herodotus: A Peer-to-Peer Web Archival System, MIT Masters's Thesis, June 2002

[9] J. Cho and H. Garcia-Molina, Parallel Crawlers, WWW, 2002

[10] Looking up data in P2P systems Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, Communications of the ACM, February 2003/Volume 46 No. 2

[11] PlanetLab: An Overlay Testbed for Broad-Coverage Services, Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman, PlanetLab Tech Report PDN-03-009, January 2003.

[12] OceanStore: An Architecture for Global-Scale Persistent Storage. John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Appears in Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000.

[13] On the Feasibility of Peer-to-Peer Web Indexing and Search. Jinyang Li, Boon Thau Loo, Joseph Hellerstein, M. Frans Kaashoek, David Karger, and Robert Morris. 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)