Parallel Crawling for Online Social Networks

Duen Horng Chau, Shashank Pandit, Samuel Wang, Christos Faloutsos
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{dchau, shashank, samuelwang, christos}@cs.cmu.edu

ABSTRACT

Given a huge online social network, how do we retrieve information from it through crawling? Even better, how do we improve the crawling performance by using parallel crawlers that work independently? In this paper, we present the framework of parallel crawlers for online social networks, utilizing a centralized queue. To show how this works in practice, we describe our implementation of the crawlers for an online auction website. The crawlers work independently, therefore the failing of one crawler does not affect the others at all. The framework ensures that no redundant crawling would occur. Using the crawlers that we built, we visited a total of approximately 11 million auction users, about 66,000 of which were completely crawled.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Architecture; H.2 [**Information Systems**]: Information Storage and Retrieval

General Terms

Performance

Keywords

Web Crawler, Web Spider, Parallelization, Online Social Networks

1. INTRODUCTION

As the Web grows, parallel crawlers are needed to meet the need of downloading and storing the massive amount of Web data. Many search engines have implemented their own versions of parallel crawlers to index the Web. However, scientific research on the topic of parallel crawler remains relatively little.

Cho and Garcia-Molina [1] had proposed architectures for parallel crawlers for the Web, together with some metrics for evaluating them. Specifically, they listed three general architectures to avoid redundant crawling among parallel crawlers: the *independent* architecture where no coordination exists among crawlers, the *dynamic assignment* architecture where there is a central coordinator, and the *static assignment* architecture where the Web is partitioned and assigned to each crawler, and the crawlers coordinate among themselves (without a central coordinator) once crawling has started. The authors had left dynamic assignment as future work – which is the architecture that we will be using in this paper, and we will discuss why this is particularly suitable for crawling social networks.

An online social network consists of individuals who are linked to the others in the same network. Some well-known examples of online social network include LinkedIn[1] which helps people build professional connections, and Friendster[2] which helps build personal relationships – for making friends, dating, etc. Online auction sites, are in fact, also social networks, where the auction users take part in buying and selling activities. Indeed, eBay[3], the largest auction site in the world, having more than 212 million registered users, might also be the largest online social network.

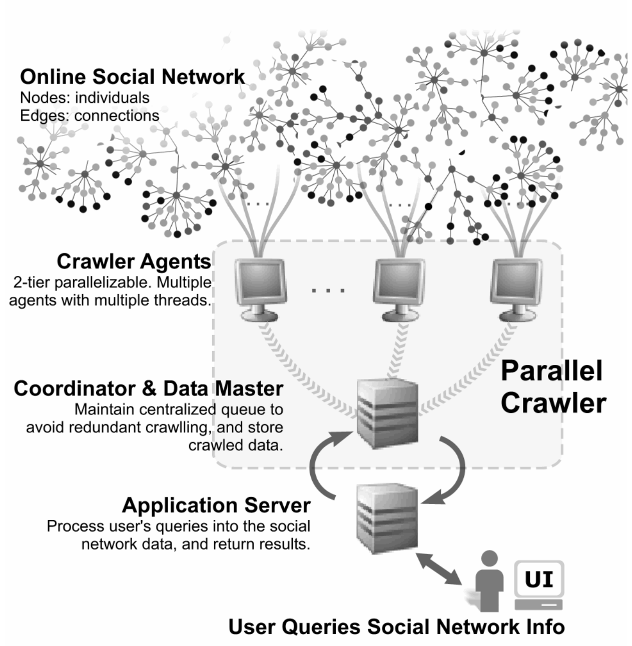

Online Social Network
Nodes: individuals
Edges: connections
Crawler Agents
2-tier parallelizable. Multiple agents with multiple threads.
Coordinator & Data Master
Maintain centralized queue to avoid redundant crawlling, and store crawled data.
Parallel Crawler
Application Server
Process user's queries into the social network data, and return results.
UI
User Queries Social Network Info

Figure 1. Overview of the parallel crawler. Also shown is the application server which typically exists to process user queries into the crawled data.

Online social networks are part of the Web, where a lot of interesting phenomena take place; and many of them are worth studying, or paying attention to. For example, on an online auction, we may want to find out the patterns of fraudulent or suspicious transactions among users.

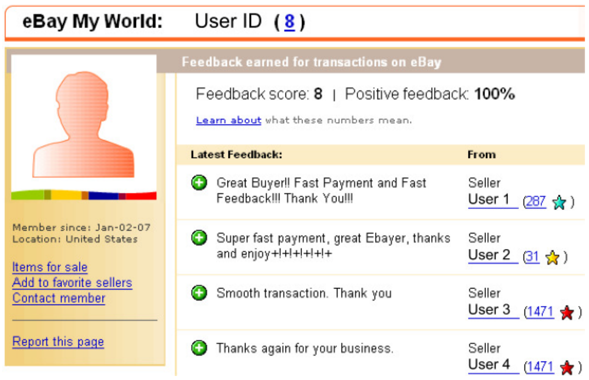
But before such analysis can be done, we need to gather the data that describes the social networks. These networks are often huge, therefore crawling these networks could be both challenging and interesting. However, there has been little documented work on the crawling of these data. Heer and Boyd [2] mentioned their use of a crawler to gather Friendster data for their Vizster social

[1] http://www.linkedin.com

[2] http://www.friendster.com

[3] http://www.ebay.com

network visualization system, but they did not go into the details of the design and implementation of the crawler.



eBay My World: User ID (8)

Feedback earned for transactions on eBay

Feedback score: **8** | Positive feedback: **100%**

Learn about what these numbers mean.

Member since: Jan-02-07
Location: United States

Items for sale
Add to favorite sellers
Contact member

Report this page

Latest Feedback:	From
Great Buyer!! Fast Payment and Fast Feedback!!! Thank You!!!	Seller User 1 (287)
Super fast payment, great Ebayer, thanks and enjoy+!+!+!+!+!+	Seller User 2 (31)
Smooth transaction. Thank you	Seller User 3 (1471)
Thanks again for your business.	Seller User 4 (1471)

Figure 2. A sample eBay user profile page, listing the recent feedback received by the user. User IDs have been edited to protect privacy.

2. CRAWLING SOCIAL NETWORKS

Online social networks are part of the Web, but their data representations are very different from general web pages. The web pages that describe an individual in an online social network are typically well-structured, as they are usually automatically generated, unlike general web pages which could be authored by any person. Therefore, we can be very certain about what pieces of data we can obtain after crawling a particular individual's web pages. Here we use eBay as an example social network. Figure 2 shows the profile web page of a user on eBay. From the page, we can obtain the user's ID, date of registration, location and other personal details. Besides this *local* information, there are usually explicit links that we can use to trace the user's connections to the others. Referring back to Figure 2, the list of feedback received by the user is shown, including the IDs of the users who left the feedback, which are hyperlinked to those users' profile pages. Thus, by crawling these hyperlinks, we can construct the graph of connections between all the users in the social network.

We crawled these user data in a breadth-first fashion. We used a queue data structure to store the list of *pending* users which had been seen but not crawled. Initially, a seed set of users was inserted into the queue. Then at each step, the first entry of the queue is popped, all feedbacks for that user were crawled, and every user who had left a feedback (but was not yet seen) was enqueued. Once all of the user's feedbacks were crawled, the user was marked as visited, and stored in a separate queue.

2.1 Parallelizing via Centralized Queue

We enhanced the naive breadth-first strategy to make it parallelizable by migrating the queue to a *master* machine, while the crawling of web pages is distributed across several *agent* machines which could be distributed. Each *agent* requests the *master* for the next available user to crawl, and returns the crawled feedback data for this user to the master. The master maintains global consistency of the queue, and ensures that a user is crawled only once. To ensure consistency and scalability of the queue data structure, we decided to use a MySQL database as the platform for the master. This allows us to add new agents without suffering any downtime or configuration issues, while maintaining a proportional increase in performance. Further, each agent itself can open arbitrary number of HTTP connections, and run several different crawler threads. Thus, the crawler architecture allows for two tiers of parallelism – the master controls multiple agents in parallel, while each agent itself uses multiple threads for crawling. Importantly, the failing of one crawler does not affect the operation of the others. Sometimes, we may want to control which user should be crawled next, such as when we want to obtain the most updated information of a user. To make this happen, we can simply insert the user's ID to the front of the centralized queue so that the next available crawler will process the user immediately.

We implemented our eBay crawler in Java, which amounted to about 1000 lines of code. The master stored all of the data in a MySQL 5.0.24 database with the following schema:

```
User      (uid, username, date_joined, location,
          feedback_score, is_registered_user,
          is_crawled)

Feedback  (feedback_id, user_from, user_to, item,
          buyer, score, time)

Queue     (uid, time_added_to_queue)
```

We started the crawling on October 10, and stopped it on November 2. In this period, we visited a total of 11,716,588 users, 66,130 of which were completely crawled. The bottleneck in limiting the download rate was at the speed of the Internet connection, but not at the centralized queue.

3. CONCLUSION

We presented the framework and implementation of parallel crawlers for crawling online social networks. Making use of the fact that social network data are well-structured, and that each individual is identified by a unique identifier, a centralized queue implemented as a database table is conveniently used to coordinate the operation of all the crawlers to prevent redundant crawling. This offers two tiers of parallelism, allowing multiple crawlers to be run on each of the multiple agents, where the crawlers are not affected by any potential failing of the other crawlers. Furthermore, manual override in prioritizing the crawling sequences is possible with simple insertion of a user's identifier to the front of the queue.

4. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants No. IIS-0326322 IIS-0534205. This work is also supported in part by the Pennsylvania Infrastructure Technology Alliance (PITA), an IBM Faculty Award, a Yahoo Research Alliance Gift, with additional funding from Intel, NTT and Hewlett-Packard. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

5. REFERENCES

[1] Cho, J., and Garcia-Molina, H. Parallel crawlers. In Proceedings of the Eleventh International World Wide Web Conference, 2002.

[2] Heer, J., and Boyd, D. Vizster: Visualizing Online Social Networks. IEEE Symposium on Information Visualization (InfoVis), 2005