Data Partition and Job Scheduling Technique for Balancing Load in Cluster of Web Spiders

Surasak Sanguanpong and Kasom Koth-arsa
Applied Network Research Lab, Department of Computer Engineering,
Faculty of Engineering, Kasetsart University
Bangkok 10900, Thailand
e-Mail: {Surask.S,g4265077}@ku.ac.th

Abstract

Search engines primary rely on web spiders to collect large amount of data for indexing and analysis. Data collection can be performed by several agents of web spiders running in parallel or distributed manner over a cluster of workstations. This parallelization is often necessary in order to cope with a large number of pages in a reasonable amount of time. However, while keeping a good parallelism, too many requests should not send from nodes to the same web server in a short period of time. In the environment of a cluster of web spiders, it requires an effective synchronization and scheduling technique between nodes in the cluster. In this paper, we propose a method of URL partition among the node of cluster running web spider and a scheduling technique to dispatch URLs to spider's thread using a simple but efficient URLs scramble. Our approach guarantees that only one node in the cluster can contact a web server in a time, i.e., avoid overloading remote web servers.

Key-Words: Search Engine, Spider, Crawler, Scheduling, Cluster

1. Introduction

Search engines are one of the primary ways that Internet users find information. A conventional search engine consists of 3 main components: information gathering (spider or crawler), information organizer (indexer), and information agent (query). The spider is responsible for downloading web pages from the Internet and stores the data on the local disks. The indexer uses the data collected from the spider to build databases. The query enables users to query the indexed database with the help of a query form.

The spider is one of the key components of search engines. The design and implementation of a web spider might seem to be simple. But it is not a case for large-scale web spiders, where several hundred of pages per second have to be downloaded. One could implement a very high-speed web spider, which has a capability to massively download at a rate of thousand pages per second. However, without careful design considerations, this could create overloading problems on destination web servers and often lead to phone calls or e-mail complaints. In the worse situation, access from web spiders may be restricted due to such behavior. It has been recognized that web spiders should respect the netiquette as designated by the Robot Exclusion Standard [1].

As the size of the web grows, single crawling process approach can not gather pages in a reasonable of time. Advanced spiders run with multiple processes in parallel as well as utilize distributed fashion over a cluster of workstations to maximize download rate. However, it has to ensure that, while keeping a good parallelism, too many requests should not send from nodes to the same web server in a short period of time. In the environment of a cluster of web spiders, it requires an effective synchronization and scheduling technique between nodes in the cluster.

This paper presents a scheduling and synchronization methodology for the cluster of web spiders. We implement the scheduling and synchronization under our cluster of web spiders, called *KSpider*. This paper is organized as follows: Section 2, we mention about related works. Section 3, we provide the architecture, Section 4 we give experimental results and we conclude the paper in the last section.

2. Related works

Several methods can be implemented to distribute URLs to each node in the cluster to achieve load balancing. Site-hash based partition function [2, 3] is

one of such method. The hash value of the site name of a URL (e.g., *moe.go.th* in *http://moe.go.th/index.html*) is computed and its URL is assigned to a spider process. In this scheme, web pages in the same site will be allocated to the same node. Therefore, it still requires other mechanism to control the request rate on each node of the clusters. Because of URLs generated from parsing often point to the same web server due to the locality of link structure [4].

A technique called domain-based throttling [5] can solve this problem. Instead of fetching URLs in the order they were parsed, the URLs will be organized in a list and stored in the reverse order of hostname (e.g. *th.go.moe*). Then, the list will be sorted and is manipulated using *k*-way unshuffle permutation before sending the list to be processed. This technique improves the load balancing property to the target web server, but it rather requires processing time to reorder URLs.

From our knowledge, several design and implement of distributed web spider found in literatures lack of two important scenarios. First, a mechanism in which different nodes in the cluster should not contact the same web server at the same time. Second, an efficient method for URLs reordering method to reduce the effect of link locality is needed. Both of scenarios will provide a baseline of well-behavior distributed web spiders. In the next section, we describe the architecture of *KSpider* and how we partition the set of URLs to distribute and schedule those URLs.

3. Architecture

KSpider's architecture is based on Beowulf cluster [6]. The cluster is connected to the Internet through the campus network. The overall architecture of *KSpider* is shown in Figure 1.

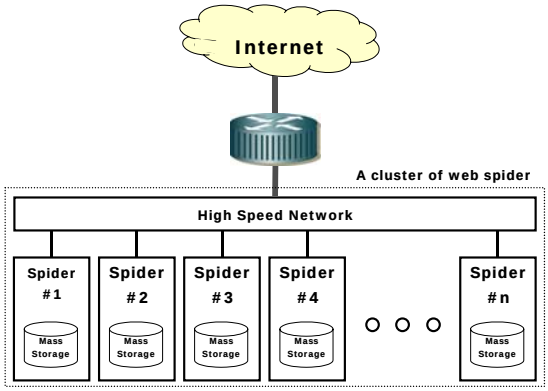

Figure 1. Overall system architecture

Each node (*Spider#1*, *Spider#2*,…, *Spider#n*) in the cluster needs to communicate and synchronize to each other. A node has to exchange discovered URLs which belonging to other nodes. *KSpider* has no central node; each node could execute its jobs independently with different set of data.

3.1 URLs partition

Conceptually, data (web pages, images, and other file type) downloaded from the web servers are distributed over the nodes in the cluster. For any given URL, there is only one node that is responsible to fetch the URL and keep data referenced by that URL.

Let *Nnode* be a total number of nodes inside the cluster. Each node is assigned with a unique identifier, starting from *0* to *Nnode*-1. A simple hash function based on the summation of every character in the URL is used to distribute all URL among the nodes in the cluster. The hash function guaranteed that every node has a disjoint set of the URLs. When a node discovered a new URL, it computes a hashed value of the URL using the hash function in the equation (1). The result of hashing is an identifier of the node in the cluster which has to handle that URL.

$$hash(url) = \sum_{i=0}^{i<len(url)} url[i] \bmod Nnode \qquad (1)$$

To better clarify this concept, suppose there are five nodes in the cluster and let S_N denote the set of URLs belonging to node *N*. The hash will logically separate the whole URLs into five disjoint sets. Each set is represented by a column of S_N, (from S_0 to S_4) as shown in Figure 2.

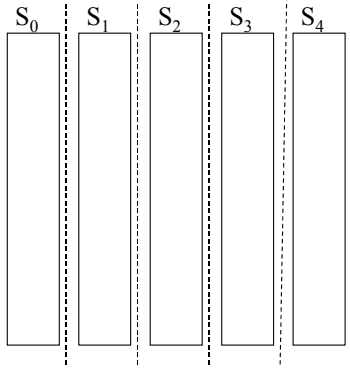

Figure 2. A disjoint set of the URLs among 5 nodes

3.2 Phase partition

Refer to the sets of URLs distributed to the nodes in Figure 2; each node will have of sequence of URLs belonged to the same server. Hence, it is likely that every node may download web data from the same web severs at the same time. Should there are several nodes running simultaneously, it would increasingly generate heavy loads on destination servers. To avoid this situation, a technique called 'phase swapping' is proposed. This technique does not only prevent the overload of the web servers, but also largely reduced the number of URLs the spider has to manage in any given time. The underlie concept of phase swapping is to group the URLs belonged to same web servers together and let each node works on the different set of servers at a time. At a pre-defined constant period, every node synchronously swaps the 'working phase' to a new set of web servers; hence it is called phase swapping.

To assign a working phase, we utilize another hashing function based on the summation of every character from the host name portion of an URL, as shown in the equation (2). The set of URLs from Figure 2 will be rehashed using this hash function. The *Nphase* in the equation is the bound of the hashing function or a number of phases.

$$hash(url) = \sum_{i=0}^{i<len(url)} host(url)[i] \bmod Nphase \quad (2)$$

Suppose that the number of phases is designated to be 6 (*Nphase*=6), hence the second hash function will splits each column of the URLs from Figure 2 into two-dimensional view as shown in Figure 3. Each row represents different phases (set of web servers). The notation P_K denotes the set of URLs in phase *K*.

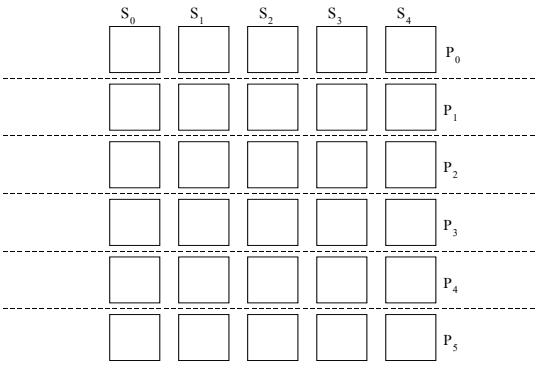

Figure 3. The set of the URLs is further divided by the second hash function

At any given time, only one working set (highlighted) is active on any given node (column), as shown in Figure 4. This means that different nodes in the cluster will not contact the same web server at the same time because they are assigned to handle different working set of web servers.

Note that the number of phases must be equal to or greater than the number of nodes. If the number of phases is less than the number of nodes, some nodes will fetch URLs from the same set of web servers. The optimum number of phases should be equal to the number of nodes in the cluster. Excessive number of phases results in the unbalanced of workload. Experimental results of load distribution will be described later in Section 4.

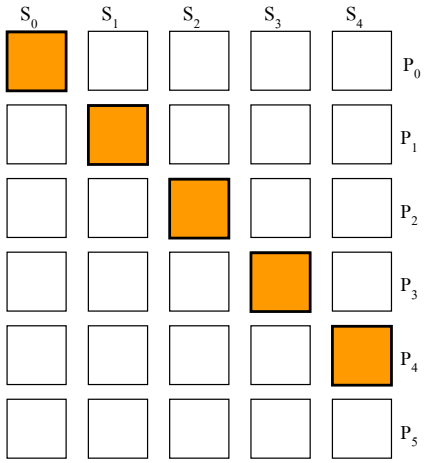

Figure 4. The active set of URLs

3.3 Phase swapping

To let a spider node change its URL working set from a set to another set, a phase swapping is introduced. Initially, the node S_n is assigned to handle the phase P_n, i.e, there are working set of (S_0, P_0), (S_1, P_1), (S_2, P_2), ... (S_N, P_N) where N is the number of nodes and $N \leq Nphase$.

The active sets are changed by the system's wall clock, as shown in Figure 5. Let T_m be the time to swap from phase m to phase $m+1$. When the first time of phase swapping occurs (from T_0 to T_1), the active set of URLs from every node are switched to the next phase from (P_m to P_{m+1}). When the swapping reaches the last phase P_{Nphase}, the next phase is switched back to the first phase P_0 again, i.e., the phase is rotated in a round-robin fashion.

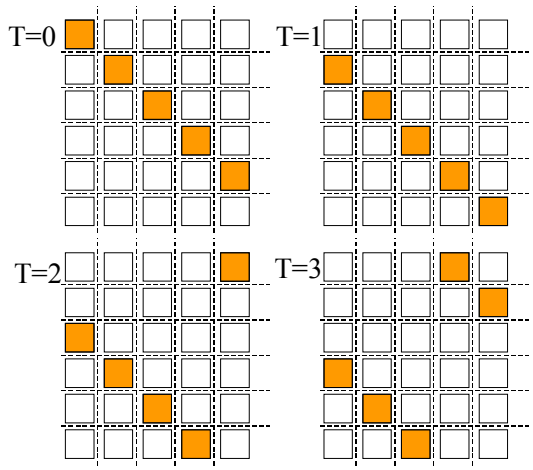

Figure 5. The active set of URLs are changed by swapping phase at different time

In order to let the phase swapping in every node occur at the same time, every nodes need to synchronize the timing. *KSpider* utilizes the Network Time Protocol (NTP) [7] for the phase swapping. The NTP is the standard time synchronization protocol that is efficient and very accurate. There is no master node or any implicit synchronization between nodes.

3.4 Scheduling

The scheduler plays an important role for the execution of downloading. It selects URLs from a set of URLs in the active phase and sends them to the threads responsible for downloading. Logically, the scheduler views all URLs as an array. This array grows corresponded to each new encountered URL. The scheduler takes each URL by scanning and uses array's index for referencing to a corresponded URL. The range of scanning is controlled by a sliding window, *sWin*. The window size, which is adjustable, is currently set to 250,000 unit using the rule of thumb. This access method is quite similar to the FIFO queue. However, the size of array is always increased according to the number of URLs. No element is destroyed along the operation.

The structure of array is very simple. It is the array of integer value, called `URLStates` as illustrated in Figure 6. After the scheduler has already picked each URL out from the array, it assigns the state to that URL. There are four possible states of URLs as the following:

`STATE_IN_LIST:` Any newly discovered URL is in this state. It means that the URL is currently in the list of URLs, but has not been yet scheduled for downloading.

`STATE_IN_PROGRESS:` Any already scheduled URL, but it has not yet complete downloaded.

`STATE_COMPLETED:` Any already downloaded URL either succeeded or failed.

`STATE_STALLED:` Any scheduled URL that can not be downloaded due to server timeout.

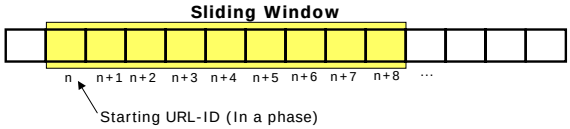

Figure 6. The `URLStates` array

URLs which have been selected from an array are not directly sent to the spider's threads. But they will be cached in a special buffer called a `Bucket`. The structure of `Bucket` is a two-dimension array as shown in Figure 7.

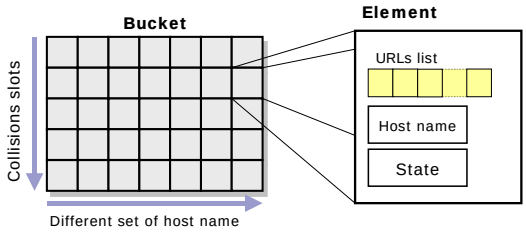

Figure 7. `Bucket` and its `Element`

Each URL will be pushed into the bucket using hashing function of the host name portion. The hash function is shown in equation (3). Hence, every `Element` in the same bucket's column is in the same set of host names. This hashing is nearly identical to the hash functions used to split the URLs into phases, except the divider *Ncolumn*. The number of column is set to 1999 in our current implementation. Its appropriate range should be lie from 199 to 4999. It is important to use a prime number that is not equal to the number of phases; otherwise the bucket will not be balanced.

$$hash(host) = \sum_{i=0}^{i<len(host)} host[i] \bmod nColumn \quad (3)$$

Note that the number of column is obviously less than the number of existing host name; hence hashing collision can not be avoided. Each row of the Bucket is used to handle hashing collisions. In the implementation, the number of row is set to be 8. The optimum range should be from 4 to 16.

Each `Element` in the `Bucket` is not a simple data type but a complex structure of three values, i.e. `List`, `Hostname` and `State`. The `List` is an array of URLs which all belong to the same web servers denoted by `Hostname`. The `State` is an integer value used to denote the status of `Bucket`'s `Element`.

For the sake of clarity, it is worth to explain the operations based on above mentioned objects again. The scheduler selects a URL and hashes it using the equation (3). Each hashed URL will be put into the corresponded `Bucket`'s `Element`. The number of URLs in the `List` of each `Element` independently increases until the `List` is full. Then, the URLs in the `List` are immediately dispatched the spider's threads for downloading. The purpose of `Bucket` is, therefore, very like the phase separation, i.e. to group the same URLs belonging to the same server until it reaches a fair amount of number (20 in the implementation). The benefit of this method is to reduce the number of connection sent to the web servers. Once there are sufficient numbers of URLs belonging to the same server, the HTTP persistent connection is fully utilized. That is, only one connection is created to the web server for downloading several URLs. Moreover, URLs are spread out randomly in the `Bucket`, waiting to be scheduled with less overhead than other method liked sorting.

The last part of the `Element` is the `State`. Its function is to indicate the status of `Element`'s processing. The `State` can be one of the four following possible states: '`STATE_FREE`', '`STATE_QUEUING`', '`STATE_FETCHING`', and '`STATE_TIMEWAIT`'.

The state '`STATE_FREE`' indicates that the element in the `Bucket` is free; this means that the `Bucket`'s element is available for the scheduler to store any URL into it. Once the scheduler occupied this slot, it will change the state to '`STATE_QUEUING`'. The scheduler will also set the `URLStates` to indicate that this URL is in the '`InProgress`' state.

The state '`STATE_QUEUING`' indicates that this element in the `Bucket` is queuing, i.e. waiting for more URLs to be put into.

If the list becomes full, the whole list is sent to the spider threads, and the state will be changed to '`STATE_FETCHING`'.

When a thread has finished a URL fetching, the `Element` will be set from '`STATE_FETCHING`' to '`STATE_TIMEWAIT`'. This state will prevent the spider to continuously fetching from the same server. The scheduler will periodically reset any '`STATE_TIMEWAIT`' to '`STATE_FREE`'. The state diagram of the state of the `Element` in the `Bucket` is shown in Figure 8.

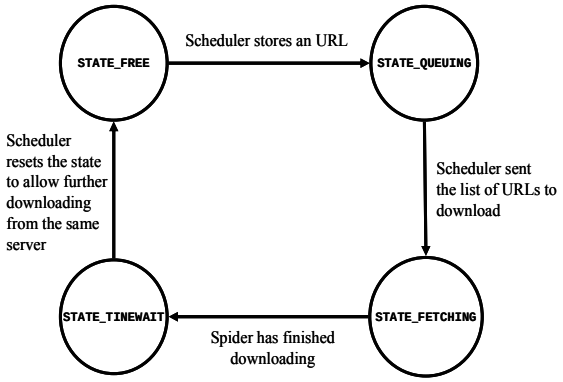

Figure 8. Cycle of the state in the `Element` of scheduler's `Bucket`

4. Experimental Results

URLs partitions as describe above are implemented inside *KSpider* and the performance result are measured. Four set of Athlon XP 1500+ CPU with 768 Mbytes DDR DRAM, six of 35 GBytes, 7200 RPM SCSI hard disk and a gigabit Ethernet interface card. They are connected together using gigabit Ethernet switch. The operating system is RedHat 7.3 with Linux kernel version 2.4.18.

The gathering speed increased when more threads or more machines is used. There are some factors that limit the gathering speed in the test environment. One factor is that the spider's CPU utilization which is rapidly saturated because every spider's thread has to manage a fast stream of web pages. Furthermore, the incoming stream of data has to be parsed and compressed which again consume CPU resources. Actually, the CPU utilization reaches 100% when the

number of threads is closed to 300. *KSpider* achieves an average download rate of 618 URLs/second with sustained 6 Mbytes/second with four machines. This result yields an estimate speed of 53 million pages per day under Thai Internet infrastructure.

Refer to the hashing functions that split the working set of URLs as described in Section 3.1 and 3.2. The first hashing function uses the entire URL to allocate URLs to spider's nodes. From our experimental with 6 million URLs under 2 to 10 nodes of the cluster, the function perfectly distributes the workload as illustrated in Figure 9.

The second hashing function (refer to equation (2)), uses the host name portion of the URL to allocate URLs to the phase slot. The hashing results are not as balanced as the first one, as shown in Figure 10. This is because some web servers rather have much more URLs than the others.

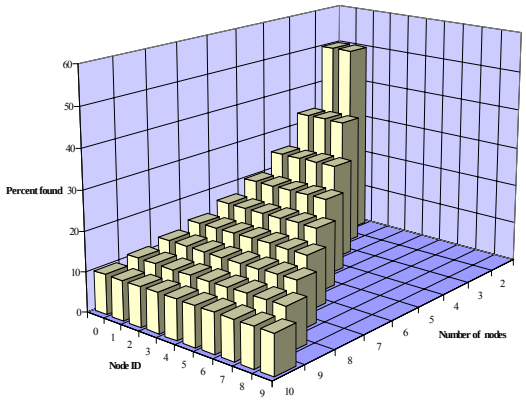

Figure 9. Distribution of URLs in each node

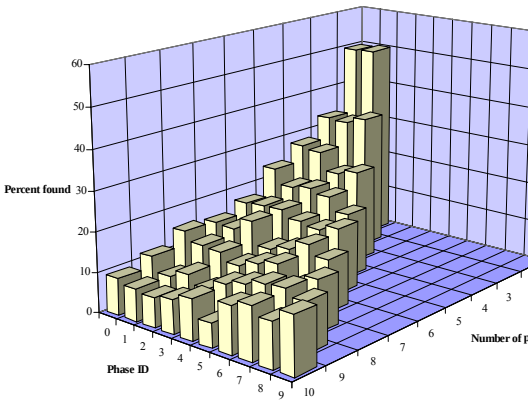

Figure 10. Distribution of URLs in each phase

5. Conclusions and Future Works

In this paper, we propose a method of data partition among the node of cluster running web spider. Both URL-hash and site-hash based partitions are utilized. The URL-hash based partition is used to distributed workload for the cluster. The site-hash based partition is to group the URLs belonged to same web servers together. The phase swapping accompanied by clock synchronization allows each node work on the different set of servers. This assures that only one node in the cluster is allowed to contact a web server at a time. Furthermore, we propose a scheduling technique to dispatch URLs to spider's threads using a simple but efficient URLs scrambling based on hashing. We implement the partitions and scheduling and show the experimental results. We have implemented several applications based on *KSPider*, such as web links analyzer, web classification, and web content filtering. Currently we are porting *KSpider* to the 64 bits machines.

6. References

[1] M, Koster. *Guidelines for Robot Writer*. 1993. http://www.robotstxt.org/wc/guidelines.html

[2] J. Cho and H. Garcia-Molina. Parallel Crawlers. *In Proc. of International World-Wide Web Conference*, 2002.

[3] P. Boldi, B. Codernotti, M. Santini, and S.Vigna. UbiCrawler : A Scalable Fully Distributed Web Crawler, *In Proc. of the 8th Australian World Wide Web Conference*, 2002.

[4] S. Chakrabarti, B. Dom, R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, David Gibson, and J. Kleinberg. Mining the Web's Link Structure. *IEEE Computer*, 32(8):60–67, 1999.

[5] V. Shkapenyuk and T.Suel. Design and Implementation of a High-Performance Distributed Web Crawler, *In Proc. of the 18th International Conference on Data Engineering (ICDE)*, 2002.

[6] T. Sterling., D. J. Becker, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. E. Packer. Beowulf: A Parallel Workstation for Scientific Computation, *In Proc. of International Conference on Parallel Processing*, 1995.

[7] D.L. Mills. Internet time synchronization: the Network Time Protocol, *IEEE Trans. Communications COM-39*, 1991.