Data-Parallel Web Crawling Models*

Berkant Barla Cambazoglu, Ata Turk, and Cevdet Aykanat

Department of Computer Engineering, Bilkent University
06800, Ankara, Turkey
{berkant,atat,aykanat}@cs.bilkent.edu.tr

Abstract. The need to quickly locate, gather, and store the vast amount of material in the Web necessitates parallel computing. In this paper, we propose two models, based on multi-constraint graph-partitioning, for efficient data-parallel Web crawling. The models aim to balance the amount of data downloaded and stored by each processor as well as balancing the number of page requests made by the processors. The models also minimize the total volume of communication during the link exchange between the processors. To evaluate the performance of the models, experimental results are presented on a sample Web repository containing around 915,000 pages.

1 Introduction

During the last decade, an exponential increase has been observed in the amount of the textual material in the Web. Locating, fetching, and caching this constantly evolving content, in general, is known as the crawling problem. Currently, crawling the whole Web by means of sequential computing systems is infeasible due to the need for vast amounts of storage and high download rates. Furthermore, the recent trend in construction of cost-effective PC clusters makes the Web crawling problem an appropriate target for parallel computing.

In Web crawling, starting from some seed pages, new pages are located using the hyperlinks within the already discovered pages. In parallel crawling, each processor is responsible from downloading a subset of the pages. The processors can be coordinated in three different ways: *independent*, *master-slave*, and *data-parallel*. In the first approach, each processor independently traverses a portion of the Web and downloads a set of pages pointed by the links it discovered. Since some pages are fetched multiple times, in this approach, there is an overlap problem, and hence, both storage space and network bandwidth are wasted. In the second approach, each processor sends its links, extracted from the pages it downloaded, to a central coordinator. This coordinator, then assigns the collected URLs to the crawling processors. The weakness of this approach is that the coordinating processor becomes a bottleneck.

Our focus, in this work, is on the third approach. In this approach, pages are partitioned among the processors such that each processor is responsible from

* This work is partially supported by The Scientific and Technical Research Council of Turkey (TÜBİTAK) under project EEEAG-103E028.

C. Aykanat et al. (Eds.): ISCIS 2004, LNCS 3280, pp. 801–809, 2004.

fetching a non-overlapping subset of the pages. Since some pages downloaded by a processor may have links to the pages in other processors, these inter-processor links need to be communicated in order to obtain the maximum page coverage and to prevent the overlap of downloaded pages. In this approach, each processor freely exchanges its inter-processor links with the others.

The page-to-processor assignment can be hierarchical or hash-based. The hierarchical approach assigns pages to processors according to the domain of URLs. This approach suffers from the imbalance in processor workloads since some domains contain more pages than the others. In the hash-based approach, either single pages or sites as a whole are assigned to the processors. This approach solves the load balancing problem implicitly. However, in this approach, there is a significant communication overhead since inter-processor links, which must be communicated, are not considered while creating the page-to-processor assignment.

The page-to-processor assignment problem has been addressed by a number of authors. Cho and Garcia-Molina [3] used the site-hash-based assignment technique with the belief that it will reduce the number of inter-processor links when compared to the page-hash-based assignment technique. Boldi et al. [2] applied the consistent hashing technique, a method assigning more than one hash values for a site, in order to handle the failures among the crawling processors. Teng et al. [7] used a hierarchical, bin-packing-based page-to-processor assignment approach. In this work, we propose two models based on multi-constraint graph partitioning for load-balanced and communication-efficient parallel crawling.

The rest of the paper is organized as follows. In Section 2, the proposed parallel crawling models are presented. In Section 3, we provide some implementation details about our parallel Web crawler. In Section 4, experimental results are presented. Finally, we conclude in Section 5.

2 Web Graph Partitioning

2.1 Graph Partitioning Problem

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ [1] is defined as a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. Every edge $e_{ij} \in \mathcal{E}$ connects a pair of distinct vertices v_i and v_j. Multiple weights $w_i^1, w_i^2, \ldots, w_i^M$ may be associated with a vertex $v_i \in \mathcal{V}$. A cost c_{ij} is assigned as the cost of an edge $e_{ij} \in \mathcal{E}$.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is said to be a K-way partition of $\mathcal{G}$ if each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, parts are pairwise disjoint, and the union of the K parts is equal to $\mathcal{V}$. A partition Π is said to be balanced if each part $\mathcal{V}_k$ satisfies the balance criteria

$$W_k^m \leq W_{\text{avg}}^m(1+\epsilon), \text{for } k=1,2,\ldots,K \text{ and } m=1,2,\ldots,M. \tag{1}$$

In Eq. 1, each weight W_k^m of a part $\mathcal{V}_k$ is defined as the sum of the weights w_i^m of the vertices in that part. W_{avg}^m is the weight that each part should have in the case of perfect load balancing. ϵ is the maximum imbalance ratio allowed.

In a partition Π of $\mathcal{G}$, an edge is said to be cut if its pair of vertices fall into two different parts and uncut otherwise. The cutsize definition for representing the cost $\chi(\Pi)$ of a partition Π is

$$\chi(\Pi) = \sum_{e_{ij} \in \mathcal{E}} c_{ij} \ . \tag{2}$$

After these definitions, the K-way, multi-constraint graph partitioning problem [5,6] can be stated as the problem of dividing a graph into two or more parts such that the cutsize is minimized (Eq. 2) while the balance criteria (Eq. 1) on the part weights is maintained. This problem is known to be NP-hard.

2.2 Page-Based Partitioning Model

A major assumption in our models is that the crawling system runs in sessions. Within a session, if a page is downloaded, it is not downloaded again, that is, each page can be downloaded just once in a session. The crawling system, after downloading enough number of pages, decides to start another download session and recrawls the Web. For efficient crawling, our models utilize the information (i.e., the Web graph) obtained in the previous crawling session and provide a better page-to-processor mapping for the following crawling session. We assume that between two consecutive sessions, there are no drastic changes in the Web graph (in terms of page sizes and the topology of the links).

We describe our parallel crawling models on the sample Web graph displayed in Figure 1. In this graph, which is assumed to be created in the previous crawling session, there are 7 sites. Each site contains several pages, which are represented by small squares. The directed lines between the squares represent the hyperlinks between the pages. There may be multi-links (e.g., (i_1, i_3)) and bidirectional links between the pages (e.g., (g_5, g_6)). In the figure, inter-site links are displayed as dashed lines. For simplicity, unit page sizes and URL lengths are assumed.

In our page-based partitioning model, we represent the link structure between the pages by a page graph $\mathcal{G}^{\mathrm{p}} = (\mathcal{V}^{\mathrm{p}}, \mathcal{E}^{\mathrm{p}})$. In this representation, each page p_i corresponds to a vertex v_i. There exists an undirected edge e_{ij} between vertices v_i and v_j if and only if page p_i has a link to page p_j or vice versa. Multi-links between the pages are collapsed into a single edge. Two weights w_i^1 and w_i^2 are associated with each vertex v_i. The weight w_i^1 of vertex v_i is equal to the size (in bytes) of page p_i, and represents the download and storage overhead for p_i. The weight w_i^2 of vertex v_i is equal to 1, and represents the overhead for requesting p_i. The cost c_{ij} of an edge $e_{ij} \in \mathcal{E}^{\mathrm{p}}$ is equal to the total string length of the links (p_i, p_j) and (p_j, p_i) (if any) between pages p_i and p_j. This cost corresponds to the volume of communication performed for exchanging the links between pages p_i and p_j in case p_i and p_j are mapped to different processors.

In a K-way partition $\Pi^{\mathrm{p}} = (\mathcal{V}_1^{\mathrm{p}}, \mathcal{V}_2^{\mathrm{p}}, \ldots, \mathcal{V}_K^{\mathrm{p}})$ of the page graph $\mathcal{G}^{\mathrm{p}}$, each vertex part $\mathcal{V}_k^{\mathrm{p}}$ corresponds to a subset $\mathcal{P}_k$ of pages to be downloaded by processor P_k. That is, every page $p_i \in \mathcal{P}_k$, represented by a vertex $v_i \in \mathcal{V}_k^{\mathrm{p}}$, is fetched and stored by processor P_k. In this model, maintaining the balance on part weights W_k^1 and

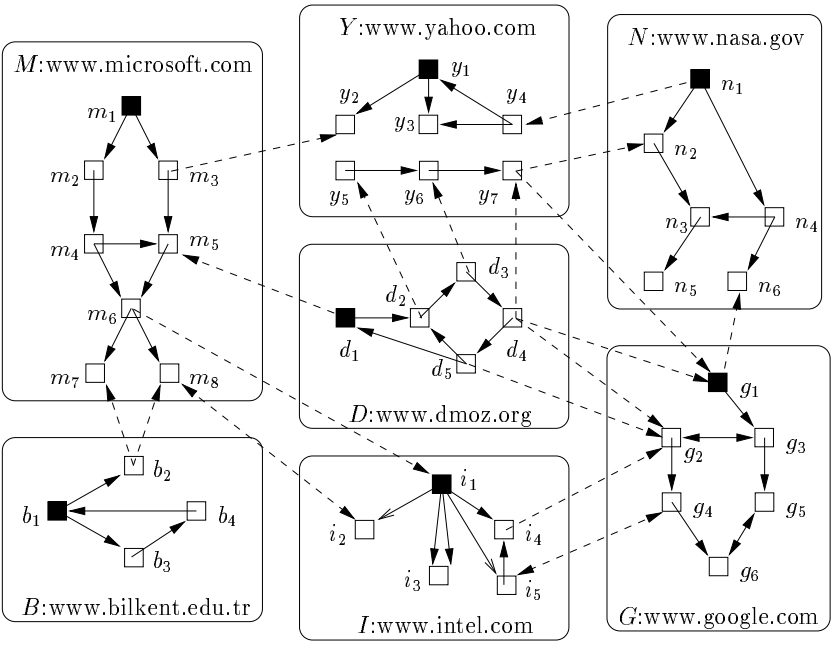

Fig. 1. An example to the graph structure in the Web

W_k^2 (Eq. 1) in partitioning the page graph $\mathcal{G}^{\mathrm{p}}$, effectively balances the download and storage overhead of processors as well as the number of page download requests issued by processors. Minimizing the cost $\chi(\Pi^{\mathrm{p}})$ (Eq. 2) corresponds to minimizing the total volume of inter-processor communication that will occur during the link exchange between processors.

Figure 2 shows a 3-way partition for the page graph corresponding to the sample Web graph in Figure 1. For simplicity, unit edge costs are not displayed. In this example, almost perfect load balance is obtained since weights (for both weight constraints) of the three vertex parts $\mathcal{V}_1^{\mathrm{p}}$, $\mathcal{V}_2^{\mathrm{p}}$, and $\mathcal{V}_3^{\mathrm{p}}$ are respectively 14, 13, and 14. Hence, according to this partitioning, each processor P_k, which is responsible from downloading all pages $p_i \in \mathcal{P}_k^{\mathrm{p}}$, is expected to fetch and store almost equal amounts of data in the next crawling session. In Figure 2, dotted lines represent the cut edges. These edges correspond to inter-processor links, which must be communicated. In our example, $\chi(\Pi^{\mathrm{p}}) = 8$, and hence, the total volume of link information that must be communicated is 8.

2.3 Site-Based Partitioning Model

Due to the enormous size of the Web, the constructed page graph may be huge, and hence it may be quite costly to partition it. For efficiency purposes, we also propose a site-based partitioning model, which considers sites instead of pages as the atomic tasks for assignment. We represent the link structure between

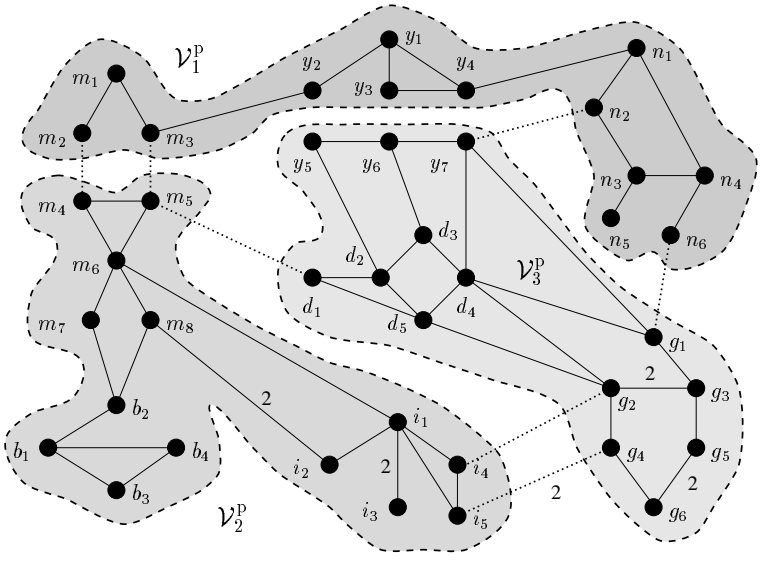

Fig. 2. A 3-way partition for the page graph of the sample Web graph in Figure 1

the pages by a site graph $\mathcal{G}^{\mathrm{S}} = (\mathcal{V}^{\mathrm{S}}, \mathcal{E}^{\mathrm{S}})$. All pages belonging to a site S_i are represented by a single vertex $v_i \in \mathcal{V}^{\mathrm{S}}$. The weights w_i^1 and w_i^2 of each vertex v_i are respectively equal to the total size of the pages (in bytes) and the number of pages hosted by site S_i. There is an edge e_{ij} between two vertices v_i and v_j if and only if there is at least one link between any pages $p_x \in S_i$ and $p_y \in S_j$. The cost c_{ij} of an edge $e_{ij} \in \mathcal{E}^{\mathrm{S}}$ is equal to the total string length of all links (p_x, p_y) and (p_y, p_x) between each pair of pages $p_x \in S_i$ and $p_y \in S_j$. All intra-site links, i.e., the links between the pages belonging to the same site, are ignored.

In a K-way partition $\Pi^{\mathrm{S}} = (\mathcal{V}_1^{\mathrm{S}}, \mathcal{V}_2^{\mathrm{S}}, \ldots, \mathcal{V}_K^{\mathrm{S}})$ of graph $\mathcal{G}^{\mathrm{S}}$, each vertex part $\mathcal{V}_k^{\mathrm{S}}$ corresponds to a subset $\mathcal{S}_k$ of sites whose pages are to be downloaded by processor P_k. Balancing the part weights (Eq. 1) and minimizing the cost (Eq. 2) has the same effects with those in the page-based model.

Figure 3 shows a 2-way partition for the site graph corresponding to the sample Web graph in Figure 1. Vertex weights are displayed inside the circles, which represent the sites. Part weights are $W_1^1 = W_1^2 = 17$ and $W_2^1 = W_2^2 = 24$ for the two parts $\mathcal{V}_1^{\mathrm{S}}$ and $\mathcal{V}_2^{\mathrm{S}}$, respectively. The cut edges are displayed as dotted lines. The cut cost is $\chi(\Pi^{\mathrm{p}}) = 1+1+3 = 5$. Hence, according to this partitioning, the total volume of communication for the next crawling session is expected to be 5.

3 Implementation Details

We developed a data-parallel crawling system, which utilizes the proposed models. The crawler is implemented in C using MPI libraries for message passing.

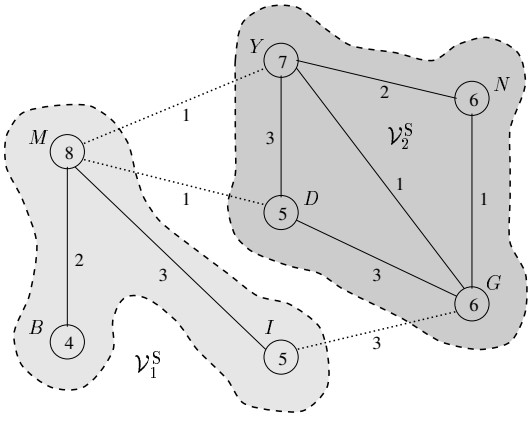

Fig. 3. A 2-way partition for the site graph of the sample Web graph in Figure 1

In our parallel crawling system, each crawling processor uses synchronous I/O and concurrently crawls the set of pages it is responsible from. Processors run several threads to fetch data from multiple servers simultaneously. Threads perform three different tasks: domain name resolution, page download, and URL extraction from downloaded pages. Intermediate data such as downloaded URLs, page-content hashes, and resolved DNS entries, which must be stored in memory, are kept in dynamic trie data structures. FIFO queues are used for coordinating the data flow between threads of a processor.

As explained in Section 2, the page-to-processor assignment is determined using page- or site-based partitioning models prior to the crawling process. The state-of-the-art graph partitioning tool MeTiS [4] is used for partitioning the constructed page and site graphs. The resulting part vectors are replicated at each processor. Whenever a URL which has not been crawled yet is discovered, the processor responsible from the URL is located by the part vector. If a newly found URL which is not listed in the part vector is discovered, the discovering processor becomes responsible from crawling that URL.

4 Experimental Results

In the experiments, the multi-constraint, multi-level k-way partitioning algorithm of MeTiS is used. The imbalance tolerance is set to 5% for both weight constraints. Due to the randomized nature of the algorithms, experiments are repeated 8 times, and the average values are reported. Results are provided for load imbalance values in storage and page request amounts of processors as well as the total volume of inter-processor communication in link exchange. We compared the proposed models with the hash-based assignment techniques. Experiments are conducted on the K values 8, 16, 24, 32, 40, 48, 56, and 64. As the test dataset, the sample (8 GB) Web collection provided by Google Inc. [8]

Table 1. The load imbalance values in storage amounts of processors

K	Page-based		Site-based	
	GP-based	Hash-based	GP-based	Hash-based
8	3.31	1.04	4.59	15.32
16	4.02	1.73	4.72	22.94
24	4.44	1.88	4.74	30.15
32	4.66	2.53	4.75	36.61
40	4.68	2.64	4.76	41.22
48	4.69	2.93	4.76	44.59
56	4.76	3.70	4.76	52.73
64	4.76	3.84	4.76	54.19

is used. This collection contains 913,570 pages and 15,820 sites. Average vertex degrees are respectively 4.9 and 10.5 for the page and site graphs created.

Table 1 displays the load imbalance values observed in storage amounts of processor for the graph-partitioning-based (GP-based) and hash-based techniques. Table 2 shows the imbalance values observed for the number of page download requests issued by processors. Experiments on page-based assignment show that the hash-based approach performs slightly better than our GP-based model in balancing both the storage overhead and the number of page download requests. In site-based assignment, the GP-based model outperforms the hash-based approach, whose imbalance rates deteriorate with increasing K values. This is basically due to the high variation in the sizes of the sites in the dataset used. Since solution space is more restricted in the site graph, the site-based GP model produces slightly inferior load imbalance rates compared to the page-based GP model.

Table 3 presents the total volume of link information that must be communicated among the processors for different techniques. As expected, an increasing

Table 2. The load imbalance values in page requests made by processors

K	Page-based		Site-based	
	GP-based	Hash-based	GP-based	Hash-based
8	3.44	0.78	4.57	3.84
16	4.01	1.15	4.75	6.17
24	4.50	1.40	4.75	7.03
32	4.68	1.56	4.75	8.89
40	4.74	1.66	4.76	10.78
48	4.74	1.95	4.76	12.16
56	4.76	2.09	4.76	13.49
64	4.76	2.34	4.76	13.73

Table 3. The total volume of communication (in bytes) during the link exchange

	Page-based		Site-based	
K	GP-based	Hash-based	GP-based	Hash-based
8	64,489	5,306,461	131,129	1,001,602
16	69,643	5,685,823	151,264	1,070,871
24	71,385	5,811,600	163,454	1,095,860
32	75,285	5,875,123	180,751	1,108,891
40	82,038	5,912,733	187,987	1,114,957
48	80,038	5,938,494	193,011	1,117,739
56	92,947	5,956,059	213,642	1,122,955
64	92,467	5,969,393	234,272	1,126,470

trend is observed in communication volumes as K increases. Site-based hashing results in around 5 times less communication than page-based hashing. This is due to the fact that many inter-processor links are eliminated since sites act as clusters of pages, and almost 4 out of 5 page links turn out to be an intra-processor link when site-based hashing is employed.

According to Table 3, the proposed GP-based models perform much better in minimizing the total communication volume. However, in contrast to the hash-based techniques, the site-based GP model causes an increase in the communication volume. This can be explained by the sparsity of our dataset and the simpler (relative to the page graph) site graph topology which causes a reduction in the solution space. Due to the sparsity of our dataset, there are many pages which do not link each other although they are associated with the same site. By working on the coarser site graph, the MeTiS graph partitioning tool fails to utilize the good edge cuts that cross across the sites (e.g., in Figure 2, pages y_1, y_2, y_3, and y_4 are mapped to $\mathcal{V}_1^{\mathrm{p}}$ while y_5, y_6, and y_7 are mapped to $\mathcal{V}_3^{\mathrm{p}}$). Consequently, the site-based GP model results in partitions with higher cut costs and hence communication volumes.

5 Conclusion

In this paper, we presented two models, based on multi-constraint graph partitioning, for data-parallel Web crawling. Compared to the hash-based assignment techniques, the proposed models produced similar load imbalance values for storage overheads and page download requests of processors while producing superior results in minimizing the total volume of communication in inter-processor link exchange. Currently, we are about to start a large crawl of the Web using the developed parallel Web crawler on a 48-node PC cluster. This will allow us to repeat the experiments presented in this paper on a larger collection of pages and verify the validity of our theoretical results in practice.

References

1. Berge, C.: Graphs and hypergraphs. North-Holland Publishing Company (1973)
2. Boldi, P., Codenotti, B., Santini, M., Vigna, S.: Ubicrawler: A scalable fully distributed Web crawler. In: Proceedings of AusWeb02, the Eighth Australian World Wide Web Conference. (2002)
3. Cho J., Garcia-Molina, H.: Parallel crawlers. In: Proceedings of the 11th World Wide Web conference (WWW11), Honolulu, Hawaii. (2002) 124–135
4. Karypis, G., Kumar, V.: MeTiS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices. Technical Report, University of Minnesota (1998)
5. Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing **48**(1) (1998) 96–129
6. Schloegel, K., Karypis, G., Kumar, V.: Parallel multilevel algorithms for multi-constraint graph partitioning. In: Proceedings of the 6th International Euro-Par Conference on Parallel Processing. (2000) 296–310
7. Teng, S., Lu, Q., Eichstaedt, M., Ford, D., Lehman, T.: Collaborative Web crawling: Information gathering/processing over Internet. In: 32nd Hawaii International Conference on System Sciences. (1999)
8. `http://www.google.com/`