An Efficient Partition-Based Parallel PageRank Algorithm

Bundit Manaskasemsak and Arnon Rungsawang
Massive Information & Knowledge Engineering
Department of Computer Engineering, Faculty of Engineering
Kasetsart University, Bangkok 10900, Thailand.
{un, arnon}@mikelab.net

Abstract

PageRank becomes the most well-known re-ranking technique of the search results. By its iterative computational nature, the computation takes much computing time and resource. Researchers have then devoted much attention in studying an efficient way to compute the PageRank scores of a very large web graph. However, only a few of them focus on large-scale PageRank computation using parallel processing techniques. In this paper, we propose a Partition-based parallel PageRank algorithm that can efficiently run on a low-cost parallel environment like the PC cluster. For comparison, we also study the other two known techniques, as well as propose an analytical discussion concerning I/O and synchronization cost, and memory usage. Experimental results with two web graphs synthesized from the .TH domain and the Stanford WebBase project are very promising.

1. Introduction

The unceasing growth of the World Wide Web nowadays provides much burden to search engine builders to engineer new ranking techniques to return better final search results. The PageRank approach to ranking webpages has been one of the most successful algorithms ever known since its announcement by the Google™ [3]. PageRank is both a compute-intensive algorithm and a computing resource hunger. With billions of existing webpages, despite computing the subset of PageRank scores would still take days. Moreover, the webpages continually updated, added, or removed, the frequent re-computation of the PageRank scores is often necessary to maintain the relevance of the search results. Also, in the context of topic sensitive or personalized web search [6, 8], the large number of PageRank scores needs to be recomputed to reflect the user preferences. Inevitably, the efficient PageRank computation is required.

Much recent research focuses on algebraic techniques. For example, Arasu et al. [1] exploit web's bow-tie structure and use Gauss-Seidel algorithm in computation. Kamvar et al. [10] assume web graph's local block structures of many intra-domain links, and compute those structures separately before recombining results to yield the global ranks. They also avoid re-computing the sub-component of the already fixed PageRank scores [9], and accelerate the computation by periodically subtracts of estimates of the non-principal eigenvectors from the current iteration [11]. Haveliwala and Kamvar [7] improve the convergence rate by examining the second eigenvalue. Lee et al. [12] collapse all dangling nodes in the web graph into one block and non-dangling nodes into another block, compute the PageRank scores separately and concatenate them to get the final result.

In addition, there are other extensive studies in terms of the system architectures. For example, Boldi and Vigna [2] first compress the large web graph structure and compute the PageRank scores in main memory. Both Chen et al. [4] and Haveliwala [5] efficiently compute the PageRank scores by reducing the I/O sequences of scan operations of the web graph on the external memory. Sankaralingam et al. [14] exploit the distributed P2P architecture to speed-up and enable the incremental computation, while Rungsawang and Manaskasemsak [13] also suggest a solution on the PC cluster.

In this paper, we propose an efficient PageRank algorithm on a low-cost parallel environment like the PC cluster. We also study the algorithms proposed by Chen et al. [4] and Haveliwala [5] for comparison, and adapt both of them to run on the PC cluster. Moreover, we provide an analytical discussion of the performance in terms of I/O and synchronization cost, and memory usage. During experiments, we test the three algorithms with two web data sets; one is synthesized from our own crawler within the .TH domain, and another is from the Stanford WebBase project [16], using a subset of the F32 cluster of the AIST (Japan) under the ApGrid [15]. The results are clearly promising.

Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems (ICPADS'05)
0-7695-2281-5/05 $20.00 © 2005 **IEEE**
IEEE COMPUTER SOCIETY

We organize our paper as follows. Sections 2 and 3 introduce basic background and the proposed PageRank algorithm. Section 4 mentions the two related algorithms. Section 5 describes the experiments, discusses the results, and an analytical study. Section 6 concludes the paper.

2. Basic PageRank background

The concept of PageRank inspires from human behavior of voting. Pages recursively voted or referenced by authors via hyperlinks constitute a gigantic web graph of the Internet. Thus pages mostly voted by many authors, i.e., pointed by many other pages, will have high voting scores and are considered to be the most interesting pages to be ranked at the beginning list of the search results.

Following the above intuitive concept, we can formulate the basic algorithm of PageRank as follows. Let T be the total number of pages in the web graph, $Rank(u)$ represent the rank score of a page u, N_u be the number of pages which page u points out (called later "out-degree"), and S_v represent the set of pages pointing to page v. If a page v has many other pages u pointing to, then the rank score of v can recursively be computed by:

$$\forall_v Rank_{i+1}(v) = \frac{(1-\alpha)}{T} + \alpha \sum_{u \in S_v} \frac{Rank_i(u)}{N_u} \qquad (1)$$

Here α, called "damping factor", is the transitional probability of the random surfer model [3].

To compute the PageRank scores, the input web graph must initially be converted into a binary link structure file L [13] as illustrated textually in Figure 1. Each number represents the web URL.

src_id (4 bytes)	out_degree (4bytes)	dest_id (4 bytes each)
1	4	9 102 256 324
2	5	3 178 203 278 345
3	5	5 10 196 313 335
4	3	2 285 299

Figure 1. The binary link structure file L.

3. Partition-based parallel PageRank algorithm

To accelerate the PageRank computation of a large web graph, we equally partition the large binary link structure file L by source URL number (i.e., *src_id*) into β files, named later by $\tilde{L}_i$; $0 \le i < \beta$. Each file will be allotted to compute within a PC processor. Figure 2(a) exemplifies textually the first three $\tilde{L}_i$. Let T be the number of pages in the underlying web graph. At each processor, we need to allocate an array of floating point V_i in main memory, having T/β entries, to represent the portion of the source rank vector of its assigned source URLs; and to create a file P_i (called later the "packet"), to store pairs of destination URLs and their corresponding rank scores, to represent the destination rank vector.

Figure 3(a) depicts the parallel PageRank computation during an iteration using four processors. The P_i, $\tilde{L}_i$ and V_i correspond to the packet file, the partitioned binary link structure file, and the portion of the source rank vector, residing at processor i, respectively.

During each iteration, a new packet P_i at each processor has been created by first scanning the corresponding partition $\tilde{L}_i$ from disk, and then computing the new destination rank scores with the current portion of the source rank vectors V_i in main memory. Consecutively, the synchronization of all rank scores occurs, i.e., new computed destination rank scores in P_i which correspond to the source URLs residing in the other processors have been read from disk and transmitted to destination processors. Algorithm 1 as follows concludes the underlying technique at one processor in the PC cluster. To compare with the other two algorithms discussed in the next section, we let all algorithms iterate for 50 loops during the experiments. We hereafter name the proposed algorithm, the "Partition-based parallel PageRank" algorithm.

I/O cost: During an iteration, each processor has to read the partitioned file $\tilde{L}_i$, compute the new destination rank scores and write a new created packet P_i back to disk, and then re-read the packet P_i from disk during synchronization. The total I/O operation cost will be:

$$C_{I/O,Partition} = \sum_{0 \le i < \beta} \left(\left|\tilde{L}_i\right| + 2 \cdot \left|P_i\right| \right) = |L| + 2 \cdot |P| \qquad (2)$$

Here, the summation of all β portions of $\tilde{L}_i$ is equal to the size of the binary link structure file L itself, and we let the summation of all packet files P_i be P.

Memory usage: Each processor has to allocate a fix size of memory to fit the portion of the source rank vector V_i. Therefore, the total memory usage will be:

$$C_{Mem,Partition} = \sum_{0 \le i < \beta} |V_i| = |V| \qquad (3)$$

Since we partition the source rank vector into β portions, the summation of all portions V_i will be equal to the size of the source rank vector V itself.

Synchronization cost: After the new destination rank scores have been computed during the iteration, the synchronization process has to be done. Since an entry of a packet P_i is two times larger than an entry of V_i, the total synchronization cost among processors will be:

$$C_{Syn,Partition} = \sum_{0 \le i < \beta} 2 \cdot \left(\frac{|P_i|}{2} - |V_i| \right) = |P| - 2 \cdot |V| \qquad (4)$$

0-7695-2281-5/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

Algorithm 1. Partition-based parallel PageRank algorithm.

$\forall_u V_i[u] = \frac{1}{T}$
for *round* = 1 ... 50 {
  create the new packet file P_i
  while ($\tilde{L}_i$ is not end of file) {
    scanLink ($\tilde{L}_i.src_id$, $\tilde{L}_i.out_degree$, $\tilde{L}_i.dest_id_1$,
      $\tilde{L}_i.dest_id_2$, ..., $\tilde{L}_i.dest_id_{\tilde{L}_i.out_degree}$)
    for j = 1 ... $\tilde{L}_i.out_degree$
      writePacket (*pair of* ($\tilde{L}_i.dest_id_j$, $\frac{V[\tilde{L}_i.src_id]}{\tilde{L}_i.out_degree}$))
  }
  while (P_i is not end of file) {
    scanPacket (*pair of* (*id*, *score*))
    if (*id* is in assigned number) then
      $V_i[id] = V_i[id] + score$
    else
      synchronize (*pair of* (*id*, *score*))
  }
  $\forall_v V_i[v] = \frac{(1-\alpha)}{T} + (\alpha \times V_i[v])$
}

4. Related works

Since both Block-based [5] and Split-Accumulate algorithms [4] are based on the same PageRank computational family called the "Power method" like ours, we study them in detail for comparison. However, these two algorithms utilize different format of the binary link structure files.

4.1. Block-based algorithm

To adapt the Block-based technique to run on the PC cluster, we here propose to equally partition the link structure file L by destination URL number (*dest_id*) into β files, named later by $\hat{L}_i$; $0 \le i < \beta$. Figure 2(b) illustrates textually the example of the first three $\hat{L}_i$. Note that each record has an additional 4-byte integer for filed "num" to represent the number of destination URLs that the source URL in that partition points to.

During the computation, each processor has to allocate an array of floating point V_i' in main memory, having T/β entries, to keep the portion of the destination rank scores; and to create a source rank vector file V to store the scores of all source URLs. Figure 3(b) depicts the Block-based PageRank computation using four processors.

I/O cost: During an iteration, each processor compute the new destination rank scores V_i' by scanning the partitioned file $\hat{L}_i$ and the source rank vector V from disk, and write all portion of V_i' back to disk after synchronization. Thus, the total I/O operation cost will be:

$$C_{I/O,Block} = \sum_{0\le i<\beta} \left|\hat{L}_i\right| + \beta \cdot |V| + \sum_{0\le i<\beta} \beta \cdot |V_i'| = (1+\varepsilon+\varepsilon') \cdot |L| + 2\beta \cdot |V| \quad (5)$$

Here we write the size of $\hat{L}$ in terms of $(1+\varepsilon+\varepsilon')$ times the size of L, where ε and ε' represent the additional space needed for the field "num" and the redundant *src_id* residing in the other portions of $\hat{L}_i$. Normally, ε and ε' are ≈0.1 [5].

Memory usage: Each processor only allocates an array V_i' for the new computed destination rank scores. Thus, the total memory usage will be:

$$C_{Mem,Block} = \sum_{0\le i<\beta} |V_i'| = |V'| = |V| \quad (6)$$

Synchronization cost: During the synchronization process, the new computed destination rank scores V_i' at each processor have to be sent to the others. Therefore, the synchronization cost among processors will be:

$$C_{Syn,Block} = \sum_{0\le i<\beta} (\beta-1) \cdot |V_i'| = (\beta-1) \cdot |V| \quad (7)$$

4.2. Split-Accumulate algorithm

To reduce I/O cost from scanning a large size of the source rank vector V from disk in the Block-based algorithm, Chen et al. [4] first create the reversed binary link structure file. To adapt the Spit-Accumulate technique, we here then equally partition that reversed binary link structure file by source URL number into β files, named later by $\overline{L}_i$; $0 \le i < \beta$. Figure 2(c) illustrates textually the example of the first three $\overline{L}_i$. However, the creation of the reversed binary link structure file for a large web graph would take days of CPU time.

During the computation, each processor has to allocate an array of floating point V_i in main memory, having T/β entries, to hold the portion of source rank scores, and an array of another 4-byte integer O_i to hold the number of out-degree of every source URL corresponding to V_i; and to create a packet file P_i to represent the new computed destination rank scores. Figure 3(c) depicts the Split-Accumulate PageRank computation using four processors.

I/O cost: During an iteration, each processor has to scan the partitioned file $\overline{L}_i$, compute the new destination rank scores and write a new packet P_i back to disk, and then re-read that packet file during the synchronization

IEEE COMPUTER SOCIETY

src_id	out_degree	dest_id
1	4	9 102 256 324
2	5	3 178 203 278 345
3	5	5 10 196 313 335

File $\tilde{L}_0$ ($1 \le$ src_id ≤ 100)

src_id	out_degree	dest_id
101	4	1 65 102 109
102	3	13 109 256
103	1	5

File $\tilde{L}_1$ ($101 \le$ src_id ≤ 200)

src_id	out_degree	dest_id
201	1	256
202	3	2 5 278
203	3	193 235 256

File $\tilde{L}_2$ ($201 \le$ src_id ≤ 300)

(a) Partition-based

src_id	out_degree	num	dest_id
1	4	1	9
2	5	1	3
3	5	2	5 10

File $\hat{L}_0$ ($1 \le$ dest_id ≤ 100)

src_id	out_degree	num	dest_id
1	4	1	102
2	5	1	178
3	5	1	196

File $\hat{L}_1$ ($101 \le$ dest_id ≤ 200)

src_id	out_degree	num	dest_id
1	4	1	256
2	5	2	203 278
4	3	2	285 299

File $\hat{L}_2$ ($201 \le$ dest_id ≤ 300)

(b) Block-based

dest_id	in_degree	src_id
1	2	11 12
2	3	4 8 13
3	1	2

File $\bar{L}_0$ ($1 \le$ src_id ≤ 100)

dest_id	in_degree	src_id
1	2	101 163
3	3	133 156 178
5	2	103 136

File $\bar{L}_1$ ($101 \le$ src_id ≤ 200)

dest_id	in_degree	src_id
2	2	202 223
4	3	204 245 288
5	3	202 223 224

File $\bar{L}_2$ ($201 \le$ src_id ≤ 300)

(c) Split-Accumulate

Figure 2. The partitioned binary link structure files $\tilde{L}_i$, $\hat{L}_i$, and $\bar{L}_i$.

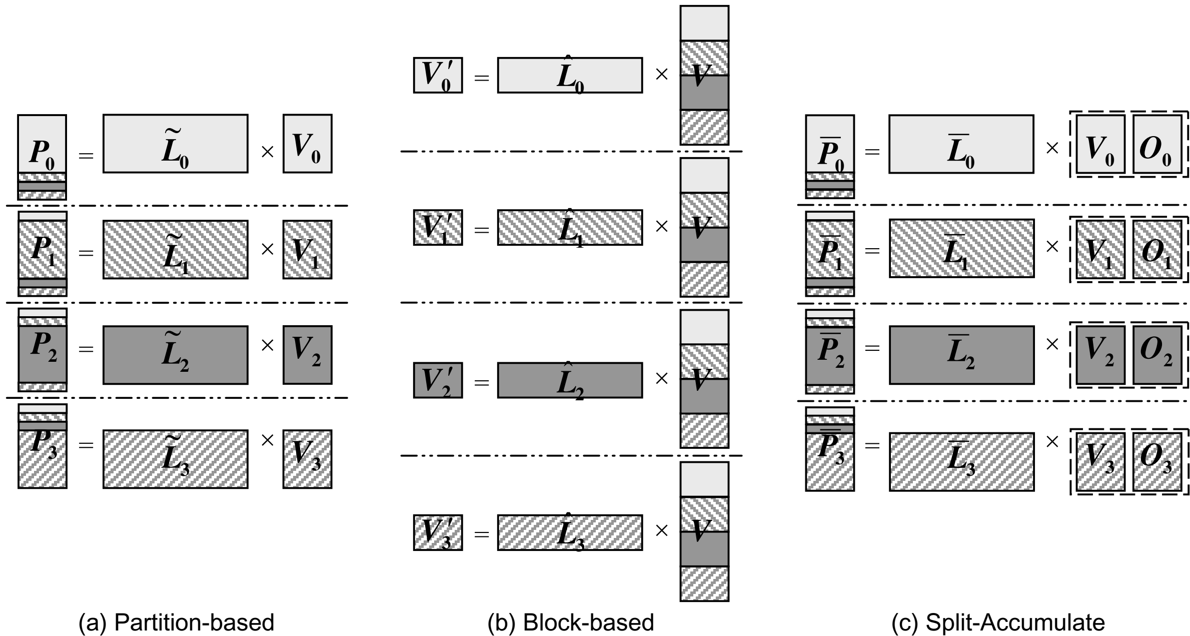

Figure 3. Parallel PageRank computation using 4 processors.

process. Thus the total I/O operation cost will be:

$$C_{I/O,Split} = \sum_{0 \le i < \beta} \left(\left| \bar{L}_i \right| + 2 \cdot \left| P_i \right| \right) = \left(1 + \varepsilon' \right) \cdot \left| L \right| + 2 \cdot \left| P \right| \quad (8)$$

Here, we also write the size of $\bar{L}$ in term of the binary link structure file L for comparison. The ε' here represents the additional space needed for the redundant *dest_id* residing in the other partitions $\bar{L}_i$, as mentioned.

Memory usage: For the Split-Accumulate algorithm, the total memory usage will be:

$$C_{Mem,Split} = \sum_{0 \le i < \beta} \left(\left| V_i \right| + \left| O_i \right| \right) = \left| V \right| + \left| O \right| = 2 \cdot \left| V \right| \quad (9)$$

Synchronization cost: During the synchronization process, the portion of the new computed rank scores in packet file P_i has to be sent to the other processors. Thus, the total synchronization cost among processors will be:

$$C_{Syn,Split} = \sum_{0 \le i < \beta} 2 \cdot \left(\frac{\left| P_i \right|}{2} - \left| V_i \right| \right) = \left| P \right| - 2 \cdot \left| V \right| \quad (10)$$

0-7695-2281-5/05 $20.00 © 2005 **IEEE**

IEEE COMPUTER SOCIETY

5. Experimental results and discussion

5.1. Experimental setup

Machine setup: We ran our experiments on sixteen PC machines, a subset of the F32 cluster running at the AIST (Japan) under the ApGrid contract [15]. Each machine is equipped with double 3.0 GHz Intel Xeon CPUs, 4GB of main memory, and a SCSI hard disk. All machines run the Linux RedHat 8.0, and are networked via the Gigabit Ethernet. All three algorithms were implemented in C language using the standard MPICH message passing version 1.2.5.

Data setup: We tested with two sets of web graphs. The first one (TH-DB), crawled in January 2003 within the .TH domain, contains around 10.9 million pages, 97 million hyperlinks. The second one (SF-DB), synthesized from the Stanford WebBase repository [16], consists of 28 million pages, 227 million hyperlinks. During experiments, all of these data have been converted into corresponding binary link structure files $\tilde{L}_i$, $\hat{L}_i$, and $\bar{L}_i$.

5.2. Experimental results

We divide the study of performance evaluation into four parts: (i) the pre-processing time needed to build the binary link structure files, (ii) the I/O cost and memory usage spent during the computation, (iii) the size of packets transferred among processors during synchronization, and (iv) the average running time needed per iteration.

Pre-processing time: Figure 4 concludes the total time needed to prepare the binary link structure files.

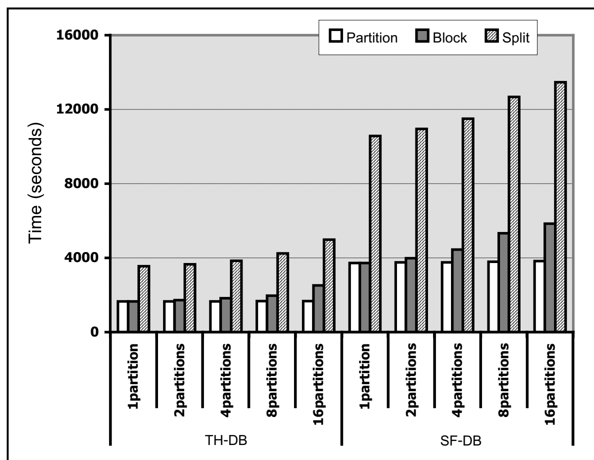

Figure 4. Time needed to prepare the binary link structure files.

The binary link structure files of the Partition-based and Block-based algorithms can directly be built from the input web graph database, and their construction spends almost the same time when the number of processors used during the computation is small. On the other hand, the time needed to build the binary link structure files for the Split-Accumulate algorithm is much longer, since the data in the input web graph needs to be first reversed.

I/O cost and memory usage: We ran all three algorithms using 1, 2, 4, 8 and 16 processors. Figure 5 concludes the total I/O operation cost per iteration. With more processors, the I/O cost of the Block-based algorithm increases very fast, while the I/O cost of the Split-Accumulate algorithm increases slightly faster than that of the Partition-based algorithm. We can see that these results follow the analysis previously mentioned in Equation (2), (5), and (8).

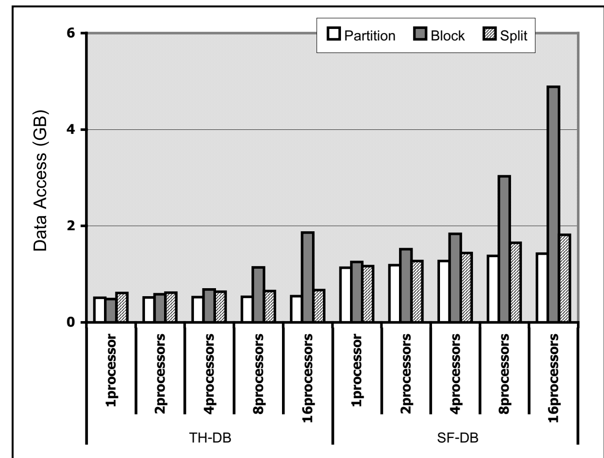

Figure 5. I/O cost per iteration.

Figure 6 reports the total memory usage during computation. The memory needed for the Split-Accumulate algorithm is around two times larger than those of the other two algorithms. These results also follow the analysis previously mentioned in Equation (3), (6), and (9).

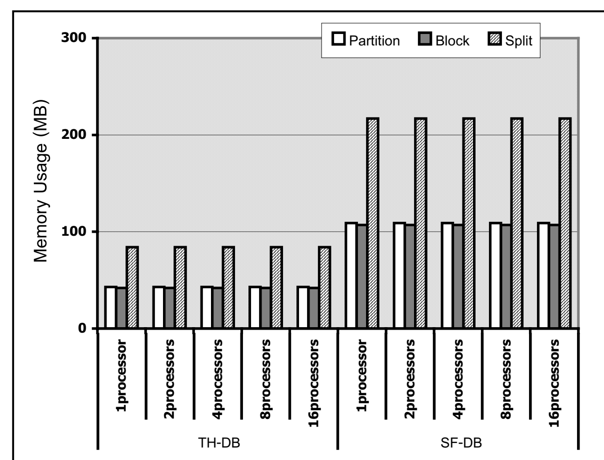

Figure 6. Memory usage during the computation.

0-7695-2281-5/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

Synchronization packets: Figure 7 concludes the amount of transferred data between processors during the synchronization.

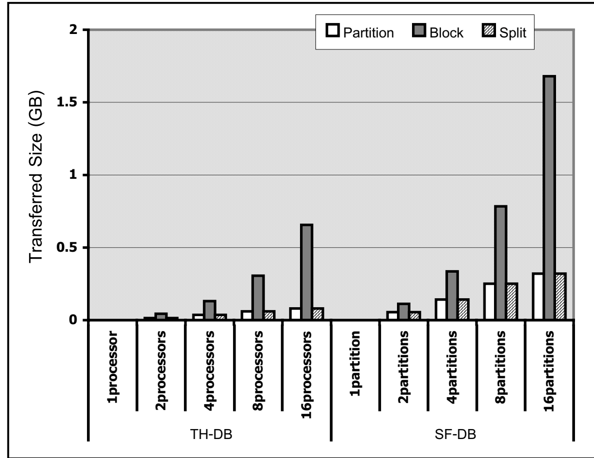

Figure 7. Amount of data transferred between processors during the synchronization.

The results show that the number of processors used for computation has much more effect on the amount of data transferred between processors for the Block-based algorithm than that of the other two algorithms. These results also follow the analysis mentioned in Equation (4), (7) and (10).

Average running time: Figure 8 concludes the average running time per iteration. This running time includes the time needed for both rank score computation and synchronization. When the number of processors used for computation is increased, the total running time is decreased. However, when the amount of processors increases to a certain number, the total running time stops decreasing but begins to increase. This degraded effect comes from that fact that all the three algorithms will spend most of their time to synchronize the rank scores between processors.

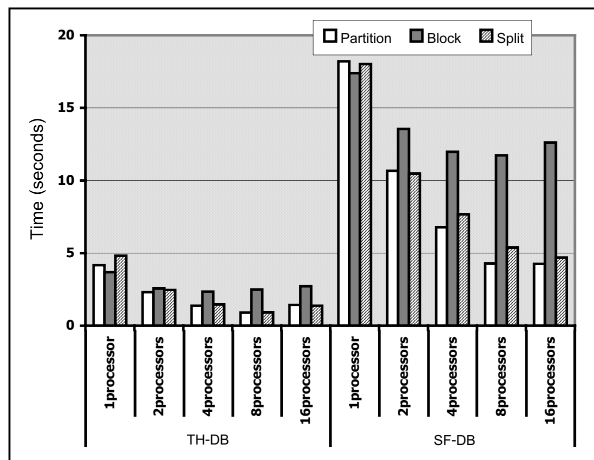

Figure 8. Average running time per iteration.

5.3. Analytical discussion

From the experimental results, we can clearly see that the Partition-based and the Split-Accumulate algorithms have better promising future in parallel computation than the Block-based one, when considering I/O and synchronization cost, and the overall running time. In addition, our Partition-based algorithm spends less computational cost than the Split-Accumulate algorithm, since it is unnecessary to perform a reversed web graph to build the binary link structure file during the pre-processing step.

For an ideal case, suppose that an input web graph can be partitioned into β files, and each partitioned web graph has a property called "Equally Dense and Strongly Connected Cluster" (EDSCC). Let $\Phi = \{C_0, C_1, \ldots, C_{\beta-1}\}$ be a set of partitioned clusters; $C_0 = (V_0, E_0)$, $C_1 = (V_1, E_1)$, …, $C_{\beta-1} = (V_{\beta-1}, E_{\beta-1})$. Each cluster C_i has m_i vertices (i.e., m_i URLs or pages) and n_i edges (i.e., n_i hyperlinks). Therefore, an ideal EDSCC web graph must have the following properties:

- For every cluster, $n_0 \cong n_1 \cong \ldots \cong n_{\beta-1}$.
- For all $i \neq j$, any hyperlink $u \to v$, if $u \in V_i$ then $v \notin V_j$; $0 \le i < \beta$ and $0 \le j < \beta$.

The first property leads to the perfect load balancing when each processor is responsible for computing with the same amount of links, while the second one provides the zero synchronization cost since all nodes in a cluster has no out-link to the other nodes outside. We now analytically discuss the three algorithms in the following way.

I/O cost: For an EDSCC, the total packet size $|P|$ will be equal to the size of the destination rank vector $|V'|$. Thus the Equation (2) can be rewritten as:

$$C_{I/O,Partition} = |L| + 2 \cdot |V'| = |L| + 2 \cdot |V| \tag{11}$$

By the same way, the Equation (8) can also be rewritten as:

$$C_{I/O,Split} = |L| + 2 \cdot |V'| = |L| + 2 \cdot |V| \tag{12}$$

On the other hand, the worst case occurs when all nodes in the input web graph is fully connected, thus each processor has to transmit every new computed score to other processors during the synchronization process. Thus the I/O cost of the Equation (2) and (8) can be rewritten as:

$$C_{I/O,Partition} = |L| + 2\beta \cdot |V'| = |L| + 2\beta \cdot |V| \tag{13}$$

$$\begin{aligned} C_{I/O,Split} &= (1+\varepsilon') \cdot |L| + 2\beta \cdot |V'| \\ &= (1+\varepsilon') \cdot |L| + 2\beta \cdot |V| \end{aligned} \tag{14}$$

0-7695-2281-5/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

Synchronization cost: By the same reason, for an EDSCC, the Equation (4) and (10) can be rewritten as:

$$C_{Syn,Partition} = |V'| - |V| = 0 \quad (15)$$

$$C_{Syn,Split} = |V'| - |V| = 0 \quad (16)$$

And for the case of the fully connected input web graph, the Equation (4) and (10) can be rewritten as:

$$C_{Syn,Partition} = \beta \cdot |V'| - |V| = (\beta - 1) \cdot |V| \quad (17)$$

$$C_{Syn,Split} = \beta \cdot |V'| - |V| = (\beta - 1) \cdot |V| \quad (18)$$

Note that for the worst case, the I/O cost (i.e., Equation (13) and (14)) and the synchronization cost (i.e., Equation (17) and (18)) of the Partition-based and the Split-Accumulate algorithms are approximately the same as those of the Block-based algorithm written in Equation (5) and (7).

Running time: For an ideal case like an input web graph that can be partitioned into EDSCCs, the I/O cost is constant while the synchronization cost is zero. Therefore, the running time of the Partition-based and the Split-Accumulate algorithms will linearly decrease when the number of processors increases. However for the worst case, their I/O and synchronization cost will be varied by the number of processors. If we continually increase the number of processors, the running time will also continually increase, or the overall speedup performance will be rapidly drop.

6. Conclusion

Recently many studies have been focused on improving the final search results of the modern search engines in correspond with the unceasing growth of the World Wide Web. Web link analysis, like PageRank, becomes a very successful technique behind the Google™. However, computing PageRank scores for a very large web graph is not trivial.

In this paper, we have proposed a Partition-based PageRank algorithm that is suitable to run on the parallel environment like the PC cluster. For comparison reason, we also propose to study the other two known PageRank algorithms and adapt them to run on the PC cluster. We implement the three algorithms and exploit sixteen PC machines to compute the PageRank scores of the two test data sets. We also propose a complete analytical discussion in term of I/O and synchronization cost, as well as memory usage.

From the experimental results, we can conclude that our Partition-based algorithm spends less cost than the other two algorithms. However, from the analytical point of view, if the input web graph can be partitioned into several equally dense and strongly connected clusters (EDSCCs), our algorithm will run on any number of processors with the optimal cost. Therefore, in our future work, we look forward to studying the possibility to divide any input web graph into several clusters, which each cluster has the nearly EDSCC properties; and reexamining results with experiments.

7. References

[1] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin, "PageRank Computation and the Structure of the Web: Experiments and Algorithms", In *Proceedings of the 11th World Wide Web Conference*, poster track, 2002.

[2] P. Boldi and S. Vigna, "WebGraph Framework I: Compression Techniques", In *Proceedings of the 13th World Wide Web Conference*, 2004.

[3] S. Brin and L. Page, "The Anatomy of a Large-scale Hypertextual Web Search Engine", In *Proceedings of the 7th World Wide Web Conference*, 1998.

[4] Y. Chen, Q. Gan, and T. Suel, "I/O-Efficient Techniques for Computing PageRank", In *Proceedings of the 11th International Conference on Information and Knowledge Management*, 2002.

[5] T.H. Haveliwala, "Efficient Computation of PageRank", Technical Report, Computer Science Department, Stanford University, 1999.

[6] T.H. Haveliwala, "Topic-Sensitive PageRank", In *Proceedings of the 11th International World Wide Web Conference*, 2002.

[7] T.H. Haveliwala and S.D. Kamvar, "The Second Eigenvalue of the Google Matrix", Technical Report, Computer Science Department, Stanford University, 2003.

[8] G. Jey and J. Wisdom, "Scaling Personalized Web Search", In *Proceeding of the 12th International World Wide Web Conference*, 2003.

[9] S.D. Kamvar, T.H. Haveliwala, and G.H. Golub, "Adaptive Methods for the Computation of PageRank", Technical Report, Computer Science Department, Stanford University, 2003.

[10] S.D. Kamvar, T.H. Haveliwala, C.D. Manning, and G.H. Golub, "Exploiting the Block Structure of the Web for Computing PageRank", Technical Report CSSM-03-02, Computer Science Department, Stanford University, 2003.

[11] S.D. Kamvar, T.H. Haveliwala, C.D. Manning, and G.H. Golub, "Extrapolation Methods for Accelerating PageRank Computations", In *Proceedings of the 12th International World Wide Web Conference*, 2003.

[12] C. Lee, G.H. Golub, and S.A. Zenios, "A Fast Two-Stage Algorithm for Computation PageRank and Its Extensions", Technical Report SCCM-03-15, Computer Science Department, Stanford University, 2003.

[13] A. Rungsawang and B. Manaskasemsak, "PageRank Computation using PC Cluster", In *Proceedings of the 10th European PVM/MPI User's Group Meeting*, 2003.

[14] K. Sankaralingam, S. Sethumadhavan, and J.C. Browne, "Distributed PageRank for P2P Systems", In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, 2003.

[15] The ApGrid, http://www.apgrid.org/, 2004.

[16] The Stanford WebBase Project, http://www-diglib.stanford.edu/~testbed/doc2/WebBase/, 2004.