An Effective Data Placement Strategy for XML Documents

SCISM, South Bank University
103 Borough Road
London SE1 OAA
zhuy@sbu.ac.uk

Abstract. As XML is increasingly being used in Web applications, new technologies need to be investigated for processing XML documents with high performance. Parallelism is a promising solution for structured document processing and data placement is a major factor for system performance improvement in parallel processing. This paper describes an effective XML document data placement strategy. The new strategy is based on a multilevel graph partitioning algorithm with the consideration of the unique features of XML documents and query distributions. A new algorithm, which is based on XML query schemas to derive the weighted graph from the labelled directed graph presentation of XML documents, is also proposed. Performance analysis on the algorithm presented in the paper shows that the new data placement strategy exhibits low workload skew and a high degree of parallelism.

Keywords: Data Placement, XML Documents, Graph Partitioning, and Parallel Data Processing.

1 Introduction

As a new markup language for structured documentation, XML (eXtensible Markup Language) is increasingly being used in Web applications because of its unique features in data representation and exchange. The main advantage of XML is that each XML file can have a semantic schema and makes it possible to define much more meaningful queries than simple, keyword-based retrievals. A recent survey shows that the number of XML business vocabularies has increased from 124 to over 250 in six months [1]. It can be expected that data in XML format would be largely available throughout the Web in the near future. As Web applications are time vulnerable, the increasing size of XML documents and the complexity of evaluating XML queries pose new performance challenges to existing information retrieval technologies. The use of parallelism has shown good scalability in traditional database applications and provides an attractive solution to process structured documents [2]. A large number of XML documents can be distributed onto several processing nodes so that a reasonable query response time can be achieved by processing the related data in parallel.

In parallel data processing, effective data placement has drawn a lot of attention because it has a significant impact on the overall system performance. The data placement strategy for parallel systems is concerned with the distribution of data between different nodes in the system. A poor strategy can result in a non-uniform distribution of the load and the formation of bottlenecks [3]. In general, determining the optimal placement of data across nodes for performance is a difficult problem even for the relational data model [4]. XML documents introduce additional complexity because they do not have a rigid, regular, and complete structure. Although some XML documents may have a DTD (Document Type Definition) file to specify their structures and the W3C (World Wide Web Consortium) is working on the XML Schema standard, either DTD or XML Schema is an optional companion to the XML documents. We cannot expect that every XML document on the Web is a v*alid* XML file, which means that it conforms to a particular DTD or XML Schema.

In this paper, we use the labelled directed graph model to represent XML data. A graph partition algorithm is explored to maximise the parallelism among the different processing nodes in a shared-nothing architecture where each node has its own memory and disks. The distribution of the data is dependent on the queries applied to the data. XML queries are based on path expressions because of its lack of schema information. As path expressions access data in a navigational manner, elements along the objective path should be placed together to minimise communication cost. At the same time, data relative to the same query should be distributed evenly to different nodes to achieve the load balance. These two objectives are both considered in the new proposed data placement strategy. Moreover, the new strategy is based on the unique features of XML documents and the distribution of XML query sets. This paper also presents the performance analysis on the new data placement strategy.

The remainder of the paper is organised as follows: Section 2 presents the related work and motivations of the study. Section 3 describes the XML data model and the algorithm for deriving the weighted graph of XML documents. Section 4 proposes a new graph-partitioning algorithm based on the features of XML documents. Section 5 analyses the performance of the new algorithm. Section 6 concludes the paper and discuss pending research issues.

2 Related Work and Motivations

Effective parallelisation of data queries requires a declustering of data across many disks so that parallel disk I/O can be obtained to reduce response time. A poor data distribution can lead to a higher workload, load imbalance and hence higher cost.

Various data placement strategies have been developed by researchers to exploit the performance potential of shared-nothing relational database systems. Since the complexity of the problem is NP-complete [5], heuristics are normally used to find a nearly optimal solution in a reasonable amount of time. According to the criteria used in reducing costs incurred on resources such as network bandwidth, CPUs, and disks, data placement strategies can be classified into three categories, which are network traffic based [6], size based [7], and access frequency based [8]. The main idea of these approaches is to achieve the minimal load (e.g. network traffic) or a balance of load (e.g. size, I/O access) across the system using a greedy algorithm. Our algorithm is a combination of the network traffic based and access frequency based strategy, because it aims to minimise the communication cost and to maximise the intra-operation parallelism.

In parallel object-oriented database systems, data placement strategy is also critical to the system performance and is far more complex. [9] pointed out that in designing a data placement method for a parallel object-oriented databases, two major factors that most of the time contradict each other must be taken into account: minimising communication cost and maintaining load balance. [4] used a greedy similarity graph partitioning algorithm to assign object into different processing nodes aiming to minimise inter-node traversals and maximise parallelism. This algorithm attempts to place objects that have a higher degree of similarity on different disks, where two objects are more similar if they are accessed together in a navigational manner but less similar if the two objects can be accessed together in a parallel manner. Although the paper gives an equation to compute the similarity between two nodes, there's no definite method for getting the weights between two nodes.

Data placement strategies in both relational and object-oriented parallel database systems could be helpful to the study of the data placement strategy for XML documents. The idea of our data placement strategy for XML data is similar to those in parallel object-oriented databases. But we focus on how to construct the weighted graph from the original XML document, which forms the basis of the graph partitioning algorithm. The objective of the research is trying to find a nearly optimal data distribution so that the system throughput and resource utilisation can be maximised. Our graph partition algorithm is based on the multilevel graph partition algorithm for its efficiency and accuracy. The unique features of XML documents and XML queries have been studied to provide the foundation for the graph partition.

3 Graph Model of XML Data

3.1 Labelled Directed Graph

The latest W3C working draft on XML Information Set (InfoSet) [10] provides a data model for describing the logical structure of a well-formed XML 1.0 document. In this model, an XML document's information set consists of a number of Information Items, which are abstract representations of some components of an XML document. For example, in the XML document of figure 2, there are three different types of information item: document information items, element information items, and attribute information items. The specification presents the information set as a tree and accordingly the information items as the node of the tree. Any information item in the XML document can be reached by recursively following the properties of the root information item. Similar to the data model used in Lore [11], we extended the InfoSet data model to a directed labelled graph, where the vertices in the graph represent the information items and arcs represent the semantic links between the information items.

```
<Publications>
  <Proceeding>
        <Conference>VLDB</Conference>
        <Year>1999</Year>
        <Location>Edinburgh</Location>
        <Article id='A1' reference='A2 A3' >
              <Title>Query Optimization for XML </Title>
              <Author>Jason McHugh </Author>
              <Author>Jennifer Widom</Author>
        </Article>
  </ Proceeding >
  <Proceeding>
        <Conference>ICDT</Conference>
        <Year>1997</Year>
        <Article id="A2">
              <Title>Querying Semi-Structured Data</Title>
              <Author>Serge Abiteboul</Author>
        </Article>
</ Proceeding >
  <Proceeding>
        <Conference>ICDE</Conference>
        <Year>1998</Year>
        <Article id='A3' Reference='A2'>
              <Title>Optimizing Regular Path Expressions Using Graph
                      Schemes</Title>
              <Author>Mary F. Fernandez</Author>
              <Author>Dan Suciu</Author>
        </Article>
  </ Proceeding >
</Publications>
```

Fig. 1. An example for XML documents

Figure 2 describes the graph presentation of the XML document in Figure 1. We use the definition in [12] as our definition for the labelled directed graph:

Definition 3.1 Let L be an arbitrary set of labels. A tuple $G = (V, A, s, t, l)$ is a L-labelled directed graph, if V is a set of vertices, A is a set of arcs, s and t are total functions from A to V assigning each arc its source and target vertex, and l is a total label function from A to L assigning each arc a label.

We can see that the labelled directed graph of single XML document is actually a graph with a unique root. Any vertex in the graph can be reached from the root by following a certain *path*.

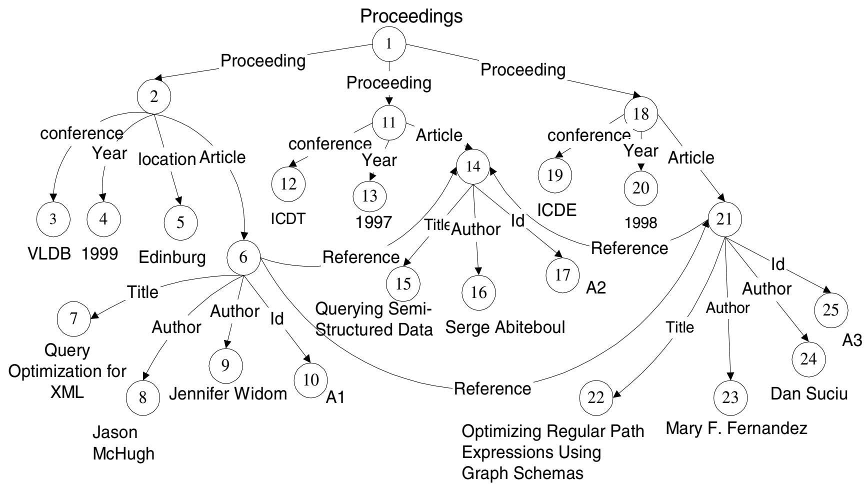

Fig. 2. The labelled directed graph representation for the XML document in Figure 1.

Definition 3.2 A nonempty sequence $(v_{i0}, a_{i1}, v_{i1},..., a_{im}, v_{im})$ is called a *path* in the graph $G = (V, A, s, t, l,)$ if $s(a_{ij}) = v_{ij}$ 1 and $t(a_{ij}) = v_{ij}$ for all positive j m, and all arcs and all vertices in that sequence are pair wise distinct.

3.2 Weighted Graph

Query languages for XML documents generally utilise *path expressions* to exploit the information stored in XML documents. Path expressions are algebraic representations of sets of paths in a graph and are specified by a sequence of nested tags. For example, the path expression „proceeding.article.title" for the XML document in Figure 1 refers to the titles of all articles published in all proceedings. As shown in [12], an XML query can also be presented by a labelled directed graph. Two XML queries and their graph presentations were shown in Figure 3. The elements in the graph are labelled with predicates, where the predicate *true()* serves as a wildcard.

Definition 3.3 Given a set of unary predicates P, a tuple $G_q = (V_q, A_q, s_q, t_q, l_q)$ is a query schema if the elements are labelled with predicates $(l : V_q \quad A_q \quad P)$.

The graphs in Figure 3 can act as schemas, which partly describe the structure of the XML document. If the predicate in a schema is true for the corresponding vertices and arcs in an instance, we say that the instance conforms to the schema. The answer to a query of XML documents is the union of all instances conforming to the query schema. If those instances could be evenly distributed among several different disks and therefore could be accessed in parallel during the query processing, the response time for a query would be largely shortened. Meanwhile, one instance should avoid spanning multiple partitions to reduce the communication cost. These two objectives conflict because the first one tries to distribute vertices across as many partitions as possible, while the second one tries to group the relevant vertices together.

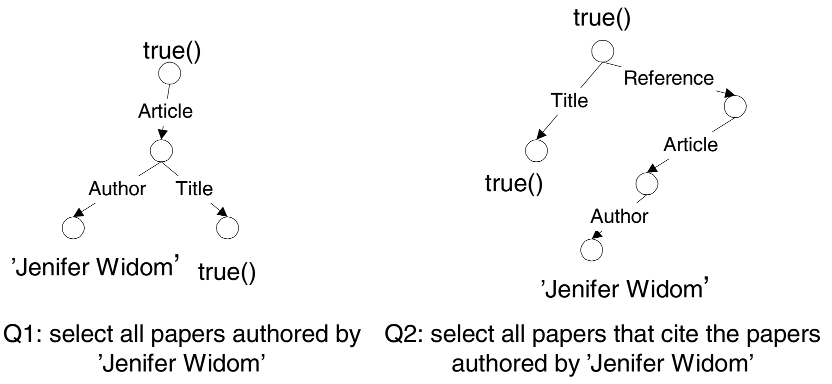
true()
Article
Author
Title
'Jenifer Widom'
true()
Q1: select all papers authored by 'Jenifer Widom'
true()
Title
Reference
true()
Article
Author
'Jenifer Widom'
Q2: select all papers that cite the papers authored by 'Jenifer Widom'

Fig. 3. Two typical XML queries and their corresponding graph based presentation

The data placement of XML documents over different sites can be viewed as a graph partitioning problem. Each edge between two vertices in the graph is associated with a *weight* to describe the frequency of traversals on it. The higher the weight is, the more possible it is to assign the two vertices to the same partition. In our algorithm, the weight between two vertices reflects two factors. One is the possibility of the two vertices to be accessed together in sequential manner, and another one is the likelihood of two vertices to be access together in parallel manner.

With the knowledge on the distribution of the XML query set, a weighted graph could be derived based on the labelled directed graph defined in section 3.1.

Definition 3.4 $G_w = (V, E, r, w)$ is the weighted graph for a labelled directed graph $G = (V, A, s, t, l)$, if E is a set of edges, r is a total functions from A to V assigning each edge its vertices, and w is a total weight function assigning each edges in E a number to describe the traversal frequency of that edge. For each $e \quad E$, there exists at least one arc $a \quad A$ with the same vertices as e.

Algorithm 3.1 describes the method to derive the weighted graph from the original labelled directed diagram based on the query distributions. In this algorithm, if the arcs in the labelled directed graph are traversed in a query, the weights between the corresponding vertices are computed based on the query frequency. If there is more than one instance that conforms to a query schema, the arcs between any two instances are studied to compute the weight of the edges that connect these two instances. The value of parameter is an adjustable number between zero and one, which indicates the relative benefit by increasing the degree of parallelism compared with lowering communication cost. If the communication overhead is high, a higher value for can be chosen.

Algorithm 3.1 Assigning Weight Algorithm

```
Input:
    The labelled directed graph $G = (V, A, s, t, l)$
    The graph presentation $G^q = \{G_1^q, G_2^q, ..., G_n^q\}$
    Where, $G_i^q = (V_i^q, A_i^q, s_i^q, t_i^q, l_i^q), i = 1,2,...,n$
    Query distribution: $= \{ _1, _2,..., _n\}$, where $\overset{n}{\underset{i=1}{}} = 1$
    Adjustable parameter $,0 < \quad < 1$
Output: the weighted graph $G^w = (V, E, r, w)$
Begin
    Initialise the weight for each edge with zero
  For each query $G_i^q$ in the query set
    For each $G_{ij} = (V_{ij}, A_{ij}, s, t, l), j = 1,2...,m$ conforms to $G_i^q$
      For each $a \quad A_{ij}$
        If $s(a) > t(a)$ Then $u = t(a), v = s(a)$
          Else $u = s(a), v = t(a)$
        End If
        $w(u,v) = w(u,v) + \quad _i \; 100$
      End For
      For each $G_{ik} = (V_{ik}, A_{ik}, s, t, l), j < k \quad m$
        If exists $a \quad A$ ,$\{s(a) \quad V_{ij}, t(a) \quad V_{ik} \mid t(a) \quad V_{ij}, s(a) \quad V_{ik}\}$
          If $s(a) > t(a)$ Then $u = t(a), v = s(a)$
            Else $u = s(a), v = t(a)$
          End If
          $w(u,v) = w(u,v) \quad (1 \quad ) \quad _i \; 100$
        End If
      End For
    End For
  End For
End
```

4 Graph Partitioning Algorithm

The graph partitioning problem is NP-complete [13], and heuristics are required to obtain reasonably good partitions. The problem is to decluster the graph into n partitions, such that each partition has roughly equal number of vertices and the number of traversals between different partitions is minimised. In the case of XML parallel processing, we aim at achieving lowest communication cost and gaining load balance among different processing nodes.

[13] introduced a multilevel graph partitioning algorithm, which generally consists of three phases: coarsening phase, partitioning phase, and uncoarsening phase. The graph is first coarsened down to a few hundred vertices, a bisection of this much smaller graph is computed, and then this partition is projected back towards the original graph. This algorithm is suitable for XML graph partitioning because vertices to be accessed in navigational manner could coalesce firstly to make sure that they are assigned to the same processing nodes. Experiments presented in [13] also showed that the multilevel algorithm outperforms other approaches both in computation cost and partition quality. Our new data placement strategy is based on a multilevel graph partitioning approach with the consideration of features of XML documents and XML query distributions.

The goal of the coarsening phase is to reduce the size of a graph by collapsing the matching vertices together. The edges in this set are removed, and the two vertices connected by an edge in the matching are collapsed into a single vertex whose weight is the sum of the weights of the component vertices. The method used to compute the matching is crucial, because it will affect both the quality of the partition, and the time required during the uncoarsening phase. [13] described a heuristic known as heavy-edge matching (HEM) which tries to find a maximal matching that contains edges with large weight. The idea is to randomly pick an unmatched node, select the edge with the highest weight over all valid incident edges, and mark both vertices connected by this edge as matched. Because it collapsed the heaviest edges, the resulting coarse graph is loosely connected. Therefore, the algorithm can produce a good partition of the original graph.

[14] argued that the HEM algorithm may miss some heavy edges in the graph because the nodes are visited randomly. To overcome this problem, they proposed a heaviest-edge matching by sorting the edges by their weights and visiting them in decreasing order of weight. HEM and its variants reduce the number of nodes in a graph by roughly a factor of 2 at each stage of coarsening. If r (instead of 2) nodes of the graph are coalesced into one at each coarsening step, the total number of steps can be reduced form $\log_2(n/k)$ to $\log_r(n/k)$. [14] used an algorithm called heavy-triangle matching (HTM), which coalesces three nodes at a time so that they can get 20% time saving.

We call our coarsening algorithm HSM (Heaviest Schema Matching). Algorithm 4.1 describes the details of the algorithm. In HSM, the vertices are no longer visited in random order. The edges are sorted by their weight and the vertices with the maximum weighted edge are selected to do the matching first. According to algorithm 3.1, there is an edge between two vertices in the weighted graph only if there is an arc between them in the labelled directed graph. In the other word, the neighbour of a vertex v in the weighted graph can be accessed together with v by following a certain path. It is reasonable to collapse the matching vertex with its neighbour together as many as possible if the weight between them is high enough. This strategy can improve the efficiency of the coarsen phase.

The coarsen phase stops when the number of nodes in the coarser graph is small enough. The coarsened graph G_i is made of *multivertices* and edges that have not been merged. A vertex in graph G_i is called multivertices if it contains more than one vertex of G. In a weighted graph, the weight of each edge indicates the possibility for the corresponding vertices to be accessed together both in sequential mode and in parallel mode. On the other hand, the weight also reflects the workload under the query distribution. When two vertices are collapsed together, we need to keep the weight information of the edge being merged. Therefore, we introduced a new notation $\overline{w}$ to denote the weight of the multivertices in the coarsened graph. The workload of each partition will be determined by the sum of the weight of edges and multivertices in that partition.

Definition 3.5 $G_k^w = (V_k, E_k, r_k, w_k, \overline{w})$ is the coarsened graph of a weighted graph $G^w = (V, E, r, w)$, if $V_k \quad V \quad \overline{V}$, where $\overline{V}$ is mad up of multivertices that are created by collapsing vertices from V, and $\overline{w}$ is a total weight function assigning each multivertices in $\overline{V}$ a number to describe the workload of that vertex.

Algorithm 4.1 Coarsening Graph

```
Input:  The labelled directed graph $G = (V_0, A, s, t, l)$
        The weighted graph $G_0 = (V_0, E_0, r, w)$ for $G$
Output: Coarser graph $G_n = (V_n, E_n, w)$ with $N$ vertices
Begin
   $i = 0$
   Do while the number of vertices in $G_i = (V_i, E_i, r, w)$ is greater than $N$
      Sort the edges of $E_i$ in descending order by their weights
      Assume $(u, v) \quad E_i$ is one of edges with maximum weight
      Call $collapse\_vertices(u, v)$ to get the new vertices $v' \quad V_{i+1}$
      $i = i + 1$
   End Do
End
$collapse\_vertices(u, v)$ {
   $V_{i+1} = V_i \quad \{u, v\}$
   $E_{i+1} = E_i \quad (u, v)$
   Build a new vertex $v'$
   Compute the workload for the new vertex:
   $\overline{w}(v') = \overline{w}(v') + w(u, v)$
   For each neighbour $x \quad V_i$ of $u$ and $v$
      If $(u, x) \quad E_i$ and $(v, x) \quad E_i$ Then
```

```
        $E_{i+1} = E_i \quad \{(u,x),(v,x)\} + (x,v')$
        $w(x,v') = w(u,x) + w(v,x)$
      Else
        $E_{i+1} = E_i \quad (\{u \mid v\},x) + (x,v')$
        $w(x,v') = w(\{u \mid v\},x)$
      End If
      If $w(u',x) \quad w(u,v)$ then
        Call $collapse\_vertices(u,v)$ to get the new vertices $v' \quad V_{i+1}$
      End If
    End For
}
```

The second phase of a multilevel graph partitioning algorithm is to compute a balanced bisection of the coarsened graph. [13] evaluated four different algorithms for partitioning the coarser graph. The basic idea of those algorithms is to form a cluster of highly connected nodes. We choose the *graph growing* heuristic for the partitioning phase. The heuristic computes a partition by recursively bisecting the graph into two sub-graphs of appropriate weight. To bisect a graph, we pick up a multivertices with the highest weight first, find its neighbours and neighbours' neighbours in a heaviest-edge-first manner until the workload of the new partition reach the average workload of the graph.

Algorithm 4.2 Partitioning Graph

```
Input:  Coarser graph $G_n = (V_n, E_n, w)$ with $N$ vertices
        The number of processors: $m$
Output: The partition function $P$ assigning each vertex $v \quad V_n$ to
        one of $m$ partitions
Begin
  $i = 2$
  $workload(G_n) = \underset{e \quad E_n}{\quad} w(e) + \underset{v \quad V_n}{\quad} \overline{w}(v)$
  Do while $i < m$
      $Average\_workload = \frac{worklaod(Gn)}{i}$
      For $j = 1$ to $i/2$ do
        Sort the multivertices in $V_n^j$ in descending order by their weights
        Assume $v \quad V_n^j$ is one of multivertices with maximum weight
        $workload(G_n^j) = \overline{w}(v)$
        $V_n^{j*2} = \{v\}$
```

```
          Do while  $\sum_{e\ E_n^{j\ 2}} w(e) + \sum_{v\ V_n^{j\ 2}} \overline{w}(v)) < Average\_workload$
            Assume $u$ is the neighbour of $V_n^{j\ 2}$ with the maximum edge weight
            $V_n^{j\ 2} = V_n^{j\ 2} \quad \{u\}$
          End Do
        End For
        $i = i \quad 2$
    End Do
End
```

During the third phase of multilevel graph partitioning, the partition of the coarsest graph G_k is projected back to the original graph by going through the graphs $G_{k\ \ 1}, G_{k\ \ 2}, \ldots, G_1$. The purpose of a partition refinement algorithm is to select two subsets of vertices, one from each part such that when swapped the resulting partition has smaller edge-cut. Many algorithms associate with each vertex v a quantity called *gain*, which is the decrease in the edge-cut if v is moved to the other part. These algorithms proceed by repeatedly selecting vertices with the highest gains from each part and updating the gains of the remaining vertices. Assuming P is the initial partition of the graph, the gain of a vertex is defined as the following:

$$g_v = \sum_{P(v)\ \ P(u)} w(v,u) \quad \sum_{P(v)=P(u)} w(v,u), \text{ where } (v,u) \quad E \qquad \textbf{(1)}$$

If v is moved to the other partition, the gain of its neighbours should be modified. The algorithm stops when there's no vertex with positive gain value left.

5 Performance Analysis

We used the DBLP [15] data set as our experiment data. The DBLP data set collects about 140,000 entries for published literature on database research area. The original DBLP database stored each entry in a separate XML file and organised them by multiple directories according to its origination. We parsed the files into entities, which represent the vertices in the graph, and tags, which are the labels in the graph. The hierarchy of the directories is also reflected in the graph representation. We specially checked the *cite* entity in each document and linked it to the corresponding vertices in the graph. The final graph for partitioning test contains 1,693,444 vertices and 1,802,158 arcs. We used the query set in [16] to test our algorithms, and the query frequency was also specified.

Table 1. Description of query sets and relative query fequencies for the experiment

	Query Description	Result Number	Frequency			
			Case1	Case2	Case3	Case4
SQ_1	Select the authors for a given title	8	30%	20%	20%	10%
SQ_2	Select all papers authored by Michael Stonebraker	169	20%	10%	5%	10%
SQ_3	Select all papers authored by Michael Stonebraker or Jim Gray	242	20%	10%	10%	15%
SQ_4	Select all papers published between 1990 and 1994	47,527	5%	5%	10%	10%
JQ_1	Select all papers by Jim Gray that are quoted by Michael Stonebraker	2	10%	30%	20%	10%
JQ_2	Select all papers that quoted Michael Stonebraker's papers and were published between 1990 and 1994	513	10%	20%	25%	30%
JQ_3	Select all pairs of papers that cite one another	108,717	5%	5%	10%	15%

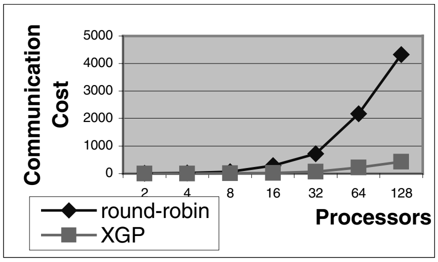

Fig. 4. Communication costs of different numbers of processors

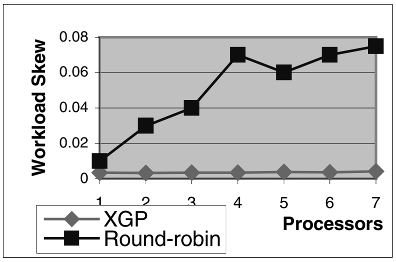

Fig. 5. Workload skews of different numbers of processors.

For convenience, we briefly called our XML graph partitioning algorithm XGP. As the objectives of the XGP algorithm are to reduce the communication cost and lower workload skew, these two measures have been tested to check the quality of the algorithm. Figure 4 compares the communication costs when the round-robin algorithm and the XGP algorithm are used for data partitioning. The communication cost is indicated by the numbers of remote requested pages. We can see that the XGP algorithm produces less communication cost than the round-robin algorithm does. Figure 5 shows workload skews among the processing nodes. The workload skew is indicated by the difference between the workload of each partition to the average workload. It is defined in formula (2) and (3).

$$Avg\_workload = \frac{{}_{e\ E}\ w(e)}{m} \qquad (2)$$

$$workload\_skew = \frac{\sum_{i=1}^{m} \left| workload(Pi) \quad avg\_wokload \right|}{m} \qquad (3)$$

It can be seen that the workload skew for the XGP algorithm doesnot change much with the increase of the number of processing nodes. XGP also produces a low workload skew, which is much less than the round-robin algorithm does.

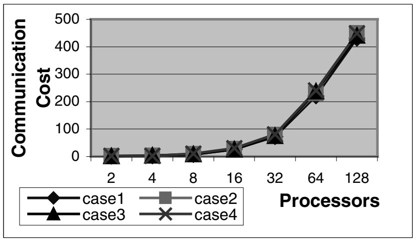

Fig. 6. Communication costs of different numbers of processors.

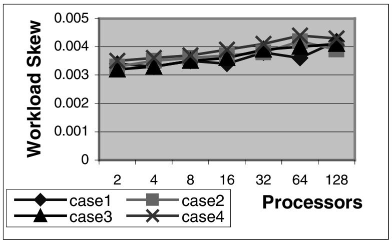

Fig. 7. Workload skews of different numbers of processors

The last experiment was to test the impact of query distributions to the workload balance. We tested the communication costs and workload skews caused by partitioning with different query frequency distributions. Figure 6 and Figure 7 show that XGP algorithm performs well under all the four cases showed in table 1. We can see that the communication costs and workload skews of four cases are quite close. Because the weight used in the XGP algorithm is dependent on the query frequency, the partitions for different query distributions will change accordingly.

6 Conclusion

In this study, we have developed a data placement strategy for the XML documents on parallel processing systems. This approach is based on the multilevel graph partitioning algorithm with consideration of the unique features of XML data and XML queries. A new algorithm is proposed for deriving the weighted graph from the labelled directed graph by using the implied schema information from XML queries. According to our approach, entities to be accessed by navigation in a query would be assigned to the same processing node, and instances accessed by the same query are distributed evenly along all the processing nodes. In the coarsening phase of the multilevel graph partitioning algorithm, all vertices in the neighourhood of the selected matching vertex are coalesced based on their edge weight. This criterion speeds up the procedure of the coarsening and reduces the possibility of assigning vertices to be accessed by navigation to different processing nodes. In the partitioning phase, the weights of multivertices are used to evenly distribute the workload of a query. The performance analysis shows that the partition produced by our algorithm could greatly reduce the communication cost and lower workload skew. In our future work, we will focus on the parallel processing of XML queries and the XML query optimisation with the consideration of different data placement strategies.

References

[1] Alan Kotok, An updated Survey of XML Business Vocabularies, http://www.xml.com/lpt/a/2000/8/02/ebiz/extensible.html, August 2000.
[2] D.B. Skillicorn, Structured Parallel Computation in Structured Documents, external technical Report, 1995.
[3] Manish Mehta, David J. DeWitt, Data placement in shared-nothing parallel database systems, VLDB Journal (1997) 6: pages 53–72.
[4] Zhen He and Jefferey Xu Yu, Object Placement in Parallel OODBMS, Proceedings of the Tenth Australasian Database Conference, pages 101-114, Auckland, New Zealand, January 1999.
[5] D. Sacca and G. Widerhold, Database partitioning in a cluster of processors. In proceedings of the 9th VLDB Conference, pages 242-247, Florence, Italy, Oct 31-Nov2, 1983.
[6] P. Apers. Data allocation in distributed database systems. ACM Transactions on Database Systems, 13(3): 263-304, September 1988.
[7] K.A. Hua, C. Lee, and H.C. Young, An efficient load balancing strategy for shared-nothing database systems. In proceedings of DEXA'92 conference, pages 469-474, Valencia, Spain, 1992.
[8] G. Copeland, W. Alexander, E. Boughter, and T. Keller, Data placement in Bubba, in Proceedings of ACM SIGMOD Conference, pages 99-108, Chicago, Illinois, June 1988.
[9] David Taniar, Toward an Ideal Data Placement Scheme for High Performance Object-Orented Database Systems, Proceedings of high-performance Computing and Networking, International Conference and Exhibition, HPCN Europe, pages 508-517, Amsterdam, 1998.
[10] Roy Goldman, Jason McHugh, Jennifer Widom, From Semistructured Data to XML: Migrating the Lore Data Model and Query Language, Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99), pages: 25-30, Philadelphia, Pennsylvania, June 1999.
[11] XML Schema Part1: Structures, W3C Candidate Recommendation, http://www.w3.org/TR/xmlschema-1, 24 October 2000.
[12] A. Bergholz. "Querying Semistructured Data Based On Schema Matching", Doctoral dissertation, Department of Computer Science, Humboldt University Berlin, Berlin, Germany, January 2000.
[13] George Karypis and vipin Kumar, A Fast and High Quality Multilevel Scheme for Partitioning Irregular graphs, SIAM Journal on Scientific Computing, Vol. 20 Number 1, pages: 359-392. 1998.
[14] A. Gupta, Fast and Effective Algorithms for Graph Partitioning and Sparse-matrix Ordering, Vol. 41, NO. 1-2, IBM Journal of Research & Development, pages: 171-184, 1997.
[15] DBLP maintained by M. Ley. http://www.informatik.uni-trier.de/~ley/db/index.html, December 2000.
[16] Feng Tian, David J. Dewitt, Jianjun Chen, and Chun Zhang, The Design and Performance Evaluation of Alternative XML Storage Strategies, http://www.cs.wisc.edu/niagara/vldb00XML.pdf, December 2000.