Controlling the Queue Size in Web Crawling

Carlos Castillo  
Università di Roma  
“La Sapienza”  
Rome, Italy  
castillo@dis.uniroma1.it

Alberto Nelli  
Università di Roma  
“La Sapienza”  
Rome, Italy  
anelli@tim.it

Alessandro Panconesi  
Università di Roma  
“La Sapienza”  
Rome, Italy  
ale@dsi.uniroma1.it

ABSTRACT

We study the size of the queue of pending pages during a crawl of a large subset of the Web. We show how an adequate handling of the larger hubs found on the Web can allow us to save 50% of the size of the queue while preserving 100% of coverage and good quality in the collection.

1. INTRODUCTION

Web crawlers are used by Web search engines to visit Web pages automatically, by recursively following links. A queue of pages to be visited is initialized with a given set of starting pages, and the newly discovered pages are added to this queue. The queue may become very large mostly due to the existence of a large number of dynamic pages, that is, pages that are generated automatically by querying a data source.

As the Web has potentially an infinite number of pages and most of them will never be visited by the crawler, we aim at keeping the queue of waiting pages relatively small. The typical prioritization policy is breadth-first search, and under this policy, the number of pending pages can be very high. We used the WebBase [5] repository of 118 million pages and simulated a breadth-first and depth-first visit of its nodes. Both visits could reach 95 million pages starting from an arbitrary, but fixed, set of nodes (in the following, we assume this to be 100% of coverage).

As we can see in Figure 1, in breadth-first search [6] the queue size grows very fast, up to a maximum of roughly 15% of the total number of pages (which can be in the order of billions of pages in the case of the full Web), whereas in depth-first search the queue tends to be much smaller. However, depth-first search cannot be used in practice for several reasons: it requires to focus the crawler on a few sites, which might be against typical “politeness” policies; it fails to capture pages with high quality [2] and it might be too prone to get caught into artificially crafted Web page loops, also known as “crawler traps”.

Reducing memory usage in Web crawling has a number of benefits. First, in practice most of the pages in the queue will never be visited; second, in the case of parallel crawlers [4] that must exchange URLs, the amount of URLs exchanged can be reduced; third, in the case of focused crawlings [3] by personal Web crawlers (that run on normal desktop PCs instead of large servers), keeping the usage of resources of the crawler small is important. The goal of our research if to find a strategy that has the good properties of breadth-first search in terms of quality and politeness with Web servers, but that uses a queue size comparable to that of depth-first search.

2. SAMPLING OUT-LINKS

Unrestricted breadth-first search generates very large queues because some nodes are very large “hubs” and have many out-links. A first approach is to take a random subset of the out-links from large hubs and ignore the rest, hoping to find other links to the ignored pages somewhere else on the Web graph.

Using this technique we can reduce the queue size, but unfortunately many pages have only one in-link so the fraction of pages that are lost tends to be quite large, as shown in Table 1. For instance, in the last column we can see that selecting a maximum of 8 links per page we reduce the maximum queue size to 36% of its size in breadth-first search, but lose almost half of the pages, which is unacceptable. In our experiments, taking a fraction of the out-links, instead of a maximum fixed number, does not improve the coverage significatively. Also, the cost-benefit trade-off of this technique seems to behave in a linear way.

Table 1: Taking up to a maximum number of out-links per page can save in terms of the queue size, but comes at the cost of a much lower coverage.

<table>
<thead>
<tr>
<th>Max. outlinks</th>
<th>All</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. queue size</td>
<td>0.14</td>
<td>0.13</td>
<td>0.12</td>
<td>0.11</td>
<td>0.08</td>
<td>0.05</td>
</tr>
<tr>
<td>Normalized Coverage</td>
<td>100%</td>
<td>92%</td>
<td>86%</td>
<td>79%</td>
<td>57%</td>
<td>36%</td>
</tr>
<tr>
<td>Coverage</td>
<td>100%</td>
<td>93%</td>
<td>90%</td>
<td>82%</td>
<td>70%</td>
<td>53%</td>
</tr>
</tbody>
</table>

Copyright is held by the author/owner(s).
3. SECONDARY QUEUE FOR LARGE HUBS

Now we are going to show a strategy that can reduce the queue size while preserving coverage and quality of the collection. The key is to remember the nodes with high outdegree and scanning their out-links again when we are out of links during the crawl.

If a node has less than \( K \) out-links, then we add all of those links to the queue of pages to visit. If a node has more than \( K \) out-links, we pick at random only \( K \) of them, but put the source node in a secondary queue for re-visiting it later. When the primary queue becomes empty, we take all the nodes in the secondary queue and re-visit them.

This strategy behaves basically as breadth-first and can provide full coverage using only half the size of the queue, as shown in Figure 2. In the figure, we are showing the combined size of both the primary and the secondary queue. The combined size is dominated by the size of the primary queue, as the out-degree exhibits a power-law so the fraction of nodes with many out-links is very small.

We tested this technique in other Web collections made available by the Laboratory of Web Algorithmics (Dipartimento di Scienze dell’Informazione, Università degli studi di Milano, available online at http://law.dsi.unimi.it/). We used two snapshots of 40 million pages from the .it domain and 18 million pages from the .uk domain. In both cases, the maximum queue size was also reduced by roughly a half, showing the robustness of our approach.

A few nodes have to be visited more than once. In Table 2 we show the maximum queue size, and the fraction of nodes that have to be re-visited in the WebBase collection, for different values of \( K \). There is an interesting trade-off between the number of maximum out-links taken per page and the maximum queue size. For instance, taking \( K = 16 \), the maximum queue size grows from 50% to 64% but only 1% of the pages have to be visited more than once.

Note that doing a second network request to the larger nodes may not be necessary: the URLs contained in the pages with many out-links can be stored on disk as (compressed) text, and then added to the crawler’s queue later, when necessary. In fact, it should be done in that way in large collections, as the Web is a very dynamic environment and the nodes in the secondary queue may change.

4. FUTURE WORK

We have also studied other techniques involving selective visits of pages. A promising approach we have tested is to select only a fraction of the out-links of home pages and pages in the first few levels of a Web site, but selecting all of the out-links of deeper pages. The reason is that pages deeper inside a Web site have fewer in-links, so they are more difficult to find. This approach also leads to savings in the queue size without having a large impact on the coverage.

Table 2: Maximum queue size and fraction of re-visits for the strategy with a secondary queue. The coverage is always 100%.

<table>
<thead>
<tr>
<th>Max. outlinks (K)</th>
<th>All</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. queue size</td>
<td>0.14</td>
<td>0.12</td>
<td>0.11</td>
<td>0.09</td>
<td>0.07</td>
</tr>
<tr>
<td>Normalized</td>
<td>100%</td>
<td>86%</td>
<td>79%</td>
<td>64%</td>
<td>50%</td>
</tr>
<tr>
<td>Re-visited nodes</td>
<td>0%</td>
<td>0.1%</td>
<td>0.4%</td>
<td>1.0%</td>
<td>2.2%</td>
</tr>
</tbody>
</table>

Note that in our case we are dealing with a simulated crawl over a known subset of the Web; in the case of a large-scale search engine the techniques we have presented are even more useful as instead of growing and then shrinking, the queue of pending pages continues to grow without bounds.

This can be mixed with quality estimators of Web pages (for instance, number of in-links or on-line page importance computations [1]) so a page with a large number of out-links, but a low quality estimator, is a good candidate for sampling its out-links and adding the page to a secondary queue. We are currently studying these quality-aware strategies.

5. REFERENCES