Improving Range Query Performance on Historic Web Page Data

Geng LI
Lab of Computer Networks and Distributed Systems, Peking University
Beijing, China
ligeng@net.pku.edu.cn

Bo Peng
Lab of Computer Networks and Distributed Systems, Peking University
Beijing, China
pb@net.pku.edu.cn

Abstract—This paper is about the performance of range queries on historic web page data set, i.e. requests into a data set of web pages that keeps record of historic versions of HTML data of URLs on the web for a subset of data, the URLs and the timestamps of which satisfy the query conditions. To keep track of all versions of every web URL, the data set could easily scale up to terabytes. Hence, systems providing query services to such a data set would require much computing resource. We show that in this scenario data storage layout has significant impact on query performance and propose storage design principles for performance improvement through quantitative approaches.

Keywords - web-scale data access, storage design, performance optimization

I. INTRODUCTION

The historic web pages on the Internet are an enormous data set. The Web InfoMall(http://infomall.cn), which aims to archive all web pages ever existed in China, now serves about 3B historic pages online and this number grows by 45M per month. Any system that can host data sets at this scale need be abundant in storage resources and well scalable at first hand. Secondly and more importantly, query serving on this data set need be designed efficiently, as the performance of approaches like full-set scan is obviously unacceptable. A typical query on this data set consists of two parts: the key part specifies the URL range of the web pages of interest to the client, while the time part specifies a time range in which the pages exist on the Web. The answer set to a query thus contains all the historic page data that lies within the key range and the time range, i.e. datum points within a rectangle in a two-dimensional space. As the size of the whole data set is too large to fit in the main memory of any mainstream commercial servers, it is more economic to keep the data on cheaper external storage devices such as the hard disk. However, the speed of access to hard disks is lower than main memory by several orders of magnitude, making it a bottleneck to efficient query serving. So, reducing disk access would decrease the overhead incurred in query serving. Usually, the size of the answer set to a query is quite small compared to the whole web page data set, and we would expect the amount of data accessed from hard disks to be as small as possible, ideally exactly those matching the query. With data compression techniques applied, the amount read from disks could even be smaller. As we shall see in the following sections, minimizing disk access in this scenario is a non-trivial task and the solution to this problem depends on several factors concerning data set properties and storage design strategies.

II. RELATED WORK

As mentioned in the previous section, the Web is a dynamic data set, in that new pages occur on the web and that the contents of existing web pages change over time. By convention, web pages are uniquely identified by their URLs. A data set that keeps track of historic web data thus need an extra timestamp value to identify each version of the page having the same URL on the web. To serve a data set at web scale for access, the system designer need to consider several features, the most important ones being scalability, data availability and retrieval performance in terms of access latency and throughput. Google's BigTable[1] and Yahoo!'s PNUTS[2] tackled the problem of scalability by introducing a multi-computer system which may easily add/remove new server machines to/from it without incurring single-node bottleneck in performance, and they provide availability guarantees through data replication on physically different machines. On data storage layout, Google's BigTable[1] proposed the Sorted String Table (SST) format which is capable of storing different versions of data having the same key.

We followed the above techniques in implementing BigHive[3] which was designed to support multi-versioned web-scale data access. In addition, we extended the SST format by introducing another level of index on the timestamp property to each data block (See Figure 1). The extended SST format is thus capable of performing two-dimensional range
queries more effectively since it is no longer necessary to retrieve all blocks containing data items having the same key from hard disks (the time index made it capable of searching along the time dimension). In practice, however, we have observed that there is little improvement in performance (in terms of disk access) after our optimization to the SST structure, i.e. disk access does not decrease as much as we expected compared to the original SST format. The evaluation result motivates our study into this phenomenon and is the basics to all the work in the following sections.

III. THE PERFORMANCE ISSUE

We formalize the performance measure of two-dimensional range queries in order to pin-point the major factors that have influence on it. By a quantitative approach to this issue, we are able to give an explanation to the problem why our previous attempts on the SST format failed to enhance query performance as we expected. The results of our analysis also suggest directions to work on to achieve better query performance.

A. Backgrounds, Definitions and Notations

Data on storage are usually accessed in blocks, i.e. the minimal amount of data that must be read or written by a single operation. In web-scale data serving systems like [1], the block refers to the unit of storage access by the data serving component rather than the file system's property. We adopted the same approach in implementing the BigHive, so that all data are stored in consecutive blocks in SSTs. On query evaluation, the data serving system retrieve data blocks from the underlying storage and extract data items that satisfy the query’s specifications from the retrieved blocks. As we mentioned in previous sections, the bottleneck of this process is the phase of retrieving blocks from underlying storage. Thus the performance measure need be closely related to the number of data blocks retrieved. As we shall see in the following sections, the work we have done concentrates on reducing the number blocks that need to be accessed to serve a single query.

To perform range query $q$ on a two-dimensional data space, we view the answer set as a subset of the data items whose keys satisfy query $q$’s specified key range $h$. We define the following notations:

- $c$: The block capacity, i.e. the number of data items that could be stored in one block on average, which depends on the block size configuration, data compression techniques applied, the average size of data items, etc.;
- $b(h)$: (Number of) data blocks taken by range $h$, i.e. all blocks that contain at least one data item whose key lies within $h$;
- $t(h)$: (Number of) data blocks that must be retrieved from $b(h)$ in order to serve the query. Apparently $t(h) \leq b(h)$

Thus, the performance measure of serving query $q$ is define by

- $P(q) = 1/t(h)$, i.e. the inverse of the number of blocks to read, whose worst-case value is $1/b(h)$ iff $t(h) = b(h)$

B. Performance Measure Analysis

The above performance measure can be re-written as

- $P(q) = 1/t(h) = 1/b(h) \cdot b(h)/t(h)$

This equation suggest that we can improve query performance by the following means:

- a) Decrease $b(h)$, by increasing block capacity $c$ or by performing data compression if $c$ is constant;
- b) Decrease $t(h)/b(h)$, i.e. decrease the number of blocks accessed from $b(h)$, which is the purpose of optimizing storage organization.

Our previous efforts in revising the SST format can be classified as storage organization optimization. They failed under our performance measure because the value of $t(h)/b(h)$ did not decrease as much as we expected. As we see, the extended SST format would only work if the number of data items under the same key are approximately equal. Further analysis suggest that this property does not hold true on a web data set.

C. Data Set Properties Matter

To investigate the distribution of the number of URLs having a fixed number of versions, we sampled a total of 14,475,268 historic web pages from the data set of the Web InfoMall. The sampled web pages are crawled within more than 4 years, and the sampled data set contains 9,219,047 different URLs, i.e. each URL has 1.6 versions of pages on average. After statistical analysis we have found that the
version numbers comply with a typical Power-Law distribution (Figure 2).

As the web page set is dynamic, we further analysed the trend of change of Power-Law distribution parameters, i.e. the value of slope. We evenly divide the 4-year time into 8 spans, and the previous statistics were performed on each of the accumulated time span, i.e. the first experiment was carried out on pages crawled in the first half of the first year, and the second on those crawled in the first year, and the third on those crawled in the first and second years, etc. The results showed that the value of slope changes little over time (Figure 3, 4, 5).

We are therefore able to perform a quantitative analysis on the performance of range queries on this web page data set. First of all, the number of keys (URLs) having \( x \) different versions of data items is given by

\[
U(x=k) = V \cdot k^{-\lambda},
\]

where \( V \) and \( \lambda \) are constants greater than 0.

Suppose the system has block capacity \( c \), for those keys who have less than or equal to \( c \) versions of data items, the index on the time values adds no value to data access performance because these blocks must be read for checking the right values as the index structure that we introduced only indexes boundaries of data blocks in SSTs. The following calculates the amount of data that have more than \( c \) data items under the same key.

Recall the number of blocks that contain at least one data item whose key lies within \( h \) is denoted by \( b(h) \) and that

\[
b(h) = \sum \lambda (V \cdot k^{-\lambda} \cdot k/c)
= (V/c) \cdot \sum k^{1-\lambda}
\]
\[(V/c) \cdot (\sum_{k \leq c} k^{1/\lambda} + \sum_{k > c} k^{1/\lambda}) \quad (2)\]

It follows that the total amount of data whose number of versions is greater than \(c\) is given by

\[r = \left( \frac{\sum_{k > c} k^{1/\lambda}}{\sum_{k \leq c} k^{1/\lambda}} \right) \quad (3)\]

As we previously analyzed in this section, the extend SST format only reduce the data accessed from the part where \(k > c\) in equation (2). Therefore, the total number of data blocks that need be read from underlying storage is

\[t(h) \geq (V/c) \cdot \sum_{k \leq c} k^{1/\lambda} \quad (4)\]

And the performance of query is given by

\[P(q) = \frac{1/b(h) \cdot b(h)/t(h)}{[V/c] \cdot (\sum_{k \leq c} k^{1/\lambda} + \sum_{k > c} k^{1/\lambda})} \quad (5)\]

This equation determines the upper bound of query performance for a fixed \(b(h)\). As the value of \(\lambda\) is around 2.5, we may fix this value in digging the influence of parameters \(V\) and \(c\) to parameter \(r\). The first experiment examines the influence of power law parameter \(V\), as shown in Table I.

<table>
<thead>
<tr>
<th>Value of (V)</th>
<th>Value of (r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>0.190734</td>
</tr>
<tr>
<td>20,000</td>
<td>0.212633</td>
</tr>
<tr>
<td>30,000</td>
<td>0.22353</td>
</tr>
<tr>
<td>40,000</td>
<td>0.231381</td>
</tr>
<tr>
<td>50,000</td>
<td>0.236431</td>
</tr>
<tr>
<td>60,000</td>
<td>0.240912</td>
</tr>
<tr>
<td>70,000</td>
<td>0.244285</td>
</tr>
<tr>
<td>80,000</td>
<td>0.247377</td>
</tr>
<tr>
<td>90,000</td>
<td>0.249674</td>
</tr>
<tr>
<td>100,000</td>
<td>0.25235</td>
</tr>
</tbody>
</table>

The value of \(V\) has little impact on the value of \(r\) (with \(c=8\) and \(\lambda=2.5\)).

Similarly, Table II describes the impact of parameter block capacity \(c\) on parameter \(r\).

<table>
<thead>
<tr>
<th>Value of (c)</th>
<th>Value of (r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.443967</td>
</tr>
<tr>
<td>5</td>
<td>0.370604</td>
</tr>
<tr>
<td>6</td>
<td>0.319601</td>
</tr>
<tr>
<td>7</td>
<td>0.281751</td>
</tr>
<tr>
<td>8</td>
<td>0.25235</td>
</tr>
</tbody>
</table>

The value of \(r\) decreases as \(c\) increases (with \(V=100,000\) and \(\lambda=2.5\)).

To achieve better performance on range queries, we need to increase the value of \(r\) on the one hand and decrease the value of \(b(h)\) on the other. What we conclude from previous, however, is that the parameter \(c\) has conflicting functionalities on the two sides: a larger \(c\) would decrease both \(b(h)\) and \(r\), causing the effect of optimization to degrade. Moreover, even if we fix the value of \(c\) to a relatively small value (say \(c=8\)), with \(\lambda=2.5\), the performance improvement (determined by \(r\) alone) is only 25%. These are the underlying reasons why our extended SST format failed the performance measure.

IV. STORAGE DESIGN IMPROVEMENT - ONE SOLUTION

As described in the previous section, the distribution of data is a major factor that must be considered in storage design. The extended SST did not work well because we previously assumed uniform distribution of data items under different keys. With data set properties analyzed, we adopted the Hilbert Curve indexing [4] technique in SST design. The Hilbert Curve indexing technique groups those data points whose Euclidean distances are small to consecutive data blocks, thus minimizing block access when performing two-dimensional range queries. The simulated experiment proves its effect on this particular problem. Figure 6 compares the performance of this technique to the extended SST, with parameters \(\lambda\) and \(c\) unchanged. The comparison suggests that the data blocks accessed from hard disks are no longer fixed (in extended SST, at least 80% of \(b(h)\)) but are proportional to the number really necessary to serve the query.

![Figure 6](image_url)

V. FUTURE WORK

Although effective in serving range queries, the Hilbert Curve Indexing technique is less effective in answering advanced queries such as “Find me the web page of www.pku.edu.cn nearest to Apr. 1 of 2004” than the extended SST format. In a historic data set, implementing advanced query serving might be of better use to its clients and is the subject of our study. A better solution might be the combination of the two formats of storage each responsible for answering different types of queries. But this might introduce...
new problems to the system such as data consistency. The study of new data storage formats will be ongoing.

VI. CONCLUSION

We believe two-dimensional range queries on a historic web-scale data set are a non-trivial issue. The design of data storage plays a vital role in performance. The paper proposes quantitative approaches that the designers can follow to evaluate system performance. With our experiences on solving this particular problem, we propose the following guidelines to storage design:

a) The performance measure is the only target of design or optimization. A clean performance measure definition is the first step towards promising outcomes.

b) Choose the right technique based upon deep understanding of the data set we are faced with. Quantify your data set properties if possible.

c) Try existing matured techniques that may be an acceptable (if not perfect) solution to the problem.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (Grant No. 70903008 and 60933004), ME 2008 Grant CNGI2008-122 and the National High-tech Research and Development Program (“863” Program) of China (Grant No. 2009AA01Z143).

REFERENCES


