Journal of Engineering Design and Analysis
Volume 3, Issue 2 - 2020, Pg. No. 8-12
Peer Reviewed Journal

Research Article

Client-Side Pagination for Large Dataset

Saikat Bagchi

Infosys Limited, Bhubaneswar, Odisha, India.

I N F O

E-mail Id:
sbagchi.research@gmail.com
How to cite this article:
Bagchi S. Client-Side Pagination for Large Dataset. *J Engr Desg Anal* 2020; 3(2): 8-12.

Date of Submission: 2020-11-15
Date of Acceptance: 2020-11-27

A B S T R A C T

This work attempts to provide a solution to address the challenges in fetch and represent of large set of records at client-side application. In general user interfaces represent search results in paginated data tables and use just-in-time asynchronous calls to service end points to get the page content. There are several alternative or complementary practices to support client-side pagination, e.g. Indexing of data through specialized indexing tools, cache systems, leveraging suitable algorithms for optimized data retrieval from storage etc. A well-crafted strategy for service calls and management of client-side resources, is essential for achieving reasonable system performance and user experience.

Keywords: Pagination, Client-Side Pagination, Dynamic Pagination, Pagination for Large Data, Scalable Pagination

Introduction

Enterprise systems generally process and represent large amount of information and data with intrinsic principles and objective to address stated business goals. System designers of any interactiv e system often face two key challenges retrieval of context specific data and representationto system users. The challenges become acute when large set of data need to be presented to users. A typical instance of this issue is observed in implementing search functions in applications. Designers adopt pagination techniques to provide an organized, bucketed display of subset of search results at client-side interfaces. Asynchronous techniques like AJAX,[1] help in adopting hybrid solution using both client-side and server-side pagination. While client-side pagination is generally appropriate for low volume of data, server-side pagination is apt for large data set. It is also adopted when there are accessibility restrictions for usage of client side scripting (e.g. java-script). Correct implementation of pagination is always challenging and demands good amount of effort and analysis by designers.[2]

Adoption of a hybrid solution for pagination, proves to be more useful in handling large data set. This approach cuts through all layers of a system to apply optimization techniques. Offset or value (cursor) based methods[3] may be used for retrieving data from database; placement of a well formed cache layer assists in improvement of service response time in delivering the paginated data to client, butonly server- side optimization is not enough for meeting the performance and scale related challenges. A well-crafted strategy, for service calls and optimum usage of client-side resources, isessential for achieving reasonable system performance and user experience. This work proposes one such client-side solution for pagination of large set of records.

Related Work

Paginationis a real challenge in client-side application that wants to present large set of data to users. This problem is mostly relevant for systems that display search results to users. A number of third party libraries (both opensource and commercial) are available in market for providing out-of-the-box solution for client-side pagination, but it is often observed that these libraries might not help in handling large dataset efficiently. Two of the frequently followed approaches,used for getting data from server-side applications, are: a) individual service call for fetching each page content, b) service call to fetch content for multiple pages in batch. Cao J. et al.[5] have given a comparative analysis of pagination algorithms based on large data sets

- server-side pagination algorithms used in data retrieval from database storage. Most of the available articles and analysis works, related to pagination, are mainly focused around the client-side best practices for representation of paginated view to users.[6-10] There have been some work on identification of disadvantages of pagination in.[11] There are discussions and analysis around pagination and performance issues at client-side pagination.[12,13] explains the data paging technique using backendless tool, which uses offset based approach for data retrieval at server-side. The scope of above mentioned study and analysis were mostly around the client-side representation and server-side data retrieval. There needs to be a comprehensive approach for handling challenges for client-side pagination of large dataset.

Proposed Solution for Client- Sidepagination of Large Data in Search Result

Assumptions

This work assumes that the underlying system is following client-server principles; server-side data retrievalis using offset/ value-based methods; service api end points are in place for providing data and an asynchronous methodology is adopted at client-side for interaction with service end points.Client-side cache is used for temporary storage of data in pages.

Client-side Configuration

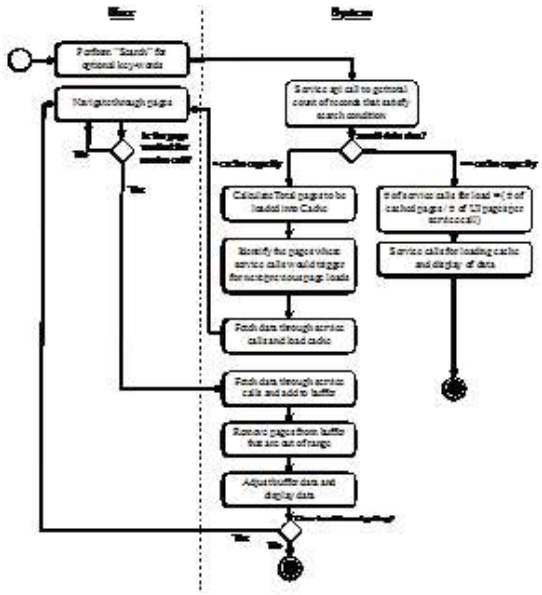


Figure 1.Activity flow of proposed solution

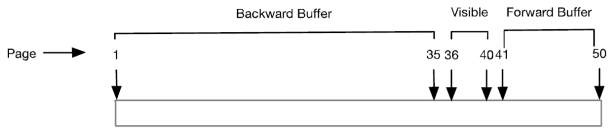


Figure 2.Cache content category

Description

Table 1.Configuration Parameters

Configuration Parameter	Description
Cache capacity	Maximum capacity of data (record set/ pages) that can be stored and maintainedin local cache (browser cache or any client-side in-memory cache)
Data size in Service response	Number of records requested from service end point in each service call
Pages in pagination table	Number of pages displayed in paginated data table to user
Page size in pagination table	Number of records displayed per page in paginated data table

Activity Flow

Figure 1, shows an activity flow of the proposed approach.

When a user performs search for any key word, a service api call resolves the total count of qualifying records in database. If the total number of qualifying records, is within the cache limit, *C*, service calls are made to fetch and load all records into client-side cache. On the other hand, if the total number of records is more than cache capacity, then *C* number of records are fetched from service, stored into cache and grouped as sequenced pages. Service calls are asynchronously triggered when a user navigates to pre-identified pages. Service response data are stored in forward/ backward buffers in cache based on the direction of navigation path. The cache content enables the application to provide a seamless experience in navigation of pages in pagination table. This is explained in more detail with example in next section. When a user navigates through pages, the buffered pages are adjusted dynamically.

During forward navigation, service calls load pages into cache in forward buffer and removes proportionate data from backward buffer to maintain volume within cache threshold. Exactly reverse operation takes place during backward navigation i.e. service calls load data into backward buffer and removes proportionate data from forward buffer.

Cache content can be categorized as forward buffer, visible pages, backward buffer. The visible pages are displayed to user in pagination table (e.g. clickable page links in pagination table in web interface), so that a user can access the page content directly. The forward buffer contains pages that a user can navigate forward in pagination table andthe backward buffer contains pages that can benavigated backward. This scheme is depicted in Figure 2.

ISSN: 2582-5607

Mathematical Notations and System Parameters

Following parameters are used in upcoming sections for explaining system functions:

Cache capacity, C_{rec}

Record count in service response, S_{rec}

Record count per page in pagination table, P_{rec}

Count of pages displayed to user, P_{page}

Pages fetched per service call, $S_{page} = \lfloor S_{rec} / P_{rec} \rfloor$

Max. pages stored in cache, $C_{page} = \lfloor C_{rec} / P_{rec} \rfloor$

Service calls for initial cache-load = $\lfloor C_{page} / S_{page} \rfloor$

Pages in forward & backward buffers = (Cpage - Ppage)

Service Calls in Forward Navigation

After initial cache load, system would identify the page positions, where service calls would be made during forward navigation and backward navigation.Service call positions during forward navigation can be computed using following scheme:

Once a user performs search function, aset of service calls asynchronously fill the client-side cache. Number of service calls required, $N_s = \lfloor$ Cpage/ Spage $\rfloor$

After initial data load into cache, S_{page} number of pages at beginning of cached pages would be visible to user and forward buffer would contain, $N_f = (C_{page} - S_{page})$ number of pages.

As user navigates forward to n^{th} page, where $S_{page} \leq n\ N_f$ the backward buffer would contain $(n - S_{page})$ pages, the visible part would have Spage and the forward buffer would contain $(C_{page}$ - n) pages.

When the number of pages in forward buffer drops below S_{page}, a service call is triggered to fetch S_{page} number of pages from service. Since cache capacity is fixed, so S_{page} number of pages are unloaded from backward buffer.

As forward navigation continues, service calls are triggered at every S_{page} intervals, till P_{total} - thpageis fetched.

Number of residual pages that are yet to be fetched after initial cache load = $(P_{total} - C_{page})$

Number of service calls needed to fetch the residual pages = $(P_{total} - C_{page})$ /

Let's denote A = (Cpage Spage) + 1 andB = $(P_{total}$ -2 $S_{page})$

Total number of service calls required for forward navigation of all records, $\lceil N_s = (B - A)/ S_{page} \rceil$

The sequence of page positions would be:

$\{A+(\chi \times S_{page}) : \chi \epsilon \{0,1.....Ns\}\}$

Service Calls in Backward Navigation

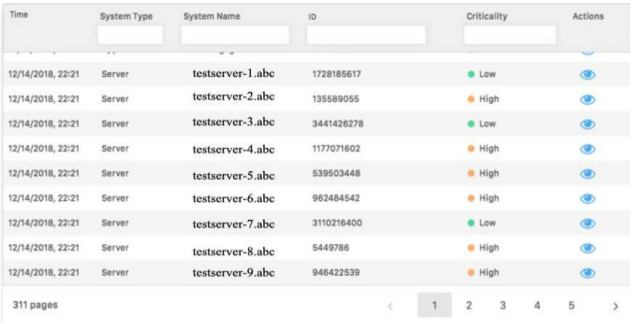
When forward navigations result in removal of cached pages from backward buffer, backward navigations trigger service calls at specific page positions to repopulate the relevant pages to backward buffer and remove pages from forward buffer.

The sequence of page positions for service call would be:

$\{\chi \times S_{page} : \chi \epsilon \{(N_s\text{-}1), (N_s\text{-}2), ..., 3,2\}\}$

Ellaboration with Example

Let's assume a user performs a search in web application which provides search result depicted in Figure 3. The paginated Table shows 5, pages at a time and there are 311 pages in total. It shows the forward and backward navigation buttons at bottom-right corner.



Time	System Type	System Name	ID	Criticality	Actions
12/14/2018, 22:21	Server	testserver-1.abc	1728185617	Low	
12/14/2018, 22:21	Server	testserver-2.abc	135589055	High	
12/14/2018, 22:21	Server	testserver-3.abc	3441426278	Low	
12/14/2018, 22:21	Server	testserver-4.abc	1177071602	High	
12/14/2018, 22:21	Server	testserver-5.abc	539503448	High	
12/14/2018, 22:21	Server	testserver-6.abc	962484542	High	
12/14/2018, 22:21	Server	testserver-7.abc	3110216400	Low	
12/14/2018, 22:21	Server	testserver-8.abc	5449786	High	
12/14/2018, 22:21	Server	testserver-9.abc	946422539	High	

311 pages 1 2 3 4 5

Figure 3.Sample paginated data

Configuration Parameters

Suppose, the system (client-side) is configured with following parameter values:

Cache capacity, $C_{rec} = 2500$

Record count in service response, $S_{rec} = 500$

Record count per page in pagination table, $P_{rec} = 50$

Count of pages displayed to user at a time, $P_{page} = 5$

Following configuration parameters are derived from above parameters:

Pages fetched per service call, $S_{page} = S_{rec}/P_{rec}$

$= \lfloor 500/50 \rfloor$

$= 10$

Max. pages stored in cache, $C_{page} = C_{rec}/ P_{rec}$

$= \lfloor 2500/50 \rfloor$

$= 50$

Service calls for initial cache-load = C_{page}/ S_{page}

$= (50/10)$

$= 5$

Pages stored in buffers (forward + backward) = (C_{page} - P_{page})

= (50 5)

= 45

Suppose, total qualifying recordsfor a given search condition = 15520, then page count in pagination table would be,

$P_{total} = \lceil 15520/ P_{rec} \rceil$

$= \lceil 15520/ 50 \rceil$

= 311.

Analysis on Service Calls and States of Paginated Data

Forward Navigation

For above mentioned example the states of the paginated data and cache buffer content would be as below. Figure 4 depicts this in detail:

Afterinitial data fetch by client when user performs search:

Visible pages to user = 1st to 5th

Pages in forward buffer = 6th to 50th

Pages in backward buffer = 0

When user navigates to (Cpage Spage) = 40th

Page the state would be as following:

Visible pages to user = 36th to 40th

Pages in forward buffer = 41st to 50th

Pages in backward buffer = 1st to 35th

When user navigates to (C_{page} - S_{page}) + 1 = (50 - 10) + 1 = 41st page, a service call is triggered to load next S_{page} = 10 pages. At the same time S_{page} = 10 pages are removed from backward buffer to avoid cache overflow. The state of the system would be as below:

Visible pages to user = 37th to 41st

Pages in forward buffer = 42nd to 60th

Pages in backward buffer = 11th to 36th

For identifying the pages where service calls are triggered, let's use the derived parmeters from Section C.3.

A= (C_{page} - S_{page}) + 1 = (50 - 10) + 1 = 41

B = (P_{total} - 2 × S_{page}) = (311 - 2 × 10) = 291

$N_s = \lceil (B - A)/ S_{page} = (291\ 41)/ 10_c \rceil = 25$

The sequence of page positions for service call would be as follows (from 2nd service call onwards after user performs search):

$\{A + (\chi \times S_{page}): x\epsilon \{0,1.....N_s\}\}$

= [41, {41 + (1 10)}, {41 + (2 10)}, {41 + (3 × 10)}..., {41 +(2.5 × 10)}]

= [41, 51, 61, 71....., 291]

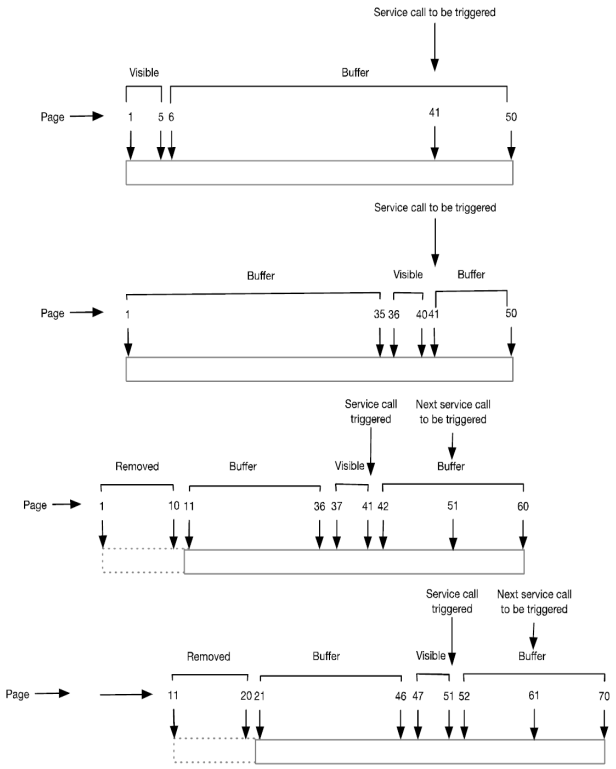


Figure 4.Pagination table, cache states and page positions for service call during forward navigation

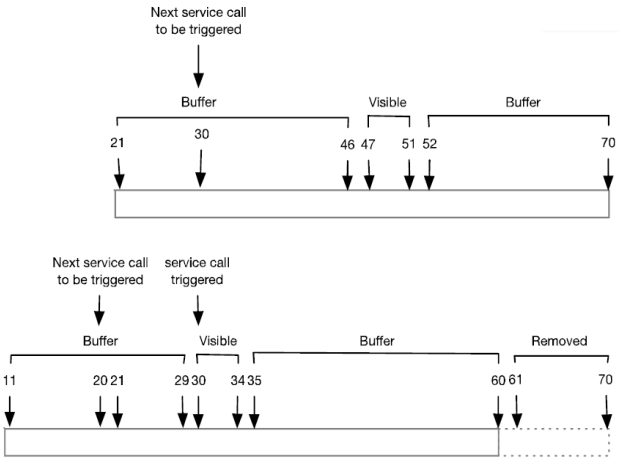


Figure 5.Pagination table, cache states and page positions for service call during backward navigation

Backward Navigation

Once the user has performed enough forward navigation so that the cache contents are updated (i.e. service calls ≥ 2 have already taken place) the backward navigation starts triggering service calls at specific pages to update cache content.

When user navigates backward to 30th page, the state becomes as below:

ISSN: 2582-5607

Visible pages to user = 30th to 34th

Pages in forward buffer = 35th to 60th

Pages in backward buffer = 11th to 29th

When user navigates back to 20thpage, the state becomes as below:

Visible pages to user = 20th to 24th

Pages in forward buffer = 25th to 50th

Pages in backward buffer = 1st to 19th

Figure 5, shows the corresponding details.

$\{\chi \times S_{page} : \chi \epsilon \{(N_s\text{-}1), (N_s\text{-}2), ..., 3,2\}\}$

$= [(25\text{-}1) \times 10, (25\text{-}2) \times 10,...,3 \times 10, 2 \times 10]$

$= [240, 230,..., 30, 20]$

Benefits

The proposed solution has both primary and auxiliary benefits. It helps in addressing the challenges associated with client-side pagination requirement for large set of search results.

The number of service calls required with this approach is low compared to any alternative approach. Extensive use of configurable parameters provide a complete control to system owners/ administrators on performance tradeoffs.

Client-side cache, with dynamic split of forward buffer, backward buffer and visible window of data, helps in faster page navigation.

Pre-load strategy in forward/ backward buffer segments help in seamless navigation experience to users.

Storage of a fixed number of pages in cache (cache capacity), helps in gaining full control on resource utilization. Fixed and configurable cache size helps in achieving a steady application performance.

Limitation

This solution works very well for gradual navigation of pages in forward or backward direction, but it would not be able to provide users Service calls are triggered at following page positions with first and last page link in pagination table.

Forward/ backward navigation keeps triggering service calls at specific pages and load/ unload of data continuous updating cache content. As a result, first page and last page content are not maintained all the time in cache.

Conclusion

Design of client-side pagination strategy is a challenge to system designers especially for handling large set of data. There are well known algorithms and approaches for handling scale oriented challenges related to pagination at server-side, but a separate strategy for client-side pagination need to be in place to accomplish an end-to-end success. The proposed solution explained in this paper helps in addressing the client-side pagination challenges. This solution follows a well-orchestrated service call and page management strategy. Extensive use of configurable parameters help in gaining full control on performance trade-offs.

Future Work

Additional capabilities, for support of first and last page in pagination table, would provide additional value in user experience. Further analysis can be performed related to performance and scale.

References

1. Oleg M. Ajax programming with Struts 2. Network World, Inc. 2019.
2. Lyndon B. Perfect PHP Pagination. Site Point. 2019.
3. Jack M. Offset and Cursor Pagination explained. 2019.
4. Technical Servies. The art of pagination - Offset vs. value based paging. Novatec. 2019.
5. Cao J, Wang W, Shu Y. Comparison of Pagination Algorithms Based-on Large Data Sets. In: Qi L. (eds) Information and Automation. ISIA 2010. Communications in Computer and Information Science, 2011; 86.
6. Sven L. Pagination - Examples and Good Practices. Smashing Magazin. 2019.
7. Abdul-Rahman A, Gimson R, Lumley J. Automatic Pagination of HTML Documents in a Web Browser. 2009.
8. Hu M, Kuang Y. Human-Machine Interface: Design Principles of Pagination Navigation in web applications. ICCSE 2014. The 9th International Conference on Computer Science & Education. 2014.
9. Wang P, Xi Y, Ma L et al. Research and Implementation of Pagination Algorithm over Massive Data Based on Ajax Technology. *IEEE Xplore* 2009.
10. Bootleg. How Paging Improves or Worsend your website. Moz. 2019.
11. Manjoo F. Stop Pagination Now. Slate 2019.
12. Eder L. Why Most Programmers Get Pagination Wrong. D Zone. 2019.
13. Piller M. How to Efficiently Load Large Data Sets in a Mobile App With Data Paging. Backendless. 2019.