# Pagination and Streaming Data within Distributed Database

In this post I would like to share my painful experience dealing with large amount of data hosted on Hadoop using Impala.

For who doesn’t know what is Impala or Hadoop checkout the following link

Problems with Distributed Database

1. Standard Pagination (Take and Skip Functionalise) with large dataset leads high latency and potential increase hardware cost (Memory)
2. Time becomes a major factor to delivery of complete message
3. Large Entity is being prepared for http wrangling

## Quick overview about Distributed Database

A **distributed database** is a [database](https://en.wikipedia.org/wiki/Database) in which [storage devices](https://en.wikipedia.org/wiki/Computer_data_storage) are not all attached to a common [processor](https://en.wikipedia.org/wiki/Processor_(computing)).[[1]](https://en.wikipedia.org/wiki/Distributed_database#cite_note-1) It may be stored in multiple [computers](https://en.wikipedia.org/wiki/Computers), located in the same physical location; or may be dispersed over a [network](https://en.wikipedia.org/wiki/Computer_network) of interconnected computers. Unlike [parallel systems](https://en.wikipedia.org/wiki/Parallel_computing), in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely coupled sites that share no physical components.

Let’s move straight to the problem; the promise is for all a distributed query system the O(cost) of a single pagination request , equivalent to requesting the entire dataset.

* Let’s see what Cloudera Team is suggesting

In Impala 1.2.1 and higher, you can combine a LIMIT clause with an OFFSET clause to produce a small result set that is different from a top-N query, for example, to return items 11 through 20. This technique can be used to simulate "paged" results. Because Impala queries typically involve substantial amounts of I/O, use this technique only for compatibility in cases where you cannot rewrite the application logic. For best performance and scalability, wherever practical, query as many items as you expect to need, cache them on the application side, and display small groups of results to users using application logic.

<https://www.cloudera.com/documentation/enterprise/5-9-x/topics/impala_offset.html>

In my opinion this wouldn’t work for long term solution as long as moving the problem from one to another, specially adding high complexity into the client, especially more client you have more your client will suffer this issue.

This issue applies not only for Impala, but for all distributed systems.

Another example of distributed data using Elastic search ?

To understand why deep paging is problematic, let’s imagine that we are searching within a single index with five primary shards. When we request the first page of results (results 1 to 10), each shard produces its own top 10 results and returns them to the coordinating node, which then sorts all 50 results in order to select the overall top 10.

Now imagine that we ask for page 1,000—results 10,001 to 10,010. Everything works in the same way except that each shard has to produce its top 10,010 results. The coordinating node then sorts through all 50,050 results and discards 50,040 of them!

You can see that, in a distributed system, the cost of sorting results grows exponentially the deeper we page. There is a good reason that web search engines don’t return more than 1,000 results for any query.

<https://www.elastic.co/guide/en/elasticsearch/guide/current/pagination.html>

Node 0

Coordinator

Select x,y,z From Table

Order by x

Offset 100 LIMIT 100

Node 1

Node 2

Node 3

Node 4

Node 5

In this scenario the client would like to get 100 rows and skip 100 checkout the next figure what exactly is happening

Node 2

Node 2

Node 5

Node 4

Node 3

Node 2

Coordinator

Each node returns top 200 records then the coordinator start to process the client request which means sorting 200 \* 6 = 600 rows by X then return 100 rows.

You can see that, in a distributed system, the cost of sorting results grows exponentially the deeper we page. There is a good reason that web search engines don’t return more than 1,000 results for any query.

I hope know you have clear Idea why Pagination is pain in the ass for distributed system;

Lets back to the main problem, which are the possible solutions

Backing to my case I would like to paginate huge data without losing big performance

How this work in SQL world

The way Offset

*…the rows are first sorted according to the <order by clause> and then limited by dropping the number of rows specified in the <result offset clause> from the beginning…*

Solution

Ingest your data into Hadoop by creating unique incremental Id (checkout this link http://community.cloudera.com/t5/Interactive-Short-cycle-SQL/Sequence-number-generation-in-impala/td-p/38126), this will drive you to use range filter.

So this means

Rather than using [offset]

Where going to use a query such as

{

Select x,y,z From Table

Where id >= 0 and id <= 100

}

## Problems

We cannot rid out of pagination when we have large dataset, but we would like to perform better ☺ Based the theory described above in order to make this happen we MUST avoid to use Take and Skip functionalities.

The way Offset

*…the rows are first sorted according to the <order by clause> and then limited by dropping the number of rows specified in the <result offset clause> from the beginning…*

First case scenario using keyset pagination I would recommend you to have a look the following links

<http://use-the-index-luke.com/no-offset>

Let’s see the other two problems we have, which are

* Time becomes a major factor to delivery of complete message
* Large Entity is being prepared for http wrangling