Sequential data access with Oracle and Hadoop: a performance comparison

Zbigniew Baranowski, Luca Canali, Eric Grancher
European Organization for Nuclear Research, IT Department, Database Group
Geneva, Switzerland
zbigniew.baranowski@cern.ch, luca.canali@cern.ch, eric.grancher@cern.ch

Abstract. The Hadoop framework has proven to be an effective and popular approach for dealing with “Big Data” and, thanks to its scaling ability and optimised storage access, Hadoop Distributed File System-based projects such as MapReduce or HBase are seen as candidates to replace traditional relational database management systems whenever scalable speed of data processing is a priority. But do these projects deliver in practice? Does migrating to Hadoop’s “shared nothing” architecture really improve data access throughput? And, if so, at what cost? Authors answer these questions—addressing cost/performance as well as raw performance—based on a performance comparison between an Oracle-based relational database and Hadoop’s distributed solutions like MapReduce or HBase for sequential data access. A key feature of our approach is the use of an unbiased data model as certain data models can significantly favour one of the technologies tested.

1. Introduction

Database schemas for online transaction processing are typically differently designed than schemas for data warehousing; this is because issuing a data warehouse queries on a database schema modelled for online transaction processing can be inefficient [1]. Indeed joining very large tables is a complex operation, de-normalised tables are often a preferred approach. Maintenance of indexes consumes a significant amount of space and resources while their usability is limited due to large sorting operation being required during the consolidation of partial results obtained from accessing large indexes. These kind of problems have affected the ATLAS Event Metadata Catalogue [2] running on a relational database. This is why the ATLAS collaboration started searching for a different solution [3].

The example above shows that keeping a big data volumes in a relational schema also used for online transaction processing is challenging. This lead to the investigation and development of several products to solve big data problem more efficiently [4].

One of the basic approaches for performing ‘ad hoc’ analytics on big data sets is storing the data in a “schema-less” flat structured format. This ensures a simple data access however it very often results in storing a lot of redundant data and moreover it forces to scan the entire data set during any data access operation every time even if only a certain piece of information is needed to be extracted. However thanks to its simplicity it can be easily distributed and scaled up which delivers a very good performance. This is why in this paper we will concentrate on the evaluation of this type of data processing using two different technologies: Hadoop and the Oracle database.

Hadoop is a popular open-source framework used for distributed processing of large data sets [5]. It consists of a distributed cluster file system (HDFS) and the MapReduce infrastructure that allows
executing of data processing jobs and distributing them among cluster nodes. A key feature of MapReduce [6] model is to deliver processing to the data; this eliminates the high network traffic between machines in the cluster. Additionally the Hadoop framework offers a NoSQL database on top of HDFS named HBase [7] that is an implementation of a distributed key value store. SQL has been developed on top of Hadoop as described in [4].

Advanced relational database management systems like Oracle running in a cluster mode (Real Application Clusters) can deliver a very good sequential data access speed thanks to the feature called Parallel Query which is able to spread a data processing cluster wide [8] [9].

As interest in Hadoop is growing at CERN (see ATLAS work [2]) and in some areas it appears as a serious candidate that could replace or complement a relational database, the authors of the paper would like to make a comparison of a sequential data accessing performance of both technologies by discussing results of series of comparable tests conducted on Hadoop and Oracle. In order to make the comparison fair and to be sure that apples were being compared with apples mentioned tests were run on exactly the same hardware. The results of these tests should also answer which technology scales better or utilises resources more efficiently.

2. Test setup
This section describes the hardware specification and software configuration of the testbed used during the exercise.

2.1. Hardware configuration

![Figure 1. Hardware configuration used for testing Hadoop and Oracle](image)

Our test setup was composed of 5 servers and 5 storage arrays connected via Storage Area Network using two Fibre Channel switches. Each server in the configuration was equipped with 2 quad core Intel Xeon 2.33GHz CPUs, 16 GB of memory and two 4Gbps FC controllers and each of them was connected to a separate FC switch. All 5 disk arrays had 12 disks with a maximum speed of around 80 MB per second and two 4Gbps FC controllers separately connected to both FC switches. On all disks in the arrays two same-sized partitions were created in order to store the Hadoop data on the first one and the Oracle data on the second. According to the hardware specification, Host Bus Adapters installed on the servers had the maximum speed of 8Gbps and both FC controllers attached to them could work in parallel at full speed; in the case of the storage machines, the HBA installed had the maximum throughput of 4Gbps which means that the total speed of a single array was limited by the speed of this hardware element. Considering the hardware configuration we can conclude that the maximum IO throughput that can obtained out of it is limited by HBAs installed in storages and it is equal to 5 x 4 Gbps = ~2,5 GB/s.
In order to implement a shared nothing architecture [10] for Hadoop we had configured the zoning on FC switches in a way that a single disk array was accessible by only one server. So in total we had to set up 5 server-storage pairs. FC switches were not a bottleneck in such configuration so we had set up a Storage Area Network architecture with 4 Gbps link between a server and 12 disks in a single array.

Configuring shared disk architecture for Oracle RAC is more natural with FC infrastructure than shared nothing as it only requires opening connectivity between all servers and all disk arrays. For this architecture the maximum IO throughput between servers and storages is equal to the maximum capacity of the hardware used in this test which is ~2.5GB/s as already mentioned above.

2.2. Software configuration
For the Hadoop testing we had installed on all 5 servers Datanode and Tasktracker software in 1.0.3 version and HBase Region Server in version 0.92.1, all taken from the Hortonworks distribution (Hortonworks Data Platform 1.1). All disks partitions dedicated for Hadoop were formatted with the ext3 file system beforehand and mounted with the noatime flag. Once the Hadoop cluster was up and running we applied some parameter tuning based on results of our initial test as well as on the best practices reported by Hadoop communities; the number of map and reduce slots available on each server were adjusted to 8 to match the number of available CPU cores on each machine. Also at the same time we fixed a task’s JVMs heap size to 768MB to fit in available memory. The HDFS file block size was increased to 1GB as it reduces the amount of map tasks to be created and increases efficiency of a sequential data access from a single device as a larger continuous chunk of a data can be read at once. In order to achieve data locality in the MapReduce processing we have configured Delayed Fair Scheduler [11] with a delay of 30 seconds. Since starting of a new map task was taking around 3 seconds we have enabled reusing of JVMs, configured a TaskTracker to send extra 'out of bench’ heartbeat to a JobTracker whenever any task has completed, we moved temporal job directories from data dedicated devices to local drives being used by an operating system; all those changes managed to reduce the starting time to less than 1 second.

For the Oracle evaluation we installed the latest 12.1 version of the Real Application Clusters database on each test machine. As a storage management solution we configured Oracle Automatic Storage Management with the allocation unit set to 4MB as it is the recommended size for storing big data sets. In order to increase the IO throughput between a single server session and the storage the ‘db file multiblock read count’ parameter was set to 512. The rest of settings including enabling of asynchronous IOs were done according to the best practice recommendations by Oracle [12].

2.3. Data set and test case
As in our test we wanted to use unbiased data we could have used a completely generic data set, just big enough and semantically the same for Hadoop and Oracle. But it was more beneficial for us to use a data with some meaning, otherwise we would not be able to crosscheck if our processing was returning right results. So in the end a collection of some physics data was used as the source set for analytics. The collection contained a catalogue of muon with theirs physics attributes expressed by integer and floating point values. For more details about the origin of data please refer to the work reported in [13]. The data in Comma Separated Value format had 532 millions of lines and 244 attributes per line and in total it occupied almost 1TB (992 MB) of space.

In our benchmarking analytic case we were counting ‘exotic’ muons in the collection. This selection is based on values of 2 attributes for each muon. The query written in SQL would look as (1) (where '/*+parallel */' statement is a hint enforcing using of Parallel Query engine and $x$ a parallelism degree).

\[
\text{select } /\ast+ \text{ parallel}(x) \ast/ \text{ count}(*') \text{ from istest.munon1tb where } \text{col9}\geq100000 \text{ and col20}=1; \tag{1}
\]
3. Testing
Testing of Oracle and Hadoop with our benchmarking analytic processing has been divided into 3 steps: loading the data to the cluster, optimizing the workload on a single machine, scaling up on the cluster.

3.1. Loading the data
Except using pure HDFS where storing files in a CSV format is the most natural and common practice, there are few possibilities for storing a semi-structured data like a CSV formatted file in a database table. Assuming that each row of a table represents a single line (entity), the easiest and universal way is to store each line from a file as one text column in a table. But such data representation requires parsing of a text row in order to retrieve an entity attributes every time it is being accessed. This problem can be avoided by using a multi-column table and storing each attribute of an entity in a separate column. Furthermore if attributes have a different data type than text (like integer, float in our case) it might be beneficial to store them in native formats by choosing appropriate columns data types. This is because there is no need of type conversion (from text) whenever an attribute is being accessed. Also it should result in more efficient space utilisation thanks to the usage of the optimal data representation. On top of each discussed variants a table or a file compression can be applied to reduce the final data set volume. In Oracle we have used the built-in columnar compression solution. For storing data files in Hadoop the LZO codec was used as it is optimized for speed [14]. Table 1 and Table 2 contain the information about the amount of a physical space that was occupied by the same data set after storing it on Oracle and Hadoop (including HBase) in various structures definitions discussed above.

The best results in terms of the data volume we managed to get out of Oracle when using a compressed multi-column table with adapted column types (418GB) The worst result was given by HBase when using of a multi-column table. As it occupied significantly more space than the original CSV data set and storing it was much more time consuming we have finally skipped it in our tests.

Table 1. Volumes of 992GB CSV file stored in different tables definitions in an Oracle database.

| Table/storage type                | Size    |
|----------------------------------|---------|
| Single column [Text]             | 1204 GB |
| Multi column [Numeric]           | 580 GB  |
| Multi column [Numeric] compressed| 418 GB  |
| Multi column [Text]              | 1203 GB |
| Multi column [Text] compressed   | 956 GB  |

Table 2. Volumes of 992GB CSV file stored on Hadoop in various data representations.

| Storage type                        | Size    |
|-------------------------------------|---------|
| HDFS file                           | 992 GB  |
| HDFS compressed with LZO            | 568 GB  |
| HBase single column table           | 1011 GB |
| HBase single column table compressed with LZO | 586 GB |
| HBase multi column table            | ~7 TB   |

3.2. Optimizing the workload on a single machine
In case of Hadoop we were using a single pair of a server and disk array and we tried to obtain the best processing throughput. For Oracle we used a single machine connected to a shared storage which was
composed of 5 disk arrays. On this part of the exercise we definitely spent more time optimizing Hadoop than Oracle.

This is because executions of our initial MapReduce code were very slow. A throughput of 202MB/s was far from our expectations as it was less than half of the maximum of what our single server hardware configuration can deliver. After checking the history of the CPU utilization on the server it was clear that our MapReduce processing was dominated by the time spent waiting on a CPU. In order to distinguish if this is a problem of a suboptimal Java code that was ran by us or maybe by misbehavior of the Hadoop’s internals we ran another set of tests with an empty body of the map function. This resulted in the throughput of 465 MB/s which was very close to the hardware limits. At this speed we were able to read the entire data set with a single machine within 36 minutes when originally it was taking 83 minutes. This test had proven that at that time something was inefficient in our initial Java implementation. At this stage we tried to move processing from Java to something else. Hadoop enables such functionality by using so called Hadoop Streaming. The feature enables to use any executable (that accept data in standard input and produce an output) for mapping and reducing phases. We had chosen Python to make a quick implementation of our benchmark analytics. Thanks to the change of source code language, the processing throughput increased from 202 MB/s to 315 MB/s (processing time was reduced to 53 minutes). The MapReduce job was still bounded by a CPU. Finally we moved back to the Java code and tried to do some manual profiling of the code. It turned out that the majority of CPU cycles were spent on splitting a data line into comma delimited records using standard split function available in the String class. According to the function description regular expressions are being used in splitting process so this may explain why it intensively uses a CPU. Replacing the function with standard StringTokenizer class did not give much difference. So we decided to write our own implementation of a string splitting which we embedded in the map function body. The change let us to obtain the data processing throughput of 420 MB/s which effectively reduce the time needed for getting the result to 40 minutes. This was quite close to the speed of read only jobs and at the same time to the limits of the hardware. In this case the time spent by processes waiting on CPU and IO operation was balanced.

Later we run the same implementation of the MapReduce job on a data compressed with LZO but since when reading of not compressed set we were already quite close to the CPU limits we were expecting that a decompression can significantly reduce the overall throughput. And in fact it happened, as the speed of 179 MB/s was the maximum that we were able to measure. Since the compressed data volume was more than half smaller the job needed 51 minutes to complete which is a promising results comparing to other approaches mentioned above. This proves that using of LZO compression is worth considering as it may reduce the data volume significantly without many penalty points added the processing speed. All MR performance results have been illustrated in Figure 2.

We have found out that using of standard scanner interface for reading data stored in HBase tables is very suboptimal. This is because by default a data access is performed with a single process that has to scan the entire data set sequentially. One of the ways to turn an HBase data accessing into a parallel operation is by running a MapReduce job against an HBase table. Since we had already implemented an optimized version of the MapReduce code we followed this direction. But in the end we have gotten much worse results comparing with those obtained on HDFS files. For uncompressed table the maximum throughput that we measured was 125 MB/s which maps to 138 minutes needed to process the entire set. In case of compressed table it was not better: 142 minutes with the speed of 68 MB/s. In both cases processes were IO-bound. Since there are no obvious tuneable options we have not tried to optimize further the workload.

Regarding an Oracle optimization, apart from parameters tuning mentioned in section 2.2 no other modifications had been done before running the benchmarking workload. With each data representation listed in Table 1 we easily manage to obtain the maximum throughput of the server’s FC controllers (2 x 4 Gbps) equal to ~750 MB/s as presented on Figure 3. However due to a different table sizes processing times were not equal. The best time belonged to the compressed table with numeric data (9 minutes and 30 seconds). The same table without compression was processed slightly
slower (13 minutes and 29 seconds). Finally tables with text data were scanned much slower (27 minutes for both). However the compressed version of the multi column text table was a bit faster (21 minutes). In all of the cases the data processing was IO-bound.

3.3. Scaling up

After obtaining optimized workload performance on a single machine we were scaling up the cluster for both shared nothing and shared disk architectures. In both cases we were getting the best results when the parallelism degree on a single server was equal to 8 which is the number of CPU cores available.

According to measured throughputs, processing of an HDFS file with MapReduce framework scales almost linearly (Figure 4). With 5 servers we managed to obtain the speed of 1516 MB/s (processing time 10 minutes and 50 seconds) for the uncompressed file and 895 MB/s (processing time 11 minutes and 10 seconds) for the compressed one. The sequential data accessing with MapReduce running against HBase did not scale much in our tests. The maximum throughput that we managed to get was 297 MB/s (1 hour needed) and 192 MB/s (52 minutes needed) for not compressed and compressed tables respectively.

The addition of 4 servers to the Oracle cluster delivered more diverse results than in the Hadoop’s case. None of the data representations have given linear scalability of the performance (Figure 5). The best result that we managed to get was while reading from a compressed multi column table with a numeric data – it was done in 4 minutes and 25 seconds at a speed of 1618 MB/s. Second best result was delivered by the same table but uncompressed – 7 minutes at a speed of 1421 MB/s. The single column text type table had similar throughput – 1457 MB/s but there was a much bigger volume to process this is why overall time was 14 minutes. The worst result was measured for the multi column text type table – 16 minutes (1275 MB/s), however compressed version of it performed much better – 11 minutes (1400 MB/s).
4. Results

In the previous section the results of a sequential data access throughput were presented for both technologies running on the same hardware. In this section we would like to directly compare obtained results talking into account various aspects.

4.1. Throughput

If we focus only on a throughput (see Figure 6) we can definitely conclude that both Hadoop and Oracle can deliver similar results. Oracle may be slightly faster if the data set consists of numeric values. Otherwise for a text processing Hadoop was delivering minimally a better throughput. In both cases we did not reach the theoretical maximum throughput of the hardware. We could observe that the main reason of that was a disk read contention which means that neither Oracle nor Hadoop’s MapReduce was able to optimally schedule a concurrent storage access.

4.2. Processing time

Data access speed has a great impact on the overall data processing time. But it is not the only factor that determines it as the same data set stored in different formats (binary representation, compression) can have not equal volume size. Moreover, different data representation of the same set can also have a different processing speed too (parsing of text record, decompression). So even though some approaches had very good data processing throughput it does not necessarily mean that they were the fastest. This is why it is important to also compare a time needed for Oracle and Hadoop to...
process same data set. According to the results of our tests Oracle had the fastest time (see Figure 7)
thanks to the numeric nature of the data which allowed accessing them in the optimal way. We managed to process the compressed table containing a numeric data with Oracle Parallel Query in 265 seconds when the best result measured on Hadoop was 650 seconds done with the compressed HDFS file. For the uncompressed one it was just 20 seconds slower. If the optimal representation for the data stored in Oracle cannot be found the results have shown that the processing of a universal representation (which is a compressed single column text type table) took 700 seconds so it was 50 seconds slower than Hadoop.

4.3. Resource utilisation
During the tests, Hadoop was always utilising 100% of the available memory and in average around 90% of a CPU time. High CPU utilisation may have a significant impact on the performance of a more complex analytic processing. In contrast Oracle was always IO bound and the CPU utilisation was not higher than 20% in any of cases. Also the amount of used memory did not cross the limit of preliminarily configured 8 GB.

4.4. Costs of deployment
Not negligible aspect of the comparison for technologies is the cost of a deployment. Hadoop is under an open source license so it is available for free. However, software support must be paid for. Oracle is a commercial software for the enterprise world and its software license and support fees are currently priced on the high end of the scale. This is why a comparison with Hadoop on this field is difficult and should be done case by case rather than comparing official price lists. Another part is with costs of a hardware acquisition. In this case costs purchasing and maintaining of shared nothing architecture are significantly lower than shared storage because there is no need of installing an extra high speed network between cluster servers and storage infrastructure.

4.5. Going beyond 5-node cluster
In section 3.3 a very good scaling up capabilities of Hadoop were emphasized. We were so interested about its scaling trend that we decided to use an opportunity to test it on a larger scale. So we repeated the same set of tests on a 50-node Hadoop cluster installation where we managed to measure the throughput of 9069 MB/s (the data processing took 112 seconds) on the “not compressed” data set and 5107 MB/s (the data processing took 114 seconds) on the compressed data set (see Figure 8).

![Figure 8. Hadoop MapReduce throughput on 50-node cluster](image)

Hardware specification:
- 2 x 8 x Intel Xeon 2.00 GHz CPU with HT,
- RAM: 128 GB,
- Storage: 3 SATA drives 7200rpm

We have not run the same test for Oracle as a shared storage configuration was not available on the cluster at the time. But because of shared disk architecture is limited by the throughput of the storage interconnection, it would be difficult to get the similar performance for Oracle as those obtained with Hadoop.
5. Conclusions

We have conducted a series of tests with the Oracle database and Hadoop on the same hardware doing the exact same type of sequential data processing which is a typical method of data accessing for big data warehouses. The results have shown that on a 5-node cluster both technologies perform similarly for this type of workload. However, Hadoop has proven that it scales much better than Oracle in our configuration.

During the testing we did not manage to get MapReduce performing on HBase as well as HDFS files or the Oracle database. This may be explained by the fact that HBase was designed to deliver good performance of data accessing by a row key, however it was interesting to observe that during our tests for sequential data access it was following behind so significantly.

Our experience with optimising MapReduce Java code has proven that writing efficient code MapReduce code for Hadoop is not trivial and there are many pitfalls that can significantly degrade overall performance. We have described an example for one of them.

We have observed that Hadoop processing uses CPU very intensively and memory even when running a simple processing. It should be noted that this software is under very active development and several of the observed problems including excessive resource utilisation or suboptimal task scheduling are being addressed in new releases. Newer more space efficient storage format are also being implemented like Parquet [15]

The work presented in this article has demonstrated that Hadoop is an interesting, heavily scalable alternative to the Oracle database in the field of data warehousing and processing large amount of data in a batch mode. However it is worth to remember that at the same time Hadoop still cannot replace a traditional relational database to support transactional multi-user workloads.

References

[1] Microsoft: http://technet.microsoft.com/en-us/library/cc917548.aspx
[2] The ATLAS Collaboration (W. Ehrenfeld et al.), 2011, Using TAGs to speed up the ATLAS analysis process (J. Phys.: Conf. Ser. 331:032007)
[3] Barberis D et al, 2014, The ATLAS EventIndex: an event catalogue for experiments collecting large amounts of data (to appear in J. Phys.: Conf. Ser., Computing in High Energy and Nuclear Physics 2013 International Conference)
[4] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel Abadi, Avi Silberschatz, and Alexander Rasin, 2009, HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. (Proc. VLDB Endow. 2, 1 , 922-933)
[5] Hadoop: http://hadoop.apache.org
[6] J. Dean and S. Ghemawat, 2004, MapReduce: Simplified Data Processing on Large Clusters (OSDI'04: Sixth Symposium on Operating System Design and Implementation)
[7] HBase: http://hbase.apache.org
[8] Hussain, Syed Jaffar, et al. 2013 "Parallel Query in RAC." Expert Oracle RAC 12c. (Apress. 353-380)
[9] http://docs.oracle.com/cd/E11882_01/server.112/e25523/parallel002.htm
[10] Stonebreaker M, 1986, The case for Shared Nothing Architecture (Database Engineering Vol 9)
[11] Matei Zaharia et al, 2010, Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling (EuroSys’10, April 13–16, 2010, Paris, France)
[12] Oracle Documentation: http://www.oracle.com/pls/db121/homepage
[13] Limper M , 2014 An SQL-based approach to Physics anaysis (to appear in J. Phys.: Conf. Ser., Computing in High Energy and Nuclear Physics 2013 International Conference)
[14] Tom White, 2012, Hadoop: The Definitive Guide, 3rd Edition (O'Reilly Media/ Yahoo Press)
[15] Parquet http://parquet.io/