2011 Workshops of International Conference on Advanced Information Networking and Applications

Performance Analysis of Hadoop for Query Processing

Tomasz Wiktor Wlodarczyk, Yi Han, Chunming Rong
Department of Computer Science and Electrical Engineering, University of Stavanger, Norway
tomasz.w.wlodarczyk@uis.no

Abstract— **Query processing using mostly various NoSQL languages becomes a significant application area for Hadoop. Despite significant work on performance improvement of these languages the performance dependence on basic configuration parameters seems not to be fully considered. In this paper we present a relatively comprehensive study into influence the basic configuration parameters have on performance of typical types of queries. We choose three queries from Lehigh University Benchmark that can represent the most typical challenges and we analyze their dependence on parameters such as: dataset size, number of nodes, number of reducers and loading overhead. The results indicate strong dependence on the amount of reducers and IO performance of the cluster, which proves the common opinion that MapReduce is IO bound. These results can help to compare performance behavior of different languages and serve as a basis for understanding the influence of configuration parameters on the final performance.**

Keywords-Hadoop, LUBM, query, performance, analysis, Amazon Web Services

I. Introduction

Query processing becomes a significant part of research and development efforts in Hadoop community. The main area consists of NoSQL approach, with some exceptions. While presenting and testing new developments various cluster configurations are used. Moreover, very often not all the parameters are disclosed e.g. number of reducers. Comprehensive performance analysis and comparison of various configurations seems to be necessary to be able to compare different results, choose optimal test settings, and also choose further research and development directions.

In our research in this field we have noticed however, that such comprehensive analysis seems to be absent. This results in difficulties in results comparison between various sources and makes the choice of best test setting at least not fully clear.

Therefore, in the scope of this paper we aim to conduct a comprehensive performance analysis for different types of queries for various configurations. We show the dependence on amount of processing nodes, types of processing nodes and amount of reducers for several sizes of datasets. We utilize three types of queries to focus on low selectivity, high selectivity and complex dependence pattern. The measurements are performed using Amazon Web Services as they provides reproducible and publically available test platform. In this context we also analyze data loading overhead.

Precise numeric values are less important with respect to the goal of this work than the trends that they visualize. Therefore all the results are presented in form of the graphs. However, numeric values can be obtained by contacting the authors.

Related Work. Dejun et al. [1] analyze response time and I/O performance of EC2. Garfinkel [2] demonstrates throughput and latency of S3 for various locations on Internet. Ibrahim et al. [3] evaluate Hadoop execution on Virtual Machines. Stewart [4] compares performance of several data query languages for Hadoop. These works, as they focus on a different level of performance analysis, can complement our work in further efforts of performance improvement.

Jiang [5] et al. study performance of MapReduce for database implementation with focus on SQL processing that utilizes, among others, indexing and fingerprinting based grouping. This is a comprehensive work that in many aspects is similar to ours. However, it focuses on a custom implementation created by authors and it assumes presence of indexing, grouping, etc., which is not a typical case in the majority of applications (i.e. NoSQL applications).

Newman et al. [6,7] show performance of their RDF store for 2 and 3 nodes cluster for different data sizes. Myung et al. [8] show performance for various cluster sizes for one size of data set and for various data sets for a fixed cluster size for authors' custom implementation of SPARQL processor. Husain et al. [9,10] demonstrate performance of their RDF graph processor on 10 nodes cluster. Sun and Jin [11] show performance of their RDF store on 11 nodes cluster for various datasets. Mika and Tummarello [12] show performance of their implementation of RDF query processing using Pig for two types of queries for three sizes of the dataset. Those work present partial results with focus on implementations created by the authors. Moreover, the study of performance influence of reducers is mostly ignored.

Contributions. The main contribution of this paper is the comprehensive performance analysis of Hadoop cluster configuration for NoSQL query processing. The experiments are performed for a wide range of different parameters to give an overview of factors that influence the performance for different query types. The additional contribution is analysis of overhead caused by data loading in Amazon Web Services, which are a popular location for Hadoop testing.

Organization of the Paper. After the Introduction, in Section 2 we describe and clarify all the main terms and concepts used in the paper. Description of test configurations, types of queries used and choice of query language can be found in Section 3. In Section 4, 5 and 6 we present results for respectively high selectivity, complex dependence pattern and low selectivity query. Results for data loading overhead are presented in Section 7. We conclude the main points in Section 8.

II. BACKGROUND

In this section we describe and clarify all the main terms and concepts used in the paper.

Hadoop. The Apache Hadoop[1] project develops open-source software for reliable, scalable, distributed computing. It is based on **MapReduce** paradigm introduced by Google in 2004 that allows creating complex distributed applications by applying set of map and reduce functions. Hadoop includes subprojects that provide: distributed file system, processing framework, query languages, etc. It is commonly used in industry and it is gaining popularity for scientific applications.

Query Languages. Query languages can base on different logic and data structure concepts. In Hadoop context the most commonly applied concept is called NoSQL basing its name on the differences to classic relational database systems that utilize SQL. Two most common examples of such query languages are Pig[2] and Hive[3]. In this paper we utilize another emerging language called Cascalog[4] due to its good fit for the test queries.

Amazon Web Services[5] **(AWS).** It is a collection of the remote computing and storage services provided on the on-demand basis. They include **Elastic Compute Cloud (EC2)** that provides computing power, **Simple Storage Service (S3)** that provides storage capabilities, **Elastic MapReduce (EMR)** that automatically implements Hadoop on EC2, etc.

Lehigh University Benchmark [6] **(LUBM).** The benchmark is intended to evaluate performance of Semantic Web repositories over large data set. It consists of set of 14 queries and data generator. Some of the queries, that are language independent, focus strictly on exposing performance of various data interrelation patterns that are not specific only to Semantic Web. For our analysis we focus on three of those queries as explained in detail in Section 3.

III. TEST SETUP

In this section we describe test setup with respect to cluster, query and data configuration.

All the tests presented in this paper were performed using Amazon Elastic MapReduce together with other Amazon Web Services. We have chosen this platform, as it is widely available, so other researchers are able to perform other tests using exactly the same conditions. Have we used our local cluster the results could be still valuable; however, to the lesser extent.

A. Cluster Configuration

Number of Nodes. All the test were performed for the total amount of nodes ranging from 2 to 20. It is important to notice that it means that the actual number of worker nodes is one less than the total number as one node is occupied by the name node. As it can be seen further 20 nodes occurred to be a satisfactory high number, as most often only limited results improvement is visible over 10 nodes.

Types of Instances. For this tests m1.small and m1.large instances were used as the Hadoop nodes. The main difference between them is the higher IO performance of the m1.large instance. Other parameters like CPU, memory and disk space also differ; however, IO performance seems to be the key to the results interpretation. We made a conscious choice not to utilize other, usually more powerful, types of instances for two reasons. First of all, more powerful instances in general offer a specialized set of parameters necessary for very specific applications e.g. big memory, multi-core processor. It is quite likely that some performance gain would be observed by utilizing them; however, their usage seems contradictory to our goal of providing generic and widely applicable results. Secondly, the cost of more powerful instances is significantly higher than the cost of m1.small and m1.large. We consider that an important argument in the context of one of the aims of MapReduce and Hadoop which is ability to run on a cluster of commodity machines.

Number of Reducers. The tests were performed for two different amounts of reducers: one reducer, the amount of nodes-1. We have actually performed tests also under different configurations; however, we do not present all of them here due to limited space. Those two configurations show in the clearest way whether performance of different queries depends on the number of reducers. However, to determine the optimal amount of reducers is a more complicated problem and it would require separate analysis, which we do not attempt here.

B. Query and Data Configuration

Choice of Queries. The tests were based on the data and queries from LUBM. This choice is not immediately obvious and it requires a few comments. LUBM is designed to test triple stores and generates RDF triples in XML format (that was transformed as explained later). It also requires some level of reasoning capabilities for some of the queries. However, several other queries do not require reasoning and they also interestingly expose potential

[1] http://hadoop.apache.org/
[2] http://pig.apache.org/
[3] http://hive.apache.org/
[4] https://github.com/nathanmarz/cascalog
[5] http://aws.amazon.com/
[6] http://swat.cse.lehigh.edu/projects/lubm/

performance issues in generic query scenarios. In particular, Query 1 focuses on high selectivity scenario where big amount of data is loaded and then selection is performed basing on the simple dependence between two files. Query 2 focuses on complex interdependency pattern between files. Finally, Query 14 demonstrates a low selectivity scenario. Those three queries allow analyzing influence of the amount of nodes, their parameters (mostly IO performance) and number of reducers on query execution times. One typical operation that is omitted is simple filtering; however, one can argue that as it could be implemented in the map operation it would not have a significant influence on the results.

Amount of Data. The tests were performed for 6 different sizes of datasets. They corresponded to LUBM data generated for 50, 100, 200, 500, 1000 and 6000 universities. The data were transformed to obtain the structure analogous to Abadi [13] that was the most convenient from the point of view of query execution. The smallest file size was 13.6MB for one of the files in Query 2 for 50 universities. The largest file was 2.9GB in Query 14 for 6000 universities. In the S3 load test we report only on the load times of files greater than 100MB to improve the clarity of the graph.

Query Execution. The test queries: LUBM 1, 2 and 14 were executed using Cascalog. We decided to utilize this query language as it seemed to have natural correspondence with the original LUBM query structure. However, those queries could be also easily executed using Pig or Hive. Some performance differences could be expected and they are partially covered in [4]. For all the scenarios data was first loaded from S3 to HDFS and further queried (read and written) from HDFS. Load times are reported in Section 7, query times in Sections 4 to 6.

IV. High Selectivity Performance

In this section we present and describe the performance of a query with high selectivity.

Basing on all the figures in the paper one can notice that biggest performance improvements can be seen up to 8 nodes, after what they become relatively small. By comparing Figure 1. with Figure 2., and Figure 3 with Figure 4. one can notice that the performance of high selectivity scenario is not significantly dependent on the amount of reducers. At the same time utilizing nodes with high IO performance provides significant benefits with time decrease of the level of 2 up to 4 depending on the configuration.

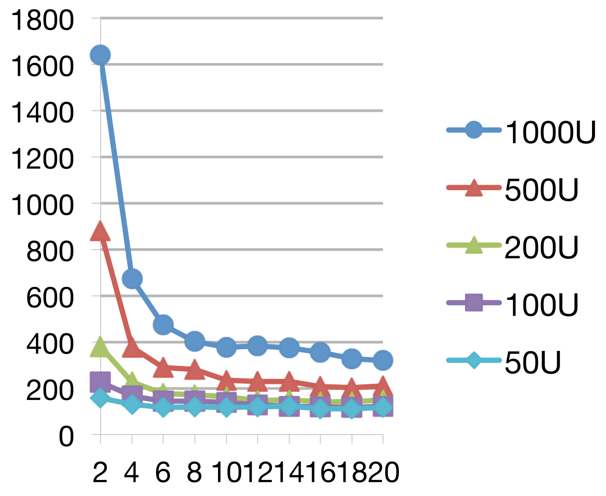

Figure 1. Execution time of Q1 in seconds dependent on the amount of nodes for 50 to 1000 Universites with 1 reducer for m1.small instances

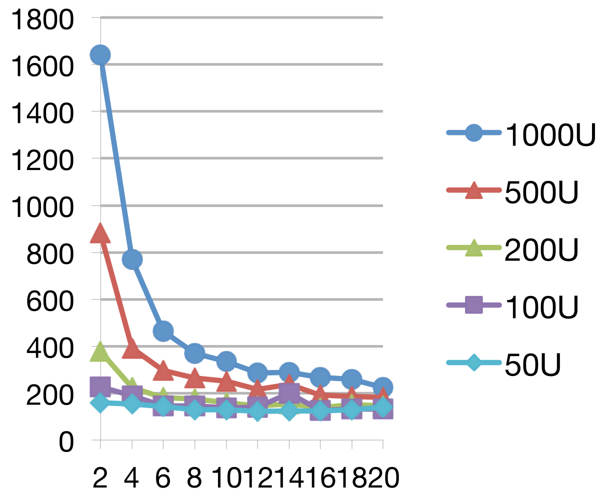

Figure 2. Execution time of Q1 in seconds dependent on the amount of nodes for 50 to 1000 Universites with nodes-1 reducers for m1.small instances

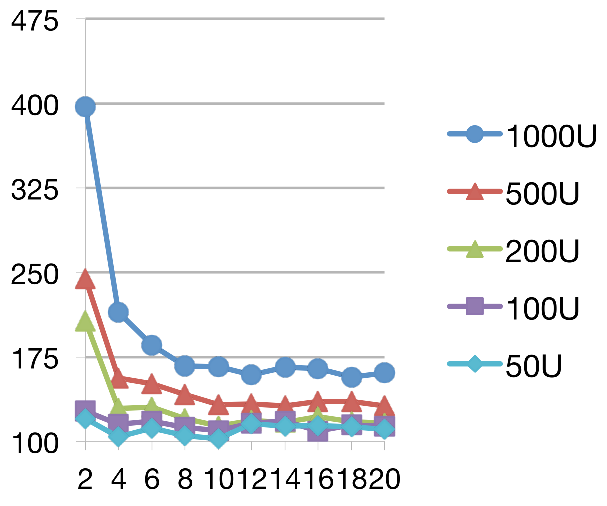

Figure 3. Execution time of Q1 in seconds dependent on the amount of nodes for 50 to 1000 Universites with 1 reducer for m1.large instances

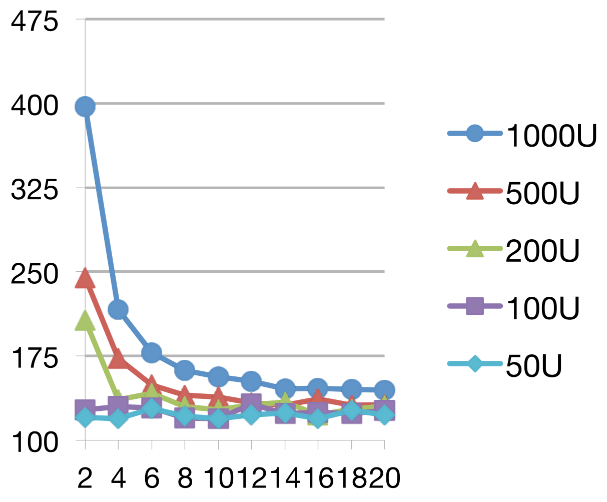

Figure 4. Execution time of Q1 in seconds dependent on the amount of nodes for 50 to 1000 Universites with nodes-1 reducers for m1.large instances

V. Complex Interdependency Performance

In this section we present and discuss performance results for query with complex interdependency pattern between files.

First of all, strong dependency on the amount of reducers can be observed. This can be most drastically seen on the Figure 5. where the performance actually decreases for more than 6 nodes. The reason for it might be the bigger need for communication between mappers and the reducer due to complexity of the query, despite relatively smaller inputs. However, not enough bandwidth is available due to low IO performance of EC2 m1.small instances. With more reducers performance is significantly increased (up to 10 times in some cases).

Secondly, further dependence on the IO performance of the cluster can be observed. Apart from the general decrease in times between m1.small and m1.large instances as observed in the previous section, additional effects can be noticed. If we compare Figure 5. with Figure 7. we can notice that high IO performance allows to partly compensate for the fact of having just one reducer by improving communication with it. Further, increased amount of reducers in such a case improves performance (Figure. 8); however, to relatively lesser extent as in the case of low IO performance (Figure 6.).

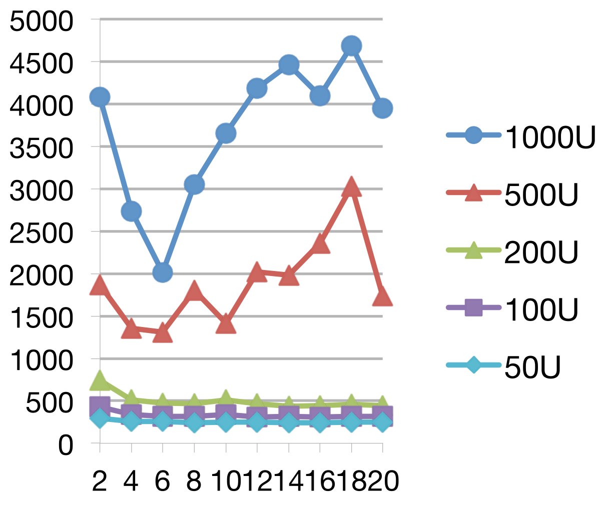

Figure 5. Execution time of Q2 in seconds dependent on the amount of nodes for 50 to 1000 Universites with 1 reducer for m1.small instances

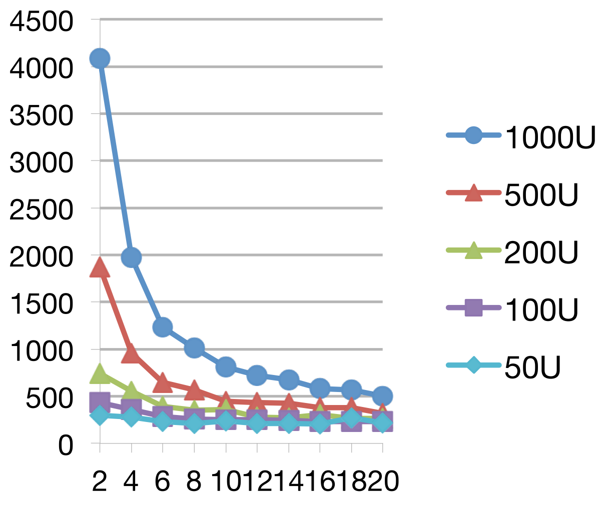

Figure 6. Execution time of Q2 in seconds dependent on the amount of nodes for 50 to 1000 Universites with nodes-1 reducers for m1.small instances

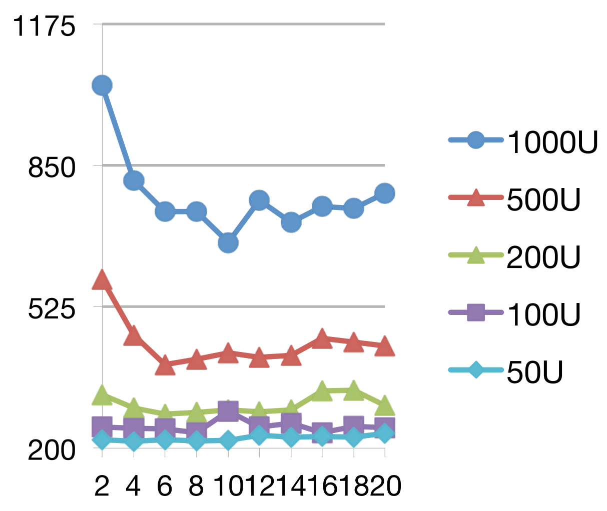

Figure 7. Execution time of Q2 in seconds dependent on the amount of nodes for 50 to 1000 Universites with 1 reducer for m1.large instances

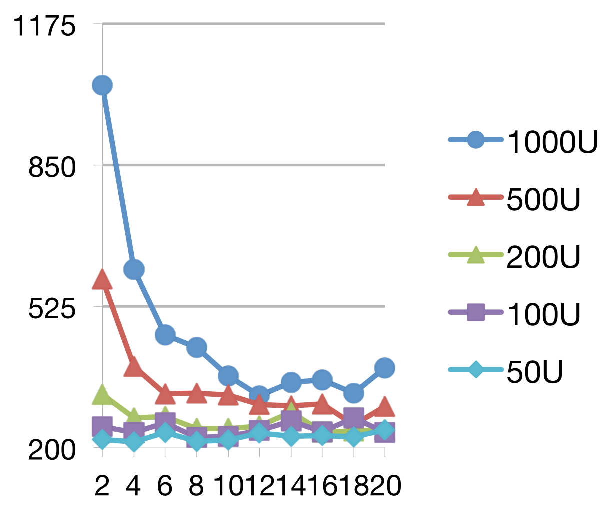

Figure 8. Execution time of Q2 in seconds dependent on the amount of nodes for 50 to 1000 Universites with nodes-1 reducers for m1.large instances

VI. Low Selectivity Peformance

In this section we present and discuss performance for query with low selectivity, so effectively very high output.

First of all, similar trends as in the previous sections can be noticed. However, after closer inspection, we observe the difference between IO performance with respect to network performance and disk performance. It is important to note that the following analysis bases just on the presented data. Further tests with precise disk and network monitoring should be performed to confirm the conclusions.

When comparing one-reducer configurations, Figure 9. and 11., we can notice that configuration with m1.large instances with high IO performance can avoid performance decrease with high growing number of nodes. In this situation higher network performance comes into effect. However, it does not show any benefits of those additional nodes. This can be caused by the limitation of the disk performance, as only one reducer has to write a relatively big output file to the HDFS. In such a case having more of cheaper machines with lower IO performance can provide slightly better results, as observed on Figures 10. and 11. from 10 nodes up.

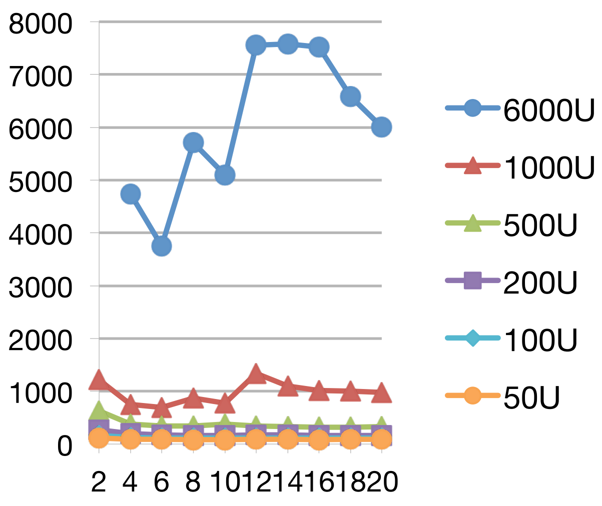

Figure 9. Execution time of Q14 in seconds dependent on the amount of nodes for 50 to 1000 Universites with 1 reducer for m1.small instances

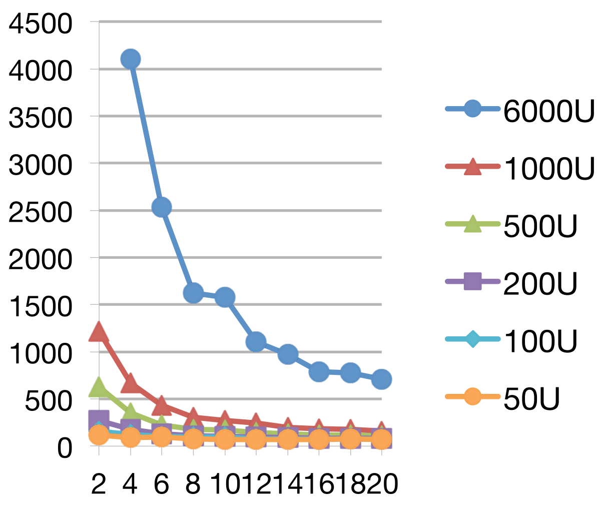

Figure 10. Execution time of Q14 in seconds dependent on the amount of nodes for 50 to 1000 Universites with nodes-1 reducers for m1.small instances

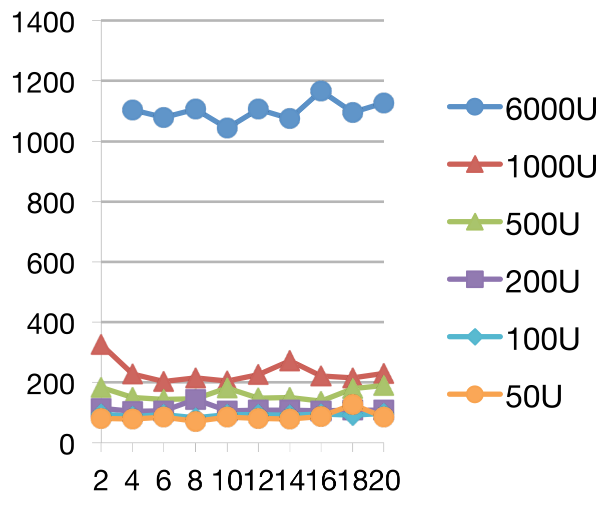

Figure 11. Execution time of Q14 in seconds dependent on the amount of nodes for 50 to 1000 Universites with 1 reducer for m1.large instances

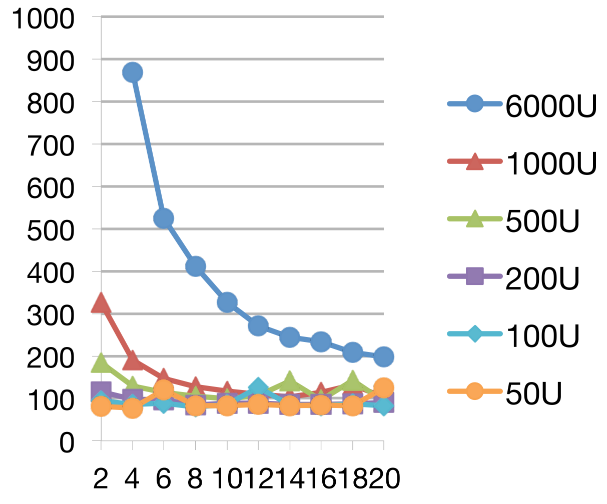

Figure 12. Execution time of Q14 in seconds dependent on the amount of nodes for 50 to 1000 Universites with nodes-1 reducers for m1.large instances

VII. Loading Performance

In this section we present and discuss load time overhead when utilizing S3. This is particularly important for AWS-based applications, as S3 is the default permanent storage service for AWS and its performance can impact performance of the whole application.

Figure 13. presents results of loading files with sizes ranging from 100 to 2900 MB on a m1.small instances depending on the amount of nodes. The biggest relative performance gain is observed in the range up to 8 nodes; though, further nodes also improve performance.

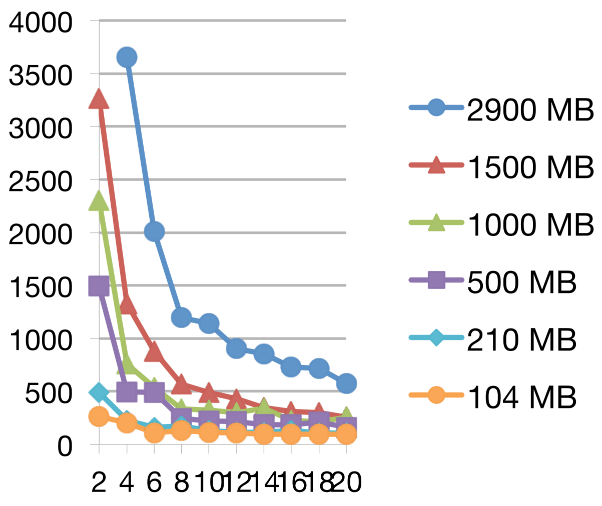

Figure 13. Load time in seconds dependent on the amount of nodes for different files sizes for m1.small instances

It is important to correlate these results with result from previous section in particular Figure. 10. For all the previous tests data was preloaded from S3 to HDFS, even though, it is not necessary for proper functioning of EMR. However, not doing so would significantly lower the performance. In Figure 14. one can see the results for the respective load time from Figure 13. added on top of results from Figure 10. What clearly illustrates the influence of the overhead of S3 load time for the total performance of the query. Therefore, for any application where the file will be loaded more than once we recommend not using S3 directly, but preloading data into HDFS first. Similar conclusion can be drawn with respect to saving any intermediate data, though we do not present the results for that analysis here.

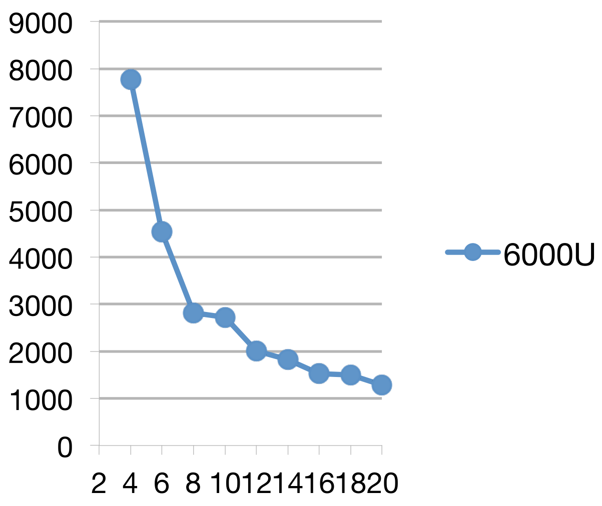

Figure 14. Load and query time in seconds dependent on the amount of nodes for different files sizes for m1.small instances

Figure 15. presents results of loading files with sizes ranging from 100 to 2900 MB on a m1.large instances depending on the amount of nodes. The biggest relative performance gain is observed in the range up to 8 nodes; though, further nodes also improve performance. One can notice that load time experience similar patterns as for m1.small instances, however, the general performance is significantly improved, by the factor ranging from 2 to 4 depending on the size of the file and amount of nodes. This clearly illustrates the dependence of any data-intensive MapReduce application on the IO performance of the cluster.

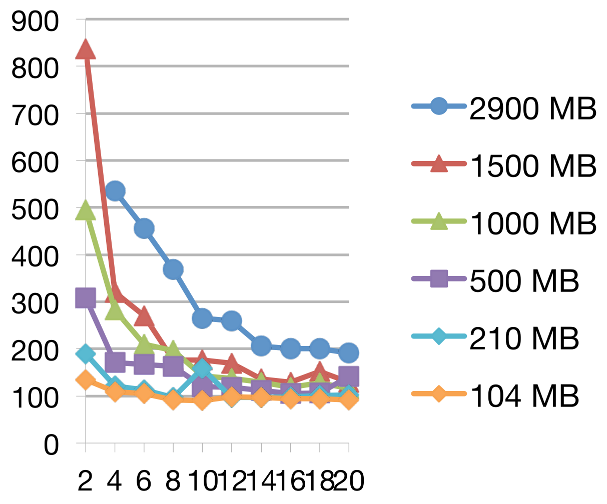

Figure 15. Load time in seconds dependent on the amount of nodes for different files sizes for m1.large instances

VIII. Conclusions

In this paper we presented a comprehensive performance analysis of Hadoop cluster configuration for NoSQL query processing.

We have shown that especially in the case of complex queries the amount of reducers and the IO network performance of the cluster are important. In case of queries with very high output, IO disk performance is the limiting factor but it can be improved by applying more reducers. Additionally, we have shown that for applications using S3 the load time overhead is significant and that processing should not be performed on S3 directly.

Moreover, we have demonstrated that in most of the cases having bigger amount of cheaper machines can be more beneficial than small amount of more powerful machines. In particular, that is the case for the reducer bound scenarios from Sections 5. and 6. Such an approach is more cost effective, as the cost of m1.large instance is double of the cost of m1.small.

These results prove a common opinion that MapReduce applications are IO bound.

Future work will consist of analysis that would point to the optimal amount of reducers for different application. Moreover, we would like to measure more precisely the difference between IO disk and network performance.

We trust that this paper provided an analysis that will help with decisions about cluster configuration and further performance improvements of query languages for Hadoop.

References

[1] J. Dejun and G.P.C. Chi, “EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications.”

[2] S.L. Garfinkel, “An Evaluation of Amazon’s Grid Computing Services: EC2, S3 and SQS,” *CENTER FOR*, 2007.

[3] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi, “Evaluating MapReduce on Virtual Machines: The Hadoop Case,” *Cloud Computing*, M. Jaatun, G. Zhao, and C. Rong, eds., Springer Berlin / Heidelberg, 2009, pp. 519-528.

[4] R. Stewart, “Performance and Programmability of High Level Data Parallel
Processing Languages: Pig, Hive, JAQL & Java-MapReduce,” 2010.

[5] J. Dawei, C.O. Beng, S. Lei, and W. Sai, “The Performance of MapReduce: An In-depth Study,” vol. 3, 2010, pp. 472-483.

[6] A. Newman, Y. Li, and J. Hunter, “A scale-out RDF molecule store for improved co-identification, querying and inferencing” Available: http://espace.library.uq.edu.au/view/UQ:188605.

[7] A. Newman, Y. Li, and J. Hunter, “Scalable Semantics - the silver lining of cloud computing” Available: http://espace.library.uq.edu.au/view/UQ:175239.

[8] J. Myung, J. Yeon, and S. Lee, “SPARQL basic graph pattern processing with iterative MapReduce,” *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud - MDAC '10*, Raleigh, North Carolina: 2010, pp. 1-6.

[9] M. Farhan Husain, P. Doshi, L. Khan, and B. Thuraisingham, “Storage and Retrieval of Large RDF Graph Using Hadoop and MapReduce,” *Cloud Computing*, M. Jaatun, G. Zhao, and C. Rong, eds., Springer Berlin / Heidelberg, 2009, pp. 680-686.

[10] M.F. Husain, L. Khan, M. Kantarcioglu, and B. Thuraisingham, “Data Intensive Query Processing for Large RDF Graphs Using Cloud Computing Tools,” *2010 IEEE 3rd International Conference on Cloud Computing*, Miami, FL, USA: 2010, pp. 1-10.

[11] Jianling Sun and Qiang Jin, “Scalable RDF store based on HBase and MapReduce,” *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, 2010, pp. V1-633-V1-636.

[12] P. Mika and G. Tummarello, “Web Semantics in the Clouds,” *Intelligent Systems, IEEE*, vol. 23, 2008, pp. 82-87.

[13] D.J. Abadi, A. Marcus, and B. Data, “Scalable semantic web data management using vertical partitioning,” *IN VLDB*, 2007, pp. 411--422.