Summary Paper of *Don't Hold My Data Hostage – A Case For Client Protocol Redesign*

Authors of Original Paper: Mark Raasveldt and Hannes Mühleisen

Hannes Pohnke
pohnke@tu-berlin.de
TU Berlin
Berlin, Germany

ABSTRACT

Traditionally, database query processing and ML tasks are executed on separate, dedicated systems, but the current trend goes towards integrated data analysis pipelines that combine both tasks. In state of the art systems, orchestration of those two tasks still is inefficient due to expensive data transfer and missed global optimization potential. The paper we are summarizing, "Don't Hold My Data Hostage – A Case For Client Protocol Redesign" (DHMDH) by Mark Raasveldt and Hannes Muhleisen, addresses this problem by investigating the high cost of transferring large data from databases to the client programs, which can be much more time consuming than the actual query execution. The authors explore and analyse current serialization methods, that are used in database systems and identify their inefficiencies through various experiments. They also introduce a new columnar serialization method that can significantly enhance data transfer performance. By improving the data transfer, this approach could be a step towards efficiently combining database and machine learning systems.

KEYWORDS

Databases, Client Protocols, Data Export

ACM Reference Format:

Hannes Pohnke. 2018. Summary Paper of *Don't Hold My Data Hostage – A Case For Client Protocol Redesign*: Authors of Original Paper: Mark Raasveldt and Hannes Mühleisen. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Having to transfer a lot of data from a database to a client program is a standard procedure when conducting complex machine learning (ML) tasks. The problem is that this transfer process is very slow, especially because database servers are typically not located on the same system as the ML workloads.

When the DHMDH paper was introduced in 2017 most research in this field, including the authors [?], focused on performing computations within the database, avoiding the need for data transfer. Since then a lot of approaches for optimizing the transfer arose, which can be broadly categorized in server-side and client-side optimizations [?]. Raasveldt and Mühlheisen chose a client-side approach, as they already realized in 2017 that the biggest and easiest to implement optimization potential lies in addressing the overhead of (de)serialization of the data. They use the term of result set serialization (RSS), which refers to converting data into a format suitable for transfer and show its significant impact on system performance by comparing the transfer time of a basic SQL query with different data management systems.

Back in the day popular ML tools, including RapidMiner, Weka, R's ML packages, and Python-based toolkits like SciKit-Learn and TensorFlow, only supported bulk data transfer from databases [?]. Therefore, users had to manually load data that was already in a table format into these tools. Because of the mentioned inefficiencies this often resulted in smaller data samples being used, which is generally a bad thing in ML.

This paper examines state of the art serialization formats and explores the design space of those formats in client protocol design. Key contributions include:

- Benchmarking current RSS methods, to identify data transfer inefficiencies.
- Investigating techniques for creating efficient serialization methods.
- Proposing a new column-based serialization method with significant performance improvements.

2 STATE OF THE ART

All remote database systems use client protocols to manage communication between server and client. This process begins with an initial authentication and an exchange of meta data. Then the client can send queries to the server. Once the server processes the query, it has to serialize the query data into a result set format, send it over the socket to the client, and then the client deserializes the data to use it. As we pointed out, the time spent on these steps is significantly influenced by the design of the result set format. We will now explore the serialization formats used by leading systems and evaluate them.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

2.1 Overview

To evaluate the performance of various databases for large result set exports, experiments on systems like MySQL [36], PostgreSQL [32], IBM DB2 [37], "DBMS X", MonetDB [5], Hive [33], and MongoDB [23] were conducted. MySQL's client protocol with GZIP compression ("MySQL+C") was tested separately. Of course there are more database systems, but many of those adopt client protocols from more popular databases to reuse existing implementations. This selection of systems therefore seems representative for the state of the art in 2017.

The experiments focused on isolating the duration for RSS and data transfer. The TPC-H benchmark's SF10 lineitem table was loaded into each database, and data retrieval times were measured using ODBC connectors (JDBC for Hive). Netcat (nc) [12] was used as a baseline for efficient data transfer without any database overheads. The results we can see in Table 1, showed that transferring data as CSV via Netcat was drastically faster than using any tested database system. This undermines that the most time-consuming part was RSS and transfer. MongoDB had the highest overhead due to its document-based style, while MySQL+C transferred the least data due to compression.

Table 1: Time taken for result set (de)serialization + transfer when transferring the SF10 lineitem table.

System	Time (s)	Size (GB)
(Netcat)	*(10.25)*	*(7.19)*
MySQL	**101.22**	7.44
DB2	169.48	7.33
DBMS X	189.50	6.35
PostgreSQL	201.89	10.39
MonetDB	209.02	8.97
MySQL+C	391.27	**2.85**
Hive	627.75	8.69
MongoDB	686.45	43.6

2.2 Network Impact

Lorem Ipsum und so ...

3 SUMMARY

Fazit nochmal labern was impact ist und wie sich anscheinend in database solutions durchsetzen aus mehreren argumenten wenn das tatsächlich der fall ist? kp ich glaube sogar nicht so wie sich connector x anhört

4 ANALYSIS

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because LaTeX now knows how many pages to expect for this document.