Conference Paper Title*

*Note: Sub-titles are not captured in Xplore and should not be used

Alexandros Ionitsa
Dept. of Electrical & Computer Engineering
National Technical University of Athens
Athens, Greece
el19193@mail.ntua.gr

Emmanouil Emmanouilidis
Dept. of Electrical & Computer Engineering
National Technical University of Athens
Athens, Greece
el19435@mail.ntua.gr

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

With the term ”Big Data” we refer to data sets whose scale, diversity and complexity require new architecture, techniques, algorithms and analytics to manage them and extract value from them. Big data has transformed the way organizations handle information, providing unprecedented amounts of context and insights.

However, big data makes queries slow and is very difficult and expensive to manage. So, this rapid evolution of big data has forced companies and organizations to put a lot of effort on developing new specialised tools designed to manage and analyze those large data sets in an efficient manner. In response to these challenges, new cutting-edge solutions emerged, including Trino (formerly known as PrestoDB).

Trino is an open-source distributed SQL query engine designed to query large data sets distributed over one or more heterogenous data sources. Among others, Trino has the ability to run federated queries that query tables in different data sources such as MySQL, PostgreSQL, Cassandra, Kafka, Redis and many more.

Over the last few years, Trino has draught a lot of attention and nowdays it is widely used by many companies and organizations. Therefore, it is significant to evaluate its performance under different scenarios and circumstances.

In this paper, we benchmark the performnace of Trino across different types of queries, data locations and underlying storage technologies. In more detail, we establish a Trino cluster consisting of four nodes, a coordinator and three workers where each of them is equipped with a different storage system: PostgreSQL on worker 1, Cassandra on worker 2 and Redis on worker 3. The diversity among these three database management systems (DBMS) helps us cover three different data models: Relational, Wide-Column and Key-value provding better results. After, the installation and set up of those three specific stores, we generate and load data using the well-known TPC-DS benchmark which is also utlized for query generation. The size of the data is large enough so that it won't fit in memory, in the order of several GB & milions of records and the queries are diverse enough including both, simple and complex ones (multiple joins and aggregations). After generating and loading the data, we pose the generated queries into each of the data sources and we come up with some initial measures. Based on those measures we devise different data distribution stategies and we measure the performance (query latency, optimizer plan) over different number of workers and different data distribution plans. Our final goal is to idneitfy how the distributed execution engine and optimization works under a varying ammount of data, data distributed in different engines and with queries of varying difficulty.

This paper is backed by a GitHub repository, that contains scripts and howtos on how to replicate our Trino cluster, generate and load the data and generate and perform the queries. Additionally, the repository includes the results of our measurements, along with a detailed analysis to provide a thorough understanding of the outcomes.

Identify applicable funding agency here. If none, delete this.

II. Source Code

As mentioned before, the project is backed by a GitHub repository with scripts and howtos which are accessible through the following link:

GitHub Repository

III. Cluster Configuration

A. Infrastructure Overview

For our project we used four virtual machines that were provided by okeanos-knossos. okeanos-knossos is a GRNET'S cloud service (IaaS) for the Greek Research and Academic Community that gives access to resources such as virtual machines and virtual networks.

B. Virtual Machines Specification

1) **Number of Virtual Machines:** A total of four virtual machines were utilized.
2) **Virtual Machine Specifications:**

- **RAM:** Each virtual machine boasted 8GB of RAM.
- **Memory:** 30GB of memory per virtual machine.
- **Operating System:** The virtual machines were equipped with Ubuntu 22.04.

C. Networking

For networking we were given one public IPv4 address, which was attached to our master node which also behaves as the Trino coordinator. In order to allow the communication between the cluster nodes, we set up a local area network (LAN: 192.168.0.0/23). In more details, 192.168.0.2 was assigned to the coordinator, 192.168.1.1 to node 1 (PostgreSQL), 192.168.1.2 to node 2 (Cassandra) and 192.168.1.3 to node 3 (Redis) as shown in the figure below:

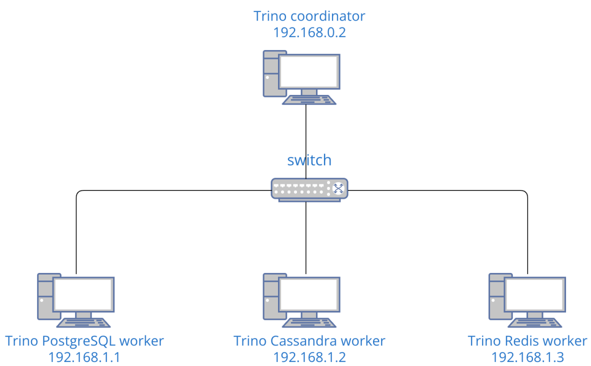

Fig. 1. Cluster's Network Topology

IV. Technology Stack

A. Overview

- **Trino:** Was utilized as a distributed query engine
- **PostgreSQL:** Database management systems (DBMS) on first node.
- **Cassandra:** DBMS on second node.
- **Redis:** DBMS on third node.
- **TPC-DS Benchmark:** Tool utilized for data and query generation.

B. Trino

Trino, formerly known as PrestoDB, is an open-source distributed SQL query engine, build in Java. It is designed to query large ammount of data distributed over one or more heterogeneous data sources, using distributed queries.

It was firstly designed as an alternative to tools that query Hadoop Distributed File System (HDFS) using pipelines of MapReduce jobs such as Hive or Pig. However, Trino has been extended to operate over different kinds of data sources, including traditional relational database such as PostgreSQL and other data sources such as Cassandra. Consequently, Trino has the ability to query data lakes housing open column-oriented data file formats like ORC or Parquet. Moreover, it excels in executing federated queries that span tables in different data sources like MySQL, PostgreSQL, Cassandra, Redis and MongoDB. Furthermore, Trino is also deisgned to handle data warehousing and analytics tasks and its functionalitites extend to data analysis, the aggregation of huge data sets and the generation of reports.

As a distributed query engine, Trino processes data in parallel across multiple servers and it runs on a cluster of servers that contains two types of nodes, a coordinator and a worker. Users interact with the coordinator using their SQL query tool, and the coordinator collaborates with workers to access connected data sources. The configuration for this access is stored in catalogs, where parameters like host, port and passwords are defined. Acting as the orchestrator, the coordinator distributes workloads in parallel across all workers. Each worker node runs Trino in a seperate JVM instance and processing is further parallelized through threads.

The Trino coordinator is responsible for parsing and optimizing statements, planning queries and managing worker nodes and it is the only node to which clients can submit statements for execution. It maintains oversight of worker activity, coordinates query execution and translates the logical query models into connected tasks. Additionally, coordinator communicates with clients and workers through a REST API and can be configured to also serve as a worker.

On the other hand, Trino workers are servers responsible for executing tasks and processing data. They fetch data from connectors and exchange intermediate data with each other. The coordinator is responsible for fetching results from workers and returning the final reuslts to the client. Like coordinator, workers use a REST API with which they communicate to each other.

Accessing many different data sources, querying data and returning results to client in a distributed way, is a very challenging task. For this purpose Trino utilizes many different components the most significant and critical of which are connectors, catalogs, schema and table.

Connectors play a crucial role as they are plugins that enable communication with different data sources and storage systems. Trino supports various connectors for traditional SQL databases like MySQL, PostgreSQL, Oracle, for NoSQL databases liek MongoDB and Cassandra and for modern data lakes like Hive, Iceberge and Delta Lake. As a result, users can query data from a large variety of different data sources.

Schemas are a way to organize tables. Together, a catalog and schema define a set of tables that can be queried. When accessing Hive or a relational database such as MySQL with Trino, a schema translates to the same concept in the target database.

A table is a set of unordered rows, which are organized into named columns with types. This is the same as in any relational database. The mapping from source data to tables is defined by the connector.

Catalogs in Trino, are logical containers that hold metadata about the available sources, including schemas, tables and connectors. Trino can connect to multiple catalogs, each representing a distinct set of data sources and their associated metadata. Users can switch between catalogs in their queries and access data from different environments without the need to change their SQL statements.

Summarizing the above, the figure below, illustrates Trino's cluster architecture:

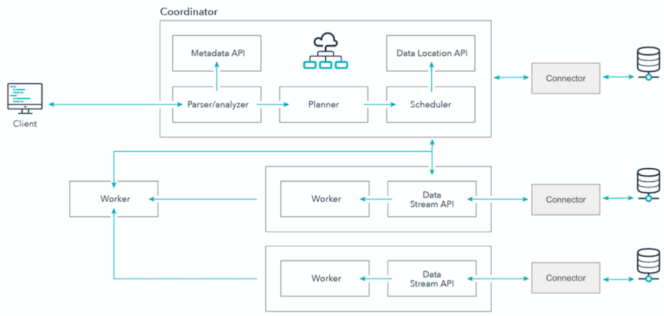

Fig. 2. Trino Cluster Architechture

C. PostgreSQL

PostgreSQL, also known as Postgres, is a highly regarded free and open-source relational database management system (RDBMS) that emphasizes extensibility and SQL compliance. The system is designed to handle a diverse range of workloads, from single machines to large-scale data warehouses or web services with numerous concurrent users. PostgreSQL operates seamlessly on major operating systems, including Linux, FreeBSD, OpenBSD, macOS, and Windows. Some key features that make PostgreSQL very interesting and powerful, are the followings:

- **SQL Compliance and Extensibility:** PostgreSQL allows users to define their own data types, operators, functions, and aggregates. This extensibility provides flexibility for developers to customize and extend the database according to their specific requirements.
- **ACID Properties:** The system ensures data reliability through transactions with Atomicity, Consistency, Isolation and Durability (ACID) properties, enhancing data integrity and consistency.
- **Advanced Functionality:** PostgreSQL supports a rich set of features such as automatically updatable views, materialized views, foreign keys providing flexibility and control over database operations.
- **Concurrency Control:** PostgreSQL employs Multi - Version Concurrency Control (MVCC), which allows multiple transactions to occur simultaneously without interfering with each other. This ensures high concurrency and scalability.
- **Advanced Query Optimization:** The query planner and optimizer in PostgreSQL are sophisticated, enabling efficient execution plans for complex queries. It includes features like join optimization, subquery optimization, and parallel query execution.

These features make PostgreSQL a very powerfull and attractive relational DBMS and that's the reason we selected it for our cluster.

D. Cassandra

Cassandra is an open - source, distributed, wide-column store, NoSQL database management system, designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Some of its main features are listed below:

- **Distirbuted:** Every node in the cluster has the same role and there is no master. Also there is no single point of failure and data is distributed across the cluster, so each node is responsible for different data.
- **Replication - Fault Tolerance:** Cassandra supports replication across multiple datacenters, providing lower latency for users. Data is automatically replicated to multiple nodes the number of which can be easily configured.
- **Scalability:** Cassandra scales horizontally and is designed in such a way, that allows linear increase in write and read throughput by adding new machines in the cluster.
- **Eventual consistency:** Cassandra is classified as an AP system, meaning that availability and partition tolerance are generaly considered to be more important than consistency. However, Cassandra manages eventual consistency of reads, upserts and deletes through Tombstones.
- **Cassandra Query Language:** Cassandra introduced the Cassandra Query Language (CQL), which is similar to SQL and therefore it can execute complex queries efficiently.
-

These feature, in addition to its efficiency and easy scalability contribute to Cassandra being a very powerful and widely used NoSQL database and makes it a perfect database to experiment on. In our project, we also examine Cassandra's scalability so we consider two versions of Cassandra cluster: a single node cluster and a two-node cluster with replication factor of two.

E. Redis

Redis (Remote Dictionary Server) is an open-source in-memory storage, used as a distributed, in-memory key-value database, cache and message broker, with optional durability. Because it holds all data in memoery and because of its design, Redis offer low-latency read and writes, making it particurarly suitable for cases that require a cache. Redis one of the most popular database systems and it is frequently used in combination with other databases in order to leverage the strengths of each database for specific use cases. Some key features are listed below:

- **In - Memory Storage:** Redis stores data in-memory, which allows for extremely fast read and write operations.
- **Data Structures:**Redis supports a variety of data structures, including strings, hashes, lists, sets and sorted sets. Each data type comes with its own set of operations.
- **Persistence:** While Redis is an in-memory store, it provides options for persistence. It can be configured to periodically write data to disk, ensuring data durability.

- **Replication:** Redis supports master-slave replication, allowing for the creation of replicas of the master Redis server. This provides high availability and fault tolerance, as the slaves can take over if the master fails.
- **Paritioning:** Redis can be partitioned across multiple nodes, enabling horizontal scaling. This is particularly useful for large datasets and high-throughput scenarios.

Redis is utilized in our project in order to store a subset of the data set and compare its efficiency and speed.

F. TPC-DS Benchmark

V. Data Distribution Strategy

VI. Process and Methodology

VII. Results and Analysis

VIII. Conclusion

IX. References

A. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: "Wb/m^2" or "webers per square meter", not "webers/m^2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm^3", not "cc".)

B. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{1}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

C. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

D. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.

- Be aware of the different meanings of the homophones “affect” and “effect”, “complement” and “compliment”, “discreet” and “discrete”, “principal” and “principle”.
- Do not confuse “imply” and “infer”.
- The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the “et” in the Latin abbreviation “et al.”.
- The abbreviation “i.e.” means “that is”, and the abbreviation “e.g.” means “for example”.

An excellent style manual for science writers is [7].

E. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

F. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced.

G. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 3”, even at the beginning of a sentence.

Fig. 3. Example of a figure caption.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

TABLE I

TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a]Sample of a Table footnote.

ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.