F1 Query

F1 Query by Google provides the ability to make queries over multiple data sources and gives the abstraction that they are at single source to the user. The data at Google can be in different file format or at different storage places such as Bigtable, Spanner, Spreadsheet. It supports OLTP, OLAP and ETL pipelines for data warehouses. It has low latency, high throughput, supports parallelism and supports complex solutions. F1 Query is derived from F1 which is a distributed relational database. It is used for advertisement data. F1 query separates the query processing from the storage providing various use cases. It claims that the data networks in Google remove the throughput latency for accessing remote data hence there's no network bottleneck. The data is distributed in small chunks over the Colossus File system. It supports UDF, UDA and TVFs for complex business logic.

User sent query through the client libraries to the F1 servers. It has F1 master which acts like a cluster manager. F1 query executes short queries on immediate single node available(single-threaded execution kernel), larger queries in distributed execution mode(multi-threaded) and largest queries are run using MapReduce framework. Query execution converts the query into dag that are optimized at physical and logical layer. To support heterogenous data sources, it uses protocol buffer. Querying protocol buffers can have similar problems faced with xml or json. However, JSON is in human readable format and dynamically typed, protocol buffers are statically typed and similar to bson in mongodb. The query planner is intelligent and all unnecessary fields are removed(I guess similar to all columnar databases). For storing the result of the queries, it also supports external data sinks. It has various options such as file format and remote storage service. The default option is as a file in Colossus file system. F1 query compiles with SQL 2011 standard, supports protocol buffer and shares dialects with other system at Google. It supports both interactive and batch mode. The batch mode supports asynchronous execution using the MapReduce framework. The query optimizer here is inspired by the Spark's catalyst planner. It calls the Google sql resolver and produce AST, series of conversions as shown in fig 8., which gets converted to Final execution plan. The intermediate steps are discussed in detail in section 5. It also supports Lua for adhoc queries. Hence, google has provided a versatile system that lets the user reuse the same SQL query to smaller data in interactive mode or huge amount of data in batch mode. F1 queries has certain pitfalls because of its row-oriented execution kernel as compared to vectorised execution engine and is working on to improve it. It is also working on taking some improving its local caching layer, scale-in, use cost-based optimization rule and in-memory analytics.

Comments:
I would have appreciated some more details in how the data latency is avoid and

how much time difference does it take for accessing local vs remote data.
“In practice, queries that run for up to an hour are sufficiently reliable” - are the google servers so reliable for making this bold statement? also failed queries constantly keep retrying, but lets say if there’s a semantic error in the query and that’s why its failing. Will it be stuck In an infinite loop or it will give up after some time?
Does the client-server architecture use the grpc protocol like the exchange operation or the http protocol?
Since the data is moved in partition strategy, I am guessing its kind of similar like shuffling in mapReduce. Isn’t that a costly operation?
The interactive query support is useful because it enables to get the results instantly in real-time instead of waiting for the entire result to be available.
This is the first time I heard about the programming interface Lua. I did a quick google and chatgpt search and found out that it is mainly used for embedded system and web development. Is there any actual significant importance of having Lua connectors or is it one of the many features discussed in section 6.