dremio

Dremio Architecture Guide

Architecting for Performance and Cost Efficiency

Why Dremio

The data lake market is experiencing rapid growth. More data is getting stored in data lake cloud storage such as AWS S3 and Microsoft ADLS. By their nature, data lakes differ from data warehouses in that they do not require a rigorous schema for data to be defined before loading data, which makes it easier for organizations to collect raw data from many systems for analysis. However, with data lake adoption comes challenges; traditional SQL engines like Presto and Athena are too slow to query data directly from data lakes for most analytical use cases outside of some edge ad-hoc queries, so most organizations prefer to use a combination of data warehouses, extracts, cubes and ETL to make the data from data lake storage more usable and accessible for analytical queries. However, when data warehouse systems scale in usage, the cost of ownership gets incredibly expensive. Additionally, there's a certain amount of vendor lock-in because data warehousing solutions are proprietary systems.

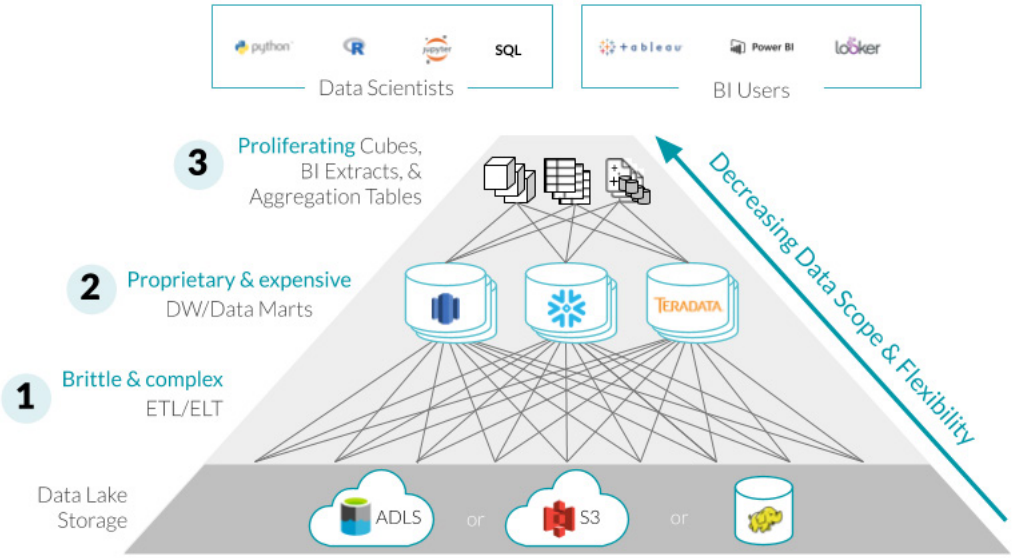

Figure 1 - Data movement leads to high cost, increased complexity and relinquished control

This layering-on of data warehouses and associated technologies and processes creates a complex and costly infrastructure for data analytics, as seen in Figure 1 above. This is unfortunate, because cloud-based data lake storage is inexpensive, easy to provision and scale, and usually already has an attached ecosystem of tools and solutions. In many organizations, it's also the first place where data lands after it's generated by operational systems. If data lake storage could be queried directly in a performant way, a lot of the additional solutions that are put in place to get around these performance problems could be simplified or removed altogether.

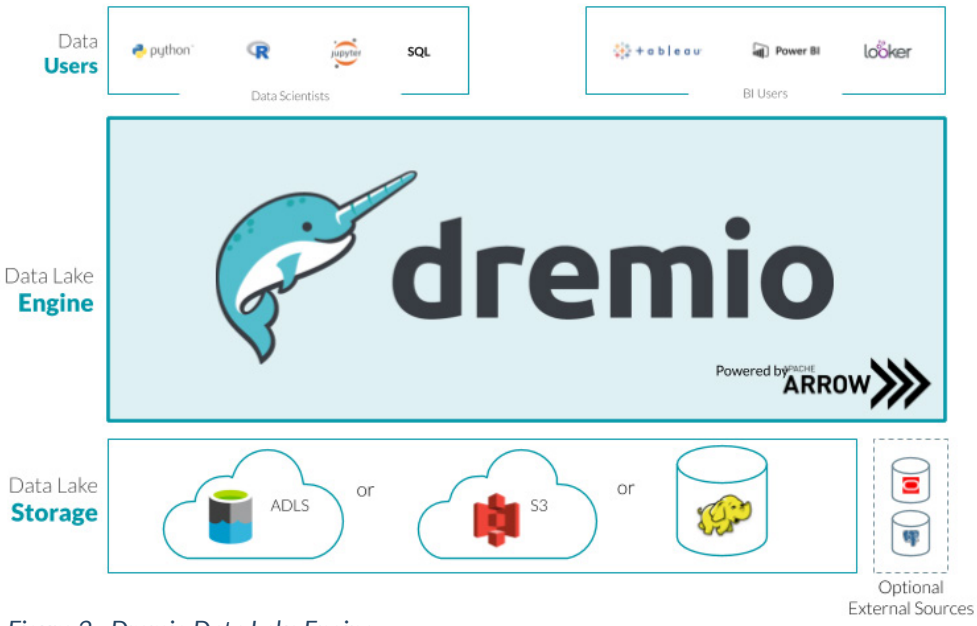

Figure 2 - Dremio Data Lake Engine

As seen in Figure 2 above, Dremio's Data Lake Engine provides that performance, together with a self-service semantic layer that makes it easy for even non-technical users to access and analyze data. It does this in a way that maintains the flexibility that's inherent in data lake storage; you don't have to copy and move your data to a third party or put it in a proprietary format. Dremio provides:

- Lightning-fast queries directly on data lake storage. Dremio technologies like Data Reflections, Columnar Cloud Cache (C3) and Predictive Pipelining work alongside Apache Arrow powered by Gandiva to execute queries directly on data lake storage at interactive speed. These query acceleration technologies combine to deliver 4-100x faster performance compared to traditional data lake engines like Presto.
- A self-service semantic layer. This abstraction layer enables data engineers to apply security and business meaning while enabling analysts and data scientists to explore data and derive new virtual datasets.
- Flexibility and openness. Dremio lets you avoid vendor lock-in, query data directly across clouds or on-prem, and keep your data in storage that you own and control. In this way, you maximize your flexibility and freedom to use your data as you see fit.
- Deep infrastructure cost savings. Dremio combines query acceleration and highly elastic compute resources to provide significant performance and cost benefits. For example, a 4x average increase in speed results in a 75% or more infrastructure cost reduction vs. Presto at the same level of performance.

Dremio Core Technologies

Dremio, at its core, utilizes high-performance columnar storage and execution, powered by Apache Arrow (columnar in-memory) with Gandiva (LLVM-based execution kernel), Apache Arrow Flight (high-speed distributed protocol) and Apache Parquet (columnar on-disk). Dremio has deep knowledge and experience with high-performance analytics and is the co-creator and current maintainer of Apache Arrow, Gandiva and Arrow Flight projects.

APACHE ARROW, ARROW FLIGHT AND GANDIVA

Apache Arrow is an open source project that enables columnar in-memory data processing and interchange. It also leverages the execution kernel *Gandiva* to compile queries to a vectorized code optimized for modern CPUs. Dremio was part of the founding team behind Arrow, which now includes committers from various organizations including IBM, Cloudera, Databricks, Hortonworks, Intel, MapR, and Two Sigma.

Dremio is the first data lake engine built from the ground up on Apache Arrow. Internally, the data in memory is maintained off-heap in the Arrow format, and *Arrow Flight* introduced an RPC API that returns query results as Arrow memory buffers.

A variety of other projects have embraced Arrow as well. Python (Pandas) and R are among these projects, enabling data scientists to work more efficiently with data. For example, Wes McKinney, creator of the popular Pandas library, recently *demonstrated* how Arrow enables Python users to read data into Pandas at over 10 GB/s.

APACHE PARQUET

Apache Parquet is an open source project that enables columnar data storage. It has emerged in recent years as the most common columnar format in the Hadoop and AWS ecosystems. Unlike Apache Arrow, which is optimized for in-memory storage and efficient processing in the CPU, Parquet is optimized for on-disk storage. For example, it utilizes encoding and compression schemes, such as dictionary and run-length encoding, to minimize overall footprint and storage I/O.

Dremio includes an ultra high-performance, vectorized Parquet reader that reads Parquet-formatted data from disk into Arrow-formatted data in memory. The Parquet reader enables fast processing of raw data as well as Data Reflections. It includes state-of-the-art capabilities such as:

- Intelligent predicate push-downs and page pruning
- In-place operations without decompressing data
- Vectorized processing
- Zero memory copies

A Dremio Instance

Dremio features an elastic scale-out *architecture*. It is designed to scale from one to thousands of nodes in a single instance. *Common deployment patterns* include:

- **Hosted Kubernetes environment:**
  - Amazon Elastic Container Service for Kubernetes (Amazon EKS)
  - Azure Kubernetes Service (AKS)
- **Hosted environment through provisioning templates:**
  - AWS Cloud Formation template
  - Azure Resource Management template
- **Shared multi-tenant environment:**
  - Hadoop using YARN
  - MapR using YARN
- **Standalone on-premises instance**

INSTANCE NODE TYPES

There are two distinct node types in a Dremio instance, and the services property *determines* whether the node is enabled with the master-coordinator or engine role. Each node type can be scaled independently; see Figure 3 below for the Dremio deployment architecture and Figure 4 for the Dremio functional architecture. The two node types are:

- Coordinators. These nodes are responsible for coordinating query planning, managing metadata, serving Dremio's UI and handling client connections. Client applications, such as BI or data science tools, connect to and communicate with coordinators. Coordinators are highly available and can be scaled up to process more concurrent clients. **Note** that all Dremio instance nodes with Dremio coordinator services must have the master-coordinator role enabled. A node with only the coordinator role enabled is not supported.
- Engines. These nodes are responsible for query execution. Client applications do not connect to engines. Engines can be scaled up to process large data volumes and more concurrent queries. Because engines are stateless, deployments can treat these nodes as elastic resources and scale the system dynamically.

Learn more about how high availability works in the Dremio instance *here.*

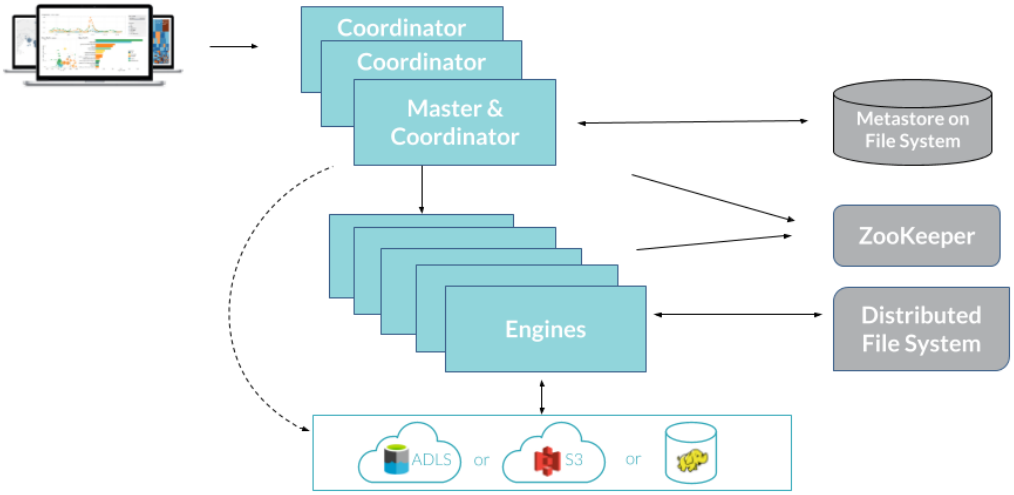

Figure 3 - Dremio deployment architecture

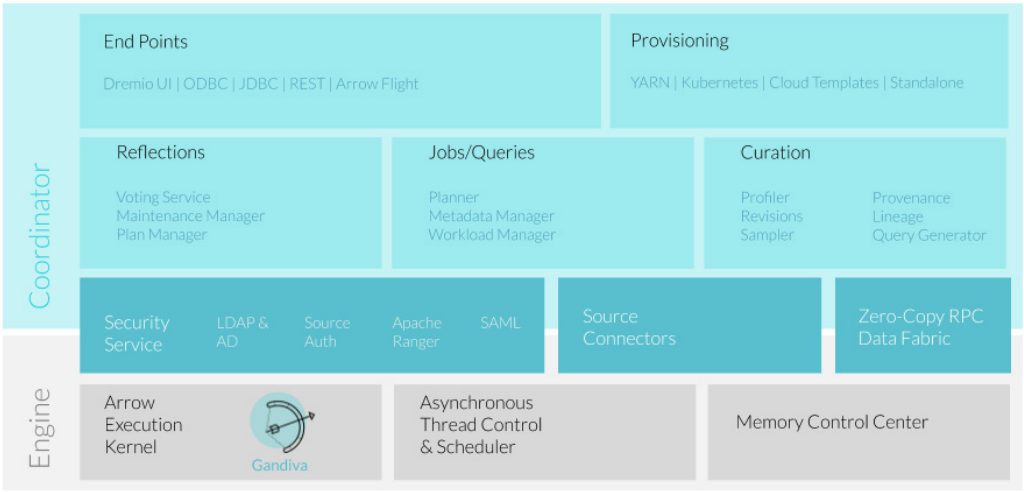

Figure 4 - Dremio functional architecture

Query Acceleration

There are two general types of analytical queries: ad-hoc and dashboard/ reporting. Dremio's Data Lake Engine has multiple acceleration technologies, allowing it to execute those queries extremely fast and efficiently. Ad-hoc queries are dynamic and interactive by nature and require on-demand direct access to the source data. Dremio's Arrow-based engine powered by Gandiva, C3 and predictive pipelining deliver ad-hoc query results up to 4x faster than traditional SQL engines(see Figure 5). Dashboarding/reporting queries are well-defined and usually are getting executed against curated datasets. For these queries, Dremio offers additional acceleration technologies such as Data Reflections to make these queries up to 100x faster compared to traditional SQL engines.

THE LIFE OF A QUERY

Client applications can issue queries to Dremio over ODBC, JDBC or REST. A query might involve one or more datasets, mostly residing in data lake storage but potentially in optional, relatively small external data sources. Dremio utilizes the following primary techniques to reduce the amount of processing required for a query:

- Direct data lake storage queries. The optimizer uses Apache Arrow to read data directly from data lake storage into the Arrow buffer with massively parallel readers and predictive pipelining at high speed and extreme concurrency. Also, the optimizer interrogates C3 to fetch automatically cached data, achieving NVMe-level performance.
- Push-downs into optional external data sources. The optimizer considesr the capabilities of the underlying external data source and the relative costs. It then generates a plan that performs stages of the query, either in the external source or in Dremio's distributed execution environment, to achieve the most efficient overall plan possible.
- Optional acceleration via Data Reflections. The optimizer uses *Data Reflections*, a highly optimized physical representation of source data, if available for all or portions of the query when this produces the most efficient overall plan. In many cases, the entire query can be serviced from Data Reflections.

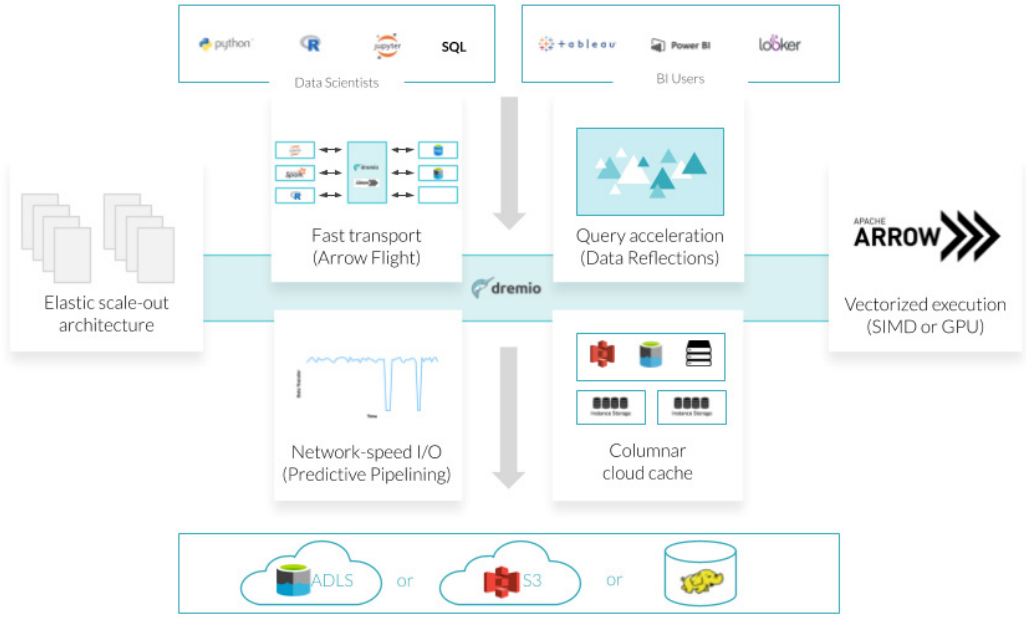

Figure 5 - Dremio query acceleration

QUERY EXECUTION PHASES

The life of a query includes the following phases (see Figure 6 for sequence diagram):

1. Client submits query to coordinator via ODBC/JDBC/REST

2. Planning:

a. Coordinator parses the query into Dremio's universal relational model.

b. Coordinator considers available statistics on data sources to develop query execution plan, as well as functional abilities of the source.

c. Coordinator rewrites query plan to use (1) any available Data Reflections, considering ordering, partitioning and distribution of the Data Reflections; and (2) the available capabilities of the data source.

3. Execution:

a. Engines read data into Arrow buffers from sources in parallel. The data typically come directly from data lake storage-based data sources. In some cases, they come from Data Reflections (Parquet files) also stored in data lake storage, or external data source(s). When reading from an external source, the engine submits the native queries (e.g., Elasticsearch Painless Script, Microsoft Transact-SQL) as determined by the optimizer in the planning phase.

b. Engines execute the rewritten query plan.

c. One engine merges the results from one or more engines and streams the final results to the coordinator.

4. Client receives the results from the coordinator.

Note that all data operations are performed on the engine node, enabling the system to scale to many concurrent clients using only a few coordinator nodes.

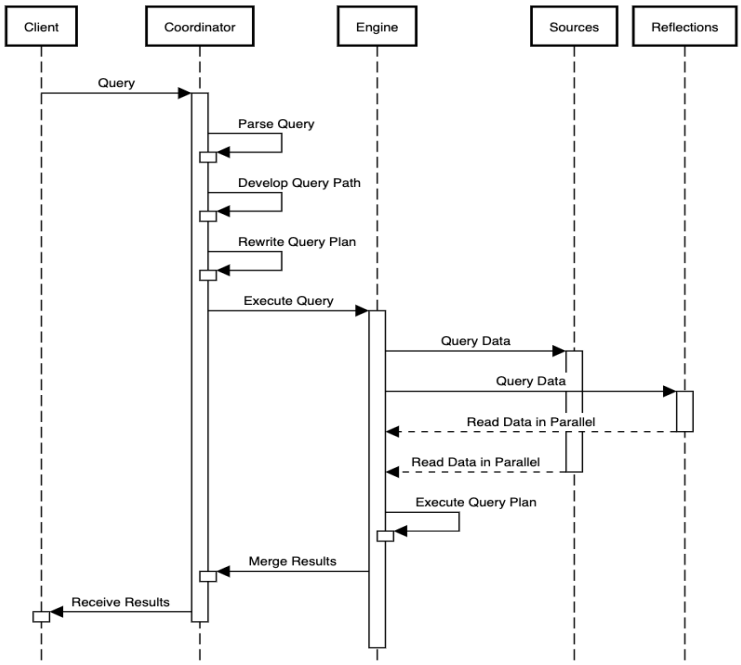

Figure 6 - Query execution phases

QUERY PUSH-DOWNS

Although most data sources reside in data lake storage, where necessary, Dremio can push down processing into external relational and non-relational data sources. Non-relational data sources typically do not support SQL and have limited execution capabilities. A file system, for example, cannot apply predicates or aggregations. The Dremio optimizer understands the capabilities of each data source. When it is most efficient, Dremio pushes as much of a query to the underlying source as possible and perform the rest in its own distributed execution engine.

DATA REFLECTIONS

Dremio accelerates queries by utilizing highly-optimized physical representations of source data called *Data Reflections.* The Data Reflection Store can live on cloud storage such as Amazon S3, Azure ADLS, HDFS or direct-attached storage (DAS). The Data Reflection Store size can exceed that of physical memory. This architecture enables Dremio to accelerate more data at a lower cost, resulting in a much higher cache hit ratio compared to traditional memory-only architectures. Data Reflections are automatically utilized by Dremio's cost-based optimizer at query time.

Note that Data Reflections are invisible to end users. Unlike OLAP cubes, aggregation tables and BI extracts, the user does not explicitly connect to a Data Reflection. Instead, users connect to virtual data sets (i.e., views of data) in Dremio's semantic layer, and issue queries against Dremio's logical model. Dremio's query optimizer automatically accelerates the query by taking advantage of Data Reflections that are suitable for the query based on the optimizer's dependency graph and cost analysis.

By default, Dremio utilizes its high-performance distributed execution engine, leveraging columnar in-memory processing (via Apache Arrow) and advanced push-downs into the underlying data sources where available (see Figure 7).

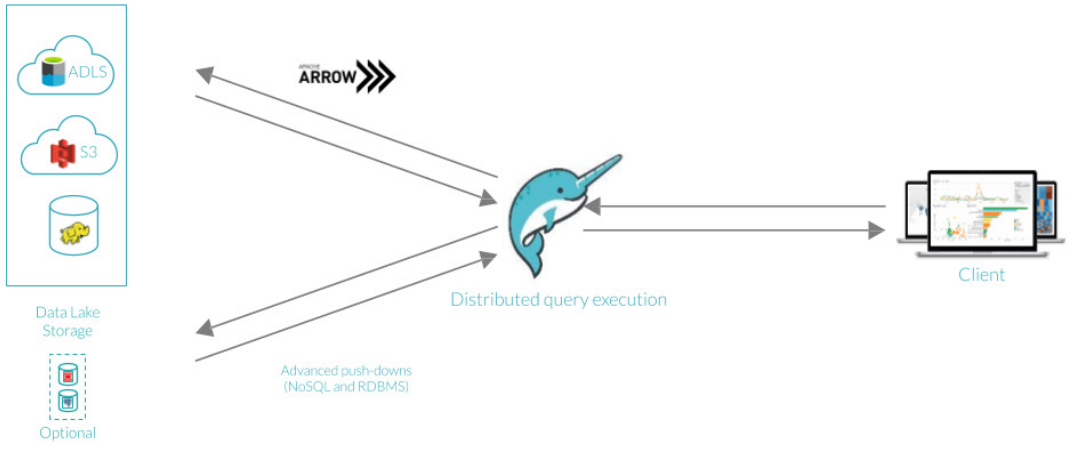

Figure 7 - Default query execution path

In some cases where the entire query can be satisfied by Data Reflections without consulting the data source, the plan is rewritten by the optimizer to utilize the Data Reflections (see Figure 8).

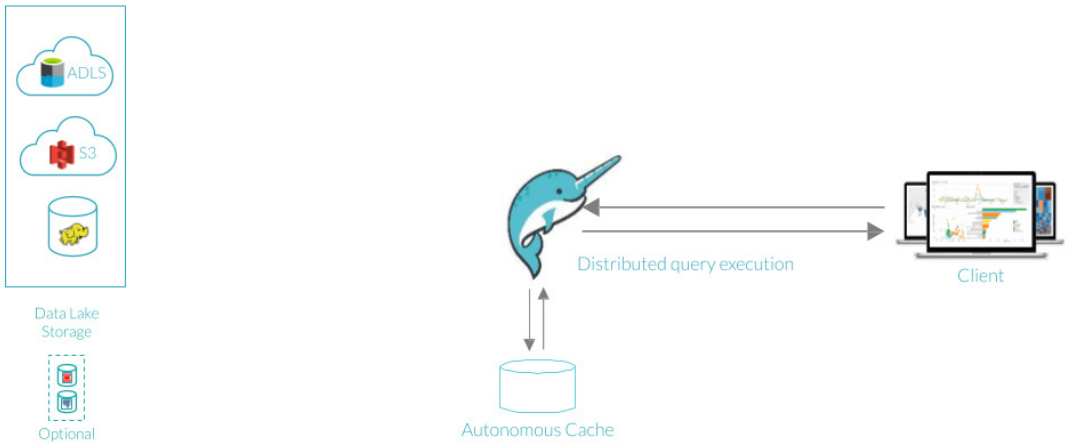

Figure 8 - Query execution with Data Reflection acceleration

There are, of course, situations where the optimal plan includes a combination of data from the data source and one or more Data Reflections (see Figure 9).

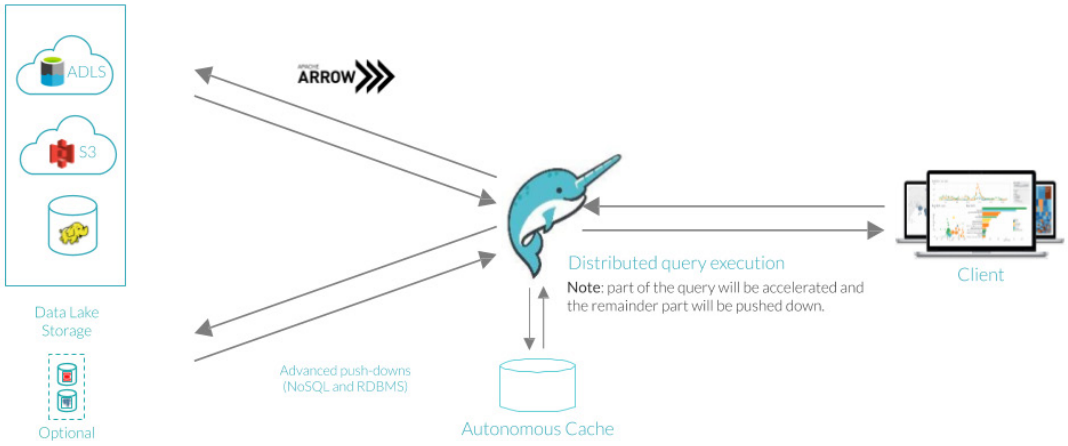

Figure 9 - Query execution with a partial Data Reflection acceleration

A Data Reflection is represented as a logical plan and the corresponding physical materialization. The logical plan may be based on one or more physical datasets (e.g., S3 bucket, log file, log directory, Hive table, Oracle table). From a management standpoint, a Data Reflection is anchored to a single physical or virtual dataset, although at query time it can accelerate queries on other datasets as well.

The physical materialization of a Data Reflection is based on Apache Parquet, with a variety of surrounding optimizations such as column-level statistics. Data Reflections can be sorted, partitioned, aggregated and distributed by specific columns.

To understand how Data Reflections work, we must answer the following independent questions:

1. How are Data Reflections managed? How are they created and refreshed?
2. How are Data Reflections used to accelerate queries?
3. How are Data Reflections physically stored and optimized?

Managing Data Reflections

From a management standpoint, Data Reflections are always anchored to a specific physical or virtual dataset in the system. While SQL queries on dataset X are often accelerated via Data Reflections anchored to dataset X, the optimizer is free to use any other Data Reflection to accelerate such queries. In addition, Data Reflections anchored to dataset X may be used by the optimizer to accelerate queries on dataset Y.

There are two types of Data Reflections:

- Raw Reflections: a projection of one or more columns of the anchor dataset. The data can be sorted, partitioned and distributed by different columns of the anchor dataset
- Aggregation Reflections: an aggregation on one or more columns of the anchor dataset. The Data Reflection is defined by dimensions and measures and contains aggregate-level data for each of the measures such as count, sum, min and max. The data can be sorted, partitioned and distributed by different columns of the anchor dataset.

There may be unique circumstances in which it is desirable to create a custom Data Reflection, which is similar to a materialized view in a relational database. In such cases, simply create a new virtual dataset with the SQL query that defines the desired materialization, and create a single raw Reflection that includes all columns in the virtual dataset.

Refreshing Data Reflections

Dremio automatically refreshes Data Reflections to ensure data freshness. There are three refresh methods:

Full refresh. This method is most suitable for mutating datasets.

- Incremental refresh for file-based sources. This method is most suitable for large, append-only datasets that are represented as a collection of files (e.g., S3). The system automatically identifies new files in the directory.
- Incremental refresh for table-based sources. This method is most suitable for large, append-only data sets represented as tables or collections (e.g., Oracle, Elasticsearch, MongoDB). A monotonically increasing column, such as a created_at timestamp or sequential primary key is required.
- Internally, Dremio maintains a dependency graph (DAG) that defines the order in which Data Reflections are refreshed. The dependencies are calculated based on relational algebra, and the actual refresh start time takes into account the expected amount of time required to complete the entire refresh cycle.

Note that this graph-based approach reduces the end-to-end cycle time, as well as the compute resources required to complete the cycle. In addition, by leveraging one Data Reflection to refresh another Data Reflection, the system can avoid resource-intensive reads on operational databases more than once.

External Data Reflections

For unique cases, Dremio supports the notion of External Data Reflections. Users can create and maintain Data Reflections using an external process such as Apache Spark, then register the External Data Reflection in Dremio. Dremio considers these Data Reflections in its cost-based analysis to accelerate queries. Any data source that Dremio supports can be used for External Data Reflections.

External Data Reflections are useful in cases where it makes sense to manage the creation outside of the Dremio process (e.g., processing that runs for many hours or days, or when existing processes are already in place that create optimized representations of data for specific query patterns). The advantage of registering these resources in Dremio is that it simplifies the experience for data consumers, and provides additional capabilities for securing and governing access to the data, as well as tracking data lineage.

Using Data Reflections to Accelerate Queries

Dremio includes a cost-based optimizer that not only plans and optimizes queries but also explores opportunities to utilize Data Reflections to reduce the query cost (i.e., accelerate the query). When a new query arrives, the optimizer considers all Data Reflections and automatically rewrites the query plan to utilize Data Reflections when possible (see Figure 10). The optimizer utilizes a two-phase algorithm:

1. Data Reflection pruning. The optimizer disregards Data Reflections that are irrelevant because their logical plans have no physical datasets in common with the query's logical plan.

2. Subgraph matching. The optimizer uses an innovative hierarchical graph algorithm to match sub-graphs of the query's logical plan with the logical plans of Data Reflections.

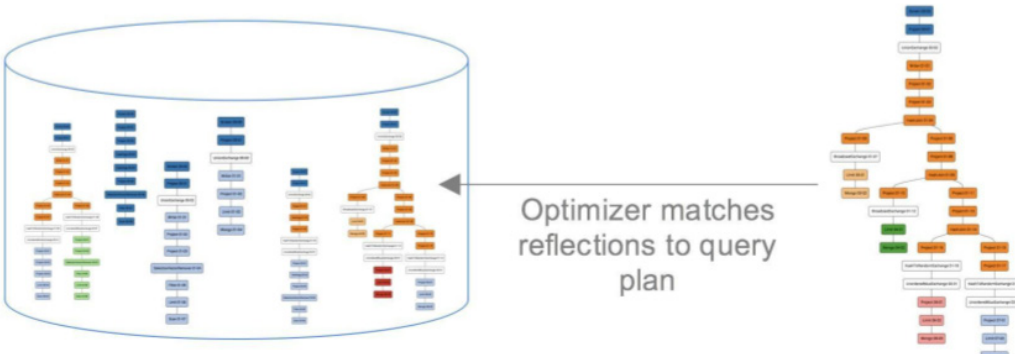

Figure 10 - Subgraph matching

Offloading Operational Databases

Most operational databases are designed for write-optimized workloads. Furthermore, these deployments must address stringent SLAs, and any downtime or degraded performance can significantly impact the business. As a result, operational systems are frequently isolated from processing analytical queries. In these cases, Dremio can execute analytical queries using Data Reflections, which provide the most efficient query processing possible while minimizing the impact on the operational system.

Self-Service Semantic Layer

Dremio establishes views into data (called virtual datasets) in a semantic layer on top of your physical data (see Figure 11), so data analysts and engineers can manage, curate and share data while maintaining governance and security—but without the overhead and complexity of copying data. Connect any BI or data science tool, including Tableau, Power BI, Looker and Jupyter Notebooks, to Dremio and start exploring and mining your data lake for value.

Dremio's semantic layer is fully virtual, indexed and searchable, and the relationships between your data sources, virtual datasets and transformations and all your queries are maintained in Dremio's data graph, so you know exactly where each virtual dataset came from. Role-based access control makes sure that everyone has access to exactly what they need (and nothing else), and SSO enables a seamless authentication experience.

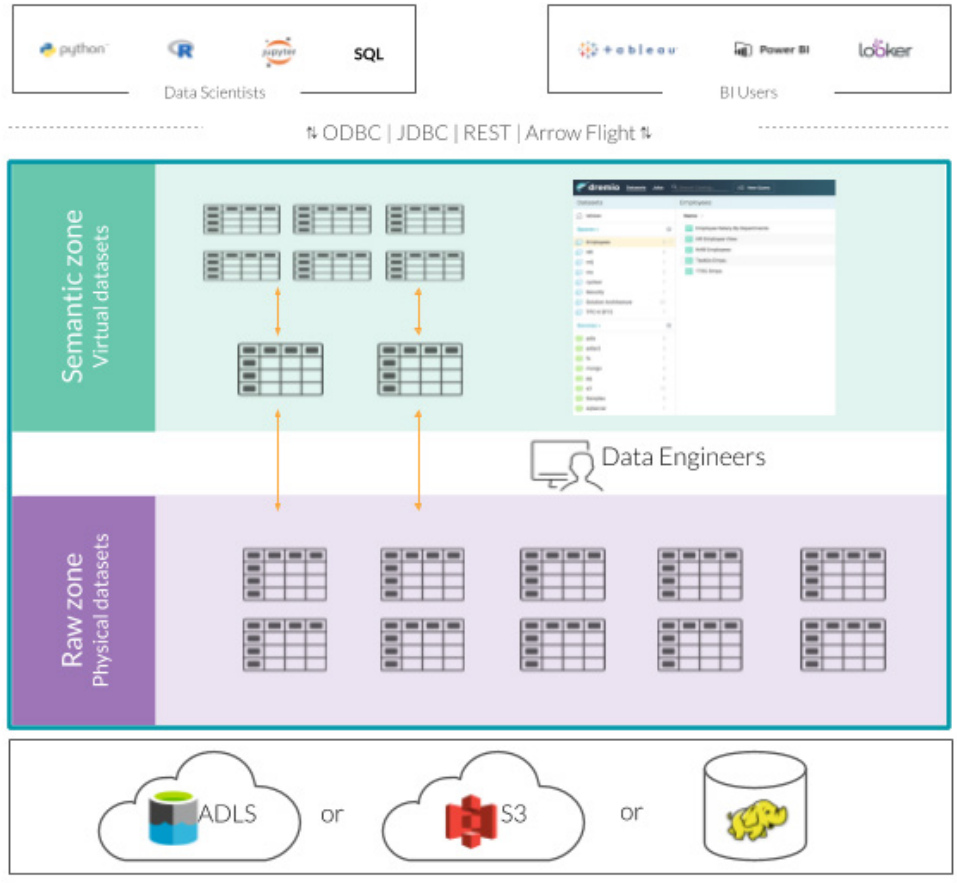

Figure 11 - Dremio semantic layer architecture

SECURITY AND DATA GOVERNANCE

Security and data governance are critical to any enterprise. However, the increasing complexity and demand for data often lead to data sprawl, resulting in significant risk. Dremio, the self-service semantic layer, enables business analysts and data scientists to discover, curate, accelerate and share data without having to export copies of the data into ungoverned systems, including disconnected spreadsheets, ungoverned BI servers and private databases. This reduces the risk of unauthorized data access as well as data theft.

Dremio provides a variety of security and governance capabilities, including:

- Row & column access control on any source
- Data masking
- Data provenance and data lineage
- Authentication based on LDAP/AD
- Role-based access control
- Auditing
- Encryption

DATA PROVENANCE & LINEAGE

Dremio's Data Graph shows the provenance and lineage of every dataset in the system, including those in a user's personal space. IT can easily understand how a dataset was created, transformed, joined and shared, as well as the full lineage of these steps between datasets. This information is captured and tracked by Dremio automatically.

AUTHENTICATION

Dremio supports the following *authentication modes:*

- Local - Dremio manages users internally.
- LDAP - Dremio connects to an existing LDAP-based directory service such as Active Directory. Dremio relies on the directory service for verifying credentials and checking group membership.
- Azure AD - Dremio authenticates users through Azure Active Directory Single Sign On (SSO).
- OpenID - Dremio authenticates users through third-party identity providers using OpenID protocol.

ACCESS CONTROL

Dremio supports fine-grained access control:

- Physical dataset permissions control which users and/or groups can query a specific physical dataset (e.g., HDFS directory, Hive table, Elasticsearch type).
- Virtual dataset permissions control which users and/or groups can query a specific virtual dataset.
- Column-level permissions can be used to restrict access to sensitive columns in a dataset for specific users.
- Row-level permissions can be used to restrict access to a subset of the records in a dataset for specific users.
- Masking of sensitive data is applied dynamically at query time based on LDAP/ AD group membership and other user-defined rules.

Virtual datasets are the recommended mechanism for managing access to data. A user who owns a specific dataset may create a derived virtual dataset, including only a subset of the columns or records of the original dataset. The user can then deny access to the original dataset while enabling access to the derived virtual dataset.

AUDITING

Dremio tracks and records user activity, including all query executions. It serves as a single entry point that shows who's accessing what data, when and how. For example, the Jobs section of the UI provides details on all query history, enabling IT to monitor the system for suspicious activity and identify cases of unauthorized data access.

ENCRYPTION

For encryption on the wire, Dremio leverages both TLS (SSL) and Kerberos. When connecting to a secure Hadoop cluster, Dremio communicates securely with the Hadoop services via Kerberos. For other data sources, Dremio supports the standard wire-level encryption scheme of the source system. For encryption at rest, Dremio leverages the encryption capabilities of the Data Reflection Store (e.g., Amazon S3, Azure ADLS, Hadoop).

ABOUT DREMIO CORPORATION

Dremio's Data Lake Engine delivers fast query speed and a self-service semantic layer operating directly against data lake storage. Dremio eliminates the need to copy and move data to proprietary data warehouses or create cubes, aggregation tables and BI extracts, providing flexibility and control for Data Architects, and self-service for Data Consumers. For more information, visit www.dremio.com.

Founded in 2015, Dremio is headquartered in Santa Clara, CA. Investors include Cisco Investments, Lightspeed Venture Partners, Norwest Venture Partners and Redpoint Ventures. Connect with Dremio on GitHub, LinkedIn, Twitter, and Facebook.