Apache Calcite: A Foundational Framework for Optimized

阿帕奇方解石:优化的基础框架

Query Processing Over Heterogeneous Data Sources

异构数据源上的查询处理

Edmon Begoli

Edmon Begoli

Oak Ridge National Laboratory (ORNL)

橡树岭国家实验室(ORNL)

Oak Ridge, Tennessee, USA begolie@ornl.gov

美国田纳西州橡树岭 begolie@ornl.gov

Jesús Camacho-Rodríguez

赫苏斯·卡马乔-罗德里格斯

Hortonworks Inc.

霍顿工程公司。

Santa Clara, California, USA jcamacho@hortonworks.com

美国加州圣克拉拉 jcamacho@hortonworks.com

Julian Hyde

朱利安·海德

Hortonworks Inc.

霍顿工程公司。

Santa Clara, California, USA jhyde@hortonworks.com

美国加州圣克拉拉 jhyde@hortonworks.com

Michael J. Mior

Michael J. Mior

David R. Cheriton School of Computer Science University of Waterloo Waterloo, Ontario, Canada mmior@uwaterloo.ca

加拿大安大略省滑铁卢大学计算机科学学院

Daniel Lemire

丹尼尔·雷米尔

University of Quebec (TELUQ) Montreal, Quebec, Canada lemire@gmail.com

加拿大魁北克省蒙特利尔魁北克大学 lemire@gmail.com 分校

ABSTRACT

摘要

Apache Calcite is a foundational software framework that provides query processing, optimization, and query language support to many popular open-source data processing systems such as Apache Hive, Apache Storm, Apache Flink, Druid, and MapD.Calcite's ar-chitecture consists of a modular and extensible query optimizer with hundreds of built-in optimization rules, a query processor capable of processing a variety of query languages, an adapter ar-chitecture designed for extensibility, and support for heterogeneous data models and stores (relational, semi-structured, streaming, and geospatial).This flexible, embeddable, and extensible architecture is what makes Calcite an attractive choice for adoption in big-data frameworks.It is an active project that continues to introduce support for the new types of data sources, query languages, and approaches to query processing and optimization.

Apache 方解石是一个基础软件框架，为许多流行的开源数据处理系统提供查询处理、优化和查询语言支持，如 Apache Hive、Apache Storm、Apache Flink、Druid 和 MapD。方解石的体系结构包括一个具有数百个内置优化规则的模块化可扩展查询优化器、一个能够处理多种查询语言的查询处理器、一个为可扩展性设计的适配器体系结构，以及对异构数据模型和存储(关系、半结构化、流式和地理空间)的支持。这种灵活的、可嵌入的和可扩展的体系结构使得方解石成为大数据框架中一个有吸引力的选择。这是一个活跃的项目，它将继续引入对新型数据源、查询语言以及查询处理和优化方法的支持。

argued that specialized engines can offer more cost-effective per-formance and that they would bring the end of the "one size fits all" paradigm.Their vision seems today more relevant than ever.Indeed, many specialized open-source data systems have since be-come popular such as Storm [50] and Flink [16] (stream processing), Elasticsearch [15] (text search), Apache Spark [47], Druid [14], etc.

认为专用引擎可以提供更具成本效益的性能，它们将终结“一刀切”的模式。他们的愿景今天似乎比以往任何时候都更加重要。事实上，许多专门的开源数据系统已经变得流行起来，比如风暴[50]和弗林克[16](流处理)，弹性搜索[15](文本搜索)，阿帕奇火花[47]，德鲁伊[14]，等等。

As organizations have invested in data processing systems tai-lored towards their specific needs, two overarching problems have arisen:

随着各组织为满足其特定需求而投资数据处理系统，出现了两个主要问题:

• The developers of such specialized systems have encoun-tered related problems, such as query optimization [4, 25] or the need to support query languages such as SQL and related extensions (e.g., streaming queries [26]) as well as language-integrated queries inspired by LINQ [33].With-out a unifying framework, having multiple engineers inde-pendently develop similar optimization logic and language support wastes engineering effort.

这种专门系统的开发人员遇到了相关的问题，如查询优化[4，25]或需要支持查询语言，如 SQL 和相关扩展(如流查询[26])以及受 LINQ 启发的语言集成查询[33]。没有统一的框架，让多个工程师独立开发相似的优化逻辑和语言支持会浪费工程精力。

• Programmers using these specialized systems often have to integrate several of them together.An organization might rely on Elasticsearch, Apache Spark, and Druid.We need to build systems capable of supporting optimized queries across heterogeneous data sources [55].

使用这些专用系统的程序员通常必须将几个系统集成在一起。一个组织可能依赖弹性搜索、阿帕奇火花和德鲁伊。我们需要构建能够支持跨异构数据源的优化查询的系统[55]。

Apache Calcite was developed to solve these problems.It is a complete query processing system that provides much of the common functionality—query execution, optimization, and query languages—required by any database management system, except for data storage and management, which are left to specialized engines.Calcite was quickly adopted by Hive, Drill [13], Storm, and many other data processing engines, providing them with advanced query optimizations and query languages.For example, Hive [24] is a popular data warehouse project built on top of Apache Hadoop.As Hive moved from its batch processing roots towards an interactive SQL query answering platform, it became clear that the project needed a powerful optimizer at its core.Thus, Hive adopted Calcite as its optimizer and their integration has been growing since.Many other projects and products have followed suit, including Flink, MapD [12], etc.

阿帕奇方解石就是为了解决这些问题而开发的。它是一个完整的查询处理系统，提供了任何数据库管理系统所需的许多通用功能一查询执行、优化和查询语言，但数据存储和管理除外，这些功能留给了专门的引擎。方解石很快被蜂巢、钻头[13]、风暴和许多其他数据处理引擎所采用，为它们提供了先进的查询优化和查询语言。例如，Hive [24]是一个流行的建立在 Apache Hadoop 之上的数据仓库项目。随着 Hive 从它的批处理根转移到一个交互式的 SQL 查询应答平台，很明显，这个项目的核心需要一个强大的优化器。因此，蜂巢采用方解石作为它的优化器，它们的集成一直在增长。许多其他项目和产品也紧随其后，包括 Flink、MapD [12]等。

http://calcite.apache.org/docs/powered_by

http://calcite.apache.org/docs/powered_by

CCS CONCEPTS

CCS CONFERENCES

• Information systems → DBMS engine architectures;

信息系统→数据库管理系统引擎架构;

KEYWORDS

关键词

Apache Calcite, Relational Semantics, Data Management, Query Algebra, Modular Query Optimization, Storage Adapters

阿帕奇方解石，关系语义学，数据管理，查询代数，模块化查询优化，存储适配器

1 INTRODUCTION

1 导言

Following the seminal System R, conventional relational database engines dominated the data processing landscape.Yet, as far back as 2005, Stonebraker and Çetintemel [49] predicted that we would see the rise a collection of specialized engines such as column stores, stream processing engines, text search engines, and so forth.They

继开创性的系统 R 之后，传统的关系数据库引擎主导了数据处理领域。然而，早在 2005 年，Stonebraker 和 thentimel[49]就预测，我们将会看到专业化引擎的兴起，例如列存储、流处理引擎、文本搜索引擎等等。她们

SIGMOD'18, June 10-15, 2018, Houston, TX, USA

2018 年 6 月 10 日至 15 日，美国德克萨斯州休斯顿

Furthermore, Calcite enables cross-platform optimization by exposing a common interface to multiple systems.To be efficient, the optimizer needs to reason globally, e.g., make decisions across different systems about materialized view selection.

此外，方解石通过将一个公共接口暴露给多个系统来实现跨平台优化。为了提高效率，优化器需要进行全局推理，例如，在不同的系统之间做出关于物化视图选择的决策。

Building a common framework does not come without chal-lenges.In particular, the framework needs to be extensible and flexible enough to accommodate the different types of systems requiring integration.

建立一个共同的框架并非没有挑战。特别是，该框架需要足够的可扩展性和灵活性，以适应需要集成的不同类型的系统。

We believe that the following features have contributed to Cal-cite's wide adoption in the open source community and industry:

我们相信以下特性有助于 Cal-cite 在开源社区和行业中的广泛采用:

• Open source friendliness.Many of the major data pro-cessing platforms of the last decade have been either open source or largely based on open source.Calcite is an open-source framework, backed by the Apache Software Founda-tion (ASF) [5], which provides the means to collaboratively develop the project.Furthermore, the software is written in Java making it easier to interoperate with many of the latest data processing systems [12, 13, 16, 24, 28, 44] that are often written themselves in Java (or in the JVM-based Scala), especially those in the Hadoop ecosystem.

开源友好。过去十年中，许多主要的数据处理平台要么是开源的，要么主要基于开源。方解石是一个开源框架，由阿帕奇软件基金会(ASF) [5]支持，它提供了合作开发项目的方法。此外，该软件是用 Java 编写的，使得它更容易与许多最新的数据处理系统[12，13，16，24，28，44]进行互操作，这些系统通常是用 Java(或基于 JVM 的 Scala)编写的，尤其是在 Hadoop 生态系统中。

• Multiple data models.Calcite provides support for query optimization and query languages using both streaming and conventional data processing paradigms.Calcite treats streams as time-ordered sets of records or events that are not persisted to the disk as they would be in conventional data processing systems.

多种数据模型。方解石提供了使用流和传统数据处理范例的查询优化和查询语言的支持。方解石将数据流视为按时间顺序排列的记录或事件集，它们不会像传统的数据处理系统那样持久存储在磁盘上。

• Flexible query optimizer.Each component of the opti-mizer is pluggable and extensible, ranging from rules to cost models.In addition, Calcite includes support for multiple planning engines.Hence, the optimization can be broken down into phases handled by different optimization engines depending on which one is best suited for the stage.

灵活的查询优化器。optimizer 的每个组件都是可插拔和可扩展的，从规则到成本模型。此外，方解石还支持多个规划引擎。因此，优化可以被分解成由不同的优化引擎处理的阶段，这取决于哪一个最适合该阶段。

• Cross-system support.The Calcite framework can run and optimize queries across multiple query processing systems and database backends.

跨系统支持。方解石框架可以跨多个查询处理系统和数据库后端运行和优化查询。

• Reliability.Calcite is reliable, as its wide adoption over many years has led to exhaustive testing of the platform.Calcite also contains an extensive test suite validating all components of the system including query optimizer rules and integration with backend data sources.

可靠性。方解石是可靠的，因为多年来它的广泛应用导致了对平台的彻底测试。方解石还包含一个广泛的测试套件，用于验证系统的所有组件，包括查询优化器规则和与后端数据源的集成。

• Support for SQL and its extensions.Many systems do not provide their own query language, but rather prefer to rely on existing ones such as SQL.For those, Calcite provides sup-port for ANSI standard SQL, as well as various SQL dialects and extensions, e.g., for expressing queries on streaming or nested data.In addition, Calcite includes a driver conforming to the standard Java API (JDBC).

支持 SQL 及其扩展。许多系统不提供自己的查询语言，而是更喜欢依赖现有的语言，如 SQL。为此，方解石提供了对 ANSI 标准 SQL 的支持，以及各种 SQL 方言和扩展，例如，用于表达对流或嵌套数据的查询。此外，方解石还包括一个符合标准 Java 应用编程接口(JDBC)的驱动程序。

The remainder is organized as follows.Section 2 discusses re-lated work.Section 3 introduces Calcite's architecture and its main components.Section 4 describes the relational algebra at the core of Calcite.Section 5 presents Calcite's adapters, an abstraction to define how to read external data sources.In turn, Section 6 describes Calcite's optimizer and its main features, while Section 7 presents the extensions to handle different query processing paradigms.Sec-tion 8 provides an overview of the data processing systems already

其余的组织如下。第 2 节讨论了相关的工作。第 3 节介绍了方解石的结构及其主要成分。第 4 节描述了方解石核心的关系代数。第 5 节介绍了方解石的适配器，一个定义如何读取外部数据源的抽象。接下来，第 6 节描述了方解石的优化器及其主要特性，而第 7 节介绍了处理不同查询处理范例的扩展。第 8 节已经概述了数据处理系统

using Calcite.Section 9 discusses possible future extensions for the framework before we conclude in Section 10.

使用方解石。在第 10 节结束之前，第 9 节讨论了框架未来可能的扩展。

2 RELATED WORK

2 相关工作

Though Calcite is currently the most widely adopted optimizer for big-data analytics in the Hadoop ecosystem, many of the ideas that lie behind it are not novel.For instance, the query optimizer builds on ideas from the Volcano [20] and Cascades [19] frameworks, incorporating other widely used optimization techniques such as materialized view rewriting [10, 18, 22].There are other systems that try to fill a similar role to Calcite.

尽管方解石目前是 Hadoop 生态系统中最广泛采用的大数据分析优化器，但其背后的许多想法并不新颖。例如，查询优化器建立在火山[20]和瀑布[19]框架的基础上，结合了其他广泛使用的优化技术，如物化视图重写[10，18，22]。还有其他系统试图扮演类似方解石的角色。

Orca [45] is a modular query optimizer used in data manage-ment products such as Greenplum and HAWQ.Orca decouples the optimizer from the query execution engine by implementing a framework for exchanging information between the two known as Data eXchange Language.Orca also provides tools for verifying the correctness and performance of generated query plans.In contrast to Orca, Calcite can be used as a standalone query execution engine that federates multiple storage and processing backends, including pluggable planners, and optimizers.

Orca [45]是一个模块化查询优化器，用于 Greenplum 和 HAWQ 等数据管理产品。Orca 通过实现一个称为数据交换语言(Data eXchange Language)的框架，将优化器与查询执行引擎分离开来。Orca 还提供了用于验证生成的查询计划的正确性和性能的工具。与 Orca 不同的是，方解石可以用作独立的查询执行引擎，联合多个存储和处理后端，包括可插拔规划器和优化器。

Spark SQL [3] extends Apache Spark to support SQL query exe-cution which can also execute queries over multiple data sources as in Calcite.However, although the Catalyst optimizer in Spark SQL also attempts to minimize query execution cost, it lacks the dynamic programming approach used by Calcite and risks falling into local minima.

Spark SQL [3]扩展了 Apache Spark，以支持 SQL 查询执行，它也可以在多个数据源上执行查询，如在方解石中。然而，尽管火花 SQL 中的 Catalyst 优化器也试图最小化查询执行成本，但它缺乏方解石所使用的动态编程方法，并且有陷入局部极小值的风险。

Algebricks [6] is a query compiler architecture that provides a data model agnostic algebraic layer and compiler framework for big data query processing.High-level languages are compiled to Algebricks logical algebra.Algebricks then generates an opti-mized job targeting the Hyracks parallel processing backend.While Calcite shares a modular approach with Algebricks, Calcite also includes a support for cost-based optimizations.In the current version of Calcite, the query optimizer architecture uses dynamic programming-based planning based on Volcano [20] with exten-sions for multi-stage optimizations as in Orca [45].Though in prin-ciple Algebricks could support multiple processing backends (e.g., Apache Tez, Spark), Calcite has provided well-tested support for diverse backends for many years.

Algebricks [6]是一个查询编译器架构，它为大数据查询处理提供了一个数据模型不可知的代数层和编译器框架。高级语言被编译成代数逻辑代数。然后 Algebricks 生成一个优化作业，目标是 Hyracks 并行处理后端。尽管方解石与 Algebricks 共享一种模块化方法，但方解石也支持基于成本的优化。在方解石的当前版本中，查询优化器体系结构使用基于火山[20]的动态编程规划，并扩展了奥卡[45]中的多阶段优化。尽管在原则上，Algebricks 可以支持多个加工后端(例如，Apache Tez，Spark)，但方解石多年来一直为不同的后端提供了经过良好测试的支持。

Garlic [7] is a heterogeneous data management system which represents data from multiple systems under a unified object model.However, Garlic does not support query optimization across differ-ent systems and relies on each system to optimize its own queries.

大蒜[7]是一个异构数据管理系统，它在一个统一的对象模型下表示来自多个系统的数据。但是，大蒜不支持跨不同系统的查询优化，而是依赖每个系统来优化自己的查询。

FORWARD [17] is a federated query processor that implements a superset of SQL called SQL++ [38].SQL++ has a semi-structured data model that integrate both JSON and relational data models whereas Calcite supports semi-structured data models by repre-senting them in the relational data model during query planning.FORWARD decomposes federated queries written in SQL++ into subqueries and executes them on the underlying databases accord-ing to the query plan.The merging of data happens inside the FORWARD engine.

FORWARD [17]是一个联邦查询处理器，它实现了一个名为 SQL++的超集。SQL++有一个半结构化数据模型，它集成了 JSON 和关系数据模型，而方解石支持半结构化数据模型，方法是在查询计划期间将它们表示在关系数据模型中。FORWARD 将用 SQL++编写的联邦查询分解为子查询，并根据查询计划在底层数据库上执行它们。数据合并发生在转发引擎内部。

Another federated data storage and processing system is Big-DAWG, which abstracts a wide spectrum of data models including relational, time-series and streaming.The unit of abstraction in

另一个联合数据存储和处理系统是 Big-DOUGH，它抽象了一系列数据模型，包括关系模型、时间序列模型和流模型。抽象的单位

Apache Calcite SIGMOD'18, June 10-15, 2018, Houston, TX, USA

2018 年 6 月 10 日至 15 日，美国德克萨斯州休斯顿

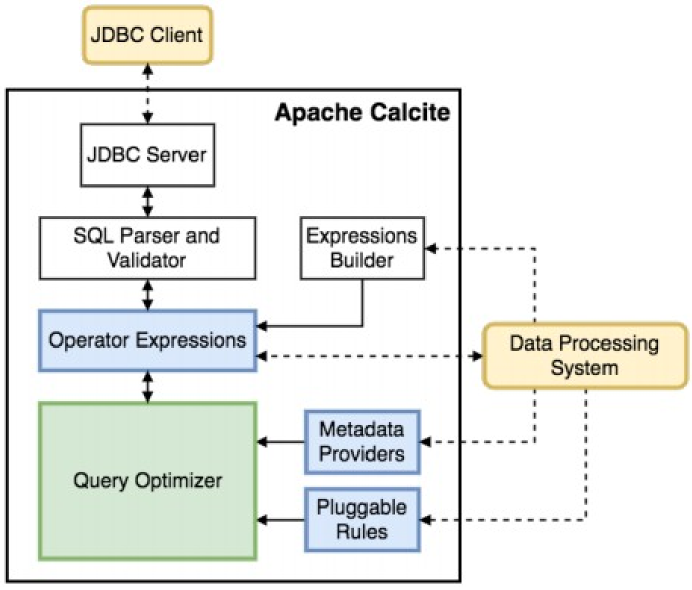


Figure 1: Apache Calcite architecture and interaction.

图 1:阿帕奇方解石建筑和互动。

BigDAWG is called an island of information.Each island of informa-tion has a query language, data model and connects to one or more storage systems.Cross storage system querying is supported within the boundaries of a single island of information.Calcite instead provides a unifying relational abstraction which allows querying across backends with different data models.

大狗被称为信息之岛。每个信息孤岛都有一种查询语言和数据模型，并连接到一个或多个存储系统。在单个信息孤岛的边界内支持跨存储系统查询。相反，方解石提供了统一的关系抽象，允许使用不同的数据模型跨后端进行查询。

Myria is a general-purpose engine for big data analytics, with advanced support for the Python language [21].It produces query plans for other backend engines such as Spark and PostgreSQL.

Myria 是大数据分析的通用引擎，对 Python 语言有高级支持[21]。它为其他后端引擎(如 Spark 和 PostgreSQL)生成查询计划。

3 ARCHITECTURE

3 ARCHITY

Calcite contains many of the pieces that comprise a typical database management system.However, it omits some key components, e.g., storage of data, algorithms to process data, and a repository for storing metadata.These omissions are deliberate: it makes Calcite an excellent choice for mediating between applications having one or more data storage locations and using multiple data processing engines.It is also a solid foundation for building bespoke data processing systems.

方解石包含许多组成典型数据库管理系统的部分。但是，它省略了一些关键组件，例如数据存储、处理数据的算法以及存储元数据的存储库。这些遗漏是有意的:它使方解石成为在具有一个或多个数据存储位置的应用程序和使用多个数据处理引擎之间进行协调的绝佳选择。这也是构建定制数据处理系统的坚实基础。

Figure 1 outlines the main components of Calcite's architecture.Calcite's optimizer uses a tree of relational operators as its internal representation.The optimization engine primarily consists of three components: rules, metadata providers, and planner engines.We discuss these components in more detail in Section 6.In the figure, the dashed lines represent possible external interactions with the framework.There are different ways to interact with Calcite.

图 1 概述了方解石建筑的主要组成部分。方解石的优化器使用关系运算符树作为其内部表示。优化引擎主要由三个组件组成:规则、元数据提供程序和计划引擎。我们将在第 6 节中更详细地讨论这些组件。在图中，虚线表示与框架可能的外部交互。有不同的方式与方解石相互作用。

First, Calcite contains a query parser and validator that can translate a SQL query to a tree of relational operators.As Calcite does not contain a storage layer, it provides a mechanism to define table schemas and views in external storage engines via adapters (described in Section 5), so it can be used on top of these engines.

首先，方解石包含一个查询解析器和验证器，可以将一个 SQL 查询转换成一个关系运算符树。由于方解石不包含存储层，它提供了一种通过适配器在外部存储引擎中定义表模式和视图的机制(如第 5 节所述)，因此它可以在这些引擎之上使用。

Second, although Calcite provides optimized SQL support to systems that need such database language support, it also provides optimization support to systems that already have their own lan-guage parsing and interpretation:

其次，尽管方解石为需要这种数据库语言支持的系统提供了优化的 SQL 支持，但它也为已经拥有自己的语言解析和解释的系统提供了优化支持:

• Some systems support SQL queries, but without or with lim-ited query optimization.For example, both Hive and Spark initially offered support for the SQL language, but they did not include an optimizer.For such cases, once the query has been optimized, Calcite can translate the relational expres-sion back to SQL.This feature allows Calcite to work as a stand-alone system on top of any data management system with a SQL interface, but no optimizer.

有些系统支持 SQL 查询，但没有或只有有限的查询优化。例如，Hive 和 Spark 最初都支持 SQL 语言，但是它们没有包含优化器。在这种情况下，一旦查询得到优化，方解石就可以将关系表达式转换回 SQL。这个特性允许方解石作为独立系统工作在任何数据管理系统之上，具有一个 SQL 接口，但是没有优化器。

• The Calcite architecture is not only tailored towards op-timizing SQL queries.It is common that data processing systems choose to use their own parser for their own query language.Calcite can help optimize these queries as well.Indeed, Calcite also allows operator trees to be easily con-structed by directly instantiating relational operators.One can use the built-in relational expressions builder interface.For instance, assume that we want to express the following Apache Pig [41] script using the expression builder:

方解石体系结构不仅是为优化 SQL 查询而定制的。数据处理系统通常选择使用自己的解析器作为自己的查询语言。方解石也可以帮助优化这些查询。事实上，方解石还允许通过直接实例化关系运算符来轻松构造运算符树。可以使用内置的关系表达式生成器界面。例如，假设我们想使用表达式生成器来表达以下 Apache Pig [41]脚本:

emp = LOAD 'employee_data ' AS (deptno , sal);emp_by_dept = GROUP emp by (deptno);

EMP = LOAD ' employee _ data ' AS(dept no，sal)；EMP _ by _ dept = GROUP EMP by(dept no)；

emp_agg = FOREACH emp_by_dept GENERATE GROUP as deptno , COUNT (emp .sal) AS c , SUM (emp .sal) as s ;dump emp_agg ;

EMP _ agg = FOREACH EMP _ by _ dept GENERATE GrouP as dept no，COUNT (emp。AS c，SUM (emp)。sal)as s；转储 emp _ agg

The equivalent expression looks as follows:

等效表达式如下所示:

final RelNode node = builder .scan (" employee_data ")

最终 RelNode 节点=构建器。扫描(“员工数据”)

.aggregate (builder .groupKey (" deptno ") ,

。聚合(构建器。group key(“dept no”)，

builder .count (false , "c") ,

建筑商。计数(错误，“c”)，

builder .sum (false , "s", builder .field ("sal")))

建筑商。sum (false，“s”，构建器。字段(“sal”))

.build ();

。构建();

This interface exposes the main constructs necessary for building relational expressions.After the optimization phase is finished, the application can retrieve the optimized rela-tional expression which can then be mapped back to the system's query processing unit.

该接口公开了构建关系表达式所需的主要构造。优化阶段完成后，应用程序可以检索优化的关系表达式，然后将该表达式映射回系统的查询处理单元。

4 QUERY ALGEBRA

4 QUERY 代数

Operators.Relational algebra [11] lies at the core of Calcite.In addition to the operators that express the most common data manip-ulation operations, such as filter, project, join etc., Calcite includes additional operators that meet different purposes, e.g., being able to concisely represent complex operations, or recognize optimization opportunities more efficiently.

操作员。关系代数[11]是方解石的核心。除了表示最常见的数据处理操作的操作符，如过滤、项目、连接等。，方解石包括满足不同目的的附加操作符，例如，能够简明地表示复杂的操作，或更有效地识别优化机会。

For instance, it has become common for OLAP, decision making, and streaming applications to use window definitions to express complex analytic functions such as moving average of a quantity over a time period or number or rows.Thus, Calcite introduces a window operator that encapsulates the window definition, i.e., up-per and lower bound, partitioning etc., and the aggregate functions to execute on each window.

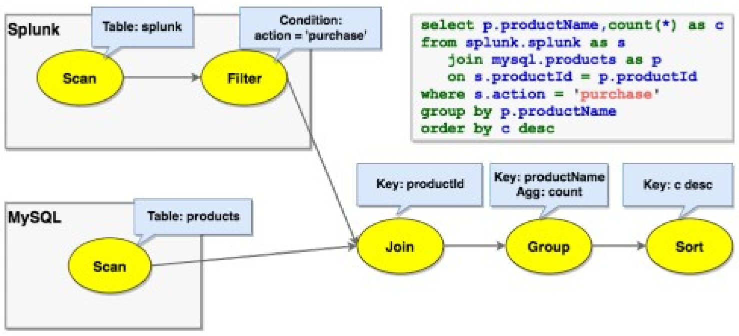
例如，对于 OLAP、决策制定和流应用程序来说，使用窗口定义来表示复杂的分析函数已经变得很普遍，例如一个数量在一个时间段或一个数字或行上的移动平均。因此，方解石引入了一个窗口操作符，它封装了窗口定义，即上限和下限，分区等。，以及要在每个窗口上执行的聚合函数。

Traits.Calcite does not use different entities to represent logical and physical operators.Instead, it describes the physical properties associated with an operator using traits.These traits help the opti-mizer evaluate the cost of different alternative plans.Changing a trait value does not change the logical expression being evaluated, i.e., the rows produced by the given operator will still be the same.

特征。方解石不使用不同的实体来表示逻辑和物理运算符。相反，它使用特征描述了与操作员相关的物理属性。这些特征有助于优化者评估不同备选方案的成本。改变特性值不会改变被评估的逻辑表达式，即由给定运算符产生的行仍然是相同的。

SIGMOD'18, June 10-15, 2018, Houston, TX, USA E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. Mior, and D. Lemire

2018 年 6 月 10 日至 15 日，美国得克萨斯州休斯顿市



```sql
select p.productName,count(*) as c
from splunk.splunk as s
   join mysql.products as p
   on s.productId = p.productId
where s.action = 'purchase'
group by p.productName
order by c desc
```

Figure 2: A Query Optimization Process.

图 2:查询优化过程。

During optimization, Calcite tries to enforce certain traits on relational expressions, e.g., the sort order of certain columns.Rela-tional operators can implement a converter interface that indicates how to convert traits of an expression from one value to another.

在优化过程中，方解石试图在关系表达式上实施某些特征，例如，某些列的排序顺序。相关运算符可以实现转换器接口，该接口指示如何将表达式的特征从一个值转换为另一个值。

Calcite includes common traits that describe the physical proper-ties of the data produced by a relational expression, such as ordering, grouping, and partitioning.Similar to the SCOPE optimizer [57], the Calcite optimizer can reason about these properties and exploit them to find plans that avoid unnecessary operations.For exam-ple, if the input to the sort operator is already correctly ordered— possibly because this is the same order used for rows in the backend system—then the sort operation can be removed.

方解石包含描述由关系表达式产生的数据的物理特性的共同特征，例如排序、分组和划分。类似于 SCOPE 优化器[57]，方解石优化器可以对这些属性进行推理，并利用它们来找到避免不必要操作的计划。例如，如果排序操作符的输入已经被正确排序——可能是因为这是后端系统中用于行的相同顺序——那么排序操作可以被删除。

In addition to these properties, one of the main features of Calcite is the calling convention trait.Essentially, the trait represents the data processing system where the expression will be executed.Including the calling convention as a trait allows Calcite to meet its goal of optimizing transparently queries whose execution might span over different engines i.e., the convention will be treated as any other physical property.

除了这些特性之外，方解石的主要特征之一是召唤惯例特性。本质上，这个特征代表了表达式将被执行的数据处理系统。将调用约定作为一种特性，允许方解石实现其透明优化查询的目标，这些查询的执行可能跨越不同的引擎，即该约定将被视为任何其他物理属性。

For example, consider joining a Products table held in MySQL to an Orders table held in Splunk (see Figure 2).Initially, the scan of Orders takes place in the splunk convention and the scan of Products is in the jdbc-mysql convention.The tables have to be scanned inside their respective engines.The join is in the logical convention, meaning that no implementation has been chosen.Moreover, the SQL query in Figure 2 contains a filter (where clause) which is pushed into splunk by an adapter-specific rule (see Section 5).One possible implementation is to use Apache Spark as an external engine: the join is converted to spark convention, and its inputs are converters from jdbc-mysql and splunk to spark convention.But there is a more efficient implementation: exploiting the fact that Splunk can perform lookups into MySQL via ODBC, a planner rule pushes the join through the splunk-to-spark converter, and the join is now in splunk convention, running inside the Splunk engine.

例如，考虑将 MySQL 中的产品表连接到 Splunk 中的订单表(见图 2)。最初，对订单的扫描按照 splunk 约定进行，对产品的扫描按照 jdbc-mysql 约定进行。这些表格必须在各自的引擎中扫描。连接符合逻辑约定，这意味着没有选择任何实现。此外，图 2 中的 SQL 查询包含一个过滤器(where 子句)，该过滤器由特定于适配器的规则推入 splunk(参见第 5 节)。一种可能的实现是使用 Apache spark 作为外部引擎:连接被转换为 Spark 约定，其输入是从 jdbc-mysql 和 splunk 到 Spark 约定的转换器。但是还有一个更有效的实现:利用 splunk 可以通过 ODBC 在 MySQL 中执行查找的事实，一个计划器规则通过 splunk 到火花转换器来推动连接，并且连接现在是 Splunk 约定，在 Splunk 引擎内部运行。

5 ADAPTERS

5 ADAPTERS

An adapter is an architectural pattern that defines how Calcite incorporates diverse data sources for general access.Figure 3 de-picts its components.Essentially, an adapter consists of a model, a schema, and a schema factory.The model is a specification of the physical properties of the data source being accessed.A schema is the definition of the data (format and layouts) found in the model.The data itself is physically accessed via tables.Calcite interfaces

适配器是一种架构模式，它定义了方解石如何合并不同的数据源以供一般访问。图 3 描述了它的组件。本质上，适配器由模型、模式和模式工厂组成。该模型是正在访问的数据源的物理属性的规范。模式是模型中数据(格式和布局)的定义。数据本身通过表进行物理访问。方解石界面

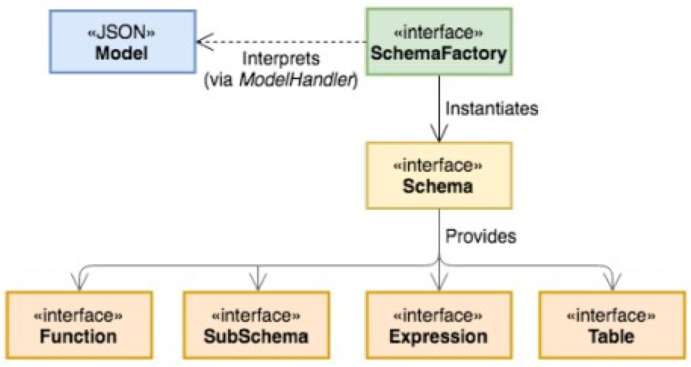


Figure 3: Calcite's Data Source Adapter Design.

图 3:方解石的数据源适配器设计。

with the tables defined in the adapter to read the data as the query is being executed.The adapter may define a set of rules that are added to the planner.For instance, it typically includes rules to convert various types of logical relational expressions to the corre-sponding relational expressions of the adapter's convention.The schema factory component acquires the metadata information from the model and generates a schema.

使用适配器中定义的表在执行查询时读取数据。适配器可以定义一组添加到规划器的规则。例如，它通常包括将各种类型的逻辑关系表达式转换为适配器约定的对应关系表达式的规则。模式工厂组件从模型中获取元数据信息，并生成一个模式。

As discussed in Section 4, Calcite uses a physical trait known as the calling convention to identify relational operators which cor-respond to a specific database backend.These physical operators implement the access paths for the underlying tables in each adapter.When a query is parsed and converted to a relational algebra expres-sion, an operator is created for each table representing a scan of the data on that table.It is the minimal interface that an adapter must implement.If an adapter implements the table scan operator, the Calcite optimizer is then able to use client-side operators such as sorting, filtering, and joins to execute arbitrary SQL queries against these tables.

正如第 4 节所讨论的，方解石使用一种称为调用约定的物理特性来识别与特定数据库后端相对应的关系运算符。这些物理操作符实现了每个适配器中底层表的访问路径。当查询被解析并转换为关系代数表达式时，为每个表创建一个运算符，表示对该表上数据的扫描。这是适配器必须实现的最小接口。如果适配器实现了表扫描操作符，那么方解石优化器就能够使用客户端操作符(如排序、过滤和连接)对这些表执行任意的 SQL 查询。

This table scan operator contains the necessary information the adapter requires to issue the scan to the adapter's backend database.To extend the functionality provided by adapters, Calcite defines an enumerable calling convention.Relational operators with the enumerable calling convention simply operate over tuples via an iterator interface.This calling convention allows Calcite to im-plement operators which may not be available in each adapter's backend.For example, the EnumerableJoin operator implements joins by collecting rows from its child nodes and joining on the desired attributes.

该表扫描操作符包含适配器向适配器后端数据库发出扫描所需的必要信息。为了扩展适配器提供的功能，方解石定义了一个可枚举的调用约定。具有可枚举调用约定的关系运算符只是通过迭代器接口对元组进行操作。这种调用约定允许方解石实现在每个适配器的后端可能不可用的操作符。例如，EnumerableJoin 运算符通过从其子节点收集行并连接到所需的属性来实现连接。

For queries which only touch a small subset of the data in a table, it is inefficient for Calcite to enumerate all tuples.Fortu-nately, the same rule-based optimizer can be used to implement adapter-specific rules for optimization.For example, suppose a query involves filtering and sorting on a table.An adapter which can perform filtering on the backend can implement a rule which matches a LogicalFilter and converts it to the adapter's calling convention.This rule converts the LogicalFilter into another Filter instance.This new Filter node has a lower associated cost that allows Calcite to optimize queries across adapters.

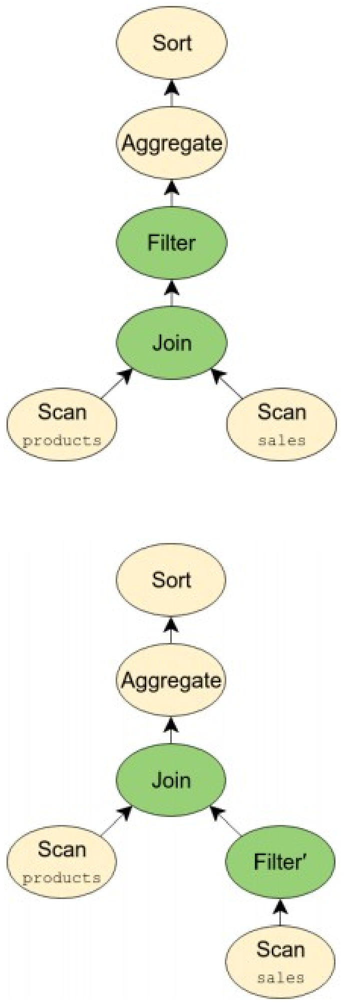
对于只涉及表中一小部分数据的查询，方解石枚举所有元组是低效的。幸运的是，相同的基于规则的优化器可以用来实现适配器特定的优化规则。例如，假设一个查询涉及对一个表进行过滤和排序。可以在后端执行过滤的适配器可以实现与逻辑过滤器匹配的规则，并将其转换为适配器的调用约定。此规则将逻辑筛选器转换为另一个筛选器实例。这个新的过滤器节点具有较低的相关成本，允许方解石优化跨适配器的查询。

The use of adapters is a powerful abstraction that enables not only optimization of queries for a specific backend, but also across multiple backends.Calcite is able to answer queries involving tables across multiple backends by pushing down all possible logic to each

适配器的使用是一个强大的抽象，它不仅能优化特定后端的查询，还能跨多个后端进行优化。方解石能够通过将所有可能的逻辑下推到每个后端来回答涉及多个后端的表的查询

Apache Calcite SIGMOD'18, June 10-15, 2018, Houston, TX, USA

2018 年 6 月 10 日至 15 日，美国德克萨斯州休斯顿



(a) Before

之前

(b) After

之后

Figure 4: FilterIntoJoinRule application.

图 4:FiltrintJoinRule 应用程序。

backend and then performing joins and aggregations on the result-ing data.Implementing an adapter can be as simple as providing a table scan operator or it can involve the design of many advanced optimizations.Any expression represented in the relational algebra can be pushed down to adapters with optimizer rules.

后端，然后对结果数据执行连接和聚合。实现适配器可以像提供表扫描操作符一样简单，也可以涉及许多高级优化的设计。关系代数中表示的任何表达式都可以用优化器规则推送到适配器。

6 QUERY PROCESSING AND OPTIMIZATION

6 查询处理和优化

The query optimizer is the main component in the framework.Calcite optimizes queries by repeatedly applying planner rules to a relational expression.A cost model guides the process, and the planner engine tries to generate an alternative expression that has the same semantics as the original but a lower cost.

查询优化器是框架中的主要组件。方解石通过对关系表达式重复应用计划规则来优化查询。一个成本模型指导着这个过程，计划引擎试图生成一个替代表达式，它与原始表达式具有相同的语义，但是成本较低。

Every component in the optimizer is extensible.Users can add relational operators, rules, cost models, and statistics.Planner rules.Calcite includes a set of planner rules to transform expression trees.In particular, a rule matches a given pattern in the tree and executes a transformation that preserves semantics of that expression.Calcite includes several hundred optimization rules.However, it is rather common for data processing systems relying on Calcite for optimization to include their own rules to allow specific rewritings.

优化器中的每个组件都是可扩展的。用户可以添加关系运算符、规则、成本模型和统计数据。计划者规则。方解石包含一组规划规则来转换表达式树。具体来说，规则匹配树中的给定模式，并执行保留该表达式语义的转换。方解石包含数百个优化规则。然而，依赖方解石进行优化的数据处理系统通常会包含自己的规则，以允许特定的重写。

For example, Calcite provides an adapter for Apache Cassan-dra [29], a wide column store which partitions data by a subset of columns in a table and then within each partition, sorts rows based on another subset of columns.As discussed in Section 5, it is beneficial for adapters to push down as much query processing as possible to each backend for efficiency.A rule to push a Sort into Cassandra must check two conditions:

例如，方解石为 Apache 卡桑-德拉[29]提供了一个适配器，这是一个宽列存储，它按表中列的子集对数据进行分区，然后在每个分区内，根据列的另一个子集对行进行排序。正如第 5 节所讨论的，为了提高效率，适配器将尽可能多的查询处理下推到每个后端是有益的。将排序推入卡珊德拉的规则必须检查两个条件:

(1) the table has been previously filtered to a single partition (since rows are only sorted within a partition) and (2) the sorting of partitions in Cassandra has some common prefix with the required sort.

(1)该表先前已经被过滤到单个分区(因为行只在一个分区内排序)和(2)在 Cassandra 中分区的排序具有一些带有所需排序的公共前缀。

This requires that a LogicalFilter has been rewritten to a CassandraFilter to ensure the partition filter is pushed down to the database.The effect of the rule is simple (convert a LogicalSort into a CassandraSort) but the flexibility in rule matching enables backends to push down operators even in complex scenarios.

这需要将一个逻辑过滤器重写为一个卡珊德拉过滤器，以确保分区过滤器被下推到数据库。规则的效果很简单(将一个逻辑排序转换成一个卡珊德拉排序)，但是规则匹配的灵活性使得后端即使在复杂的场景中也能按下操作符。

For an example of a rule with more complex effects, consider the following query:

对于具有更复杂效果的规则示例，请考虑以下查询:

SELECT products .name , COUNT (*) FROM sales JOIN products USING (productId) WHERE sales .discount IS NOT NULL GROUP BY products .name ORDER BY COUNT (*) DESC ;

选择产品。名称，COUNT (*) FROM sales 使用(productId)加入产品销售地点。折扣不能为空按产品分组。以伯爵(*)的名义订购 desc；

The query corresponds to the relational algebra expression pre-sented in Figure 4a.Because the WHERE clause only applies to the sales table, we can move the filter before the join as in Fig-ure 4b.This optimization can significantly reduce query execution time since we do not need to perform the join for rows which do match the predicate.Furthermore, if the sales and products tables were contained in separate backends, moving the filter be-fore the join also potentially enables an adapter to push the fil-ter into the backend.Calcite implements this optimization via FilterIntoJoinRule which matches a filter node with a join node as a parent and checks if the filter can be performed by the join.This optimization illustrates the flexibility of the Calcite approach to optimization.

该查询对应于图 4a 中给出的关系代数表达式。因为 WHERE 子句只适用于销售表，所以我们可以在连接之前移动过滤器，如图 4b 所示。这种优化可以显著减少查询执行时间，因为我们不需要对与谓词匹配的行执行连接。此外，如果销售和产品表包含在单独的后端，那么在连接之前移动过滤器也有可能使适配器将过滤器推入后端。方解石通过 FilterIntoJoinRule 实现这种优化，该规则将一个过滤器节点与一个作为父节点的连接节点进行匹配，并检查该连接是否可以执行过滤器。这种优化说明了方解石优化方法的灵活性。

Metadata providers.Metadata is an important part of Calcite's optimizer, and it serves two main purposes: (i) guiding the planner towards the goal of reducing the cost of the overall query plan, and (ii) providing information to the rules while they are being applied.

元数据提供者。元数据是方解石优化器的重要组成部分，它有两个主要目的:(1)引导计划者实现降低整体查询计划成本的目标，以及(2)在应用规则时向规则提供信息。

Metadata providers are responsible for supplying that informa-tion to the optimizer.In particular, the default metadata providers implementation in Calcite contains functions that return the overall cost of executing a subexpression in the operator tree, the num-ber of rows and the data size of the results of that expression, and the maximum degree of parallelism with which it can be executed.In turn, it can also provide information about the plan structure, e.g., filter conditions that are present below a certain tree node.

元数据提供者负责向优化器提供该信息。具体来说，方解石中的默认元数据提供程序实现包含返回在运算符树中执行子表达式的总成本、该表达式结果的行数和数据大小以及可执行的最大并行度的函数。反过来，它还可以提供关于计划结构的信息，例如，特定树节点下存在的过滤条件。

Calcite provides interfaces that allow data processing systems to plug their metadata information into the framework.These systems may choose to write providers that override the existing functions, or provide their own new metadata functions that might be used during the optimization phase.However, for many of them, it is sufficient to provide statistics about their input data, e.g., number of rows and size of a table, whether values for a given column are unique etc., and Calcite will do the rest of the work by using its default implementation.

方解石提供了允许数据处理系统将元数据信息插入框架的接口。这些系统可以选择编写覆盖现有函数的提供程序，或者提供它们自己的新的元数据函数，这些函数可能在优化阶段使用。然而，对于它们中的许多，提供关于它们的输入数据的统计就足够了，例如，行数和表的大小，给定列的值是否唯一等。，方解石将通过使用其默认实现来完成剩余的工作。

As the metadata providers are pluggable, they are compiled and instantiated at runtime using Janino [27], a Java lightweight com-piler.Their implementation includes a cache for metadata results, which yields significant performance improvements, e.g., when we need to compute multiple types of metadata such as cardinal-ity, average row size, and selectivity for a given join, and all these computations rely on the cardinality of their inputs.

由于元数据提供者是可插入的，所以它们在运行时使用 Janino [27]进行编译和实例化，Janino 是一个 Java 轻量级编译器。它们的实现包括元数据结果的缓存，这产生了显著的性能改进，例如，当我们需要计算多种类型的元数据时，例如基数、平均行大小和给定连接的选择性，并且所有这些计算都依赖于它们的输入基数。

Planner engines.The main goal of a planner engine is to trigger the rules provided to the engine until it reaches a given objective.At the moment, Calcite provides two different engines.New engines are pluggable in the framework.

计划引擎。计划引擎的主要目标是触发提供给引擎的规则，直到它达到给定的目标。目前，方解石提供了两种不同的引擎。新的引擎可以在框架中插入。

The first one, a cost-based planner engine, triggers the input rules with the goal of reducing the overall expression cost.The engine uses a dynamic programming algorithm, similar to Volcano [20], to create and track different alternative plans created by firing the

第一个是基于成本的计划引擎，它触发输入规则，目标是降低总体表达式成本。该引擎使用一个动态编程算法，类似于火山[20]，创建和跟踪不同的替代计划创建的发射

rules given to the engine.Initially, each expression is registered with the planner, together with a digest based on the expression attributes and its inputs.When a rule is fired on an expression e1 and the rule produces a new expression e2, the planner will add e2 to the set of equivalence expressions Sa that e1 belongs to.In addition, the planner generates a digest for the new expression, which is compared with those previously registered in the planner.If a similar digest associated with an expression e3 that belongs to a set Sb is found, the planner has found a duplicate and hence will merge Sa and Sb into a new set of equivalences.The process continues until the planner reaches a configurable fix point.In particular, it can (i) exhaustively explore the search space until all rules have been applied on all expressions, or (ii) use a heuristic-based approach to stop the search when the plan cost has not improved by more than a given threshold δ in the last planner iterations.The cost function that allows the optimizer to decide which plan to choose is supplied through metadata providers.The default cost function implementation combines estimations for CPU, IO, and memory resources used by a given expression.

给予引擎的规则。最初，每个表达式都在规划器中注册，并带有基于表达式属性及其输入的摘要。当在表达式 e1 上触发一个规则并且该规则产生一个新的表达式 e2 时，规划者将把 e2 添加到 e1 所属的等价表达式 s a 的集合中。此外，计划员为新表达式生成一个摘要，该摘要将与之前在计划员中注册的摘要进行比较。如果找到了与属于集合 Sb 的表达式 e3 相关联的相似摘要，则规划者已经找到了副本，因此将 Sa 和 Sb 合并到新的等价集合中。该过程一直持续到规划器到达可配置的固定点。具体来说，它可以(I)彻底探索搜索空间，直到所有规则都应用于所有表达式，或者(ii)当计划成本在最后一次计划迭代中没有提高超过给定阈值 δ 时，使用基于启发式的方法来停止搜索。允许优化器决定选择哪个计划的成本函数是通过元数据提供者提供的。默认的成本函数实现结合了对给定表达式使用的中央处理器、输入输出和内存资源的估计。

The second engine is an exhaustive planner, which triggers rules exhaustively until it generates an expression that is no longer mod-ified by any rules.This planner is useful to quickly execute rules without taking into account the cost of each expression.

第二个引擎是一个详尽的计划器，它详尽地触发规则，直到它生成一个不再被任何规则修改的表达式。这种规划器在不考虑每个表达式的成本的情况下快速执行规则非常有用。

Users may choose to use one of the existing planner engines depending on their concrete needs, and switching from one to another, when their system requirements change, is straightforward.Alternatively, users may choose to generate multi-stage optimization logic, in which different sets of rules are applied in consecutive phases of the optimization process.Importantly, the existence of two planners allows Calcite users to reduce the overall optimization time by guiding the search for different query plans.

用户可以根据他们的具体需求选择使用现有的计划引擎，当他们的系统需求改变时，从一个引擎切换到另一个引擎是很简单的。或者，用户可以选择生成多阶段优化逻辑，其中在优化过程的连续阶段应用不同的规则集。重要的是，两个计划者的存在允许方解石用户通过引导搜索不同的查询计划来减少整体优化时间。

Materialized views.One of the most powerful techniques to accel-erate query processing in data warehouses is the precomputation of relevant summaries or materialized views.Multiple Calcite adapters and projects relying on Calcite have their own notion of mate-rialized views.For instance, Cassandra allows the user to define materialized views based on existing tables which are automatically maintained by the system.

物化视图。加速数据仓库中查询处理的最强大的技术之一是相关摘要或物化视图的预计算。多个方解石适配器和依赖方解石的项目有它们自己的个性化视图的概念。例如，卡珊德拉允许用户根据系统自动维护的现有表定义物化视图。

These engines expose their materialized views to Calcite.The optimizer then has the opportunity to rewrite incoming queries to use these views instead of the original tables.In particular, Calcite provides an implementation of two different materialized view-based rewriting algorithms.

这些引擎将它们的物化视图展示给方解石。然后，优化器有机会重写传入的查询，以使用这些视图而不是原始表。具体来说，方解石提供了两种不同的基于物化视图的重写算法的实现。

The first approach is based on view substitution [10, 18].The aim is to substitute part of the relational algebra tree with an equiva-lent expression which makes use of a materialized view, and the algorithm proceeds as follows: (i) the scan operator over the materi-alized view and the materialized view definition plan are registered with the planner, and (ii) transformation rules that try to unify expressions in the plan are triggered.Views do not need to exactly match expressions in the query being replaced, as the rewriting algorithm in Calcite can produce partial rewritings that include additional operators to compute the desired expression, e.g., filters with residual predicate conditions.

第一种方法基于视图替换[10，18]。目标是用利用物化视图的等价表达式替换关系代数树的一部分，并且算法如下进行:(I)在物化视图和物化视图定义计划上的扫描操作符被注册到计划器，并且(ii)尝试在计划中统一表达式的转换规则被触发。视图不需要与被替换的查询中的表达式完全匹配，因为方解石中的重写算法可以产生部分重写，包括计算所需表达式的附加操作符，例如带有剩余谓词条件的过滤器。

The second approach is based on lattices [22].Once the data sources are declared to form a lattice, Calcite represents each of

第二种方法基于格[22]。一旦数据源被声明形成一个晶格，方解石就代表了

the materializations as a tile which in turn can be used by the opti-mizer to answer incoming queries.On the one hand, the rewriting algorithm is especially efficient in matching expressions over data sources organized in a star schema, which are common in OLAP applications.On the other hand, it is more restrictive than view substitution, as it imposes restrictions on the underlying schema.

物化为图块，optimizer 可以使用该图块来回答传入的查询。一方面，重写算法在匹配以星型模式组织的数据源上的表达式时特别有效，这在 OLAP 应用程序中很常见。另一方面，它比视图替换更具限制性，因为它对底层模式施加了限制。

7 EXTENDING CALCITE

7 扩展方解石

As we have mentioned in the previous sections, Calcite is not only tailored towards SQL processing.In fact, Calcite provides extensions to SQL expressing queries over other data abstractions, such as semi-structured, streaming and geospatial data.Its internal operators adapt to these queries.In addition to extensions to SQL, Calcite also includes a language-integrated query language.We describe these extensions throughout this section and provide some examples.

正如我们在前面的章节中提到的，方解石不仅仅是为 SQL 处理而定制的。事实上，方解石提供了对 SQL 的扩展，表达了对其他数据抽象的查询，例如半结构化、流和地理空间数据。它的内部操作符适应这些查询。除了对 SQL 的扩展，方解石还包括一种语言集成的查询语言。我们在本节中描述了这些扩展，并提供了一些示例。

7.1 Semi-structured Data

7.1 半结构化数据

Calcite supports several complex column data types that enable a hybrid of relational and semi-structured data to be stored in tables.Specifically, columns can be of type ARRAY, MAP, or MULTISET.Furthermore, these complex types can be nested so it is possible for example, to have a MAP where the values are of type ARRAY.Data within the ARRAY and MAP columns (and nested data therein) can be extracted using the [] operator.The specific type of values stored in any of these complex types need not be predefined.

方解石支持多种复杂的列数据类型，使关系数据和半结构化数据能够混合存储在表中。具体来说，列可以是 ARRAY、MAP 或 MULTISET 类型。此外，这些复杂类型可以嵌套，因此，例如，有可能有一个值为 ARRAY 类型的映射。可以使用[]运算符提取数组和映射列中的数据(以及其中的嵌套数据)。存储在任何这些复杂类型中的特定类型的值不需要预先定义。

For example, Calcite contains an adapter for MongoDB [36], a document store which stores documents consisting of data roughly equivalent to JSON documents.To expose MongoDB data to Calcite, a table is created for each document collection with a single column named _MAP: a map from document identifiers to their data.In many cases, documents can be expected to have a common structure.A collection of documents representing zip codes may each contain columns with a city name, latitude and longitude.It can be useful to expose this data as a relational table.In Calcite, this is achieved by creating a view after extracting the desired values and casting them to the appropriate type:

例如，方解石包含 MongoDB [36]的适配器，MongoDB[36]是一个文档存储库，它存储的文档由大致相当于 JSON 文档的数据组成。为了将蒙古数据库数据展示给方解石，为每个文档集合创建了一个表，其中有一个名为_MAP 的列:从文档标识符到其数据的映射。在许多情况下，可以预期文档具有共同的结构。代表邮政编码的文档集合可能每个都包含带有城市名称、纬度和经度的列。将这些数据公开为关系表可能会很有用。在方解石中，这是通过提取所需值并将其转换为适当类型后创建视图来实现的:

SELECT CAST (_MAP ['city '] AS varchar (20)) AS city , CAST (_MAP ['loc '][0] AS float) AS longitude , CAST (_MAP ['loc '][1] AS float) AS latitude

选择 CAST (_MAP ['city '] AS varchar (20))作为城市，CAST (_MAP ['loc '][0]作为浮动)作为经度，CAST (_MAP ['loc '][1]作为浮动)作为纬度

FROM mongo_raw .zips ;

来自蒙古_raw。拉链。

With views over semi-structured data defined in this manner, it becomes easier to manipulate data from different semi-structured sources in tandem with relational data.

通过以这种方式定义半结构化数据的视图，可以更容易地操作来自不同半结构化源的数据和关系数据。

7.2 Streaming

7.2 流式传输

Calcite provides first-class support for streaming queries [26] based on a set of streaming-specific extensions to standard SQL, namely STREAM extensions, windowing extensions, implicit references to streams via window expressions in joins, and others.These extensions were inspired by the Continuous Query Language [2] while also trying to integrate effectively with standard SQL.The primary extension, the STREAM directive tells the system that the user is interested in incoming records, not existing ones.

方解石提供一流的流查询支持[26]，基于一组流特定的标准 SQL 扩展，即流扩展、窗口扩展、通过连接中的窗口表达式隐式引用流等。这些扩展受到连续查询语言[2]的启发，同时也试图与标准的 SQL 有效地集成。STREAM 指令是主要的扩展，它告诉系统用户对传入的记录感兴趣，而不是现有的记录。

SELECT STREAM rowtime , productId , units FROM Orders

从订单中选择流行时间、产品标识、单位

WHERE units > 25;

其中单位> 25;

Apache Calcite SIGMOD'18, June 10-15, 2018, Houston, TX, USA

2018 年 6 月 10 日至 15 日，美国德克萨斯州休斯顿

In the absence of the STREAM keyword when querying a stream, the query becomes a regular relational query, indicating the system should process existing records which have already been received from a stream, not the incoming ones.

当查询一个流时，如果没有 STREAM 关键字，查询就变成了一个常规的关系查询，表明系统应该处理已经从一个流中接收到的现有记录，而不是传入的记录。

Due to the inherently unbounded nature of streams, window-ing is used to unblock blocking operators such as aggregates and joins.Calcite's streaming extensions use SQL analytic functions to express sliding and cascading window aggregations, as shown in the following example.

由于流固有的无界性，窗口被用来解除阻塞操作，如聚合和连接。方解石的流扩展使用 SQL 分析函数来表示滑动和级联窗口聚合，如下例所示。

SELECT STREAM rowtime , productId , units ,

选择流行时间，产品标识，单位，

SUM(units) OVER (ORDER BY rowtime PARTITION BY productId

合计(单位)超过(按行排序按产品标识划分

RANGE INTERVAL '1' HOUR PRECEDING) unitsLastHour FROM Orders ;

距离间隔' 1 '小时之前)单位从订单中减去;

Tumbling, hopping and session windowsare enabled by the TUMBLE, HOPPING, SESSION functions and related utility functions such as TUMBLE_END and HOP_END that can be used respectively in GROUP BY clauses and projections.

翻滚、跳跃和会话窗口由翻滚、跳跃、会话功能和相关实用程序功能启用，如翻滚_结束和跳跃_结束，可分别用于分组依据子句和预测。

SELECT STREAM

选择流

TUMBLE_END (rowtime , INTERVAL '1' HOUR) AS rowtime , productId ,

滚动结束(行时间，间隔' 1 '小时)作为行时间，产品标识，

COUNT (*) AS c , SUM(units) AS units FROM Orders

将(*)计算为 c，将(单位)总和计算为订单单位

GROUP BY TUMBLE (rowtime , INTERVAL '1' HOUR), productId ;

按翻转分组(行时间，间隔' 1 '小时)，产品标识；

Streaming queries involving window aggregates require the pres-ence of monotonic or quasi-monotonic expressions in the GROUP BY clause or in the ORDER BY clause in case of sliding and cascading window queries.

涉及窗口聚合的流式查询要求在滑动和级联窗口查询的情况下，在 GROUP BY 子句或 ORDER BY 子句中存在单调或准单调表达式。

Streaming queries which involve more complex stream-to-stream joins can be expressed using an implicit (time) window expression in the JOIN clause.

涉及更复杂的流到流连接的流查询可以使用 JOIN 子句中的隐式(时间)窗口表达式来表示。

SELECT STREAM o . rowtime , o . productId , o . orderId , s . rowtime AS shipTime FROM Orders AS o JOIN Shipments AS s

选择行时间、产品标识、订单标识、行时间作为装运时间从订单作为加入装运作为

ON o . orderId = s . orderId

打开订单编号=订单编号

AND s . rowtime BETWEEN o . rowtime AND o . rowtime + INTERVAL '1' HOUR ;

和之间的行时间和行时间+间隔' 1 '小时；

In the case of an implicit window, Calcite's query planner vali-dates that the expression is monotonic.

在隐式窗口的情况下，方解石的查询计划器假定表达式是单调的。

7.3 Geospatial Queries

7.3 地理空间查询

Geospatial support is preliminary in Calcite, but is being imple-mented using Calcite's relational algebra.The core of this imple-mentation consists in adding a new GEOMETRY data type which encapsulates different geometric objects such as points, curves, and polygons.It is expected that Calcite will be fully compliant with the OpenGIS Simple Feature Access [39] specification which defines a standard for SQL interfaces to access geospatial data.An example query finds the country which contains the city of Amsterdam:

地理空间支持在方解石中是初步的，但是正在使用方解石的关系代数来实现。这个实现的核心在于添加一个新的几何数据类型，它封装了不同的几何对象，如点、曲线和多边形。预计方解石将完全符合开放地理信息系统简单功能访问[39]规范，该规范定义了访问地理空间数据的 SQL 接口标准。一个示例查询查找包含阿姆斯特丹市的国家:

SELECT name FROM (SELECT name ,

从中选择名称(选择名称，

ST_GeomFromText ('POLYGON ((4.82 52.43 , 4.97 52.43 , 4.97 52.33 , 4.82 52.33 , 4.82 52.43)) ') AS " Amsterdam ", ST_GeomFromText (boundary) AS " Country " FROM country

作为“阿姆斯特丹”，作为“国家”

Tumbling, hopping, sliding, and session windows are different schemes for grouping of the streaming events [35].

翻滚、跳跃、滑动和会话窗口是对流式事件进行分组的不同方案[35]。

) WHERE ST_Contains (" Country ", " Amsterdam ");

)其中 ST_Contains(“国家”、“阿姆斯特丹”)；

7.4 Language-Integrated Query for Java

7.4 语言集成的 Java 查询

Calcite can be used to query multiple data sources, beyond just relational databases.But it also aims to support more than just the SQL language.Though SQL remains the primary database lan-guage, many programmers favour language-integrated languages like LINQ [33].Unlike SQL embedded within Java or C++ code, language-integrated query languages allow the programmer to write all of her code using a single language.Calcite provides Language-Integrated Query for Java (or LINQ4J, in short) which closely follows the convention set forth by Microsoft's LINQ for the .NET languages.

方解石可以用来查询多个数据源，不仅仅是关系数据库。但是它的目标也不仅仅是支持 SQL 语言。尽管 SQL 仍然是主要的数据库语言，但许多程序员喜欢像 LINQ [33]这样的语言集成语言。与嵌入在 Java

或 C++代码中的 SQL 不同，语言集成查询语言允许程序员用一种语言编写所有代码。方解石为 Java(或简称 LINQ4J)提供语言集成查询，它严格遵循微软的 LINQ 为。NET 语言。

8 INDUSTRY AND ACADEMIA ADOPTION

8 工业和学术界采用

Calcite enjoys wide adoption, specially among open-source projects used in industry.As Calcite provides certain integration flexibility, these projects have chosen to either (i) embed Calcite within their core, i.e., use it as a library, or (ii) implement an adapter to allow Calcite to federate query processing.In addition, we see a growing interest in the research community to use Calcite as the cornerstone of the development of data management projects.In the following, we describe how different systems are using Calcite.

方解石被广泛采用，特别是在工业中使用的开源项目中。由于方解石提供了一定的集成灵活性，这些项目选择了(I)将方解石嵌入到它们的核心中，即将其用作一个库，或者(ii)实现一个适配器来允许方解石联合查询处理。此外，我们看到研究团体对使用方解石作为开发数据管理项目的基石越来越感兴趣。在下文中，我们将描述不同的系统如何使用方解石。

8.1 Embedded Calcite

8.1 嵌入方解石

Table 1 provides a list of software that incorporates Calcite as a library, including (i) the query language interface that they expose to users, (ii) whether they use Calcite's JDBC driver (called Avatica), (iii) whether they use the SQL parser and validator included in Calcite, (iv) whether they use Calcite's query algebra to represent their operations over data, and (v) the engine that they rely on for execution, e.g., their own native engine, Calcite's operators (referred to as enumerable), or any other project.

表 1 提供了一个软件列表，这些软件将方解石合并为一个库，包括(I)它们向用户公开的查询语言接口，(ii)它们是否使用方解石的 JDBC 驱动程序(称为 Avatica)，(iii)它们是否使用方解石中包含的 SQL 解析器和验证器，(iv)它们是否使用方解石的查询代数来表示它们对数据的操作，以及(v)它们执行所依赖的引擎，例如它们自己的本地引擎、方解石的操作符(称为可枚举的)或任何其他项目。

Drill [13] is a flexible data processing engine based on the Dremel system [34] that internally uses a schema-free JSON document data model.Drill uses its own dialect of SQL that includes extensions to express queries on semi-structured data, similar to SQL++ [38].

Drill [13]是一个基于 Dremel 系统[34]的灵活的数据处理引擎，它在内部使用一个无模式的 JSON 文档数据模型。与 SQL++类似，Drill 使用自己的 SQL 方言，其中包括对半结构化数据进行查询的扩展。

Hive [24] first became popular as a SQL interface on top of the MapReduce programming model [52].It has since moved to-wards being an interactive SQL query answering engine, adopting Calcite as its rule and cost-based optimizer.Instead of relying on Calcite's JDBC driver, SQL parser and validator, Hive uses its own implementation of these components.The query is then translated into Calcite operators, which after optimization are translated into Hive's physical algebra.Hive operators can be executed by multiple engines, the most popular being Apache Tez [43, 51] and Apache Spark [47, 56].

Hive [24]最初是作为 MapReduce 编程模型之上的一个 SQL 接口而流行起来的[52]。从那以后，它已经发展成为一个交互式的查询引擎，采用方解石作为它的规则和基于成本的优化器。Hive 不依赖于方解石的 JDBC 驱动程序、SQL 解析器和验证器，而是使用自己的这些组件的实现。该查询然后被翻译成方解石运算符，优化后被翻译成蜂巢的物理代数。蜂巢操作器可以由多个引擎执行，最受欢迎的是阿帕奇 Tez [43，51]和阿帕奇 Spark [47，56]。

Apache Solr [46] is a popular full-text distributed search platform built on top of the Apache Lucene library [31].Solr exposes multiple query interfaces to users, including REST-like HTTP/XML and JSON APIs.In addition, Solr integrates with Calcite to provide SQL compatibility.

Apache Solr [46]是一个流行的全文分布式搜索平台，建立在 Apache Lucene 库[31]之上。Solr 向用户公开了多个查询接口，包括类似 REST 的 HTTP/XML 和 JSON APIs。此外，索尔与方解石集成，以提供 SQL 兼容性。

Apache Phoenix [40] and Apache Kylin [28] both work on top of Apache HBase [23], a distributed key-value store modeled after Bigtable [9].In particular, Phoenix provides a SQL interface and orchestration layer to query HBase.Kylin focuses on OLAP-style

Apache Phoenix [40]和 Apache Kylin [28]都在 Apache HBase [23]之上工作，这是一个模仿 Bigtable [9]的分布式键值存储。具体来说，菲尼克斯提供了一个查询糖化血红蛋白的 SQL 接口和编排层。麒麟专注于 OLAP 风格

System Query Language JDBC Driver SQL Parser and Validator Relational Algebra Execution Engine Apache Drill SQL + extensions ✓ ✓ ✓ Native Apache Hive SQL + extensions ✓ Apache Tez, Apache Spark Apache Solr SQL ✓ ✓ ✓ Native, Enumerable, Apache Lucene

系统查询语言 JDBC 驱动程序 SQL 解析器和验证器关系代数执行引擎 Apache Drill SQL +扩展\1000 \1000 \1000 \1000 本地 Apache Hive SQL +扩展\1000 Apache Tez，Apache Spark Apache Solr SQL \1000 \1000 \1000 本地，可枚举，Apache Lucene

Apache Phoenix SQL ✓ ✓ ✓ Apache HBase

阿帕奇凤凰 SQL \\

Apache Kylin SQL ✓ ✓ Enumerable, Apache HBase

阿帕奇麒麟 SQL \\

Apache Apex Streaming SQL ✓ ✓ ✓ Native

阿帕奇顶点流 SQL 本地

Apache Flink Streaming SQL ✓ ✓ ✓ Native

Apache Flink 流 SQL \\ 1000 \\ 1000 \\本机

Apache Samza Streaming SQL ✓ ✓ ✓ Native

Apache Samza 流 SQL \\ 1000 \\ 1000 \\本机

Apache Storm Streaming SQL ✓ ✓ ✓ Native

阿帕奇风暴流 SQL 本地

MapD [32] SQL ✓ ✓ Native

MapD[32]SQL \1000 \1000 \u Native

Lingual [30] SQL ✓ ✓ Cascading

语言[30]SQL \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

Qubole Quark [42] SQL ✓ ✓ ✓ Apache Hive, Presto

量子阱夸克[42]SQL \\\

Table 1: List of systems that embed Calcite.

表 1:嵌入方解石的系统列表。

Adapter Target language

适配器目标语言

Apache Cassandra Cassandra Query Language (CQL)

阿帕奇卡桑德拉卡桑德拉查询语言(CQL)

Apache Pig Pig Latin

阿帕奇猪猪拉丁语

Apache Spark Java (Resilient Distributed Datasets)

Apache Spark Java(弹性分布式数据集)

Druid JSON

德鲁伊 JSON

Elasticsearch JSON

弹性搜索 JSON

JDBC SQL (multiple dialects)

JDBC SQL(多种方言)

MongoDB Java

蒙古数据库 Java

Splunk SPL

Splunk SPL

Table 2: List of Calcite adapters.

表 2:方解石适配器列表。

SQL queries instead, building cubes that are declared as materialized views and stored in HBase, and hence allowing Calcite's optimizer to rewrite the input queries to be answered using those cubes.In Kylin, query plans are executed using a combination of Calcite native operators and HBase.

取而代之的是 SQL 查询，构建被声明为物化视图并存储在 HBase 中的多维数据集，因此允许方解石的优化器重写输入查询以使用这些多维数据集来回答。在麒麟中，查询计划使用方解石本地操作符和 HBase 的组合来执行。

Recently Calcite has become popular among streaming sys-tems too.Projects such as Apache Apex [1], Flink [16], Apache Samza [44], and Storm [50] have chosen to integrate with Calcite, using its components to provide a streaming SQL interface to their users.Finally, other commercial systems have adopted Calcite, such as MapD [32], Lingual [30], and Qubole Quark [42].

最近，方解石也在流媒体系统中流行起来。像 Apache Apex [1]、Flink [16]、Apache Samza [44]和 Storm [50]这样的项目已经选择与方解石集成，使用它的组件为用户提供一个流式的 SQL 接口。最后，其他商业系统已经采用了方解石，如 MapD [32]，语言[30]和量子夸克[42]。

8.2 Calcite Adapters

8.2 方解石适配器

Instead of using Calcite as a library, other systems integrate with Calcite via adapters which read their data sources.Table 2 provides

其他系统通过读取数据源的适配器与方解石集成，而不是使用方解石作为库。表 2 提供了

the list of available adapters in Calcite.One of the main key com-ponents of the implementation of these adapters is the converter responsible for translating the algebra expression to be pushed to the system into the query language supported by that system.Table 2 also shows the languages that Calcite translates into for each of these adapters.

方解石中可用适配器的列表。实现这些适配器的主要关键组件之一是转换器，它负责将要推送到系统的代数表达式转换成该系统支持的查询语言。表 2 还显示了方解石为这些适配器翻译的语言。

The JDBC adapter supports the generation of multiple SQL di-alects, including those supported by popular RDBMSes such as PostgreSQL and MySQL.In turn, the adapter for Cassandra [8] gen-erates its own SQL-like language called CQL whereas the adapter for Apache Pig [41] generates queries expressed in Pig Latin [37].The adapter for Apache Spark [47] uses the Java RDD API.Finally, Druid [14], Elasticsearch [15] and Splunk [48] are queried through REST HTTP API requests.The queries generated by Calcite for these systems are expressed in JSON or XML.

JDBC 适配器支持生成多个 SQL 数据库，包括流行的关系数据库支持的数据库，如 PostgreSQL 和 MySQL。反过来，卡珊德拉的适配器[8]生成自己的类似于 SQL 的语言，称为 CQL，而阿帕奇猪的适配器[41]生成用猪拉丁语表示的查询[37]。Apache Spark [47]的适配器使用了 Java RDD 应用编程接口。最后，德鲁伊[14]，弹性搜索[15]和 Splunk [48]是通过 REST HTTP API 请求查询的。方解石为这些系统生成的查询用 JSON 或 XML 表示。

8.3 Uses in Research

8.3 研究用途

In a research setting, Calcite has been considered [54] as a polystore-alternative for precision medicine and clinical analysis scenarios.In those scenarios, heterogeneous medical data has to be logically assembled and aligned to assess the best treatments based on the comprehensive medical history and the genomic profile of the pa-tient.The data comes from relational sources representing patients' electronic medical records, structured and semi-structured sources representing various reports (oncology, psychiatry,laboratory tests, radiology, etc.), imaging, signals, and sequence data, stored in sci-entific databases.In those circumstances, Calcite represents a good foundation with its uniform query interface, and flexible adapter architecture, but the ongoing research efforts are aimed at (i) in-troduction of the new adapters for array, and textual sources, and (ii) support efficient joining of heterogeneous data sources.

在一项研究中，方解石被认为[54]是精密医学和临床分析场景的多晶替代物。在这些情况下，必须对不同的医学数据进行逻辑组合和比对，以便根据患者的综合病史和基因组图谱来评估最佳治疗方案。数据来自代表患者电子病历的相关来源，代表各种报告(肿瘤学、精神病学、实验室检测、放射学等)的结构化和半结构化来源。)，存储在科学数据库中的成像、信号和序列数据。在这种情况下，方解石以其统一的查询接口和灵活的适配器体系结构代表了一个良好的基础，但是正在进行的研究工作旨在(I)引入用于数组和文本源的新适配器，以及(ii)支持异构数据源的有效连接。

9 FUTURE WORK

9 未来工作

The future work on Calcite will focus on the development of the new features, and the expansion of its adapter architecture:

方解石的未来工作将集中在新特性的开发和适配器架构的扩展上:

• Enhancements to the design of Calcite to further support its use a standalone engine, which would require a support for data definition languages (DDL), materialized views, indexes and constraints.

方解石设计的增强，进一步支持其使用独立引擎，这将需要支持数据定义语言(DDL)、物化视图、索引和约束。

• Ongoing improvements to the design and flexibility of the planner, including making it more modular, allowing users Calcite to supply planner programs (collections of rules or-ganized into planning phases) for execution.

不断改进规划器的设计和灵活性，包括使其更加模块化，允许用户方解石提供规划器程序(规则集合或组织成规划阶段)以供执行。

• Incorporation of new parametric approaches [53] into the design of the optimizer.

将新的参数方法[53]结合到优化器的设计中。

• Support for an extended set of SQL commands, functions, and utilities, including full compliance with OpenGIS.• New adapters for non-relational data sources such as array databases for scientific computing.

支持一组扩展的 SQL 命令、函数和实用程序，包括完全符合开放地理信息系统。非关系数据源的新适配器，如用于科学计算的阵列数据库。

• Improvements to performance profiling and instrumenta-tion.

性能分析和检测的改进。

9.1 Performance Testing and Evaluation

9.1 性能测试和评估

Though Calcite contains a performance testing module, it does not evaluate query execution.It would be useful to assess the perfor-mance of systems built with Calcite.For example, we could compare the performance of Calcite with similar frameworks.Unfortunately, it might be difficult to craft fair comparisons.For example, like Calcite, Algebricks optimizes queries for Hive.Borkar et al.[6] compared Algebricks with the Hyracks scheduler against Hive ver-sion 0.12 (without Calcite).The work of Borkar et al. precedes signif-icant engineering and architectural changes into Hive.Comparing Calcite against Algebricks in a fair manner in terms of timings does not seem feasible, as one would need to ensure that each uses the same execution engine.Hive applications rely mostly on ei-ther Apache Tez or Apache Spark as execution engines whereas Algebricks is tied to its own framework (including Hyracks).

虽然方解石包含一个性能测试模块，但它不评估查询执行。评估用方解石建造的系统的性能是有用的。例如，我们可以比较方解石与类似框架的性能。不幸的是，很难进行公平的比较。例如，像方解石一样，Algebricks 优化了对蜂巢的查询。Borkar 等人[6]将 Algebricks 与 Hyracks 调度程序进行了比较，结果显示 Hive 版本为 0.12(不含方解石)。博卡等人的工作先于对蜂巢的重大工程和建筑变革。从时间上公平地比较方解石和阿尔杰布里克似乎不可行，因为我们需要确保每个人使用相同的执行引擎。Hive 应用程序主要依赖其他 Apache Tez 或 Apache Spark 作为执行引擎，而 Algebricks 被绑定到自己的框架(包括 Hyracks)。

Moreover, to assess the performance of Calcite-based systems, we need to consider two distinct use cases.Indeed, Calcite can be used either as part of a single system—as a tool to accelerate the construction of such a system—or for the more difficult task of combining several distinct systems —as a common layer.The former is tied to the characteristics of the data processing system, and because Calcite is so versatile and widely used, many distinct benchmarks are needed.The latter is limited by the availability of existing heterogeneous benchmarks.BigDAWG [55] has been used to integrate PostgreSQL with Vertica, and on a standard benchmark, one gets that the integrated system is superior to a baseline where entire tables are copied from one system to another to answer specific queries.Based on real-world experience, we believe that more ambitious goals are possible for integrated multiple systems: they should be superior to the sum of their parts.

此外，为了评估基于方解石的系统的性能，我们需要考虑两个不同的用例。事实上，方解石既可以作为单一系统的一部分——作为加速这种系统构建的工具——也可以作为一个共同的层，用于将几个不同系统结合起来的更困难的任务。前者与数据处理系统的特性有关，因为方解石用途广泛，用途广泛，所以需要许多不同的基准。后者受到现有异构基准可用性的限制。BigDough[55]已被用于将 PostgreSQL 与 Vertica 集成，在标准基准测试中，人们会发现集成系统优于基准测试，基准测试是将整个表从一个系统复制到另一个系统，以回答特定的查询。基于现实世界的经验，我们相信集成多个系统的更宏伟的目标是可能的:它们应该优于它们各部分的总和。

10 CONCLUSION

10 结论

Emerging data management practices and associated analytic uses of data continue to evolve towards an increasingly diverse, and het-erogeneous spectrum of scenarios.At the same time, relational data sources, accessed through the SQL, remain an essential means to

新兴的数据管理实践和相关的数据分析应用继续向日益多样化和异质化的场景发展。同时，通过 SQL 访问的关系数据源仍然是实现以下目标的重要手段

how enterprises work with the data.In this somewhat dichotomous space, Calcite plays a unique role with its strong support for both traditional, conventional data processing, and for its support of other data sources including those with semi-structured, streaming and geospatial models.In addition, Calcite's design philosophy with a focus on flexibility, adaptivity, and extensibility, has been another factor in Calcite becoming the most widely adopted query opti-mizer, used in a large number of open-source frameworks.Calcite's dynamic and flexible query optimizer, and its adapter architecture allows it to be embedded selectively by a variety of data manage-ment frameworks such as Hive, Drill, MapD, and Flink.Calcite's support for heterogeneous data processing, as well as for the ex-tended set of relational functions will continue to improve, in both functionality and performance.

企业如何处理数据。在这个有点二分法的空间中，方解石发挥了独特的作用，它对传统的、常规的数据处理提供了强有力的支持，并支持其他数据源，包括半结构化、流式和地理空间模型。此外，方解石的设计理念注重灵活性、适应性和可扩展性，这也是方解石成为被大量开源框架广泛采用的查询优化器的另一个因素。方解石的动态和灵活的查询优化器，以及它的适配器体系结构允许它有选择地嵌入各种数据管理框架，如 Hive、Drive、MapD 和 Flink。方解石对异构数据处理的支持，以及对扩展的关系函数集的支持，将在功能和性能上继续改进。

ACKNOWLEDGMENTS

感谢

We would like to thank the Calcite community, contributors and users, who build, maintain, use, test, write about, and continue to push the Calcite project forward.This manuscript has been in part co-authored by UT-Battelle, LLC under Contract No. DE-AC05- 00OR22725 with the U.S. Department of Energy.

我们要感谢方解石社区、贡献者和用户，他们构建、维护、使用、测试、撰写和继续推进方解石项目。这份手稿部分是由 UT-Battelle 有限责任公司与美国能源部合作完成的，合同号为 DE-AC05-00OR22725。

REFERENCES

参考

[1] Apex.Apache Apex.https://apex.apache.org .(Nov. 2017).

[1]顶点。阿帕奇顶点。https://apex.apache.org。(2017 年 11 月)。

[2] Arvind Arasu, Shivnath Babu, and Jennifer Widom.2003.The CQL Continuous Query Language: Semantic Foundations and Query Execution.Technical Report 2003-67.Stanford InfoLab.

[2] Arvind Arasu，Shivnath Babu 和 Jennifer Widom。2003.CQL 连续查询语言:语义基础和查询执行。技术报告 2003-67。斯坦福信息实验室。

[3] Michael Armbrust et al. 2015.Spark SQL: Relational Data Processing in Spark.In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15).ACM, New York, NY, USA, 1383-1394.

[3]迈克尔·阿姆布鲁斯特等人，2015 年。火花 SQL:火花中的关系数据处理。在 2015 年 ACM SIGMOD 国际数据管理会议记录(SIGMOD '15)中。美国纽约州纽约市，1383-1394。

[4] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia.2015.Spark SQL: Relational Data Processing in Spark.In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15).ACM, New York, NY, USA, 1383-1394.

[4]迈克尔阿姆布鲁斯特，雷诺辛，程炼，尹怀，戴维斯刘，约瑟夫布拉德利，，托马卡夫卡坦，迈克尔富兰克林，阿里戈德西和马泰扎哈利亚。2015.火花 SQL:火花中的关系数据处理。在 2015 年 ACM SIGMOD 国际数据管理会议记录(SIGMOD '15)中。美国纽约州纽约市，1383-1394。

[5] ASF.The Apache Software Foundation.(Nov. 2017).Retrieved November 20, 2017 from http://www.apache.org/

[5] ASF。阿帕奇软件基金会。(2017 年 11 月)。2017 年 11 月 20 日从 http://www.apache.org/检索

[6] Vinayak Borkar, Yingyi Bu, E. Preston Carman, Jr., Nicola Onose, Till Westmann, Pouria Pirzadeh, Michael J. Carey, and Vassilis J. Tsotras.2015.Algebricks: A Data Model-agnostic Compiler Backend for Big Data Languages.In Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15).ACM, New York, NY, USA, 422-433.

[6] Vinayak Borkar，Yingyi Bu，E. Preston Carman，Jr . Nicola Onose，Till Westmann，Pouria Pirzadeh，Michael J. Carey 和 Vassilis J. Tsotras。2015.面向大数据语言的数据模型不可知的编译器后端。第六届美国计算机学会云计算研讨会论文集。美国纽约州纽约市，422-433。

[7] M. J. Carey et al. 1995.Towards heterogeneous multimedia information systems: the Garlic approach.In IDE-DOM '95.124-131.

[7] M. J. Carey 等人，1995 年。走向异构多媒体信息系统:大蒜方法。在 95 年的集成开发环境中。124-131。

[8] Cassandra.Apache Cassandra.(Nov. 2017).Retrieved November 20, 2017 from http://cassandra.apache.org/

[8]卡桑德拉。阿帕奇·卡珊德拉。(2017 年 11 月)。2017 年 11 月 20 日从 http://cassandra.apache.org/ 检索

[9] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber.2006.Bigtable: A Distributed Storage System for Structured Data.In 7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA.205-218.

[9] Fay Chang，Jeffrey Dean，Sanjay Ghemawat，Wilson C. Hsieh，Deborah A. Wallach，Michael Burrows，Tushar Chandra，Andrew Fikes 和 Robert Gruber。2006.大表:结构化数据的分布式存储系统。在第七届操作系统设计和实施研讨会(OSDI '06)上，11 月 6-8 日，美国华盛顿州西雅图。205-218。

[10] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim.1995.Optimizing Queries with Materialized Views.In Proceedings of the Eleventh International Conference on Data Engineering (ICDE '95).IEEE Computer Society, Washington, DC, USA, 190-200.

[10] Surajit Chaudhuri，Ravi Krishnamurthy，Spyros Potamianos 和 Kyuseok Shim。1995.使用实体化视图优化查询。在第十一届国际数据工程会议记录(ICDE '95)。IEEE 计算机学会，华盛顿州，DC，美国，190-200。

[11] E. F. Codd.1970.A Relational Model of Data for Large Shared Data Banks.Commun.ACM 13, 6 (June 1970), 377-387.

[11] E. F. Codd。1970.大型共享数据库的数据关系模型。通信。ACM 13，6(1970 年 6 月)，377-387。

[12] Alex Şuhan.Fast and Flexible Query Analysis at MapD with Apache Calcite.(feb 2017).Retrieved November 20, 2017 from https://www.mapd.com/blog/2017/02/ 08/fast-and-flexible-query-analysis-at-mapd-with-apache-calcite-2/

[12]亚历克斯·舒汉。快速和灵活的查询分析在地图与阿帕奇方解石。(2017 年 2 月)。检索日期:2017 年 11 月 20 日，https://www.mapd.com/blog/2017/02/ 08/快速灵活的查询分析

[13] Drill.Apache Drill.(Nov. 2017).Retrieved November 20, 2017 from http: //drill.apache.org/

[13]钻孔。阿帕奇钻。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[14] Druid.Druid.(Nov. 2017).Retrieved November 20, 2017 from http://druid.io/

[14]德鲁伊。德鲁伊。(2017 年 11 月)。2017 年 11 月 20 日从 http://druid.io/检索

[15] Elastic.Elasticsearch.(Nov. 2017).Retrieved November 20, 2017 from https: //www.elastic.co

[15]弹性。弹性搜索。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[16] Flink.Apache Flink.https://flink.apache.org .(Nov. 2017).

[16] Flink。阿帕奇·弗林克。https://flink.apache.org。(2017 年 11 月)。

[17] Yupeng Fu, Kian Win Ong, Yannis Papakonstantinou, and Michalis Petropoulos.2011.The SQL-based all-declarative FORWARD web application development framework.In CIDR.

[17]富、江文翁、帕帕康斯坦丁和彼得罗保罗斯。2011.基于 SQL 的全声明式 FORWARD web 应用程序开发框架。在 CIDR。

[18] Jonathan Goldstein and Per-Åke Larson.2001.Optimizing Queries Using Ma-terialized Views: A Practical, Scalable Solution.SIGMOD Rec.30, 2 (May 2001), 331-342.

[18]乔纳森·戈尔茨坦和佩克·拉森。2001.使用物化视图优化查询:一个实用、可扩展的解决方案。SIGMOD Rec。30，2(2001 年 5 月)，331-342。

[19] Goetz Graefe.1995.The Cascades Framework for Query Optimization.IEEE Data Eng.Bull.(1995).

[19] Goetz Graefe。1995.查询优化的级联框架。数据工程。胡说。(1995 年)。

[20] Goetz Graefe and William J. McKenna.1993.The Volcano Optimizer Genera-tor: Extensibility and Efficient Search.In Proceedings of the Ninth International Conference on Data Engineering.IEEE Computer Society, Washington, DC, USA, 209-218.

[20]戈茨·格拉夫和威廉·麦肯纳。1993.火山优化器生成器:可扩展性和高效搜索。在第九届国际数据工程会议记录中。IEEE 计算机学会，华盛顿州，DC，美国，209-218。

[21] Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu.2014.Demonstration of the Myria Big Data Management Service.In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14).ACM, New York, NY, USA, 881-884.

[21]丹尼尔·哈尔佩林、维克托·特谢拉·德·阿尔梅达、李·李乔、舒莫·楚、帕拉肖斯·库特里斯、张秀坤·莫里茨、珍妮弗·奥尔蒂斯、瓦斯托尔·鲁阿姆维博苏克、王晶晶、安德鲁·惠特克、许圣亮、马格达莱纳·巴拉津斯卡、比尔·豪和丹·苏休。2014.Myria 大数据管理服务演示。在 2014 年 ACM SIGMOD 国际数据管理会议记录(SIGMOD '14)中。美国纽约州纽约市，电话:881-884。

[22] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman.1996.Implement-ing Data Cubes Efficiently.SIGMOD Rec.25, 2 (June 1996), 205-216.

[22] Venky Harinarayan，Anand Rajaraman 和 Jeffrey D. Ullman。1996.高效实现数据立方体。SIGMOD Rec。25，2(1996 年 6 月)，205-216。

[23] HBase.Apache HBase.(Nov. 2017).Retrieved November 20, 2017 from http: //hbase.apache.org/

[23]HbBase。Apache HBase。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[24] Hive.Apache Hive.(Nov. 2017).Retrieved November 20, 2017 from http: //hive.apache.org/

[24]蜂巢。阿帕奇蜂巢。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[25] Yin Huai, Ashutosh Chauhan, Alan Gates, Gunther Hagleitner, Eric N. Hanson, Owen O'Malley, Jitendra Pandey, Yuan Yuan, Rubao Lee, and Xiaodong Zhang.2014.Major Technical Advancements in Apache Hive.In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14).ACM, New York, NY, USA, 1235-1246.

[25]尹怀，阿舒托什·肖汉，艾伦·盖茨，冈瑟·哈格莱特纳，埃里克·汉森，欧文·欧玛利，吉坦德拉·潘迪，袁媛，鲁宝·李和张晓东。2014.阿帕奇蜂巢的主要技术进步。在 2014 年 ACM SIGMOD 国际数据管理会议记录(SIGMOD '14)中。美国纽约州纽约市，1235-1246。

[26] Julian Hyde.2010.Data in Flight.Commun.ACM 53, 1 (Jan. 2010), 48-52.

[26]朱利安·海德。2010.飞行中的数据。通信。ACM 53，1(2010 年 1 月)，48-52。

[27] Janino.Janino: A super-small, super-fast Java compiler.(Nov. 2017).Retrieved November 20, 2017 from http://www.janino.net/

[27]雅尼诺。一个超小型、超快速的 Java 编译器。(2017 年 11 月)。2017 年 11 月 20 日从 http://www.janino.net/检索

[28] Kylin.Apache Kylin.(Nov. 2017).Retrieved November 20, 2017 from http: //kylin.apache.org/

[28]麒麟。阿帕奇·麒麟。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[29] Avinash Lakshman and Prashant Malik.2010.Cassandra: A Decentralized Struc-tured Storage System.SIGOPS Oper.Syst.Rev. 44, 2 (April 2010), 35-40.

[29]阿维纳什·拉克什曼和普拉尚特·马利克。2010.卡珊德拉:一个分散的结构化存储系统。SIGOPS Oper。系统。第 44 卷，第 2 期(2010 年 4 月)，第 35-40 页。

[30] Lingual.Lingual.(Nov. 2017).Retrieved November 20, 2017 from http://www.cascading.org/projects/lingual/

[30]语言。语言。(2017 年 11 月)。检索于 2017 年 11 月 20 日。cascading.org/projects/lingual/

[31] Lucene.Apache Lucene.(Nov. 2017).Retrieved November 20, 2017 from https: //lucene.apache.org/

[31]露西。阿帕奇·卢塞恩。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[32] MapD.MapD.(Nov. 2017).Retrieved November 20, 2017 from https://www.mapd.com

[32] MapD。MapD。(2017 年 11 月)。检索于 2017 年 11 月 20 日。mapd.com

[33] Erik Meijer, Brian Beckman, and Gavin Bierman.2006.LINQ: Reconciling Object, Relations and XML in the .NET Framework.In Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD '06).ACM, New York, NY, USA, 706-706.

[33]埃里克·梅耶尔、布莱恩·贝克曼和加文·比尔曼。2006.LINQ:协调对象、关系和 XML。NET 框架。在 2006 年 ACM SIGMOD 国际数据管理会议记录(SIGMOD '06)中。美国纽约州纽约市，邮编:706-706。

[34] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shiv-akumar, Matt Tolton, and Theo Vassilakis.2010.Dremel: Interactive Analysis of Web-Scale Datasets.PVLDB 3, 1 (2010), 330-339.http://www.comp.nus.edu.sg/ ~vldb2010/proceedings/files/papers/R29.pdf

[34]谢尔盖·梅尔尼克、安德烈·古巴列夫、景静龙、杰弗里·罗默、希瓦·希夫-阿库马、马特·托尔顿和西奥·瓦西拉基斯。2010.网络规模数据集的交互式分析。PVLDB 3，1 (2010)，330-339。http://www.comp.nus.edu.sg/ ~ vldb 2010/会议记录/档案/论文/R29.pdf

[35] Marcelo RN Mendes, Pedro Bizarro, and Paulo Marques.2009.A performance study of event processing systems.In Technology Conference on Performance Evaluation and Benchmarking.Springer, 221-236.

[35]马塞洛·恩·门德斯、佩德罗·比扎罗和保罗·马克斯。2009.事件处理系统的性能研究。绩效评估和基准技术会议。斯普林格，221-236。

[36] Mongo.MongoDB.(Nov. 2017).Retrieved November 28, 2017 from https: //www.mongodb.com/

[36]蒙戈。蒙古数据库。(2017 年 11 月)。检索于 2017 年 11 月 28 日

[37] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins.2008.Pig Latin: a not-so-foreign language for data processing.In SIGMOD.

[37]克里斯托弗·奥尔斯顿、本杰明·里德、乌卡什·斯里瓦斯塔瓦、拉维·库马尔和安德鲁·汤姆金斯。2008.猪拉丁语:一种不那么陌生的数据处理语言。在 SIGMOD。

[38] Kian Win Ong, Yannis Papakonstantinou, and Romain Vernoux.2014.The SQL++ query language: Configurable, unifying and semi-structured.arXiv preprint

[38] Kian Win Ong，Yannis Papakonstantinou 和 Romain Vernoux。2014.SQL++查询语言:可配置、统一和半结构化。arXiv 预印本

arXiv:1405.3631 (2014).

arXiv:1405.3631 (2014)。

[39] Open Geospatial Consortium.OpenGIS Implementation Specification for Ge-ographic information - Simple feature access - Part 2: SQL option.http: //portal.opengeospatial.org/files/?artifact_id=25355.(2010).

[39]开放地理空间联合会。地理信息的实现规范。简单特征访问。第 2 部分:SQL 选项。http://portal . open geography . org/files/? artifact_id=25355。(2010 年)。

[40] Phoenix.Apache Phoenix.(Nov. 2017).Retrieved November 20, 2017 from http://phoenix.apache.org/

[40]凤凰城。阿帕奇凤凰城。(2017 年 11 月)。2017 年 11 月 20 日从 http://phoenix.apache.org/检索

[41] Pig.Apache Pig.(Nov. 2017).Retrieved November 20, 2017 from http://pig.apache.org/

[41]猪。阿帕奇猪。(2017 年 11 月)。检索于 2017 年 11 月 20 日。apache.org/

[42] Qubole Quark.Qubole Quark.(Nov. 2017).Retrieved November 20, 2017 from https://github.com/qubole/quark

[42]量子粒子夸克。量子位夸克。(2017 年 11 月)。2017 年 11 月 20 日从 https://github.com/qubole/quark 检索

[43] Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun C. Murthy, and Carlo Curino.2015.Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications.In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.1357-1369.https://doi.org/10.1145/2723372.2742790

[43]比卡斯·萨哈、希泰什·沙阿、西达尔特·塞思、戈帕尔·维贾亚拉·哈万、阿伦·穆蒂和卡洛·库里诺。2015.阿帕奇技术中心:建模和构建数据处理应用的统一框架。2015 年 5 月 31 日至 6 月 4 日在澳大利亚维多利亚州墨尔本举行的 2015 年 ACM SIGMOD 国际数据管理会议记录。1357-1369。https://doi.org/10.1145/2723372.2742790

[44] Samza.Apache Samza.(Nov. 2017).Retrieved November 20, 2017 from http: //samza.apache.org/

[44] Samza。Apache Samza。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[45] Mohamed A. Soliman, Lyublena Antova, Venkatesh Raghavan, Amr El-Helw, Zhongxian Gu, Entong Shen, George C. Caragea, Carlos Garcia-Alvarado, Foyzur Rahman, Michalis Petropoulos, Florian Waas, Sivaramakrishnan Narayanan, Konstantinos Krikellas, and Rhonda Baldwin.2014.Orca: A Modular Query Optimizer Architecture for Big Data.In Proceedings of the

2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14).ACM, New York, NY, USA, 337-348.

[45] Mohamed A. Soliman，Lyublena Antova，Venkatesh Raghavan，Amr El-Helw，钟仙谷，申，George C. Caragea，Carlos Garcia-Alvarado，Foyzur Rahman，Michalis，Florian Waas，Sivaramakrishnan Narayanan，Konstantinos Krikellas 和 Rhonda Baldwin。2014.Orca:大数据的模块化查询优化器架构。在 2014 年 ACM SIGMOD 国际数据管理会议记录(SIGMOD '14)中。美国纽约州纽约市，337-348。

[46] Solr.Apache Solr.(Nov. 2017).Retrieved November 20, 2017 from http://lucene.apache.org/solr/

[46] Solr。阿帕奇·索尔。(2017 年 11 月)。检索于 2017 年 11 月 20 日。apache.org/solr/

[47] Spark.Apache Spark.(Nov. 2017).Retrieved November 20, 2017 from http: //spark.apache.org/

[47]火花。阿帕奇火花。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[48] Splunk.Splunk.(Nov. 2017).Retrieved November 20, 2017 from https://www.splunk.com/

[48] Splunk。啪啪。(2017 年 11 月)。检索于 2017 年 11 月 20 日。splunk.com/

[49] Michael Stonebraker and Ugur Çetintemel.2005."One size fits all": an idea whose time has come and gone.In 21st International Conference on Data Engineering (ICDE'05).IEEE Computer Society, Washington, DC, USA, 2-11.

[49]迈克尔·斯通布雷克和乌古尔·埃廷特梅尔。2005.“一刀切”:这个想法的时代已经过去。在第 21 届国际数据工程会议(ICDE'05)上。IEEE 计算机学会，华盛顿州，DC，美国，2-11。

[50] Storm.Apache Storm.(Nov. 2017).Retrieved November 20, 2017 from http: //storm.apache.org/

[50]风暴。阿帕奇风暴。(2017 年 11 月)。检索于 2017 年 11 月 20 日

[51] Tez.Apache Tez.(Nov. 2017).Retrieved November 20, 2017 from http://tez.apache.org/

[51] Tez。Apache Tez。(2017 年 11 月)。检索于 2017 年 11 月 20 日。apache.org/

[52] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy.2009.Hive: a warehousing solution over a map-reduce framework.VLDB (2009), 1626-1629.

[52]阿希什·图苏、茹瓦迪普·森·萨尔马、纳米特·贾恩、邵铮、普拉萨德·查卡、苏雷什·安东尼、刘浩、皮特·威科夫和拉古坦·穆尔西。2009.Hive:地图简化框架上的仓储解决方案。VLDB (2009)，1626-1629。

[53] Immanuel Trummer and Christoph Koch.2017.Multi-objective parametric query optimization.The VLDB Journal 26, 1 (2017), 107-124.

[53]伊曼努尔·特鲁默和克里斯托夫·科赫。2017.多目标参数查询优化。《VLDB 日报》26，1 (2017)，107-124。

[54] Ashwin Kumar Vajantri, Kunwar Deep Singh Toor, and Edmon Begoli.2017.An Apache Calcite-based Polystore Variation for Federated Querying of Heteroge-neous Healthcare Sources.In 2nd Workshop on Methods to Manage Heterogeneous Big Data and Polystore Databases.IEEE Computer Society, Washington, DC, USA.

[54] Ashwin Kumar Vajantri，Kunwar Deep Singh Toor 和 Edmon Begoli。2017.一种基于阿帕奇方解石的多元统计变量，用于联合查询异质医疗保健源。第二次研讨会，关于管理异构大数据和多项存储数据库的方法。美国 DC 华盛顿州 IEEE 计算机学会。

[55] Katherine Yu, Vijay Gadepally, and Michael Stonebraker.2017.Database engine integration and performance analysis of the BigDAWG polystore system.In 2017 IEEE High Performance Extreme Computing Conference (HPEC).IEEE Computer Society, Washington, DC, USA, 1-7.

[55]凯瑟琳·余、维贾伊·加代帕利和迈克尔·斯通布雷克。2017.BigDougpolystore 系统的数据库引擎集成和性能分析。在 2017 年 IEEE 高性能极限计算会议(HPEC)上。IEEE 计算机学会，华盛顿州，DC，美国，1-7。

[56] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica.2010.Spark: Cluster Computing with Working Sets.In HotCloud.

[56]马泰·扎哈里亚、莫沙里夫·乔杜里、迈克尔·富兰克林、斯科特·申克和扬·斯托伊察。2010.火花:带工作集的集群计算。在 HotCloud 中。

[57] Jingren Zhou, Per-Åke Larson, and Ronnie Chaiken.2010.Incorporating partition-ing and parallel plans into the SCOPE optimizer.In 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010).IEEE Computer Society, Washington, DC, USA, 1060-1071.

[57]周、佩-克·拉森和罗尼·柴肯。2010.将分区和并行计划纳入 SCOPE 优化器。2010 年 IEEE 第 26 届国际数据工程会议(ICDE，2010 年)。美国 DC 华盛顿州计算机学会，1060-1071。