以太坊合并大数据基于 DataV 的实现

课程报告

陈昊天

（浙江理工大学，21 计科 4 班，2021329600006）

摘　要：本文基于 Google BigQuery 提供的公共数据集 crypto_ethereum，结合大数据技术，对以太坊合并事件进行了详细分析。通过使用 DataV 等可视化工具，我们对合并前后的交易数量和费用、出块时间、网络哈希率、Gas 价格、智能合约部署情况及用户行为进行了数据处理和可视化展示。研究结果表明，以太坊从工作量证明向权益证明机制的转变在多个方面产生了显著影响。合并后交易数量和费用均有所下降，区块生成时间显著减少且更为稳定，网络哈希率急剧下降，平均 Gas 价格下降，智能合约部署活跃，用户活跃度和新增地址数量保持相对稳定。本文展示了大数据技术和可视化工具在金融科技领域的强大应用潜力，为区块链技术的发展和应用提供了宝贵参考。

关键词：以太坊合并　大数据　DataV　可视化　区块链技术

一、引言

（一）研究背景与意义

随着区块链技术的快速发展，以太坊作为全球第二大加密货币平台，其技术演进和市场动态对整个加密货币市场产生了深远的影响。以太坊合并作为该平台历史上的一个重要里程碑，标志着从工作量证明向权益证明机制的转变，这一变革不仅涉及技术层面的重大更新，也对网络的安全性、效率和可持续性带来了显著的影响[1]。因此，深入分析以太坊合并事件的影响，对于理解区块链技术的发展趋势、指导未来的技术研究和应用具有重要的理论和实践意义。

在大数据时代背景下，数据的规模和复杂性不断增长，如何有效利用和分析这些数据成为了一个重要课题。本研究通过利用 Google BigQuery 提供的公共数据集 crypto_ethereum，结合大数据技术，对以太坊合并事件的影响进行实证分析。这不仅能够为区块链技术的研究

者和实践者提供宝贵的数据支持和见解，也有助于推动大数据技术在金融科技领域的应用和发展。

本研究还探讨了大数据技术在区块链领域的应用潜力，特别是在处理和分析大规模区块链数据集方面的能力。通过这项研究，我们希望能够展示大数据技术在金融科技领域的实际应用价值，并为相关领域的研究者和实践者提供参考和启示。

（二）大数据时代的到来

我们正处于一个数据爆炸的时代，大数据技术的兴起标志着信息技术的新纪元。随着互联网的普及和物联网技术的发展，数据的产生速度和规模日益增长，大数据已经成为推动社会进步和商业创新的关键力量。在金融、医疗、交通、教育等多个领域，大数据的应用不断深化，对决策支持、风险管理、个性化服务等方面产生了深远的影响。

在区块链领域，以太坊作为领先的智能合约平台，其庞大的交易量和用户基数产生了海量的数据。这些数据包含了丰富的网络行为信息和市场动态，对于理解区块链技术的发展、优化网络性能、预测市场趋势具有重要的价值。如何有效收集、存储、处理和分析这些数据，成为了一个亟待解决的问题。

本研究聚焦于以太坊合并事件，通过分析 Google BigQuery 提供的公共数据集，探讨大数据技术在区块链领域的应用。这项研究有助于揭示以太坊合并对网络性能和市场反应的影响，展示了大数据技术在金融科技领域的潜力和应用前景。

（三）大数据可视化的重要性

在当前的信息时代，数据已成为企业和组织的重要资产。大数据技术的发展使得我们能够从海量数据中提取有价值的信息，但同时也带来了数据复杂性和难以直接理解的挑战。大数据可视化作为连接数据与用户理解的桥梁，其重要性日益凸显。

可视化技术能够将复杂的数据集转化为直观的图形和图表，使用户能够快速识别模式、趋势和异常。在区块链领域，尤其是以太坊这样的大型网络，数据的实时性和动态性要求我们能够即时获取和理解信息。通过可视化，我们可以更有效地监控网络状态，分析交易模式，评估市场情绪，以及预测潜在的风险。

本研究中，我们利用 DataV 等可视化工具，将从 Google BigQuery 获取的以太坊区块链数据转化为直观的展示，以便于分析和解释以太坊合并事件对网络性能、安全性、交易量、交易费用以及市场反应的影响。通过这种方式，我们能够提供深入的数据分析，使非技术背

景的决策者和利益相关者更容易地理解复杂的区块链现象。

二、项目数据分析

（一）数据来源和数据集概述

1. 数据集来源

bigquery-public-data.crypto_ethereum 数据集托管在 Google BigQuery 上，提供了对以太坊区块链数据的公开访问[2]。

Google 使用 medvedev1088/ethereum-etl 项目制作该数据集[3][4]，其基本原理如下：

（1）数据提取

ethereum-etl 使用 web3.py 库与以太坊节点通信，提取区块、交易、ERC20/ERC721 代币转账、日志、合约和内部交易等数据。提取区块数据的代码如下：

```
from web3 import Web3
web3 = Web3(Web3.HTTPProvider('YOUR_INFURA_URL'))
block = web3.eth.getBlock('latest')
```

（2）数据转换

将提取的数据转换为 CSV 或 JSON 格式，以便加载到数据库中。

```
import csv
block_data = {
    'number': block.number,
    'hash': block.hash.hex(),
    'parentHash': block.parentHash.hex(),
    'miner': block.miner,
    'timestamp': block.timestamp
}
with open('blocks.csv', mode='w') as file:
    writer = csv.writer(file)
    writer.writerow(block_data.keys())
writer.writerow(block_data.values())
```

（3）数据加载

将转换后的数据加载到 BigQuery 中。

```
from google.cloud import bigquery
client = bigquery.Client()
schema = [
    bigquery.SchemaField('number', 'INTEGER'),
    bigquery.SchemaField('hash', 'STRING'),
    bigquery.SchemaField('parentHash', 'STRING'),
    bigquery.SchemaField('miner', 'STRING'),
    bigquery.SchemaField('timestamp', 'TIMESTAMP'),
]
table_id = 'your-project.your_dataset.your_table'
job_config = bigquery.LoadJobConfig(schema=schema)
with open('blocks.csv', 'rb') as file:
    load_job = client.load_table_from_file(file, table_id,
job_config=job_config)
load_job.result()
print('Loaded {} rows into {}.'.format(load_job.output_rows, table_id))
```

2. 数据集访问

在 GCP 项目中启用 BigQuery API 即可访问数据集。

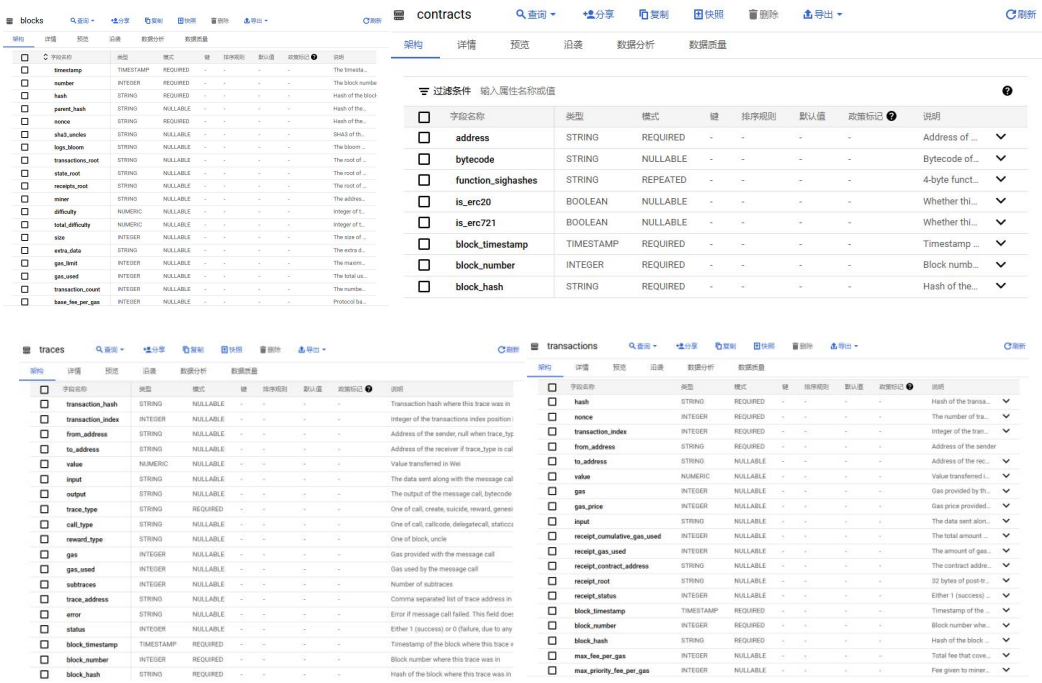
3. 数据集大小

该数据集大小随着区块链的增长而不断增加。截至 2024 年 5 月 20 日，数据集大小为 14091.61GB。

4. 数据类型

数据集由多张表组成，每张表包含不同类型的数据，包括：区块信息、交易信息、智能合约信息、ERC-20 代币转账信息、交易追踪信息。

以下展示几张较为重要的表架构信息。



5. 数据的时间范围和地理覆盖范围

（1）时间范围

数据集涵盖了从 2015 年 7 月以太坊创世区块以来的所有区块链数据，持续更新到当前区块。

（2）地理覆盖范围

没有特定的地理限制或覆盖范围，或称地理范围为全球。以太坊区块链是一个全球性的去中心化网络，节点分布在世界各地。

（二）数据处理与清洗

在使用 BigQuery 进行数据处理前，需要先简单介绍 BigQuery 的工作原理。

BigQuery 的底层技术之一是 Dremel，它是一种交互式分析工具，能够在数秒内处理 PB 级数据。Google 开发了 Dremel 以将处理时间缩短到秒级，作为 MapReduce 的有力补充。

根据 Google 论文 *Dremel: Interactive Analysis of WebScaleDatasets*，Dremel 系统有下面几个主要的特点：多级执行树架构、列式存储格式、递归聚合技术、高效的编码和压缩、类 SQL 的查询语言等[5]。

1. Dremel 的存储

数据模型基于强类型的嵌套记录。其抽象语法如下：

$$\tau = \mathbf{dom} \mid \langle A_1 : \tau[*|?], \ldots, A_n : \tau[*|?] \rangle$$

其中，τ 是一个原子类型或记录类型。dom 中的原子类型包括整数、浮点数、字符串等。记录由一个或多个字段组成。记录中的第 i 个字段具有名称 Ai 和一个可选的多重性标签。重复字段（*）在一个记录中可以出现多次。它们被解释为值的列表，即字段在记录中出现的顺序是重要的。可选字段（?）可能会在记录中缺失。

如图展示了定义 Web 文档的记录类型 Document 的模式。Document 有一个必需的整数 DocId 和一个可选的 Links，包含一个 Forward 和 Backward 条目的列表，记录了其他网页的 DocIds。一个文档可以有多个 Names，即该文档可以通过不同的 URLs 引用。Name 包含一系列的 Code 和 Country 对。模式中定义的字段形成了一个树状层次结构。

下图是这份数据在 Dremel 实际的存储的格式。

DocId

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2
80	0	2

Links.Backward

value	r	d
NULL	0	1
10	0	2
30	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

2. Dremel 的查询

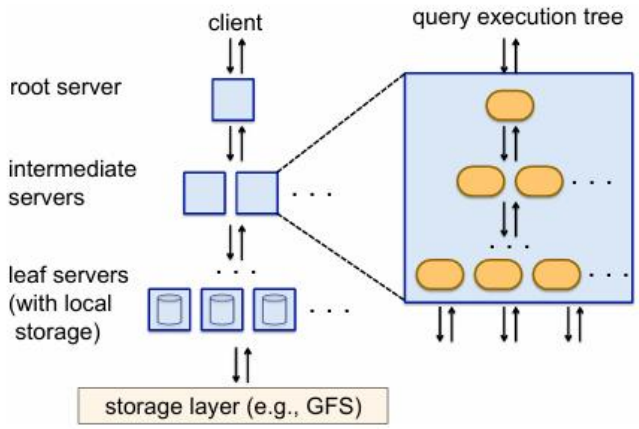
Dremel 的查询语言基于 SQL，并设计为能高效地在列式嵌套存储上实现。每个 SQL 语句以一个或多个嵌套表及其模式为输入，并生成一个嵌套表及其输出模式。如图展示了一个示例查询，该查询执行投影、选择和记录内聚合操作。

```
SELECT DocId AS Id,
  COUNT(Name.Language.Code) WITHIN Name AS Cnt,
  Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;
```

```
Id: 10                                  t1
Name
  Cnt: 2
  Language
    Str: 'http://A,en-us'
    Str: 'http://A,en'
Name
  Cnt: 0
```

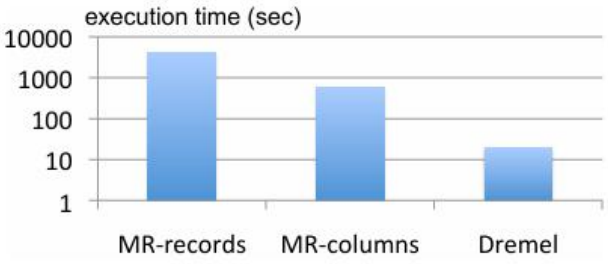
```
message QueryResult {
  required int64 Id;
  repeated group Name {
    optional uint64 Cnt;
    repeated group Language {
      optional string Str; }}}
```

Dremel 使用多级服务树来执行查询。根服务器接收传入查询，从表中读取元数据，并将查询路由到服务树的下一层。叶服务器与存储层通信或访问本地磁盘上的数据。下图展示了树形查询过程。



3. Dremel 与 MapReduce

Google 展示了在列式数据与记录导向数据上执行 MR 和 Dremel 的对比。考虑一个访问单个字段的情况，即性能收益最显著的情况。在这个实验中，计算表 T1 中字段 txtField 的平均词数。



上图展示了两个 MR 作业和 Dremel 的执行时间。两个 MR 作业都在 3000 个工作节点上运行。一个 3000 节点的 Dremel 实例用于执行查询 Q1。Dremel 和基于列的 MR 读取了约 0.5TB 的压缩列式数据，而基于记录的 MR 读取了 87TB 的数据。如图所示，通过从记录导向存储切换到列式存储，MR 的效率提高了一个数量级。通过使用 Dremel，又获得了一个数量级的提升。

4. 交易数据

在理解 BigQuery 和 Dremel 的关系后，我们可以使用 SQL 语言对数据进行处理和清洗。

编写 SQL 查询，从公共数据集中选择 2022 年 8 月 1 日至 10 月 31 日之间的以太坊交易数据，计算每天的交易数量和平均交易费用。执行查询并将结果存储为 pandas DataFrame，使用前向填充方法处理 DataFrame 中的缺失值，计算平均交易费用的四分位数和四分位距，然后去除异常值，使用 sklearn 的 StandardScaler 对交易数量和平均交易费用进行标准化处理。

```
from google.cloud import bigquery
import pandas as pd
client = bigquery.Client()
# 每日交易数量和平均交易费用
query = """
    SELECT
        DATE(block_timestamp) AS date,
        COUNT(*) AS daily_transactions,
        AVG(cast(gas_price AS FLOAT64) * cast(gas AS FLOAT64) / 1e18) AS
avg_transaction_fee
    FROM
        `bigquery-public-data.crypto_ethereum.transactions`
```

```
        WHERE
            block_timestamp BETWEEN '2022-08-01' AND '2022-10-31'
        GROUP BY
            date
        ORDER BY
            date
    """
    # 结果存储
    query_job = client.query(query)
    df = query_job.to_dataframe()
    # 处理缺失值
    df.fillna(method='ffill', inplace=True)
    # 处理异常值
    q1 = df['avg_transaction_fee'].quantile(0.25)
    q3 = df['avg_transaction_fee'].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    df = df[(df['avg_transaction_fee'] >= lower_bound) &
(df['avg_transaction_fee'] <= upper_bound)]
    # 数据转换和规范化
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    df[['daily_transactions', 'avg_transaction_fee']] =
scaler.fit_transform(df[['daily_transactions', 'avg_transaction_fee']])
    merge_date = '2022-09-15'
    df['period'] = df['date'].apply(lambda x: 'before' if x < merge_date else
'after')
```

5. 矿工奖励和出块时间

数据处理方法与交易数据相似。

```
from google.cloud import bigquery
import pandas as pd
client = bigquery.Client()
# 每日矿工奖励和区块生成时间
query = """
    SELECT
        DATE(timestamp) AS date,
        AVG(reward) AS avg_miner_reward,
        AVG(TIMESTAMP_DIFF(LEAD(timestamp) OVER (ORDER BY timestamp),
timestamp, SECOND)) AS avg_block_time
    FROM
        `bigquery-public-data.crypto_ethereum.blocks`
    WHERE
        timestamp BETWEEN '2022-08-01' AND '2022-10-31'
    GROUP BY
        date
    ORDER BY
        date
"""
query_job = client.query(query)
df = query_job.to_dataframe()
# 处理缺失值
df.fillna(method='ffill', inplace=True)
# 处理异常值
q1_reward = df['avg_miner_reward'].quantile(0.25)
q3_reward = df['avg_miner_reward'].quantile(0.75)
iqr_reward = q3_reward - q1_reward
```

```
    q1_block_time = df['avg_block_time'].quantile(0.25)
    q3_block_time = df['avg_block_time'].quantile(0.75)
    iqr_block_time = q3_block_time - q1_block_time
    lower_bound_reward = q1_reward - 1.5 * iqr_reward
    upper_bound_reward = q3_reward + 1.5 * iqr_reward
    lower_bound_block_time = q1_block_time - 1.5 * iqr_block_time
    upper_bound_block_time = q3_block_time + 1.5 * iqr_block_time
    df = df[(df['avg_miner_reward'] >= lower_bound_reward) &
(df['avg_miner_reward'] <= upper_bound_reward)]
    df = df[(df['avg_block_time'] >= lower_bound_block_time) &
(df['avg_block_time'] <= upper_bound_block_time)]
    # 数据转换和规范化
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    df[['avg_miner_reward', 'avg_block_time']] =
scaler.fit_transform(df[['avg_miner_reward', 'avg_block_time']])
```

6. 网络哈希率和难度

数据处理方法与交易数据相似。

```
    from google.cloud import bigquery
    import pandas as pd
    client = bigquery.Client()
    # 每日网络哈希率和难度
    query = """
        SELECT
            DATE(timestamp) AS date,
            AVG(difficulty) AS avg_difficulty,
            AVG(hashrate) AS avg_hashrate
        FROM (
            SELECT
```

```
            timestamp,
            difficulty,
            (difficulty / TIMESTAMP_DIFF(LEAD(timestamp) OVER (ORDER BY
timestamp), timestamp, SECOND)) AS hashrate
        FROM
            `bigquery-public-data.crypto_ethereum.blocks`
        WHERE
            timestamp BETWEEN '2022-08-01' AND '2022-10-31'
    )
    GROUP BY
        date
    ORDER BY
        date
"""
query_job = client.query(query)
df = query_job.to_dataframe()
# 处理缺失值
df.fillna(method='ffill', inplace=True)
# 处理异常值
q1_difficulty = df['avg_difficulty'].quantile(0.25)
q3_difficulty = df['avg_difficulty'].quantile(0.75)
iqr_difficulty = q3_difficulty - q1_difficulty
q1_hashrate = df['avg_hashrate'].quantile(0.25)
q3_hashrate = df['avg_hashrate'].quantile(0.75)
iqr_hashrate = q3_hashrate - q1_hashrate
lower_bound_difficulty = q1_difficulty - 1.5 * iqr_difficulty
upper_bound_difficulty = q3_difficulty + 1.5 * iqr_difficulty
lower_bound_hashrate = q1_hashrate - 1.5 * iqr_hashrate
upper_bound_hashrate = q3_hashrate + 1.5 * iqr_hashrate
```

```
    df = df[(df['avg_difficulty'] >= lower_bound_difficulty) &
(df['avg_difficulty'] <= upper_bound_difficulty)]
    df = df[(df['avg_hashrate'] >= lower_bound_hashrate) &
(df['avg_hashrate'] <= upper_bound_hashrate)]
    # 数据转换和规范化
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    df[['avg_difficulty', 'avg_hashrate']] =
scaler.fit_transform(df[['avg_difficulty', 'avg_hashrate']])
    merge_date = '2022-09-15'
    df['period'] = df['date'].apply(lambda x: 'before' if x < merge_date else
'after')
```

三、数据处理过程实现

（一）使用的大数据技术栈

1. BigQuery

BigQuery 是 Google Cloud 提供的全托管、无服务器的数据仓库解决方案，专为处理和分析大规模数据而设计。它能够支持 PB 级的数据查询和分析，使用 SQL 查询语言，使得数据科学家和分析师能够轻松执行复杂的分析任务。BigQuery 具有自动扩展和高性能的特点，能够根据用户需求动态调整计算资源，并与其他 Google Cloud 服务无缝集成，使得数据导入、导出和处理变得更加简便。其内置的机器学习功能也使得用户可以直接在数据仓库中进行预测分析，无需将数据迁移到其他平台。

2. Dremel

Dremel 是支持 BigQuery 的核心技术之一，是 Google 开发的分布式交互式查询系统。Dremel 通过采用树状结构的查询执行方式，能够在数千台服务器上并行处理查询请求，显著提高了查询性能和响应速度。与传统的分布式查询系统相比，Dremel 具有低延迟、高并发的优势，可以在几秒钟内处理数十亿行数据。这使得用户能够实时分析大量数据，从而快速做出业务决策。Dremel 的创新设计使得它不仅适用于结构化数据，还能高效处理半结构

化和非结构化数据。

3. Colossus / GFS

Colossus 是 Google 内部使用的分布式存储系统，作为 GFS 的继任者，Colossus 提供了更高的可靠性、可扩展性和性能。它设计用于支持 Google 的全球性基础设施，能够处理海量数据存储和访问需求。Colossus 采用了多副本存储和分布式数据处理技术，确保数据的高可用性和一致性，同时支持多租户环境和细粒度的访问控制。它还具备自动恢复和负载均衡功能，当硬件故障发生时，系统能够自动迁移数据和计算任务，保证服务的连续性和稳定性。

4. Borg

Borg 是 Google 内部的集群管理系统，用于调度和管理运行在数百万台服务器上的容器化应用程序。Borg 的设计目标是提供高效的资源利用率、可靠性和弹性，支持大规模分布式计算。它通过资源调度算法将计算任务分配到不同的服务器上，最大化资源利用率，同时通过冗余和故障转移机制，确保系统的高可用性。Borg 的成功经验为 Kubernetes 的开发提供了宝贵的实践基础，后者已经成为业界标准的容器编排系统。

（二）数据分析方法

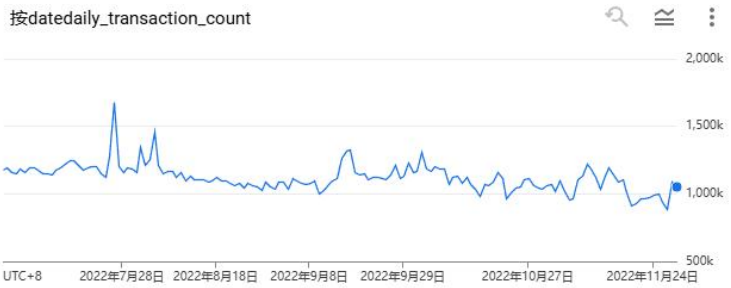
使用 BigQuery Standard SQL 进行数据检索和分析，并绘制可供参考的图像。

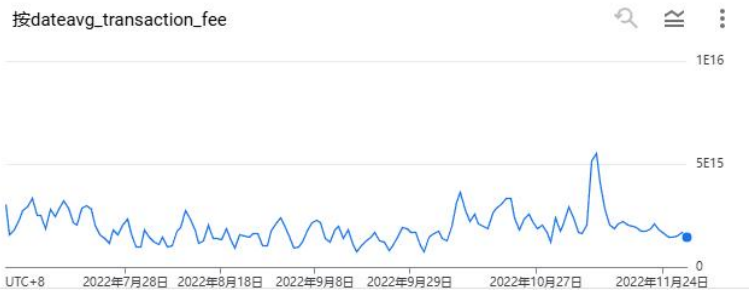
1. 交易数量和费用

比较合并前后每日的交易数量，分析合并前后交易费用的变化。其中 avg_transaction_fee 的单位为 wei，1 ETH = 1e18 wei。

```
-- 查询每日交易数量和平均交易费用
WITH daily_transactions AS (
    SELECT
        DATE(block_timestamp) AS date,
        COUNT(*) AS daily_transaction_count,
        AVG(CAST(gas_price AS FLOAT64) * CAST(receipt_gas_used AS FLOAT64)) AS
avg_transaction_fee
    FROM
        `bigquery-public-data.crypto_ethereum.transactions`
    WHERE
        block_timestamp BETWEEN '2022-07-01' AND '2022-11-30'
    GROUP BY
        date
```

```
    ORDER BY
        date
)
-- 标记合并前后时期
SELECT
    date,
    daily_transaction_count,
    avg_transaction_fee,
    CASE
        WHEN date < '2022-09-15' THEN 'before'
        ELSE 'after'
    END AS period
FROM
    daily_transactions
ORDER BY
    date;
```



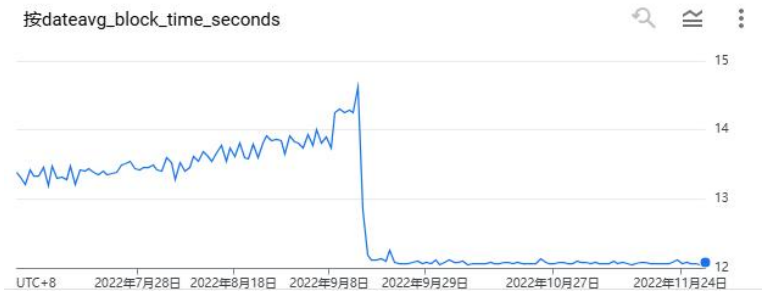


2. 出块时间

比较合并前后区块生成时间的变化。

```
-- 分析区块生成时间变化
WITH block_intervals AS (
  SELECT
    number AS block_number,
    timestamp,
    LAG(timestamp) OVER (ORDER BY number) AS previous_block_timestamp
  FROM
    `bigquery-public-data.crypto_ethereum.blocks`
  WHERE
```

```
    timestamp BETWEEN '2022-07-01' AND '2022-11-30'
  )
  SELECT
    DATE(timestamp) AS date,
    AVG(TIMESTAMP_DIFF(timestamp, previous_block_timestamp, SECOND)) AS
avg_block_time_seconds
  FROM
    block_intervals
  WHERE
    previous_block_timestamp IS NOT NULL
  GROUP BY
    date
  ORDER BY
    date;
```



3. 网络哈希率

分析合并前后的网络哈希率变化，其中 avg_hash_rate 是每日平均哈希率，单位为 Hash/s。

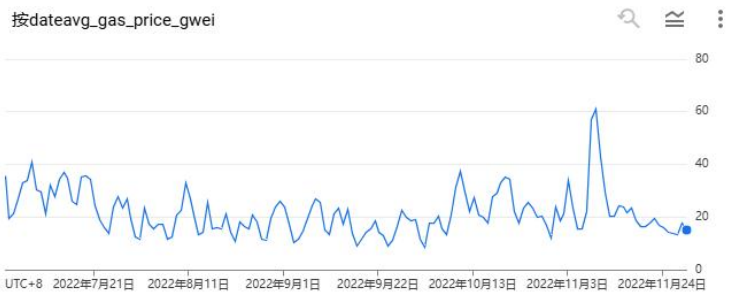
```
  -- 分析网络哈希率变化
  WITH block_intervals AS (
    SELECT
      number AS block_number,
      timestamp,
      difficulty,
      LAG(timestamp) OVER (ORDER BY number) AS previous_block_timestamp
    FROM
      `bigquery-public-data.crypto_ethereum.blocks`
    WHERE
      timestamp BETWEEN '2022-07-01' AND '2022-11-30'
  ),
  hash_rate_data AS (
    SELECT
      block_number,
      timestamp,
      difficulty,
```

```
      TIMESTAMP_DIFF(timestamp, previous_block_timestamp, SECOND) AS
block_time_seconds
    FROM
      block_intervals
    WHERE
      previous_block_timestamp IS NOT NULL
  )
  SELECT
    DATE(timestamp) AS date,
    AVG(difficulty / block_time_seconds) AS avg_hash_rate
  FROM
    hash_rate_data
  GROUP BY
    date
  ORDER BY
    date;
```

4. Gas 价格

比较合并前后的平均 Gas 价格变化。

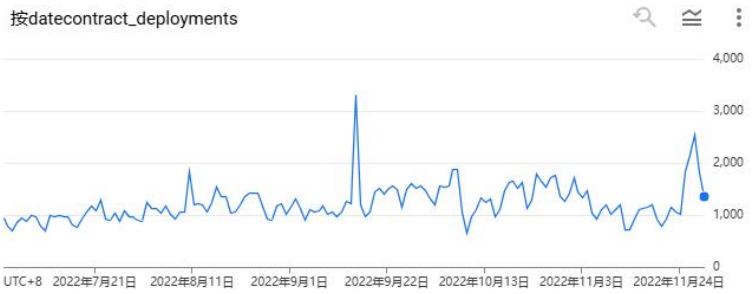
```
-- 分析平均 Gas 价格变化
SELECT
  DATE(block_timestamp) AS date,
  AVG(CAST(gas_price AS FLOAT64)) / 1e9 AS avg_gas_price_gwei
FROM
  `bigquery-public-data.crypto_ethereum.transactions`
WHERE
  block_timestamp BETWEEN '2022-07-01' AND '2022-11-30'
GROUP BY
  date
ORDER BY
  date;
```



5. 以太坊网络活动

比较合并前后的智能合约部署数量。

```
-- 分析智能合约部署数量
SELECT
  DATE(block_timestamp) AS date,
  COUNT(*) AS contract_deployments
FROM
  `bigquery-public-data.crypto_ethereum.transactions`
WHERE
  block_timestamp BETWEEN '2022-07-01' AND '2022-11-30'
  AND to_address IS NULL
GROUP BY
  date
ORDER BY
  date;
```

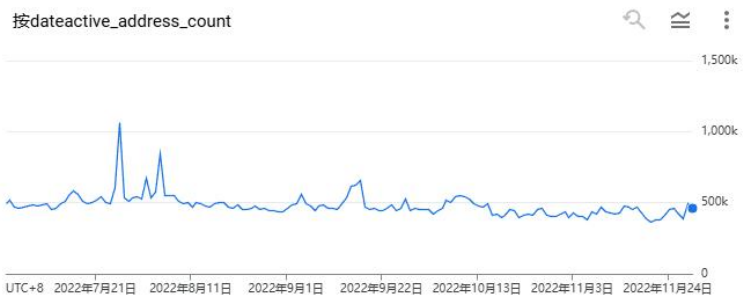


6. 用户行为

分析活跃地址数量的变化，观察用户活跃度的变化情况。分析新增地址数量的变化，看看是否有更多新用户加入。活跃地址可以通过每天进行交易的唯一地址数来衡量，新增地址可以通过每天首次出现在区块链上的地址数来衡量。

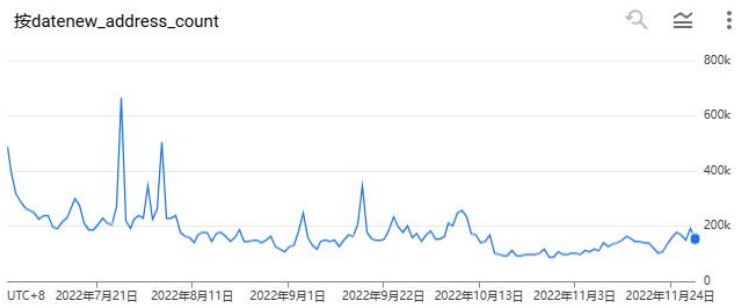
```
-- 分析活跃地址数量变化
WITH active_addresses AS (
  SELECT
    DATE(block_timestamp) AS date,
    from_address AS address
```

```
  FROM
    `bigquery-public-data.crypto_ethereum.transactions`
  WHERE
    block_timestamp BETWEEN '2022-07-01' AND '2022-11-30'
  UNION DISTINCT
  SELECT
    DATE(block_timestamp) AS date,
    to_address AS address
  FROM
    `bigquery-public-data.crypto_ethereum.transactions`
  WHERE
    block_timestamp BETWEEN '2022-07-01' AND '2022-11-30'
    AND to_address IS NOT NULL
)
SELECT
  date,
  COUNT(DISTINCT address) AS active_address_count
FROM
  active_addresses
GROUP BY
  date
ORDER BY
  date;
```




```
-- 分析新增地址数量变化
WITH first_seen_addresses AS (
  SELECT
    address,
    MIN(DATE(block_timestamp)) AS first_seen_date
  FROM (
    SELECT
      from_address AS address,
      block_timestamp
    FROM
      `bigquery-public-data.crypto_ethereum.transactions`
    WHERE
      block_timestamp BETWEEN '2022-07-01' AND '2022-11-30'
```

```
    UNION ALL
    SELECT
      to_address AS address,
      block_timestamp
    FROM
      `bigquery-public-data.crypto_ethereum.transactions`
    WHERE
      block_timestamp BETWEEN '2022-07-01' AND '2022-11-30'
      AND to_address IS NOT NULL
  )
  GROUP BY
    address
)
SELECT
  first_seen_date AS date,
  COUNT(address) AS new_address_count
FROM
  first_seen_addresses
GROUP BY
  date
ORDER BY
  date;
```



（三）主要的数据洞察

1. 交易数量和费用

①交易数量分析

合并前：每日交易数量相对稳定，有一些波动。在 2022 年 8 月初和中旬有明显的交易量峰值，合并临近时，交易数量略有上升，用户在合并前进行更多交易以防止潜在的网络不稳定性。

合并后：每日交易数量总体上有所下降，但仍然保持在一定的范围内。交易数量在合并后的波动相对较小，显示出网络在转向 PoS 机制后逐渐稳定。合并后的交易数量没有出现极

端波动，网络在过渡过程中保持了相对的稳定性。

②交易费用分析

合并前：平均交易费用有较大的波动。在 2022 年 8 月和 9 月初，交易费用有一些明显的峰值，与网络拥堵和 Gas 价格波动有关。合并前，交易费用整体上升。

合并后：平均交易费用在合并后明显下降，并保持在较低水平。合并后费用的下降表明 PoS 机制在一定程度上减少了网络拥堵。尽管有一些小的峰值，整体费用趋势仍然是下降的，网络在合并后的效率有所提高。

2. 出块时间

合并前：区块生成时间逐渐上升，平均区块时间约为 13 到 14 秒。在合并前的一段时间里，区块生成时间略有增加。

合并后：区块生成时间显著下降，平均区块时间约为 12 秒，并且非常稳定。以太坊从 PoW 转向 PoS 后，区块生成时间更加一致和稳定。PoS 机制通过验证者进行区块生成，消除了 PoW 中矿工竞争的随机性，从而大幅降低了区块生成时间的波动。

3. 网络哈希率

合并前：网络哈希率较为稳定，保持在一个较高的水平。在合并前的几个月，哈希率略有波动，但整体趋势保持稳定。

合并后：网络哈希率急剧下降到接近零。以太坊网络从工作量证明 PoW 完全转向了权益证明 PoS，哈希率在 PoS 机制中不再适用，因为 PoS 不依赖于计算能力来验证交易和生成区块。

4. Gas 价格

合并前：平均 Gas 价格有一定的波动，但大多数时间保持在 10 到 30 Gwei 之间。在某些日期，Gas 价格有明显的峰值。

合并后：平均 Gas 价格整体呈下降趋势，但仍有一些波动和短期的峰值。在合并后的某些日期，Gas 价格有较高的峰值。总体来看，合并后 Gas 价格相对更加平稳，虽然仍有波动，但波动幅度较小。

5. 以太坊网络活动

合并前：智能合约部署数量相对稳定，有一些波动。在某些日期，合约部署数量有明显的峰值。2022 年 9 月初有一个明显的高峰，可能与合并临近有关。

合并后：合并后，智能合约部署数量继续保持一定的波动，有几次较大的峰值。整体来看，合约部署数量在合并后并没有显著下降，反而在某些日期有较高的活动量，网络在合并后仍然保持活跃。

6. 用户行为

①活跃地址数量分析

合并前：活跃地址数量在几次高峰后相对稳定。在 2022 年 8 月初和中旬有明显的峰值。合并临近时，活跃地址数量略有上升。

合并后：活跃地址数量保持相对稳定，有轻微下降趋势。虽然数量有所下降，但整体变化不大，表明网络在过渡到 PoS 后依然保持一定的用户活跃度。

②新增地址数量分析

合并前：新增地址数量在 7 月初和 8 月中有明显的峰值。在合并前的几个月，新增地址数量总体呈下降趋势。

合并后：新增地址数量在合并后保持相对稳定，但总体水平有所下降。虽然新增地址数量有所减少，但网络仍在吸引新用户加入。

四、可视化设计与实现

（一）可视化目标与需求

在进行以太坊合并大数据分析时，数据的复杂性和规模决定了必须使用有效的可视化手段来展示分析结果。我们的可视化目标是通过直观、简洁的图表和图形，让用户快速理解以太坊合并事件对各项指标的影响，并能够通过交互操作深入探索数据细节。具体需求包括展示交易数量和费用、出块时间、网络哈希率、Gas 价格、智能合约部署情况以及用户行为分析等多个方面的变化。

1. 不同领域的展示需求

不同领域的展示需求各不相同。在交易数量和费用方面，需要展示合并前后每日的交易数量及平均交易费用的变化趋势；在区块生成时间方面，通过折线图展示区块生成时间的变化趋势，揭示合并对网络性能的影响；在网络哈希率方面，通过柱状图或折线图展示网络哈希率的变化，分析合并对以太坊网络算力的影响；在 Gas 价格方面，利用折线图展示平均 Gas 价格的变化，帮助用户理解交易成本的变化趋势；在智能合约部署方面，通过柱状图展

示智能合约部署数量的变化，评估合并对智能合约活动的影响；在用户行为分析方面，利用饼图和柱状图展示活跃地址和新增地址的数量变化，分析用户活跃度和新用户加入情况。

2. 用户交互设计

用户交互设计是可视化展示中的重要环节，通过良好的交互设计，可以提升用户的使用体验。可视化设计中加入了数据筛选、图表切换和详细信息显示等交互功能。用户可以通过在不同图表间进行切换获取更多维度的信息。用户可以点击图表中的某个数据点查看详细信息，帮助他们更深入地理解数据背后的含义。

（二）DataV 项目数据应用

为了实现上述可视化目标，选择阿里云 DataV 作为可视化工具。DataV 具备强大的数据处理和可视化能力，能够满足对多样化数据展示的需求[6]。

1. 数据处理流程

数据处理是可视化的基础。通过 BigQuery 从 Google BigQuery 公共数据集中提取以太坊区块链数据，并进行清洗和处理。具体流程包括数据提取、数据清洗、数据标准化和数据存储。使用 SQL 查询从 BigQuery 中提取所需数据；然后处理缺失值和异常值，确保数据的完整性和准确性；对交易数量、交易费用、出块时间等数据进行标准化处理，以便在同一尺度上进行比较；最后将处理后的数据存储为 CSV 文件或直接存储在 BigQuery 中，以便后续的可视化使用。

2. 可视化组件选择

根据不同的数据展示需求选择适合的可视化组件。具体选择包括折线图、柱状图、饼图和交互组件。折线图用于展示时间序列数据，如交易数量、交易费用、出块时间和 Gas 价格的变化；柱状图用于展示分类数据，如网络哈希率、智能合约部署数量和新增地址数量；饼图用于展示占比数据，如活跃地址占比；交互组件则包括数据筛选、图表切换和详细信息显示，以提升用户体验和数据探索能力。

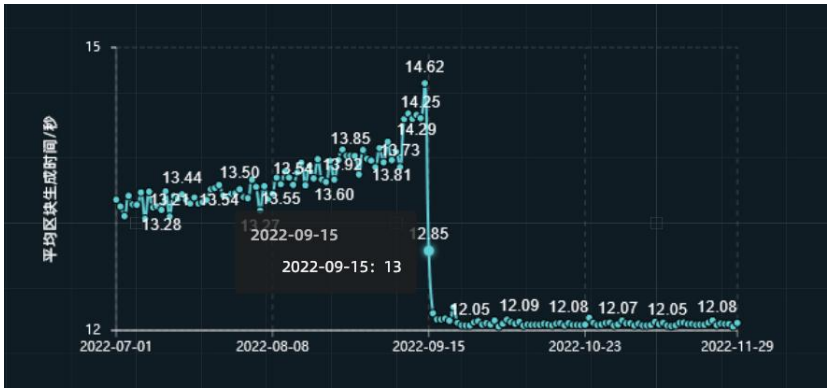
（三）可视化应用搭建

1. 设计方案

根据数据特点和用户需求，设计整体可视化方案。采用仪表盘形式展示不同指标的图表，使用户可以在一个界面中查看全部信息。设计过程中注重图表的布局和配色，确保信息传达的清晰和美观。各个图表之间设置联动关系，使得用户在操作一个图表时，其他图表能够同步更新，提供整体视角。

2. 技术实现

使用 DataV 进行技术实现。具体步骤包括数据导入、组件配置和图表联动。在 DataV 中选择相应的可视化组件，并配置数据源、图表样式和交互功能。将处理好的数据导入 DataV 平台，可以通过 API 连接 BigQuery 或者上传 CSV 文件；选择适合的可视化组件，在 DataV 中进行配置；最后设置图表间的联动关系，使用户在操作一个图表时，其他图表能够同步更新，提供整体视角。



3. 用户体验优化

通过响应式设计、图表说明、性能优化和用户反馈等方式优化用户体验。响应式设计确保仪表盘在不同设备上都有良好的显示效果；图表说明为每个图表添加说明文字，帮助用户理解数据内容；性能优化通过优化数据查询和图表渲染速度，提升用户操作的流畅度；用户反馈收集用户使用反馈，不断迭代和改进可视化设计。通过这些方式成功将以太坊合并大数据转化为直观、易于理解的可视化展示，帮助用户深入理解合并事件的影响，展示了大数据和可视化技术在金融科技领域的强大应用潜力。

五、结论

本研究通过利用 Google BigQuery 提供的公共数据集 crypto_ethereum，结合大数据技术和 DataV 可视化工具，对以太坊合并事件进行了详细的分析与展示。通过研究，我们实现了对以太坊合并前后各项指标的可视化展示，揭示了合并对网络性能、交易行为、Gas 价格、智能合约部署以及用户活跃度的影响。

研究结果表明，以太坊合并从工作量证明转向权益证明机制后，在多个方面产生了显著的变化。交易数量在合并后出现了一定的下降趋势，但整体上保持稳定，这表明网络在过渡过程中依然保持了较好的运行状态。交易费用在合并后明显下降，表明 PoS 机制在减少网络拥堵方面具有积极效果。出块时间的显著减少和稳定性提升，则展示了 PoS 机制在提升区块生成效率方面的优势。

网络哈希率在合并后急剧下降至接近零，这符合 PoS 机制的特点，因为 PoS 不再依赖计算能力来验证交易和生成区块。平均 Gas 价格在合并后呈现下降趋势，显示出合并对交易成本的有效控制。智能合约的部署数量在合并后继续保持活跃，甚至在某些时间点出现高峰，表明网络活动的持续性和用户对以太坊生态系统的信心。

用户行为分析显示，活跃地址数量在合并后保持相对稳定，尽管有轻微下降，但整体变化不大，表明用户对新机制的接受度较高。新增地址数量在合并后有所减少，但网络仍在吸引新用户加入，说明以太坊的吸引力依然强劲。

本研究通过 DataV 可视化工具，将复杂的数据转化为直观易懂的图形和图表，使用户能

够快速理解以太坊合并事件的影响。通过良好的用户交互设计，提升了数据展示的易用性和可探索性，使得非技术背景的决策者和利益相关者也能轻松理解分析结果。

综上所述，本研究展示了大数据技术和可视化工具在金融科技领域的强大应用潜力，为区块链技术的发展和应用提供了宝贵的参考和启示。未来的研究可以继续深入探索大数据技术在区块链领域的更多应用场景，进一步推动技术创新和行业发展。

参考文献：

[1] Ethereum Foundation. The Merge[EB/OL]. [2024-05-26]. https://ethereum.org/en/roadmap/merge/

[2] Google Cloud. Blockchain Analytics Documentation[EB/OL]. [2024-05-21]. https://cloud.google.com/blockchain-analytics/docs/

[3] BigQuery. Ethereum Blockchain Dataset[EB/OL]. [2024-05-21]. https://www.kaggle.com/datasets/bigquery/ethereum-blockchain/

[4] Blockchain ETL. Ethereum ETL[EB/OL]. [2024-05-21]. https://github.com/blockchain-etl/ethereum-etl/

[5] Melnik S, Gubarev A, Long J, et al. Dremel: Interactive Analysis of Web-Scale Datasets[C]. 2010.

[6] 邱忠洋,蒋骏,雷正翠等.基于DataV的气象可视化数据平台设计与实现[J].湖北农业科学,2023,62(1):182-187,195.