图查询语言调研报告

罗胤玻 胡天羽 吴书让

2024 年 1 月 21 日

人员	分工	贡献比
吴书让	PGQL、AQL 的语言学习、特性整理；图查询语言整体总结、相关开源项目调研	33%
罗胤波	Gremlin、GSQL 的语言学习、特性整理；图查询语言相关开源项目调研、编译流程总结	33%
胡天羽	nGQL、openCypher 的语言学习、特性整理；图查询优化相关文献研究、优化技术总结	33%

1 摘要

在本次研究中，我们深入调查了图查询语言的现状和发展趋势。从研读和理解各种图查询语言的文档、编写相关语言示例来分析它们的语义和功能开始；接着基于开源图查询语言编译器的代码阅读，我们总结了图查询语言编译的通用流程和特殊处理机制；最后通过阅读总结相关文献，整理出了图查询的一些常用优化方法。本研究提供了对图查询语言发展情况的全面概述，为未来在图查询语言领域的深入探索和优化提供了坚实的支持。

2 图查询语言语义研究

我们对所有常用的图查询语言都进行了初步调研，并在此基础上选取了文档较全面的六种进行初步学习。主要关注于每种语言的特性以及共同点，同时也留意是否有相关开源代码实现可供进一步研究。为了文档的清晰易懂，我们倾向于只展示每种语言特性最核心的语句部分，而不是完整的代码；理解这些图查询语言可能需要少许 SQL 基础，因为它们大部分都是基于 SQL 的理念增添对图的功能。

2.1 openCypher

2.1.1 简介

openCypher 是一个图形查询语言，主要用于查询和操作图形数据。

它最著名的应用是在 Neo4j，一个高度流行的图形数据库中。openCypher 允许用户通过其图形模型以声明性方式访问数据，这使得它适用于复杂数据关系和模式的高效查询。

2.1.2 特点

- **声明性语言：** openCypher 是一种声明性查询语言，专注于“what”而不是“how”。用户描述他们想要的数据，而不需指定具体过程。

```
MATCH (n:Person) WHERE n.name = 'Alice' RETURN n
```

此查询找到所有名为'Alice' 的'Person' 节点并返回。

- **图形模式匹配：** openCypher 的核心之一是图形模式匹配，用于识别复杂结构和关系。

```
MATCH (n:Person)-[:FRIENDS_WITH]-(friend) RETURN friend
```

查询返回与'Person' 类型节点有'FRIENDS_WITH' 关系的所有节点。

- **灵活的查询能力：** openCypher 支持多种查询，包括创建（Create）、读取（Read）、更新（Update）和删除（Delete）。

```
CREATE (n:Person \{name: 'Bob'\})
```

此命令创建名为'Bob' 的新'Person' 节点。

- **聚合和排序：** openCypher 支持数据聚合和排序，适合数据分析和报告。

```
MATCH (n:Person) RETURN n.age, COUNT(*) ORDER BY n.age
```

按年龄分组并计数'Person' 类型节点。

- **路径查询：** openCypher 可查询图中路径，适合分析网络和关系。

```
MATCH p = (n:Person)-[*]->(m) RETURN p
```

返回从'Person' 类型节点到任何其他节点的所有路径。

2.1.3 缺点

- **学习曲线：** 对于不熟悉图形数据库的用户，openCypher 的学习曲线可能较陡峭。
- **性能问题：** 在处理大规模数据集时，某些查询可能会出现性能问题。
- **标准化：** 作为较新的技术，openCypher 还在持续发展中，标准化程度有待提高。

2.1.4 相关开源项目

openCypher 的主要开源实现有两个：

官方的前端编译模块主要包含了表达式的解析以及语法树的构建，以及用于规范简化语法树的重写模块。项目主要用 Scala/Java 写成，但遗憾的是，上一次主要维护已经是近两年前：

https://github.com/opencypher/front-end

还有由 Microsoft 的开源团队维护的一个 openCypher 的转译器，用于将 openCypher 语句转译为 T-SQL，其中的 Parser 模块提供了从 openCypher 到 AST 的生成。项目主要由 C# 写成，上一次更新是五年前，但相比前者，代码更加精炼易读：

https://github.com/microsoft/openCypherTranspiler/tree/master/src/openCypherParser

2.2 nGQL

2.2.1 简介

Nebula Graph Query Language (nGQL) 是 Nebula Graph 数据库的原生查询语言。Nebula Graph 是一个开源的、分布式、易于扩展的图数据库，专门设计用于处理大规模的图形数据。nGQL 旨在提供一个高效、灵活且用户友好的方式来查询和操作图形数据。

2.2.2 特点

- **图形查询和操作：** nGQL 支持创建、检索、更新和删除图中的节点（Vertices）和边（Edges）。

```
CREATE TAG person(name string, age int);
INSERT VERTEX person(name, age) VALUES "1":("Alice", 30);

CREATE EDGE friend(degree int);
INSERT EDGE friend(degree) VALUES "1"->"2":(3);
```

- **图模式匹配：** 类似于 SQL 和 openCypher，nGQL 允许图模式匹配来探索数据间的关系。

```
MATCH (p:person) WHERE p.name == "Alice" RETURN p;
```

- **索引和全文搜索：** nGQL 支持建立索引以提高查询效率，也支持全文搜索。

```
CREATE TAG INDEX index_person ON person(name(10));

LOOKUP ON person WHERE person.name == "Alice";
```

- **支持复杂查询：** nGQL 支持聚合、排序和分组等复杂查询。

```
MATCH (p:person) RETURN p.name AS Name, COUNT(*)
    AS Friends_Count ORDER BY Friends_Count DESC;
```

2.2.3 与 openCypher 的对比

在语法上，除了一些细微差别，比如符号和函数的选用等，二者间的区别主要在于 openCypher 还支持 DML/DCL 等语言，比如可以创建、合并图数据表；而 nGQL 不支持这些，因此仅能进行查询操作，同时对查询操作的从句（例如 MATCH / WHERE）在功能上也有所限制。

2.2.4 相关开源项目

在 NebulaGraph 的实现中有一个组件为 nGQL 的 Parser，但是相关核心代码及文档有所缺失，难以深入研究：

https://github.com/vesoft-inc/nebula/tree/master/src/parser

2.3 PGQL

2.3.1 简介

PGQL 是一种由 Orcale 开发并维护的图查询语言，并且广泛应用于 Orcale 的图数据库项目上。PGQL 基于 SQL 设计，添加了针对属性图的 CRUD 操作，并且完全兼容 SQL 现有的语法。Orcale 给出了 PGQL 的完整语法定义，并且相关开源 Parser 项目维护十分完善，适合进行深度研究。

2.3.2 特点

- **图类型：**：PGQL 所专注的属性图由点集和边集组成，每个点/边可以有标签（Label）或是属性（Properties），相比 AQL 的关系图, 它更接近于我们一般熟知的图的概念。
- **图查询：**相较于 SQL，PGQL 最大的特点就是加入了两大类针对图结构的查询语句。
  - **图样匹配：**在使用 MATCH 时，可以指定任意路径样式、节点/边类型进行匹配。

    其最基础的语法类似于：`MATCH (a:person)-[e:know]->(b:person)`，这匹配了一段从 Person 类型节点 a，经过一个从 a 发出的 Know 类型的边 e，到达 Person 类型节点 b 的路径。除此之外，还可以继续添加路径，调整边的方向，指定边的个数。

    例如，这是一个稍微复杂一些的图样匹配：

    ```
    MATCH (a:person)-[e1 IS know|friend]2,3->(b)<-[e2:family]->(c)
    ```

    代表着匹配一段“从 Person a 开始，经过 2 或 3 条 Know/Friend 关系边到达 b，且 b 与 c 之间有着任意方向的 Family 类型的关系边”的路径。

    通过 MATCH 语句中的代称来表示实体，被图样匹配的的路径中还可以继续利用 WHERE 等操作筛选，这一部分 PGQL 并为大幅修改 SQL 的语义，也与图查询关系不大，不再赘述。
  - **变长路径查询：** PGQL 的语义实现了还任意指定两点间的变长路径查询，主要包括最短路径/最小花费路径两类，并且可以返回 1/k/all 三种数量的符合条件的路径，这种灵活性使得各种特殊查询语句也可以高效地编写。

    其语法类似于：`MATCH CHEAPEST 3 PATHS (a:Account) (-[e:trans]-> COST e.c)* (a)`

    这匹配了“从 Account a 出发，经过任意条 Trans 类型关系边回到 a 本身，在以边的 c 属性作为代价的情况下”最便宜的 3 条路径。
- **完善的语法定义与 SQL 语法兼容：** PGQL 的文档里有完善明晰的语法描述，方便进行语义分析或是从零制作 Parser。并且兼容绝大部分 SQL 语法，例如子查询、聚合查询、排序查询等，减轻了从 SQL 学习成本。同时他也是完整的 DDL、DCL、DML 语言，可以独自完成图数据库的增删改查（CRUD）操作。

2.3.3 小结

PGQL 实现的功能可以看作是 Cypher 语言的超集，支持的数据库相关操作也是较为完善的。但是 PGQL 并未支持类似 nGQL 索引与全文搜索等功能，在保持语言简洁易用性的同时可能会有性能下降。与 SQL 的高度相似性、完善的语法产生式定义、官方开源 Parser 项目使得他的学习门槛较低，适合进行深入研究。

2.3.4 相关开源项目

Oracle 官方实现了一个从 PGQL 到一种易于理解的 IR 的 Parser，还提供了 PGQL 语法检查、格式优化、自动补全等功能实现。主要使用 Java 编写，整体结构清晰，维护活跃，适合深入研究

https://github.com/oracle/pgql-lang/

2.4 AQL

2.4.1 简介

AQL 是一种专为 ArangoDB 设计的数据库查询语言，语法类似 SQL，但是有不少不同之处。AQL 只支持对数据的读取和修改，但不支持定义和具体操作，因此它只能被归为实现了 DML 语言，不包含 DCL/DDL。由于不是专为图数据库设计，图查询仅仅是 AQL 语言的一小部分功能，我们仅针对涉及图数据的这一部分进行了调研。

2.4.2 特点

- **图类型:**：AQL 所关注的属性图更接进关系数据库的模型，而非我们熟知的图的结构（由节点集合和边集合构成）。更具体地说，AQL 关注的属性图是由数个关系表以及它们之间的一些关系表的组成的。这一点也决定了 AQL 在处理涉及图的拓扑结构相关查询时，性能会劣于用边集/点集组成图的图查询语言。
- **图查询:**：
  - **遍历查询:**：AQL 中的图查询主要是通过遍历（Traversal）的方式实现的。通过指定从一个节点出发的 BFS/DFS 遍历，辅以深度要求/剪枝条件，来获得从起始点到遍历重点的一条路径上的所有节点和边实体。例如：`FOR vertex IN 2..3 OUTBOUND “v/a” GRAPH "g"` 就是在图 g 中枚举从顶点类 v 中的顶点 a 出发，经过 2 或 3 条路径到达的所有顶点集合。除此之外，还支持选定枚举对象（顶点/边/路径），进行筛选剪枝等高级操作，增强遍历查询的自由度与实用性。

    即使 AQL 声称自己是声明类语言，只需要声明数据，不需要关心具体实现。但相比于其他图查询语言中常见的图样匹配，通过遍历、剪枝等来搜索需求的数据明显更需要代码编写者思考有关底层的种种逻辑，且较为反直观，难以高效写出查询语句。
  - **最短路径查询:**：AQL 还提供了两点之间的 1/k/all 最短路径或是可达路径查询，并且可以指定边的属性作为权重。同时由于遍历深度也可以限制，因此也可以做到筛选一定边长范围内的路径，和 PGQL 的功能性相近

2.4.3 小结

AQL 本身只能对数据库中的图部分进行读写，而不能定义，若将研究范围扩张到 ArangoDB 全体，我们还将看到它提供了多种图存储格式（卫星图/企业图/智能图）以最大查询效率。但由于我们专注于图查询语言的语义研究，在此不作展开。

总的来看，相比其他的语言。AQL 只实现了 DML 的部分功能，同时对图的操作和设计也不如 nGQL/PGQL 等直观灵巧：例如 PGQL 可以指定一条线通过 n 段 M 类型的边，再通过 m 段 N 类型的边的路径，而由于 AQL 的遍历只能选中边集合，无法做到如此精细的查询。

同时，高度商业化的产品也加大了开源开发者参与其中的的难度，因此相关可供研究的 AQL 开源编译项目几乎不存在，难以进行下一步研究。

2.5 GSQL

2.5.1 简介

GSQL 是 TigerGraph 使用图查询语言，用于在图数据库中执行复杂的图查询和分析。GSQL 语言风格既有声明式 (declarative) 语言的特点，又有命令式 (imperative) 的特点，可以很方便发起并控制图遍历的过程。

另一方面。GSQL 具有类似于 SQL 的语法，但是它的语义和功能更适用于图数据。通过 GSQL，用户可以定义图上的顶点和边的属性，编写复杂的图算法和查询，并且可以利用 TigerGraph 的并行计算能力来高效地处理大规模的图数据。GSQL 的语法和功能使得用户能够轻松地进行图数据分析和挖掘，从而发现隐藏在数据之间的关联和模式。

2.5.2 特性

根据官方文档的介绍，GSQL 的特点如下：

- **图灵完备性：**GSQL 支持命令式编程，实现了完整的控制流语句，同时 TigerGraph 官方也提供了 GSQL 的开源算法库。
- **类 SQL 语法：**GSQL 的语法与 SQL 存在较大的相似性，有效降低了学习难度。
- **优越的性能：**利用 TigerGraph 的 MPP(Massively Parallel Processing) 功能和 GSQL 语言的累加器 (accumulators) 功能，GSQL 实现了快速的数据查询与获取

以下对上述特性做深入介绍：

图灵完备性：此处实际上是指 GSQL 完整地实现了顺序、分支、循环三种控制流。GSQL 的主要竞争力在于其完善的算法库，算法库包含 7 个部分：中心算法、分类算法、社区算法、连接预测算法、节点嵌入算法、寻路算法、相似性算法。可以看到，该库即有传统的图数据处理方法，也集成了 AI 图分析的重要算法，基本涵盖了现有的对图算法的需求。

类 SQL 语法： GSQL 与 SQL 具有相似的关键字和语法结构，但是 GSQL 支持更多的图数据结构操作。

优越的性能：官方文档中给出的数据表明：在 LDBC 社交网络基准测试（SNB）的 Scale Factor 30k 数据集上测试时，GSQL 对 36TB 的图数据做查询时可以在几分钟以内给出结果。

文档中提到，这一性能的提升得益于 TigerGraph 本身的并行性以及 GSQL 所实现的累加器 (accumulators) 功能，以下介绍 GSQL 的累加器功能以及其是如何实现的。

累加器是 GSQL 中的状态变量，是一种用于收集和聚合结果的数据结构。它们的主要作用是在 GSQL 查询过程中保存和更新中间计算结果。它的状态在查询的整个生命周期中都是可变的。它具有初始值，然后用户可以多次使用它内置的运算符 “+=”，不断地把新值累加到它身上。定义每个累加器变量时都要声明它的类型，这个类型决定了执行 “+=” 运算时会进行怎样的操作。同时，累加器分为全局累加器与局部累加器。全局累加器属于整个查询。在查询中的任何位置，语句都可以更新它的值。局部累加器属于每个顶点。只有在可以访问拥有它的顶点时才能更新它。为了区分它们，在声明时，给它们的名称使用了特殊前缀。

为了显示累加器的性能提升，以下将其与 SQL 中的 GROUP BY 做对比：

累加器在 GSQL 的 Select-from-where 查询块中的 Accum 子句部分进行值的聚合计算。结果表明，这种查询块结构可以表示所有常规 SQL 的 GROUP BY 聚合。

对于 SQL 的 5 个内置聚合函数 (MIN / MAX / SUM / COUNT / AVG)，GSQL 对应了 5 个不同的累加器类型 (MINAccum / MAXAccum / SUMAccum / SUMAccum / AVGAccum)。在接下来的例子中可以看到，累加器通过实现对聚合的精细化操作来进行优化：

假设分别对 GROUPING SET(K1)、(K2)、(K3) 进行求和、最小值和平均值聚合计算，那么 GROUPING SET 语义将强制计算每个分组集合的所有三个聚合 (其中两个是不需要的)。在 SQL 形式中，它表示为 **Accum A1 += (k1 → a1, a2, a3), A2 += (k2 → a1, a2, a3),A3 += (k3 → a1, a2,a3)**

当用户希望将特定的聚合器绑定到不同的分组集时，这将导致浪费计算。而在 GSQL 中表示为：**Accum A1 +=(k1 → a1), A2 += (k2 → a2), A3 += (k3 → a3)**

很明显，聚合器的精细绑定避免了对 k1 进行 a1、a3 计算，对 k2 进行 a1、a3 计算，以及对 k3 进行 a1、a2 计算。

2.5.3 小结

从上面的内容可以看出 GSQL 实际上也在一定程度上代表了图查询语言的一种发展方向，即：一方面拓宽图查询语言的功能，支持更多 AI 相关算法，使之更适应如今 AI 的发展；另一方面借助本身数据库的相关特性和语言的设计对之前的数据库语言做优化。

GSQL 的相关项目较少，主要是由于 TigerGraph 未开源核心代码，仅开源了算法库。

2.6 Gremlin

2.6.1 简介

Gremlin 语言是图数据库最主流的查询语言，是 Apache TinkerPop 框架下规范的图语言。研究 Gremlin 就离不开 Apache TinkerPop。

TinkerPop 是 Apache 软件基金会旗下的一个顶级开源项目，是一种开源图计算框架，既可用于联机事务处理（OLTP），又可用于联机分析处理系统（OLAP），适用于处理单一机器以及分布式环境的庞大数据。

TinkerPop 旨在为图数据库建立行业标准，使用 Gremlin 作为标准查询语言。从这一点出发，Apache TinkerPop 3.x.x 框架实现了兼容绝大多数图数据库产品的功能。

基于此，TinkerPop 实际上定义了一套框架，有 graph structure、graph provider、graph process 等部分组成。graph structure 定义了图的结构和数据表示，它是一个基于标签 (label) 的属性图 (property graph)，这种图由顶点（vertex）和边（edge）组成，每个顶点和边有且只有一个标签，同时顶点和边可以有 0 个或多个属性（key/value 值对）。但 TinkerPop 并没有提供 graph structure 的具体实现，图的顶点和边等结构具体如何存储和访问由 graph provider，比如 janusgraph、hugegraph 等具体实现。graph structure 可以说是定义了 TinkerPop 与 graph provider 的接口。graph process 是对 graph 的数据的处理，即图遍历（traversal），相当于是 TinkerPop 与用户之间的接口。Gremlin 则是一个图遍历语言（graph traversal language），用户使用 Gremlin 来对图进行遍历分析。

Gremlin 既可以写成声明式也可以写成命令式，可以兼顾非专业人员的学习曲线和专业人士对功能的要求。

2.6.2 特性

由于 Gremlin 是作为 Apache TinkerPop 框架中的重要一环存在的，其特点往往与框架有紧密的关系，在功能性、兼容性上面都有一些独到之处。

- **兼容性：** Gremlin 不存在查询语言和编码语言的分割，因为遍历可以用任何支持函数组合和嵌套的编程语言编写（每个主要编程语言都支持）。用户的 Gremlin 遍历就与应用程序代码一起编写，并受益于宿主语言及其工具（例如类型检查、语法突出显示、点完成等）提供的优势。存在各种 Gremlin 语言变体，包括：Gremlin-Java、Gremlin-Groovy、Gremlin-Python、Gremlin-Scala 等。同时也有开源插件使得 Gremlin 支持了 Cypher。
- **流式语言：** Gremlin 也是一种流式语言，以适应现在许多流式编程的高级语言，使得两者之间的转换也变的非常方便。无需像 SQL 一样，编程人员还需要熟悉一种与编程语言差异极大的语言，Gremlin 中甚至包含了所有高级编程语言中都会有的 if、if...else...、for、do...while、while 等流程控制语句。
- **功能丰富：** Gremlin 在框架内具有丰富的功能，包括对多种图数据库、图数据结构、复杂查询方法的支持。

2.6.3 小结

Gremlin 与其说是一种图查询语言，更像是一个框架，其设计理念充斥着标准化、可拓展性的需求，支持多语言、多数据库、多数据结构等特点都是其核心竞争力。另一方面，开源社区对 Gremlin 的开发是以增强其扩展性为主，使得其支持前端显示、更多的语言或数据库，其兼容性不断增强。

2.7 其他

我们还对其他一些图查询语言进行了调研，包括 GraphQL、LIKQ、GCore、SPARQL 等。然而，由于这些语言的文档相对不清晰、开源项目较为有限、以及利用率相对较低等原因，我们并未进行深入研究。

2.8 GQL 标准

与 SQL 相似，图查询语言将逐渐从百花齐放的状态迈向标准化。GQL 标准委员会致力于通过将各种繁杂的 GQL 语言规范以 ISO 标准的形式进行整合，从而实现标准化。目前，该标准涵盖了三个主要方面的约束：通用 SQL 相关定义、图样匹配以及一系列 GQL 特殊功能，例如图数据格式和图操作。ISO 标准秘书处已将 GQL 标准的最终草案日程安排上议程，预计于 2024 年 1 月 26 日开始进行草案表决。一旦通过，这一 GQL 标准将为不同语言提供统一的指导，推动图查询语言的规范化发展。

2.9 总结

经过调研，我们将各类图查询语言的异同及功能、语言差异情况汇总在了下表：

语言	语法	图类型	图样匹配	路径查询	CRUD	DDL/DCL	开源项目
nGQL	类 SQL	点/边	有	有	有	有	有
openCypher	类 SQL	点/边	有	有	有	有	有
PGQL	SQL 拓展	点/边	有	有	有	有	有
AQL	类 SQL	实体表/关系	手动遍历	有	仅查询	无	无
Gremlin	流式语言	点/边	有	有	有	有	有
GSQL	类 SQL	点/边	有	有	有	有	无

除此之外，所有语言都是声明式的语言，这也是查询语言的共性：只需描述需求何种结果，而不需要考虑如何执行。

可以观察到，受 SQL 语言的主导地位影响，大多数图查询语言在语法设计上选择了类似 SQL 或直接对 SQL 进行拓展的方式，以降低用户的学习成本。这种取向使得熟悉 SQL 的用户更容易掌握这些图查询语言，因为它们在语法和结构上具有相似性。特别值得一提的是，PGQL 甚至达到了直接兼容所有 SQL 语法的水平，进一步简化了用户在图查询领域的过渡和应用。

在功能方面，大多数语言都实现了图样匹配和路径查询，这两种功能是图查询最主要、最直观的方式。通常的实现方式是自动生成遍历语句，然后在全图范围内查找用户指定的图样；然而，不支持图样匹配的 AQL 语言选择将遍历查询和编写剪枝筛选条件的权力交给了用户。这种做法提高了用户查询的复杂性，但由于遍历操作的功能不完善，未能提供更高的自由度，反而降低了语言的可用性。

除了支持查询功能，许多语言还加入了增删改操作的支持，包括 DDL 和 DCL 的子集，使其能够直接执行数据库的定义和管理等操作。这种扩展使这些语言更趋近于完全的数据库操作语言。然而，也存在一些语言仅限于实现只读查询功能，无法对图数据本身进行任何修改。这种只读查询的限制在一定程度上影响了语言的灵活性和实用性。虽然这类语言在图数据的分析和探索方面具有一定价值，但其无法直接进行数据的更新、插入或删除等操作，限制了用户对图数据库的整体管理能力。在实际应用中，对数据的动态修改在需要及时响应变化、更新关系或进行实时数据维护的场景下往往是至关重要的。

在开源方面，我们发现有不少图查询语言以及对应的图数据库以商业化为主，并无可供学习的开源实现；而较为热门的几种图查询语言，或许是因为图查询语言的编译仍属于小众领域，除了官方的开源项目之外，很少见个人开发的编译器项目等具体实现；因此，我们只能对仅有的几个官方相关项目进行下一步研究。

总之，目前的各种图查询语言功能支持较为相近，语法和图类型上的选择也较为一致，只是在具体实现和语法细节上有不同之处。所以构建一个统一的 GQL 标准来规范图查询的核心功能，是十分合理，且有利于社区共同发展的。

3 图查询语言的编译流程

3.1 一般流程

在上面的调研中，不同的语言有不同的特性。相应的，它们在编译的流程上也存在细微的差异，以下内容深入到各大语言的代码中去对比它们的不同 (GSQL 未开源，未作讨论)。

图数据库之于图查询语言类似于机器之于高级语言。编译高级语言时需要根据机器的不同选择合适的中间代码，进而编译为可用的机器码。与之相似的图查询语言的编译也需要考虑图数据库的问题。

一般而言，有如下的步骤：

- 解析：首先，查询字符串会被解析器解析为一个 AST，这个过程包括词法分析和语法分析。词法分析将查询字符串分解成一个个 token，语法分析将这些标记组织成语法结构。
- 语义分析：在语义分析阶段，编译器会对 AST 进行静态语义检查。它会验证表达式的语法正确性、类型正确性以及标识符的存在性等。此外，还会检查查询中使用的实体、属性和关系是否在数据库中存在。
- 查询优化：在查询优化阶段，编译器会尝试对查询进行优化，以提高查询性能。这通常包括重写查询计划、选择合适的索引、确定连接顺序等操作。优化步骤的目标是生成一个执行效率较高的查询计划。

- 查询计划生成：在这一阶段，编译器根据优化后的查询计划生成可执行的查询计划。查询计划是一个描述如何从数据库中获取所需数据的执行计划，大多数时候，这一步由图数据库来完成。
- 执行查询：最后，生成的查询计划会被传递给图数据库引擎进行执行。引擎根据计划中的操作和顺序，从图数据库中检索数据并计算结果。执行过程可能涉及到遍历图结构、应用过滤条件、聚合数据等操作，最终返回查询结果。

由此可见，不同语言的编译流程处理重点区别在查询计划生成上，这类似于编程语言编译时的中间代码生成，可能不同的图数据库需要不同的查询计划格式。

3.2 细节差异

值得提出的一点是，图查询语言到图数据库与高级语言到机器最大的区别是查询语言需要实现对查询语句的解析，这是不同于分支、循环这样的控制流的地方，在之后的内容里，我们也将举例介绍。

- **openCypher** openCypher 官方对 Cypher 的语法做了进一步规范，通过 antlr 进行解析与语义检查，进而生成正确的 Cypher 查询命令；但是微软的 **openCypher Transpiler 项目**使用了一个新的结构：首先，基于 ANTLR4 构建的 openCypher 解析器和官方的 openCypher 语法，用于解析并创建一个 AST 来抽象图查询的语法结构; 然后，使用 Logical Planner 将 AST 转换为类似于关系代数的关系查询逻辑规划; 查询代码呈现器从逻辑计划生成实际的查询代码。可以看到，该项目在官方基础上对 openCypher 更进一步，在查询指令提交给图数据库前使用 Logical Planner 进行提前的优化。
- **nGQL** nGQL 开源了图查询语言以及图数据库代码。首先通过 Yacc 对代码作解析与检查，然后将查询请求提交至图数据库，经过 optimizer 和 planner 两个模块后，生成查询计划进而得到结果。
- **PGQL** PGQL 对属性图的查询做了一系列规范的语法定义，同时开发了从 PGQL 到一种易读的 IR 的编译器 pgql-lang。该编译器的结构和在课程中的一般程序语言编译器并无太大差异，依然是语言-语法树-IR 的流程。相比其他图查询语言的编译器，它将关系代数的转化和优化等部分省略，放在下游组件中实现，极大地简化了编译器结构。
- **AQL** AQL 直接内置于 ArangoDB 数据库中，首先借助 Yacc 对代码做检查，然后向图数据库发送请求，在图数据库内进行优化与计划生成。
- **Gremlin** Gremlin 本身支持多种语言，但实际上实现的方式是由这些语言实现了一套 Gremlin 的 API，这一套 API 直接接在 Apache TinkerPop 框架上，由 ThinkerPop 框架实现 API 与图数据库的交互。

3.3 小结

总体而言，不同图查询语言的编译流程在细节上存在一些差异，但在一般的编译流程中都包括解析、语义分析、查询优化、查询计划生成和执行查询等阶段。这些语言通过使用不同的解析器、语义分析器和优化器，以及针对各自特性的查询计划生成方式，实现了对图数据库的查询操作。

在细节上的差异中，openCypherTranspiler 项目通过引入 Logical Planner 对查询进行进一步优化，而 PGQL 则将优化的部分推迟到下游组件中实现，简化了编译器结构。nGQL 和 AQL 则是将查询请求提交至图数据库，在图数据库内进行优化与计划生成。这些细节上的差异反映了各个图查询语言通过对于编译流程的不同设计选择，以适应不同的使用场景和需求。

4 图查询优化

4.1 概述

相比于编译到关系代数时进行优化，图查询语言更倾向于对生成的关系代数图进行优化。同时，图查询语言和一般编程语言的差异性（譬如：图查询语言的主体是一条条顺序执行的查询语句，没有任何的循环、选择结构）导致了图查询语言的优化这一部分涉及的知识更偏向于图查询处理以及数据库这一特定专业领域，而和编程语言的通用编译优化知识（例如进行活跃变量分析、死代码删除、循环外提等）相关性不大。因此我们没有选择进行编译优化的尝试，而是主要去对图查询优化相关技术进行了调研。

图数据库的查询优化是提高查询性能和响应速度的关键。这个过程包括高效的查询设计、合理使用索引、充分利用图数据库能力等多个方面，我们详细调研了多种技术，最终选取了数据分布以及查询分解、增量处理这三个技术进行研究。

4.2 主要优化策略

4.2.1 数据分布

在进行图数据库优化技术的研究中，动态图分区方法是其中最为普遍的一种，其目的在于优化查询响应时间并降低数据存储成本。由于图数据库广泛应用于实时数据摄取和查询，因此其模式设计通常具有动态性。大多数图数据库查询涉及遍历边以获取邻居节点的信息，因此在将大型图分区为大小或权重相等的部分进行分布式存储时，需要同时最小化切割边的数量，这一问题被证明是一个 NP-Hard 问题。

综合而言，图分区技术面临的挑战在于寻找减少服务器间通信成本（主要由边切割数定义）与减轻服务器处理负载之间的平衡。尽管与编译技术的关联较小，但图分区技术却是支撑大型图数据库的核心技术，依然值得我们探讨细节。

- **基于社区结构的分区：**大多数方法依赖于社区结构对图节点进行分组，以降低边切割负载。例如，文献 [1] 提出的方法要求同一分区内的所有邻居必须在一个顶点的范围内，这降低了边切割负载，但同时增加了数据更新负载。
- **公平性要求和访问模式：**文献 [2] 引入了“公平性要求”，通过设定阈值确保每个顶点至少有一定比例的邻居在同一分区内。此外，顶点的访问模式（读取频率）被用于规定每个分区的更新策略。
- **自然社区结构和标签传播：**文献 [3, 4] 的方法利用了自然社区结构进行动态图分区。此外，还有应用于图分区的多级标签传播（MLP）方法。
- **轻量级重新分区算法：**近期研究 [5] 提出了一种动态分区算法，以解决增量分区的负载问题，并保持分区间的负载平衡。
- **考虑硬件特性的工作量预测：**与仅考虑分区间边切割数的方法不同，[6] 中的方法在预测工作量时还考虑了硬件特性，例如数据传输量、网络异构性和硬件架构提供的服务。

这些方法通过数据复制和增量分区来减轻分区负载、最小化边切割数，并持续维持分区间的负载平衡，从而提高整体查询响应时间。这些研究表明，有效的图分区方法可以显著提升大规模图数据库的性能和效率。

4.2.2 查询分解和增量处理

查询分解和增量处理是针对图模式匹配的关键优化策略，旨在降低大型图数据库中复杂查询的执行成本，减少不必要的重新编译，以提高系统的响应时间和性能。通过有效地将查询任务分解为更小的子任务并仅处理必要的增量变化，系统能够更高效地处理图数据，从而优化查询的执行效率。这种优化不仅提升了系统的整体性能，还增强了其在大规模和动态变化的图数据环境中的可扩展性。

增量处理：

- 旨在仅处理图数据中发生变化的部分，而不是重新执行整个查询过程。
- 查询跟踪图数据库中的变更，并只计算与变更相关的部分。
- 在需要频繁查询的动态环境，可以显著提高查询的响应时间和性能。

查询分解：

- 将复杂的查询任务拆分为更小、更简单的子任务，每个子任务负责处理一部分数据或特定的图模式匹配。
- 然后组装子结果以形成最终结果，并在合并子结果时避免昂贵的连接操作。
- 可以大幅减少搜索的复杂性，优化系统性能

具体实现与研究:

- [**7**]：关注了图模式匹配的增量算法，综合考虑了各种算法的实现难度、性能开销以及对系统整体稳定性的影响。
- [**8**]：为大型静态图实现了基于分解的算法，优化内存中的图探索，通过避免使用图索引来最小化服务器负载。
- [**9**]：提出了一个结合分解和增量处理的综合解决方案，特别有效于动态图中的连续查询。这种方法因其在频繁更新环境中索引代价高昂的高效性而受到关注。

查询分解和增量处理的结合为处理复杂查询提供了一种有效的方法，特别是在动态图数据库中。这种方法在效率和计算负载之间取得了平衡，非常适用于那些图数据更新频繁的场景。无论是在静态还是高度动态的环境中，都凸显了在图数据库中定制查询优化方法的重要性。

4.3 一个图查询优化例子——nGQL 的 optimizer

之前的内容中曾提到，nGQL 开源了代码的 optimizer 部分。阅读其代码可知，对于任何给定的查询，执行计划并不是唯一确定的，optimizer 的作用是在最简单、直接的计划基础之进行性能上的优化。

NebulaGraph 目前的优化器是完全的基于规则的优化 RBO，这些预设的规则的代码都在 src/-graph/optimizer 的 rules 里，它们都是针对执行计划模式的修改规则。下面以几个例子来简要说明 nGQL 是如何进行优化的。

- **边查询转换规则** (GetEdgesTransformRule)：在没有点、边索引的情况下，无起点 VID/属性条件的查询是可以通过扫点、边数据的。比如 MATCH ()-[e]->() RETURN e 扫描边的查询因为只需要返回边，所以直接扫描边是更高效的。

- **计算下推系列规则**：计算下推是指在计算存储分离的数据库系统中，涉及到读取数据的算子需要从存储层远程捞取数据再做进一步处理。而数据传输常常成为性能的瓶颈。如果下一步计算要做的事情是对数据的按条件剪枝，例如 Filter、Limit、TopN 等等。这时候，如果存储层获取数据时考虑了条件剪枝情况，则可以大大地减少数据传输量。比如说 **边扫描计算下推优化规则** (PushLimitDownScanEdgesRule)：当 ScanEdges 算子的下游是 Limit 算子的时候，把 Limit 的过滤条件嵌入到 ScanEdges 之中。

5 总结

我们主要着眼于图查询语言的现状和发展趋势，以全面而多维的方式展开了研究。首先，通过详细学习各种图查询语言的文档并尝试编写语言示例，我们理解了不同图查询语言的语法、风格和功能。这一步骤不仅考察了每种语言的独特特性，还总结了它们之间的共同之处。

接下来，我们将研究焦点转向了现有开源的图查询语言编译器，研究了在多个编译器中共同存在的图查询语言编译的一般流程。在这个过程中，我们也着重考察了某些编译器中的特殊处理机制，以及它们与一般程序语言编译的不同技术和策略。通过深入讨论这些方面，我们更全面地理解了图查询语言编译这一小众领域的独特之美。

最后，通过对相关文献的深入阅读，我们总结了图查询语言中常见的几种理论优化方法以及实际代码中出现的规则优化方案。这一阶段的研究为我们提供了在图查询语言中实施优化的理论基础，并使我们能够更好地理解在实际应用中如何提高性能和效率。

这一系列层层递进的研究，为我们提供了深入了解图查询语言的坚实基础，也为未来进一步研究和优化图数据库查询语言提供了有力支持。当前，图查询语言的领域正处于规范化的初期阶段，相关的开源编译项目也较为有限。希望本次调研的结果能够为读者提供对图查询语言全面的认识，同时激励各位在这一小众领域展开新的尝试、探索和学习，为图查询语言未来的发展注入新的活力。

参考文献

[1] J. M. Pujol , V. Erramilli , G. Siganos, X. Yang, N. Laoutaris , P. Chhabra and P. Rodriguez, "The little engine(s) that could: scaling online social networks," SIGCOMM Comput. Commun. Rev., pp. 375–386, Vol 40. No. 4 . 2010.

[2] J. Mondal and A. Deshpande, "Managing large dynamic graphs efficiently," SIGMOD Conference , pp. 145-156, 2012.

[3] L. Vaquero, . F. Cuadrado , D. Logothetis and C. Martella, "Adaptive Partitioning of Large-Scale Dynamic Graphs," Proceedings of the 4th annual Symposium on Cloud Computing (SOCC '13), 2013.

[4] L. Wang, Y. Xiao, B. Shao and H. Wang, "How to partition a billion-node graph," ICDE , pp. 568- 579, 2014.

[5] D. Nicoara , S. Kamali , K. Daudjee and L. Chen, "Hermes: Dynamic Partitioning for Distributed Social Network Graph Databases," EDBT , pp. 25-36, 2015.

[6] A. Zheng, A. Labrinidis, P. . H. Pisciuneri, P. K. Chrysanthis and P. Givi, "PARAGON: Parallel Architecture-Aware Graph Partition Refinement Algorithm," in EDBT 2016: 365-376, 2016.

[7] S. Yang, X. Yan, B. Zong and A. Khan, "Towards effective partition management for large graphs," SIGMOD Conference , pp. 517-528, 2012.

[8] F. Wenfei , L. Jianzhong, L. Jizhou , T. Zijing , W. Xin and W. Yinghui, "Incremental Graph Pattern Matching," SIGMOD, pp. 925- 936, 2011.

[9] S. Choudhury, L. B. Holder, . G. J. Chin , P. Mackey, . K. Agarwal and J. Feo, "Query Optimization for Dynamic Graphs," CoRR abs/1407.3745, 2014.

[10] Z. Sun, H. Wang, H. Wang, B. Shao and J. Li, "Efficient Subgraph Matching on Billion Node Graphs," VLDB Endowment, pp. 788-799, Volume 5 Issue 9, May 2012.

[11] K. J. Ahn, S. Guha and A. McGregor, "Graph Sketches: Sparsification, Spanners, and subgraphs," PODS, pp. 5-14, 2012.

[12] P. Zhao, C. C. Aggarwal and M. Wang, "gSketch: On Query Estimation in Graph Streams," VLDB Endowment, pp. 193-204, Volume 5 Issue 3, November 2011.

[13] J. Mondal and . A. Deshpande, "EAGr: Supporting Continuous Ego-centric Aggregate Queries over Large Dynamic Graphs," in international conference on Management of Data, 2014.

[14] A. Silberstein, J. Terrace, B. . F. Cooper and R. Ramakrishnan, "Feeding Frenzy: Selectively Materializing Users' Event Feeds," SIGMOD, 2010.

[15] Orcale Teams, https://github.com/oracle/pgql-lang/, Github, accessed Jan-2024

[16] GSQL Language Reference :: GSQL Language Reference, https://docs.tigergraph.com/gsql-ref/current/intro/

[17] Apache TinkerPop: Gremlin, https://tinkerpop.apache.org/gremlin.html