TigerGraph

原生并行图

用于实时深度关联分析的

新一代图数据库

TigerGraph

原生并行图

用于实时深度关联分析的新一代图数据库

Dr. Yu Xu

Dr. Victor Lee

Dr. Mingxi Wu

Gaurav Deshpande

Dr. Alin Deutsch

第 1 章

新型的图数据库

图数据库是所有数据管理系统中增长最快的一类。由于很早就被 Twitter、Facebook 和 Google 等公司采用，图已成为当今各行各业的企业所使用的主流技术。通过以图格式组织数据，图数据库克服了其他数据库无法克服的巨大、复杂的数据挑战。与传统的关系型数据库管理系统和近年来新兴的 NoSQL 大数据产品相比，图有着明显优势。

图数据库是数据管理软件。构成要素是点和边（两个点之间的关联）。社交关系图是图的一个简单示例。具体而言，关系型数据库也属于数据管理软件，其构成要素是表。它们都需要将数据加载到软件中，并使用查询语言或 API 才能访问数据。

关系型数据库兴起于 20 世纪 80 年代。许多商业公司（如 Oracle、Ingres、IBM）均支持数据管理的关系模型（以表为基础）。在当时，主要的数据管理需求是生成报表。

然而现如今，数据管理需求已发生改变。数据量爆炸性增长、数据分析越来越复杂、模式更改越来越频繁，再加上实时查询响应时间和更智能的数据推送要求，这种种因素使人们认识到了图模型的优势。

商业软件公司多年来一直支持图模式，例如 TigerGraph、Neo4j 和 DataStax 的 DSE Graph。这项技术在许多领域带来了颠覆性变化，例如供应链管理、电子商务推送、网络安全、欺诈检测以及先进数据分析中的许多领域。

主要优势

与关系型数据库和 NoSQL 数据库相比，定义良好的图数据库都有着明显优势。在以下章节中，我们将介绍各种图数据库之间的差异。

更好、更快速的查询和分析

无论数据大小，图数据库均可出色查询相关数据。图模型提供了内置的索引数据结构，它无需针对给定查询而加载或接触无关数据。图模式的高效率使其成为出色的解决方案，能够更好更快地进行实时大数据分析查询。这与 NoSQL 数据库（如 MongoDB、HBase/HDFS (Hadoop) 和 Cassandra）相反，NoSQL 数据库的架构针对数据湖、顺序扫描和新数据附加（不随机查找）而构建。在此类系统中，每次查询都会涉及大部分的数据文件，关系型数据库则更加糟糕，处理全量数据很慢，并且需要给表建立连接才能呈现表间的关系。

更简化、更自然的数据建模

任何学习过关系型数据库建模的人都了解，满足数据库标准化和参照完整性需要遵循严格规则。某些 NoSQL 架构走向了另一极端，将所有类型的数据放在一个大型表中。而在图数据库中，您可以定义所需的任意点类型，并定义任意边类型来表示点之间的关系。图模型则是按需定义，没有冗余的标准化和浪费。

由点及面地挖掘知识体系

知识图当然也是图。知识图是以 < 主语 >< 谓语 >< 宾语 >（如 <Pat>< 有 >< 船 >）的形式表示的各个事实的集合。以 < 主语 > 和 < 宾语 > 为点，< 谓语 > 为点之间的边，这构成了图。知识图的实际功能是查看整体：遵循一系列边、分析邻域或分析整个图。这样，您可以推断或推理出新的关系。

面向对象的思维

在图中，每个点和边都是自包含对象实例。在基于模式的图数据库中，用户定义点类型和边类型，就像对象类一样。此外，将点关联至其他点的边有点类似于对象方法，因为边说明点可以“做”什么。要处理图中的数据，需要“遍历”边，在概念上是指从一个点遍历到相邻点，保持数据的完整性。比较而言，在关系型数据库中，要关联两个记录，必须将它们相连并创建新的数据记录类型。

更强大的问题解决能力

图数据库能够解决对于关系查询不切实际和切合实际的问题。例如迭代算法（如 PageRank、梯度下降）以及其他数据挖掘和机器学习算法。某些图查询语言是图灵完备的，这意味着您可以使用它编写任何算法。不过在市场中有许多表达能力有限的查询语言。

迅速的实时扩展

如上所述，图数据库提供更出色的建模功能，从而能更好地为互联数据提供问题解决能力。但图数据库一直以来有一项劣势，那就是可扩展性不佳：以往的图数据库无法加载或存储超大数据集、无法实时处理查询，并且 / 或者无法遍历查询中两个以上的连续关联（两步以上）。

不过，现在有了新一代图数据库。基于原生并行图架构，此第三代图数据库拥有出色的速度和可扩展性，可提供以下优势：

并发查询和实时数据更新

许多以前的图系统无法实时获取新数据，因为它们构建在以牺牲写入性能来换取良好读取性能的 NoSQL 数据存储之上。但许多应用都需要实时更新，例如欺诈检测、个性化实时推荐以及任何交易或流数据应用。原生并行图可实时处理读取和写入。并行通常与并发控制相结合，以便为读取查询和图更新提供很高的每秒查询量。

深度关联分析

原生并行图设计提供的最重要优势之一就是能够在超大图上实时处理遍历多步（10 步以上）的查询。原生图存储（加快每次数据访问速度）和并行处理（同时处理多个操作）相结合，可将许多用例从不可能变为可能。

动态模式更改

原则上，图模型允许您通过定义新的点类型和边类型来描述新的数据类型和新的关系类型。或者您可能需要添加或减少属性。您可以关联多个数据集，只需加载数据集并添加一些新边即可关联。具体难易程度根据实际情况有所不同。原生并行图是彻彻底底的图，在设计时考虑了图模式发展，因此可动态处理模式更改，即在图处于使用状态时进行处理。

简单的多维度数据表示

假设您想要向实体添加地理位置属性，或者想要记录时间序列数据。您当然可以使用关系型数据库来完成。但通过图数据库，您可以选择将位置和时间视为点类型和属性。或者使用带有权重的边来明确关联在空间或时间上彼此接近的实体。可以创建一系列边以表示因果变化。与关系模型不同，无需创建多维数据集来表示多个维度。每个新点类型和边类型均表示潜在的新维度；实际边表示实际关系。让多维度表现关系成为无限可能。

高级聚合及分析

除了传统的按组划分聚合之外，原生并行图数据库还可以执行更复杂的聚合，这些聚合在关系型数据库中是不可想象或不切实际的。由于采用表模型，关系型数据库上的聚合查询受到数据分组方式的极大限制。与此相反，图的多维度性质和新型图数据库的并行处理功能可让用户高效地分割、切分、汇总、转换甚至更新数据，而无需预处理数据或使数据进入强模式。

增强的机器学习和人工智能

我们已经说过，图可以用做知识图，使人能进一步推理间接事实和知识。此外，任何图都是优化机器学习的强大武器。首先，对于无监督学习，图模型可出色检测集群和异常，因为您只需关注关联。监督学习始终需要更多、更好的训练数据，而图能够提供先前被忽视的特性，是出色的提供源。例如，实体出度或两实体之间的最短路径等简单特性可提供缺失部分以提高预测准确性。如果机器学习 / 人工智能应用需要实时决策，我们需要快速的图数据库。图数据库成功用作实时人工智能数据基础架构的关键在于：

- 支持在最新数据流入时进行实时更新
- 表达能力高且用户友好的声明性查询语言为数据科学家提供完全的数据控制能力
- 支持实时（亚秒）深度关联遍历（> 3 步），就像人脑神经元通过神经网络发送信息一样
- 横向扩展和纵向扩展以管理大型图

原生图数据库具有诸多优势，它可管理传统关系型数据库无法处理的大数据。不过，与替代旧技术的任何新技术一样，图数据库的应用仍存在障碍。这包括图数据库查询语言的非标准化。有一些不成熟的产品导致性能和实用性低于标准，减慢了图模型的应用速度。

图数据库和关系型数据库比较

存储模型

关系型数据库管理系统在表中存储信息（行 / 列）。通常，每种实体都有一个表（如学生、教师、课程、学期、注册）。在实际应用中，关系型数据库管理系统会有几十个甚至几百个表。

在图数据库中，所有信息存储为点和边（点之间的关联）的集合，而非存储在物理分隔的表格中。例如，用户可以表示为点，此点直接与若干其他点关联，包括：用户的电话号码、家庭住址、购买的产品、好友等。每个点和每条边均可存储一系列属性。

查询模型

关系型数据库管理系统的基础计算模型基于扫描行、连接行、过滤行和聚合。

图数据库的计算模型基于的是，从一系列初始点开始，通过多步来遍历图形。每一步从当前点集开始，遵循选定的关联（边）子集到达相邻点。查询详细信息将指定在何处开始、选择的边和目标点、走多少步以及在遍历期间对观察的数据执行何种操作。

分析类型

关系型数据库适于会计，涉及一两个（最多三个）表格的简单数据查找以及描述性统计。不过，它不太适于更具探索性或预测性的分析。例如，很难或无法编写 SQL 查询来回答以下问题：“这三个用户如何关联？”、“最终有多少钱从帐户 A 流入其他 3 个帐户？”、“点 A 和点 B 之间的最短 / 最廉价路线是什么？”或“如果网络中的组件 C 故障，会带来什么下游后果？”。而在图数据库系统中，所有上述类型的分析都能得到自然、高效地表达和解决。

实时查询性能

在关系型数据库中，每个表采取物理分隔方式进行存储，因此，建立两个表之间的关联相对较慢。此外，如果数据库设计者没有为外键建立索引，则建立关联将极慢。在图数据库中，关于点的一切已关联至点，因此查询性能可大大提高。

转换到图数据库

借助 Tigergraph 原生并行图数据库，可以极为轻松地设置基本图模式并向其加载数据。如果已具有符合标准的关系模式，可以轻松地将其转换为可操作的图模式。毕竟，您的实体关系模型是个图，对吗？最具挑战性的步骤是学着从图的角度（基于点之间的遍历）来思考问题如何解决（查询），而不是从关系角度来思考（基于连接表、投影和分组以提供输出表）。对于许多应用，以图的方式来思考实际上更加自然和直观，当所有信息已在图中关联时尤其如此。

TigerGraph 提供的 GraphStudio UI 是完整的端到端 SDK，此外还提供高级语言 GSQL，以便尽可能顺畅地在 TigerGraph 平台之上开发解决方案，并使步骤与 SQL 系统紧密兼容。您可以遵循以下三步流程：

1. 创建图模型（与 SQL 的“创建表”非常类似）
2. 将初始数据加载到图中（如果为表数据，则使用简单的声明语句）
3. 使用 GSQL 编写图解决方案。GSQL 的语法和 SQL 相仿并通过高级程序语言和 MapReduce 的概念增强。

关系型数据库管理系统领域的知识和技能可以应用到 TigerGraph 系统，在前两步尤其如此。TigerGraph 拥有丰富的文档和培训材料，其中提供各个垂直行业和用例的大量示例来说明这三个步骤，适用案例包括协作过滤推荐、基于内容的推荐、图算法（如 PageRank）和欺诈调查等。您会发现，从关系模式直接转换到图模式不总是最佳选择。

此外，在 TigerGraph 系统中，图还是分布式计算模型：可以将计算函数与点和边关联。在逻辑上，点和边不再只是静态数据块;它们变为活动计算元素,与我们大脑中的神经元类似。点可通过边向其他点发送消息。学会“像图一样思考”后，您将开启一个充满无限可能的新世界。

第 2 章

图数据库的市场格局

图数据库的市场格局

> ***随着图数据库的普及，许多新公司不断出现，形成了包含各种工具和技术的市场格局。我们从类别和领先解决方案切入，深入了解一下图数据库的市场格局。***

咨询公司 DB—Engines.com 的调查表明，图数据库是所有数据管理系统中增长最快的一类。Forrester 最近的一项调查表明，全球数据和分析技术决策者中有 51% 采用图数据库。

所有其他数据库类型（关系型数据库管理系统、数据仓库、文档数据库和键值数据库）主要开始于本地，在建立数据即服务之前颇受欢迎。现在，大型云服务提供商都在转向图技术，图数据库的采用速度会不断加快。

现在有许多不同类型的图数据库产品可供选择，并且每种产品都有其独特的特点和功能，因此，了解其中差异非常重要。

操作型图数据库

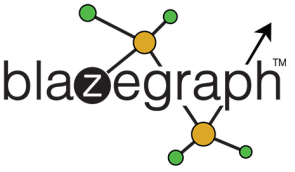
Gartner 将操作型数据库（不限于图数据库）定义为“适用于广泛企业级事务应用程序的数据库管理系统产品，以及支持作为其他事务类型的交互和观察的数据库管理系统产品”（Gartner Inc.,《操作型数据库管理系统魔力象限》（“Magic Quadrant for Operational Database Management Systems”），2016 年 10 月发布，ID：G00293203）。Bloor Research 称，这些解决方案往往是本地图存储，或者建立在 NoSQL 平台之上。它们专注于事务 (ACID) 和操作分析，对索引没有绝对要求。（Bloor Research：《图数据库和 RDF 数据库 2015》（“Graph and RDF databases 2015”）第 2 期，2015 年 9 月发布）。

操作型图数据库包括 Titan、JanusGraph、OrientDB 和 Neo4j。

知识图 /RDF

资源描述框架 (RDF) 是万维网联盟规范体系，最初是作为元数据模型设计的。它使用各种语法记号和数据序列化格式，已用作概念描述或在 Web 资源中实施的信息建模的一般方法。每一项事实都是三元的（主语、谓语、宾语），因此 RDF 数据库有时也称为三重存储。

Bloor Research 调查得出，这些图往往语义集中，基于基础（包括关系型数据库）。它们非常适合在操作环境中使用，但具有推理功能，即使在事务环境中也需要索引。(Bloor Research：《图数据库和 RDF 数据库 2015》(“Graph and RDF databases 2015”）第 2 期，2015 年 9 月发布)。



许多图数据库供应商的知识图技术基于 RDF，包括 AllegroGraph、Virtuoso、Blazegraph、Stardog 和 GraphDB。

多模态图

该类别包括支持多个模态类型的数据库。例如，常见的可能性是文件存储、键值存储或 RDF/ 图存储的三路选项。(Bloor Research：《图数据库和 RDF 数据库 2016》(“Graph and RDF databases 2016”)，2017 年 1 月发布)。多模态方法的优点是可以针对相同数据运行不同类型的查询，例如图查询和键值查询。主要缺点是性能上不能匹配专用和优化的数据库管理系统。

多模态图示例包括 Microsoft Azure Cosmos DB、ArangoDB 和 DataStax。

分析图

Bloor Research 称，分析图侧重于解决“已知的已知”问题（大多数），即实体和关系已知的情况，或者“已知的未知”甚至“未知的未知”问题。该调查公司称，“多种方法使该领域包含不同的体系结构，包括本地和非本地存储，并行化的不同方法以及高级代数的使用”（Bloor Research：《图数据库和 RDF 数据库 2015》（“Graph and RDF databases 2015”）第 2 期，2015 年 9 月发布）。

分析图的示例包括 Apache Giraph、Spark GraphX 和 Turi（以前为 GraphLab，现在由 Apple 所有）。

实时大图

实时大图是一种新的图数据库类别，用于处理海量数据和数据获取率，并提供实时分析。为了处理大型且不断增长的数据集，实时大图数据库设计为可横向扩展和纵向扩展。

无论在图中遍历的步数有多少，实时大图均可实现超过 1 亿次点或边遍历 / 秒 / 服务器和超过 10 万次更新 / 秒 / 服务器的实时大图分析。实时大图还提供实时图分析能力，以便探究、发现和预测非常复杂的关系。这代表着实时深度关联分析，可实现 10 步以上的大图遍历，以及快速的图遍历与数据更新。

实时大图示例包括 TigerGraph。

了解各项产品

总而言之，现在有许多不同类型的图数据库产品可供选择，并且每种产品都有其独特的优势，因此，随着图数据库在各个垂直行业的企业和用例中不断得到采用，了解其中差异非常重要。Forrester Research 最近的一项调查称，“全球数据和分析技术决策者中有 51% 正在实施、已经实施或正在升级他们的图数据库”（Forrester Research，《Forrester 供应商市场格局：图数据库》（“Forrester Vendor Landscape: Graph Databases”），Yuhanna，2017 年 10 月 6 日）。

随着组织开始接受图的强大功能，必须要了解可供选择的各种产品及其优势，对于特定的用例（场景）选择有针对性的产品特别重要。实时大图应用证明，图技术正在向新一代演变。这些解决方案经过专门设计，可为拥有海量数据的组织提供实时分析。

图数据库现状

	实时大图	操作型图	多模态图	分析图	RDF 图
示例	TigerGraph	Neo4j、Titan	DataStax Graph、Microsoft Azure Cosmos、ArangoDB	Apache Giraph、Turi	AllegroGraph、Virtuoso、Blazegraph、Stardog、GraphDB
关键优势和重点	实时、通用、可扩展性	通用	支持多个 NoSQL 数据模型	可遍历全图的分析	三元模型、语义查询
潜在弊端	尚无开放源供应商	性能未扩展到大图	折中，在性能上不具领先优势	在探索或交易方面不够强大	在分析或交易方面不够强大
探索（邻域搜索）	★★★	★★	★½	★	★★★
分析（全数据集读取）	★★★	★★	★½	★★★	★
交易（实时更新、并发）	★★★	★★	★½	★	★
扩展速度	★★★	★★	★	★	★

第 3 章

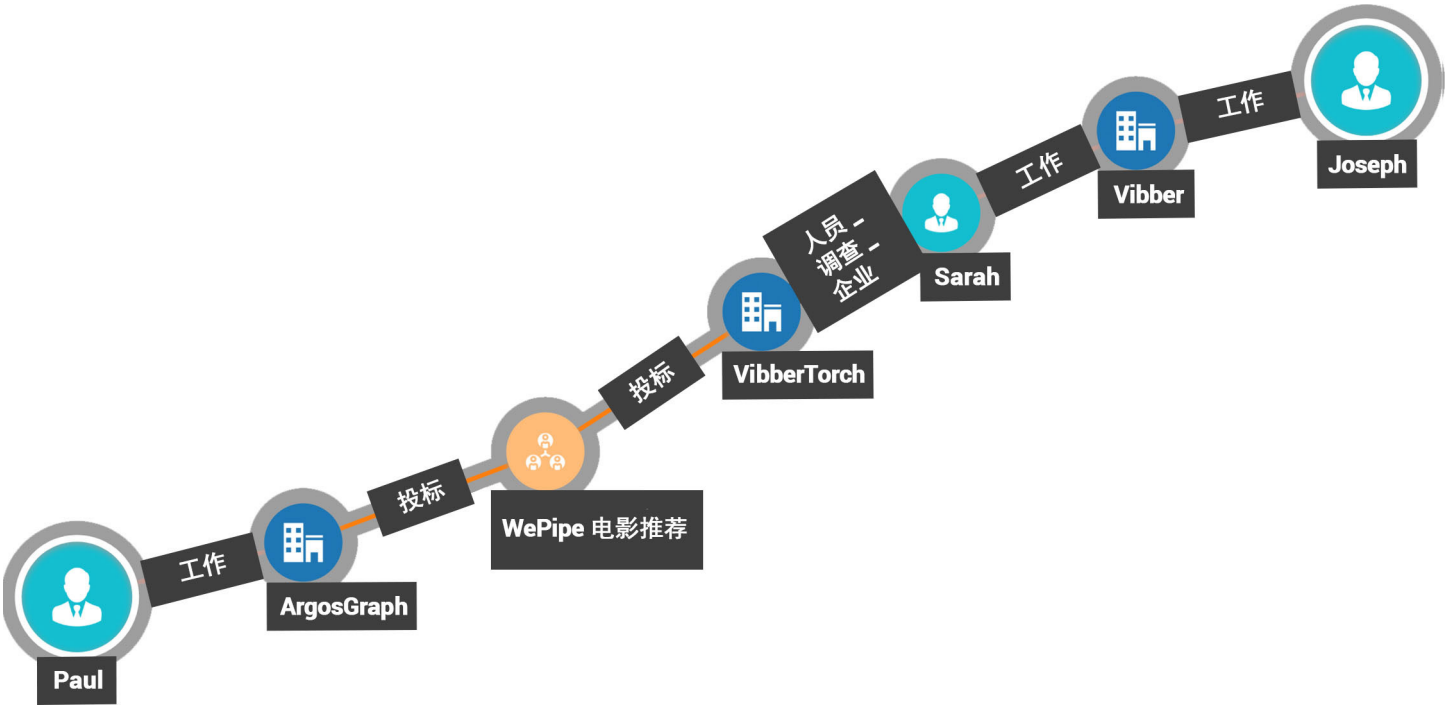
实时深度关联分析

图数据库是分析和解决大数据集间关系和关联的复杂问题的理想模型。不过，它们能否有效提供实时洞察取决于一项关键特性，那就是可在大图中实时遍历的步数（即分离度）。

图使用包括点、边和属性的形式存储数据，克服了表达海量、复杂且互联的数据的挑战。与传统的关系型数据库管理系统和近年来新兴的其他大数据产品相比，图更适于关系分析。不过，大图分析不仅需要适当存储，还需要能快速访问和处理海量图数据。

传统的图技术做不到实时分析，因为它们无法支持大图中 3 步以上的遍历。传统图技术可以处理小图（具有几百万点和边）上的多步遍历，但随着图的大小增加，它们处理连续多步的能力大幅下降。此外还有其他限制，包括难以加载海量数据以及实时获取数据的性能不佳。

随着当今企业生态系统中的实时数据普遍日益增长，现在正是图数据库应势发展的时机。



实时深度关联分析简介

当今企业需要实时图分析能力，以便探究、发现和预测复杂关系。这需要实时深度关联分析，它通过利用 3 到 10 步以上的大图遍历以及快速图遍历和快速数据更新来实现。

思考一个非常简单的个性化推荐问题："那些与你喜好相同的顾客还购买了哪些商品？"。首先，从一个人开始，该查询首先确定您浏览 / 点赞 / 购买过的商品。

第二步，寻找其他查看 / 喜欢 / 购买过这些商品的人。

第三步，找出这些人购买过的其他商品。

```
人 → 产品 →（其他）人 →（其他）产品
```

该查询要求在图上实时走 3 步，因此它超出了前几代图技术对于海量数据集的 2 步查询限制。如果再增加一层关系（如产品功能、产品在售情况），很容易将查询扩展到 4 步或更多步。

实时深度关联分析示例

由于每增加一步就能在海量数据中揭示更多新的关联和隐藏的关系，因此，实时深度关联分析代表了图分析的新里程碑和发展。特别地，实时深度关联分析在许多用例中提供巨大优势，包括反欺诈、个性化推荐、供应链优化以及当今最重要的企业应用所需的其他分析功能。我们来具体说明一下：

风险和欺诈控制

实时深度关联分析通过辨识高风险交易来打击金融犯罪。例如，从一个新的信用卡交易开始，该交易与其他实体发生关联的方式可以辨识如下：

```
新交易 → 信用卡 → 持卡人 →（其他）信用卡 →（其他）不良交易
```

该查询使用 4 步来寻找与转入交易只有一卡之隔的关联。现今的欺诈者试图将自己与已知不良行为或不良行为者之间建立迂回关联，藉以掩盖自己的活动。路径中的任何个人都可能看上去清白无辜，但如果能够在 A 到 B 之间发现多条路径，则存在欺诈的可能性就会增加。因此，不仅需要遍历深度（4 步），还需要遍历宽度（找到多条路径）。

有鉴于此，成功的风险和欺诈控制需要具备多步遍历的能力。这种遍历模式适用于许多其他使用情形，只需将交易替换为网页点击事件、电话记录或汇款即可。借助实时深度关联分析，可以揭示多个隐藏的联系，从而最大限度减少欺诈。

多维度个性化推荐

为使查询找到类似的顾客，或者提供个性化产品推荐，需要进行实时深度关联分析。请考虑以下推荐路径示例，其中所有路径均超过 3 步：

```
i
Person→ Purchases→ Products→ (other)Purchases→ (other)Persons→ (other)Products

Person → Purchases→ Products→ Categories→ (other)Products

Person→ Clicks→ Products→ Categories→ (other)Products
```

以上任何路径均可单独或彼此组合来用于例如零售网站上的个性化推荐。第一条路径选择与您购买了相同产品的其他用户所购买的产品。第二条路径确定与您购买的产品相似的产品，第三条路径确定与您浏览 / 感兴趣的产品相似的产品。推荐算法可在运行时将不同权重分配至每种类型的路径所找到的产品，从而为用户提供一组混合的推荐产品。

不过，由于当前系统的局限性，当前大部分推荐功能都是使用离线计算。这些推荐是在后台提前计算好的，不能利用最新数据执行实时的按需分析，从而丧失很多推荐机会或造成过时的推荐。新的图数据库技术现在能够解决这一问题。

电力调度、供应链物流、道路交通优化

这些应用调整各个实体和 / 或各个关联，直到建立一个动态平衡或最优的状态。例如，在国家或区域电网中，每台发电机的电力输出按如下所示调整：

```
所有实体：

   → 与周边区域的关系和相应影响

   → 更新所有实体

< 重复，直到达到稳定状态 >。
```

为了得到适当的值，需要在图数据上进行迭代计算（类似于 PageRank），直到其中一些度量值收敛为止。在这里，每个顶点组件（例如发电机）是多个从属部件的网络枢纽，从而导致多层关联。在电力分配网络，关联程度通常需要 6 步。

```
Power generator→ Transformers→ Control units→ Lower—level transformers→ Lower—
level control units→ Meters→ Power—consuming devices
```

对于此类案例，图数据库系统需要实时深度关联分析才能处理这一级别的计算。

实时决策支持和企业人工智能的变革性技术

与实时深度关联分析相比，企业通过有限的图分析所获得的信息只是冰山一角。实时深度关联分析支持数据实体之间的所有关联，提供真正变革性的技术，尤其适于具有海量数据的组织。

实时深度分析的出现代表着图分析技术的新纪元。作为图数据库变革的下一里程碑，实时深度关联分析使用户能够比以往更准确、更快速和更深入地探究、发现和预测关系。它最终可以实时支持企业人工智能应用，这是当今快速发展的竞争市场中的真正变革因素。

第 4 章

图数据库之间的差异

图数据库因其相较于关系模型的明显优势而备受关注。许多供应商纷纷进入该领域，因此，评估所有产品颇有难度。下面列出了在评估图数据库时应考虑的因素。

- **属性图或 RDF 存储。**RDF 存储是专用图数据库；属性图是通用图数据库。如果您的数据为纯 RDF，您的查询仅限于模式匹配或逻辑推理，那么 RDF 数据库能够满足您的要求。其他用户应该关注属性图。即便如此，由于可用 RDF 数据库的性能还不够强大，一些具有 RDF 知识图应用的企业仍转而使用属性图数据库。
- **装载能力。**如果数据达到千兆位或更大，装载能力是关键的差异因素。如果您计划定期将不同数据从数据湖导入图数据库，那么装载能力非常重要。加载速度和可用性都很重要。如果您没有测试数据，可以考虑公共数据集，如 15 亿条边的 Twitter 粉丝网络 (http://an.kaist.ac.kr/traces/WWW2010.html) 或 18 亿条边的 Friendster 网络 (http://snap.stanford.edu/data/com—Friendster.html)。首先，图数据库是否需要您预先定义模式？如果是，这是否直观？它是否支持您需要的数据类型，包括复合类型？数据库能否直接从数据文件提取、转换和加载 (ETL) 数据？或者您是否需要预处理数据？它能否处理多种常见输入文件格式？加载 1000 万条边需要多久？ 2000 万条边呢？数据库能否随图的大小增加而保持良好的加载速度？它是否支持并行加载以达到更高的吞吐量？它是否支持超大加载作业所需的计划和重新开始？这些问题将促使您进行评估。
- **原生图存储。**许多图数据库为非原生数据库，这意味着它们在关系表、RDF 三元组或磁盘上的键值对中存储图数据。将数据获取到内存后，它们将提供中间层 API 以模拟图视图和图遍历。与之相反，原生图数据库直接以图模型格式（点和边）存储数据。最有名的原生图包括 TigerGraph 和 Neo4j。非原生图是折中解决方案。它们虽然能更轻松地支持多模态数据库（与一个数据存储相比提供更多数据模型），但牺牲了图性能。而原生图的边和点存储模型提供内置索引，可实现更快速、更高效的图遍历。如果数据集很大，非原生图通常难以处理 3 步以上的查询。
- **有模式或无模式。**某些图数据库称，它们不需要预定义模式。某些图数据库则遵循传统关系数据库模型，需要预定义模式。无模式似乎更好（减少架构师的工作），但这会影响性能。预定义模式可实现更高效的存储、查询处理和图遍历，因此，定义模式的少许初始工作可带来几倍的价值。不过，如果需要预定义模式，请确保数据库支持模式更改（当数据库在使用中时，最好支持在线更改）。可以确保您设计的初始模式不是最终模式。
- **OLTP、OLAP 或 HTAP。**您希望图数据库执行哪类操作？某些图平台纯粹适合大规模处理 (OLAP)，例如 PageRank 或社区检测。优秀的 OLAP 图数据库应该具备高度并行的处理设计和良好的可扩展性。某些图数据库专为“点查询”而设计，从一个或几个点开始若干步遍历。但是，要成为真正的事务数据库（用于 OLTP），它应该支持 ACID 事务、并发性和很高的每秒查询量 (QPS)。如果事务包括数据库更新，则必须做到实时更新（参见下文）。极少有图数据库能同时满足上述要求，因此可考虑混合事务 / 分析处理 (HTAP) 数据库。

- **实时更新。**实时更新意味着数据库更新（添加、删除或修改）可以与数据库上的其他查询同时进行，而且更新可以很快完成(提交)和可用。常见的图更新操作包括插入 / 删除新点 / 边以及更新现有点 / 边的属性。理想情况下，实时更新应与并发控制相结合，以便多个查询和更新（事务）可以同时运行，但最终结果与事务按顺序运行时相同。大多数非原生图平台（例如，GraphX、DataStax DSE Graph）不支持实时更新，因为其数据存储系统（HDFS、Cassandra）不可变。
- **实时深度关联分析。**在如今的大多数图数据库上，查询响应速度从三步开始显著降低。然而，图数据库的真正价值在于找到那些深藏在几步之后的非表面关联。要检验一个图数据库的深度关联性能，一种简单的方法是在大型（约 10 亿条边）图上运行 k 值递增的一系列 k 步邻域大小查询。也就是说，找到距离起始点一步、两步、三步以至更多步的所有点。为集中于图遍历，而非 IO 时间，仅返回 k 步邻点的数量。
- **可扩展性和性能。**可扩展性是大数据时代的一个重要特征。数据总是层出不穷，您拥有的数据越多 / 越丰富 / 越新鲜，分析和洞察能力就会越强。用户需要确保他们的数据库能够增长。数据库可扩展性可划分为三个主要方面：
  - **软件支持** — 软件对可管理的实体数量或数据大小是否有限制?
  - **硬件配置** — 数据库是否能利用单台计算机上增加的硬件容量（纵向扩展）？数据库是否能分布在不断增长的计算机集群中（横向扩展）？
  - **纵向扩展性能** — 在一个完美的系统中，如果数据、内存和磁盘空间全部增加 S 倍，那么全图加载时间应该增加 S 倍，点查询 QPS 应该基本保持不变，分析（全图）查询时间应根据算法的计算复杂度而增加。也就是说，我们料想到运行速度减慢，但仅限于处理的数据增加的情况下。
  - **横向扩展性能** — 在一个完美的系统中，如果数据和计算机数量全部增加 S 倍，那么全图加载时间应该保持不变 (数据的增量与计算机的增量相互抵消)，点查询 QPS 应该最多增加 S 倍 (改进)，分析 (全图）查询时间应根据算法的计算复杂度除以 S 而增加。例如，一个图在 2 倍数量的计算机上增加到 2 倍的边数，PageRank 在理想情况下需要的时间不变，因为 PageRank 的计算力随着边数而等比例扩大。由于计算机间的数据传输速度比计算机内部传输速度慢，所以没有系统能达到如此完美的程度，但您可以比较系统，考察它们与理想标准的接近程度。注意：某些并行设计出色的数据库可以随着 CPU 核数的增加而扩展。严格来讲，这是纵向扩展，但其行为更类似于横向扩展。
- **企业需求。**除了图数据库自身的基本特征，企业还需要对安全性、可靠性、维护和管理的支持。其中许多要求与对传统数据库的要求是相同的：基于角色的访问控制、加密、多租户、高可用性、备份和还原等。但是，由于图数据库市场不太成熟，并非所有供应商都拥有一系列全面的企业特征。客户可能愿意在上述部分要求上灵活变通，以取得图数据库的独特优势。
- **图可视化。**在这里，图数据库所独有的决定性因素是：作为内含功能提供或通过第三方工具支持的图显示能力。优秀的视觉显示非常有助于理解数据。一个图可显示多大？如何确定布局？可视化系统是否还支持图探索或查询?

图数据库受到热捧，因此市场为消费者提供了很多选择。不管您有何要求，思考以上每个因素都将帮助您挑选出合适的图数据库系统。

第 5 章

原生并行图

不同的架构支持不同的用例

> ***无论是用于客户分析、欺诈检测、风险评估还是解决其他现实挑战，快速有效地探索、发现和预测复杂关系的能力都是当今企业的巨大竞争优势。得到这种能力不仅需要有连通的数据，还需要实时、最新的关联、检测和发现过程。组织需要能够将结构化、半结构化和非结构化数据以及大规模企业数据孤岛转换为智能、互联且可操作的数据网络，从而发现支持业务目标的关键模式和决策支持。***

拥有大量数据的企业需要实时分析，这一基本痛点推动着新兴的图数据库成为各行各业公司所推崇的主流技术。

Twitter、Facebook 和 Google 等公司很早就采用了图数据库，此后这一技术不断升温。云服务提供商巨头 Amazon、IBM 和 Microsoft 在过去两年都增加了图数据库，证明业界对于图技术简单自然的数据建模、通过易写的查询解决复杂问题以及从互联数据快速形成见解的这些优势越来越感兴趣。

图数据库擅长解决有关大数据集内关系的复杂问题。但当数据量变得非常巨大，或问题需要深度关联分析，又或者必须实时提供答案时，大多数图数据库都会在性能和分析能力上碰壁。

这是因为前几代图数据库缺乏能满足当今速度和规模需求的技术和设计。第一代设计（例如 Neo4j）不是以并行性或分布式数据库概念为核心构建的。第二代的特点是在 NoSQL 存储之上创建图视图。这些产品可以扩展到巨大的规模，但这一附加层使之丧失了巨大的潜在性能。如果没有原生图设计，执行多步查询的代价会很高，因此许多 NoSQL 平台只能提供很高的读取性能，而不支持实时更新。

TigerGraph 是新一代图数据库设计的代表，通过原生并行图打破了前几代的局限性。原生并行图可实现深度关联分析，带来以下优势：

- 加快数据加载速度以快速构建图
- 加快图算法执行速度
- 能够实时流式处理更新和插入
- 能够将实时分析与大规模离线数据处理统一起来
- 能够纵向扩展和横向扩展分布式应用

图遍历：步数越多，洞察越深

为什么需要深度关联分析？因为在图中可遍历（步）的关联越多，洞察力就越深刻。思考一下知识和社交混合图。每个点都关联到您的知识和您认识的人。直接关联（一步）能揭示您的知识。两步能揭示您的朋友和熟人的所有知识。三步呢？即可发掘每个人的相关信息。

图的优势在于认识数据集中数据实体之间的关系，这是知识发现、建模和预测的核心所在。每一步都会导致关联数量以及相应的知识量呈指数级增长。但技术难度也很大。只有高效并行执行跳跃的系统可以提供实时深度关联（多步）分析。

在第 3 章，我们详细介绍了实时深度关联分析及其增加了独特价值的部分用例：风险和欺诈控制、个性化建议、供应链优化、电力调度优化等。

在了解原生并行图的优势之后，我们看看它的应用。

TigerGraph 原生并行图设计

要能实时绘制数据实体之间的深层关联，就需要有兼顾规模和性能的新技术。并非所有声称原生或并行的图数据库都是相同的。需要许多设计决策协调配合才能实现 TigerGraph 突破性的速度和可扩展性。下面我们来看这些设计特征并讨论它们是如何结合使用的。

原生分布式图

TigerGraph 是彻头彻尾的图数据库。它的数据存储中有点、关联及其属性。市场上的某些图数据库产品实际上是在较一般化的 NoSQL 数据存储上打包而来。这种虚拟图战略在性能上受到双重打击。首先，从虚拟图操作转化为物理存储操作增加了工作量。其次，没有为图操作优化底层结构。此外，该数据库从一开始就是为支持横向扩展而设计的。

存储紧凑，访问快速

TigerGraph 没有被称作内存中数据库，因为将数据存储在内存中是首选项，而不是必需的。用户可以设置参数以指定多少空闲内存可用于存放图。如果全图无法整体放入内存，多余部分将存储在磁盘上。当然，如果全图能放入内存，能达到最佳的性能。

存储数据值的编码格式可有效压缩数据。压缩系数因图结构和数据而异，但一般的压缩系数介于 2 倍到 10 倍之间。压缩有两种优势：第一，更大量的图数据可以放入内存和 CPU 缓存。这种压缩不仅减少内存占用量，还提高 CPU 缓存命中率，从而加快整体查询性能。第二，减少巨大图用户的硬件成本。例如，压缩系数是 4 倍，那么一个组织可以将所有数据放到一台计算机中，而非四台。

解压 / 解码对最终用户来说非常快速和透明，因此压缩的好处超过了压缩 / 解压导致的少量时间延迟。一般而言，只有显示数据时需要解压。在内部使用值时，通常可能保持编码和压缩状态。

内部使用哈希索引来引用点和关联。按照大 O 记法，我们的平均访问时间是 O(1)，平均索引更新时间也是 O(1)。也就是说，对图中特定点或关联的访问速度非常快，即使图的大小增加，也能保持快速访问。此外，向图中添加新点和关联时维护索引的速度也是非常快的。

并行和共享数据

当速度成为您的目标时，有两条基本路线可选：加快每项任务的执行速度，或一次完成多个任务。后一条路就是并行。TigerGraph 在竭力快速执行每项任务的同时，还擅长并行处理，完全采用 MPP（大规模并行处理）设计架构。例如，它的图引擎使用多个工作线程和线程遍历一个图，而且可利用多核 CPU 中的每个核。

图查询的本质是“跟踪关联”。从一个或多个点开始，查看这些点的可用关联，然后追随这些关联找到一些或全部邻近点。于是，我们说您刚“遍历”了一“步”。重复该过程，找到原始点邻点的邻点，那么您就遍历了两步。由于每个点都可能有许多关联，因此这个两步遍历涉及很多路径，才能从起始点抵达目标点。图天然就适合并行执行多线程。

当然，查询不仅是只访问一个点。在简单的情况下，我们可以清点唯一两步邻点的数量或者列出其 ID。当您有多个并行计数器时，如何计算总数呢？该过程与您在现实中的做法相似：让每个计数器完成自己的份额，最后将它们的结果合并起来。

回忆一下征询唯一点数量的查询。同一点有可能被两个不同的计数器计算在内，因为达到该目标的路径不止一条。即使在单线程设计中也可能出现该问题。标准的解决方案是为每个点分配一个临时变量。变量初始化为 False。当一个计数器访问一个点时，该点的变量赋值为 True，因此其他计数器便知道不再计算它。这里的解释可能听起来很复杂，但您实际上可以使用 TigerGraph 的高级查询语言通过几行代码编写包含计数和更复杂计算的此类图遍历，代码长度甚至比这段话还短。

采用 C++ 编写的存储和处理引擎

语言的选择也会影响性能。TigerGraph 的图存储引擎和处理引擎是使用 C++ 实现的。在通用程序语言家族中，C 和 C++ 被认为比 Java 等其他语言的级别更低。这意味着了解计算机硬件如何执行软件命令的程序员可以做出明智的选择来优化性能。TigerGraph 经过精心设计，可高效使用内存并释放未使用的内存。精心的内存管理有助于 TigerGraph 在一次查询中遍历深度和广度意义上的许多关联。

许多其他图数据库产品是用 Java 编写的，优点和缺点并存。Java 程序在 Java 虚拟机 (JVM) 内运行。JVM 负责内存管理和垃圾收集（释放不再需要的内存）。虽然这很方便，但程序员难以优化内存使用率或控制未使用内存的释放时间。

GSQL 图查询语言

TigerGraph 也有自己的图查询和更新语言 GSQL。虽然 GSQL 有很多巧妙的细节，但支持高效并行计算的关键有两点：ACCUM 子句和累加器变量。

大多数 GSQL 查询的核心是 SELECT 语句，与 SQL 中的 SELECT 语句非常相近。SELECT、FROM 和 WHERE 子句用于选择和筛选一组关联或点。在此选择之后，可使用可选的 ACCUM 子句定义由每个关联或临近点执行的一组操作。我之所以说“由它们执行”而非“在它们上执行”，是因为在概念上，每个图对象都是独立的计算单位。图结构就像是一张巨大的并行计算网。图不仅是数据存储，还是查询或分析引擎。

一个 ACCUM 子句可能包含许多不同的操作或语句。这些语句可以从图形对象中读取值、执行本地计算、应用条件语句并计划图更新。为支持这些分布式查询内计算，GSQL 语言提供了累加器变量。累加器的种类有很多，但都是临时（仅存在于查询执行期间）、共享（可用于任何执行线程）和互斥（一次只能有一个线程更新它）的。例如，简单的和数累加器将用于执行上述所有邻点的邻点的计数。集累加器将用于记录所有这些邻点的邻点的 ID。累加器可在两个范围中使用：全局和每点。在前面的查询示例中，我们提到需要将每个点标记为已访问或未访问。此处将使用每点累加器。

MPP 计算模型

重申一下上述结论，TigerGraph 图既是存储模型又是计算模型。每个点和关联都可以关联一个计算函数。因此，每个点或关联同时作为并行的存储和计算单元。如果使用一般的 NoSQL 数据存储，或者不使用累加器，都无法达到这种效果。

自动划分

在当今的大数据世界中，企业需要能够将他们的数据库解决方案横向扩展到多台计算机，因为他们的数据可能增长得过大，无法经济地存储在单个服务器上。TigerGraph 能够将图数据自动划分到一个服务器集群中，并且仍然运行迅速。哈希索引不仅用于确定服务器内的数据位置，还用于确定哪个服务器。从给定点连出的所有关联都存储在同一服务器上。

分布式计算模式

TigerGraph 有一种分布式计算模式，可显著提升遍历大部分图的分析查询性能。在分布式查询模式下，要求所有服务器都参与查询；每个服务器的实际参与根据需求而定。当遍历路径从服务器 A 跨入服务器 B 时，将向服务器 B 传递它需要知道的最少量信息。在服务器 B 已经得知整个查询请求后，便可以轻松地参与分工。

我们在基准测试研究中测试了通用的 PageRank 算法。此算法是对图的计算和通信速度的严格测试，因为它遍历每个关联、计算每个点的分数并重复迭代几次遍历与计算过程。当图分布到八台服务器时，与单台服务器相比，PageRank 查询的完成速度接近快七倍。

通过原生并行图进行高性能图分析

TigerGraph 代表着图技术的新纪元，为用户带来真正的实时分析。它的技术优势支持更复杂、更个性化并且更准确的分析，还让组织能够与快速变化和扩张的数据保持同步。

全球首个且唯一的真正原生并行图 (NPG) 系统 TigerGraph 是一个完整的分布式图数据分析平台，支持 Web-Scale 的实时数据分析。TigerGraph NPG 建立在本地存储和计算的基础上，支持实时图更新，并充当并行计算引擎。TigerGraph ACID 事务保证数据一致性和正确的结果。分布式的原生并行图架构让 TigerGraph 能够达到无可匹敌的性能水平：

- 每台计算机每小时加载 100 到 200 GB 数据。
- 每台计算机每秒遍历数亿个点 / 边。
- 在亚秒时间里执行 10 多步查询。
- 每秒更新数千个点和边，每天更新数亿个。
- 横向扩展以处理无限数据，同时保持实时速度并提高加载和查询吞吐量。

原生并行图的出现是图数据库历史上的一座里程碑。通过这项技术，首个实时深度关联分析数据库成为现实。

第 6 章

在键值存储上建立图数据库?

> ***直到最近，图数据库设计也只是满足了企业的部分而非全部图分析需求。第一代图数据库（例如 Neo4j）不是为大数据而设计的。它们不能横向扩展到分布式数据库，不能并行执行，而且在大型数据集和 / 或多步查询中的数据加载和查询速度都很缓慢。***

第二代图数据库 (例如 DataStax Enterprise Graph) 是在 Apache Cassandra 等 NoSQL 存储系统上构建而成的。这解决了横向扩展的问题，但由于缺乏原生图存储设计，在大型数据集上执行图更新或多步查询时仍然速度缓慢。面对明显缺乏高性能图数据库解决方案的情况，某些组织考虑构建自己的内部图解决方案。要构建定制图解决方案，无需从零开始，现代的快速键值存储似乎是一个不错的基础。

键值存储的诱惑

由于关系型数据库太慢太僵化，才掀起了 NoSQL 革命。大数据用户需要大容量、高速度地吸收各种不同结构化的数据，并以最少的麻烦横向扩展物理基础设施。键值存储作为最简单因而也是最快速的 NoSQL 架构应运而生。键值数据库基本上是一个两列哈希表，每行有一个唯一键 (ID) 和一个与该键关联的值。搜索键域可以非常快速地返回单数据值，比关系型数据库快得多。键值存储也能很好地扩展到非常大型的数据集。键值存储易于理解，而且有很多免费的开源设计可供使用。使用快速键值存储存放图的点实体和边实体、实现更多逻辑并称之为图数据库，似乎非常简单直接。

构建自有图数据库的真实成本

在键值存储的基础上构建自有图数据库远比看起来困难得多，更不要说实现高性能了。显著的难点有两个：

1. 您在遵循第二代图数据库的架构原则，却又尝试在此基础上改进。DataStax Enterprise Graph 等产品在非原生图存储设计的固有限制下，已经尽最大努力优化性能。
2. 您真正需要的是一个完整的数据库管理系统 (DBMS)，而不仅是一个图数据库。数据库只是按特定方式组织的数据集合，以便支持数据的存储和检索。另一方面，DBMS 是通过数据库使用和管理数据的完整应用程序。DBMS 通常包括支持定义数据库模型，插入、读取、更新和删除数据，执行计算查询（不仅是读取原始数据），应用数据完整性检查，支持事务保证和并发，与其他应用交互以及执行管理功能（例如管理用户权限和安全性）。自己动手设计需要实现并验证所有这些额外的要求。

我们来看在键值存储的基础上构建高性能图的一些常见问题。

1. 数据不一致和错误的查询结果
2. 架构不匹配，性能低

3. 实现代价高，机制僵化

4. 缺乏企业支持

1. 数据不一致和错误的查询结果

键值存储通常经过高度优化，以追求高吞吐量和*单记录* 读 / 写的低延迟。因此，大多数键值存储只能保证单记录级别的 ACID（原子性、一致性、隔离性、持久性）特性。这意味着涉及多记录写入（例如客户采购、转账、拼车请求）的任何现实事务性事件面临数据一致性的风险。数据一致性不足意味着聚合两个或更多这类常见多记录事务的查询可产生不正确的结果。

如果开发人员希望构建企业可以真正信任的系统，就应该认识到，数据一致性和保证结果正确是一项非成即败的工作。实现 ACID 特性没有捷径可走。在图数据系统中，一个事务通常需要创建多个点并在新点或现有点之间添加新的关联 / 边，所有这一切都发生在一个非成即败的事务中。保证 ACID 的简单方法是一次仅执行一个操作，锁定所有其他操作。但这会导致非常低的吞吐量。

在高性能设计中，事务必须永远不能干扰、减慢或阻止任何其他现有或转入的只读查询，即使查询在遍历图的相邻部分时也是如此。**如果业务要求并发操作，那么开发人员需要设计出非阻塞性的图事务架构，在分布式键值系统内添加多项键值事务支持。**有许多开发团队尝试走捷径，但不可避免地导致了下游数据和查询的正确性问题。

2. 架构不匹配，性能低

涉及单记录的查询用键值存储可以很快完成。但随着查询的复杂度上升，键值存储查询速度会急剧下降。图遍历每走一步，应用都必须向外部键值存储发送键值检索命令。步数越多，就会生成越多的通信开销，导致查询性能降低。数据的计算（例如筛选、聚合和证据收集）不能完全向下推给键值存储。相反，它必须发生在图应用层，导致与键值存储之间的额外输入和输出传输，同时进一步拉低查询性能。此外，图应用层（而非原生并行图平台）内的计算默认是非并行的。**要达到像 TigerGraph 一样的原生并行图的并行性，开发人员需要承受额外的负担，并行化处理单台计算机上的计算，甚至承受更为沉重的责任，将计算横向扩展到多台计算机**并负责分布式系统的计划、消息传递、故障转移和其他要求。

想要高性能图的企业需要一个能分析复杂关系并揭示隐藏模式，而且查询速度快的系统。要满足这些要求，开发人员需要执行以下所有工作：在处理基本图建模（一般描述图的结构和行为）的键值应用上构建图，实现基本图数据定义和数据操作，确保事务保证，然后在此基础上开发应用查询。**也就是说，内部开发人员需要实现数据定义语言 (DDL) 和图查询语言的解释程序。**开发人员应该考虑到，支持如此高难度的工作需要大量的时间和资源，而且还要解决很多不可预见的下游问题。

3. 实现代价高，机制僵化

如果组织打算在另一数据库系统之上创建自己的图数据库系统，那么应该考虑他们的应用需要的资源、可靠性目标以及理想的适应性。

在键值系统上构建图将需要大量的时间来完成。构建自定义软件的前期成本是不可小觑的。此外，可靠性对于应用的成功至关重要。如前所述，开发人员应该深入思考如何设计、测试、部署和支持图生态系统，包括完整的数据一致性、正确的结果和高性能的复杂查询。最后，管理和开发团队应该考虑到，单一应用可能不具备高性能分布式原生图数据库产品的通用功能和灵活性。决策者应着眼于大局，思考他们的定制设计项目能否解决未来不可预见的扩张问题。现在选择正确的图形功能，您可以简化任务、减少错误并让每个人在未来几年生活得更轻松。

4. 缺乏企业级支持

最后但同样重要的是，定制设计不包含对安全可靠的企业部署至关重要的许多功能：用户授权和访问控制、日志记录、高可用性以及与其他应用的集成。其中一些企业功能可能适用于键值存储，但需要在图（应用）级别提供。

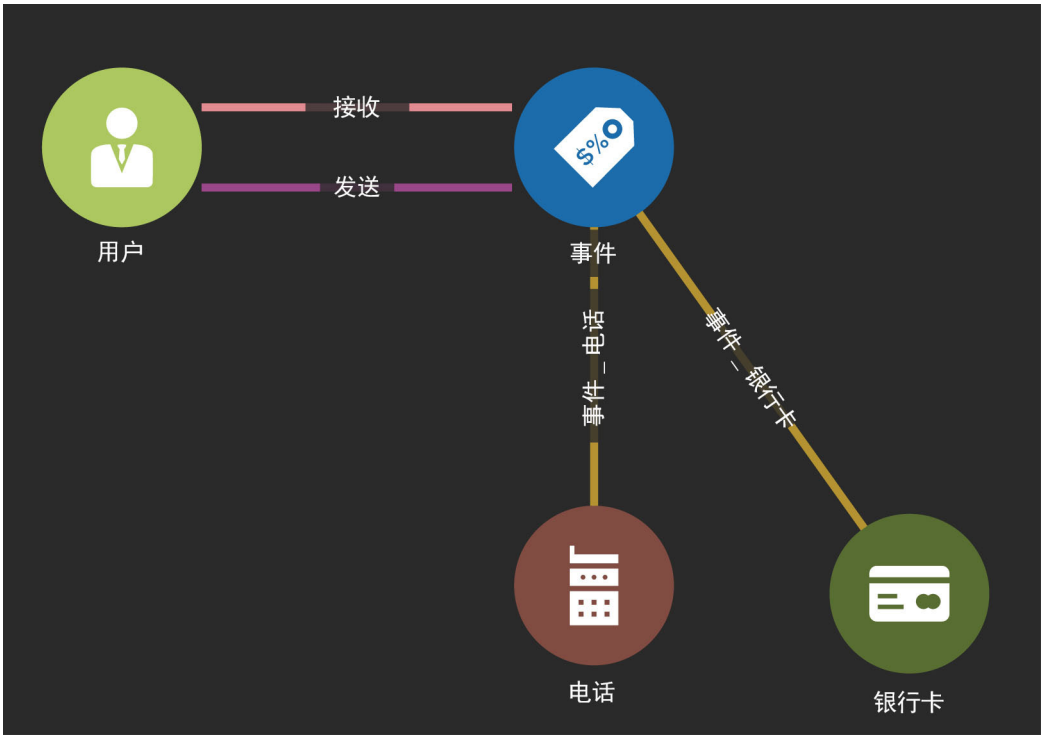
小结

总之，在键值存储之上设计应用级图是一项代价高昂且复杂的工作，不会产生高性能的结果。虽然键值存储在单键值事务中表现优异，但它们缺少 ACID 特性以及图更新所需的复杂事务功能。因此，在键值存储之上构建图数据库会导致数据不一致、查询结果错误、多步查询速度缓慢而且部署成本高而且机制僵化。

示例：更新图

以下示例显示了典型的图数据库更新中发生的操作。它说明了使用基于键值存储的图形系统时如何导致数据不一致和结果错误，并讨论了尝试构建功能性变通方法时遇到的开发难题和限制。

请思考此更新情景。一个用户在手机上使用银行卡给另一个用户转账时，便发生了实时事件。此类事务可用简单的图模式来建模，如下所示。



本例适用于许多其他用例。例如，您可以将转账事件替换为产品采购事件或服务合约事件，例如招聘或拼车服务。

转账事件 e1 可描述为四个共现实体（u1、u2、c1、p1），其中：

- u1 是转钱的用户。
- u2 是收钱的用户。
- c1 是使用的支付卡。
- p1 是使用的手机。

为简便起见，我们忽略此类事件中存在的所有其他属性（例如美元金额、位置、时间戳）。

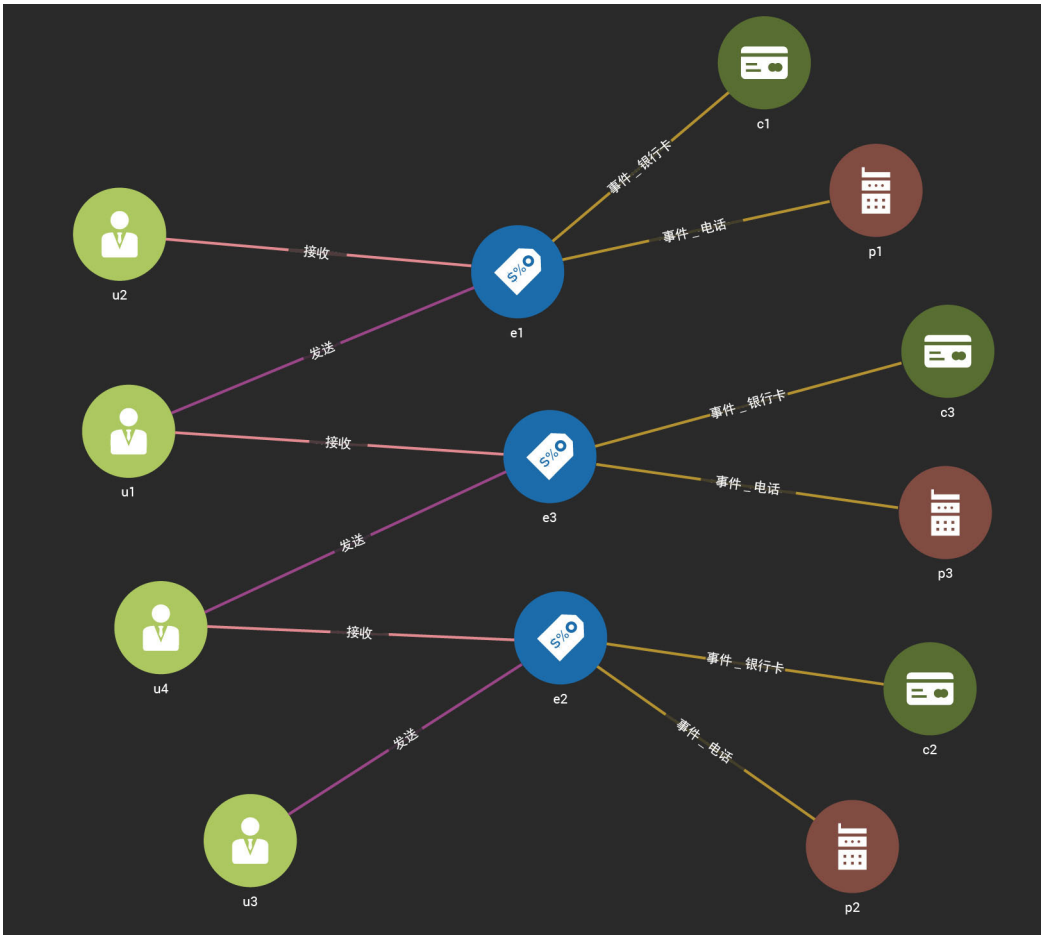
使用键值图系统可以通过多种方法建立图数据库模型。一种方法是在键值图系统中设两个表：一个点表 V 和一个边表 E。另一种方法是只有单一的点表（无边表），其中每个点拥有一行来存储点属性和边信息（边属性和关联的点）。任何一种方法所面临的事务挑战都是相似的，使用双表方法可以最好地展现出来。

要添加新事件 e1 以更新图，我们执行以下操作：

1. 对于每个实体 u1、u2、c1 和 p1，检查是否已存在。为尚不存在的每个实体创建新点（命名为 V_u1、V_u2、V_c1、V_p1）。在键值图系统方法中，相当于需要在点表 V 中插入零到四行。
2. 为事件 e1 创建新点（在下文中称作 V_e1）。在键值图系统方法中，相当于需要在点表 V 中插入一行。
3. 添加四条边（从 V_e1 分别连到 V_u1、V_u2、V_c1 和 V_p1）。为提高查询性能，可能需要在 V_u2 和 V_c1 之间添加一条边，在 V_u2 和 V_p1 之间添加另一条边。这会直接关联一位用户使用的所有卡和手机，但会让键值图系统实现起来更复杂而且更容易出错。

   我们还希望能够找到与一部手机、一个用户或一张卡关联的所有事件。这需要添加方向相反的另外四条边：分别从 V_u1、V_u2、V_c1 和 V_p1 连到 V_e1。根据实现细节，这意味着添加四个行或更新现有四行的值。在键值图系统方法中，相当于需要在边表 E 中插入八行。

参见下例中的图实例：



在处理事件 e1 并更新图后，图实例类似于上图。

由于事件 e1 作为单一事务处理，所以上述三个操作将以原子方式发生在图（相当于两个表）中，这意味着：

1. 要么所有三个操作保证在图（相当于两个表）中生效，要么没有一个在图（相当于两个表）中生效。允许执行部分而非全部操作（例如操作 1 和 2 成功完成，但操作 3 未完成）的非事务性系统将导致数据不一致和结果错误。如果任何一个或多个操作未完成，结果中将缺少边，并出现查询结果错误。
2. 在我们应用操作 1、2 和 3 过程中，这些操作完成之前，其效果对于任何其他读或写操作应不可见。这意味着，如果您有另一个查询 q1，它在事件 e1 进入系统前仍然在遍历图中的边 / 点，q1 应该不能看到 1、2 和 3 中的任何操作，即使它们在 e0 结束前成功完成也是如此。
3. 这还意味着，如果只读查询 q2 进入系统的时间晚于 e1 但在 e1 的三项操作全部完成之前，e1 的效果对于 q2 应该是不可见的。

哪些键值图变通方法是可行的?

一种变通方法是直接忽略所有数据不一致性和错误查询结果的问题，直接将键值行插入到点表 V 和边表 E 中，然后自求多福。即使公司的业务要求可以放宽，能容忍数据不一致和错误的查询结果，这种方法也将面临实现过程的梦魇。这是因为用户不知道图中是否或何时缺少一个点或边。为解决此问题，开发人员将需要增加保护代码，以确保应用不会因为遍历幽灵 / 缺失边而挂起或崩溃。

较好的解决方案是在分布式键值系统中增加多键值事务支持。但这不仅需要完成大量的工程任务，还会明显降低键值操作速度。

第三种解决方案是“安慰剂解决方案”，即不更改底层键值存储的内核，在应用级别内部增加某种锁定机制。例如，尝试向每一行添加“提交”标记或维护应用中行的“脏”列表，这种变通方法可以减少数据不一致和错误的结果。但是，此类附加功能不等同于为事务保证而设计的架构，因此数据不一致和结果错误可能始终存在（尽管很隐蔽）。人们很难保证这些变通方法能始终协调和处理每个事务。

第 7 章

查询大图

市场上有很多属性图语言，包括 Neo4j 的 Cypher、Apache TinkerPop Gremlin 和 TigerGraph 的 GSQL 等。在讨论哪种图语言最好或者如何将每种图语言的精华融合为全新而统一的语言之前，我们先退后一步，问一个更基本的问题：一种图查询语言首先要具备哪些前提条件？

我们如今的时代正在见证互联数据的洪流。互联数据能带来变革性的业务价值，这一点可以从新兴的互联网公司巨头中得到证实，包括 Google（知识图）、Facebook（社交图）、LinkedIn（职业网络）和 Twitter（兴趣图）。我们的通信应用、电子交易、智能设备和监控设备每秒都会产生海量的互联数据。因此，对图数据管理的迫切需求已然浮现。

关系型模型首次在数据库的历史上失效，因为在关系型数据库中追踪一连串的多个关联太慢了。每一步连接两个表，需要付出巨大的计算代价。向外键添加索引将加快连接速度，但关系型数据库的常规存储和处理模型仍然是一个障碍。

更重要的是，SQL（关系型数据库查询语言）甚至不支持类似“查找一个关联所连接的所有记录”的基本发现操作，因为 SQL 要求您事先知道目标实体的类型。简而言之，关系型数据库和 SQL 语言不适合从互联数据中发现和学习。

图模型解决了这一难题，带来了以下有益的功能：

- 为互联数据提供自然高效的存储模型。
- 为管理演变对象类型提供自然模型。
- 通过链和组合观察为认知 / 推理 / 学习提供自然的分析和计算模型。

图数据库是管理和利用当今庞大的互联数据的正确选择。怎样才是合适的图查询语言？

图查询语言的八项前提条件

正如 SQL 显著降低了管理关系数据的难度，精心设计、用户友好且表达充分的图查询语言也可以有力地帮助您从提出笼统的现实问题过渡到轻松制定基于图的解决方案。要挑选合适的图查询语言，我们不妨从用户（开发人员）及其所在企业的角度来思考。

- **建模**：用户希望如何对图数据建模？
- **查询**：组织希望从图中得到哪种结果？
- **开发**：开发人员用图语言表达意图的简单程度如何？

这些都是非常笼统的问题，所以为了解答这些问题，我们会将其进一步分解。下面我们提出了一种图查询语言要满足如今的苛刻需求而要具备的八项前提条件。

1 基于模式并可以动态更改模式

查询语言实际上始于所查询的数据结构。图模型的强大体现在它能轻松反映数据对象的自然组织属性。这种数据组织也称为模式，是信息的一部分。因此，优秀的图语言应该支持定义清晰的模式，这些模式包含用户定义的点类型和边类型，而点和边都有自己的一组特征或属性。语言应该支持广泛的基本和复合数据类型。因为数据模型在现实中随着时间而改变，所以语言和系统还应支持简单高效的模式更改，特别是图数据库保持在线时。

2 高级（声明式）图遍历

SQL 语言是声明式的，意味着用户寻找数据时不必关心数据库软件如何执行查询。这让用户不必再编写分步算法代码来执行任务。要设计一种精妙的图查询语言，也可以应用同样的范式。例如，一个“查找此模式”的简单查询应通过定义该模式并定义搜索空间来表达，而不必解释如何执行搜索。

3 图遍历的算法控制

图的功能远不止简单的查询，所以必须要支持对于图遍历的算法控制。在复杂分析和数据修改过程中，每一步都很可能存在数据更新或者需要引用上一步状态的操作或者计算。例如，思考 PageRank、社区发现和中心性测量等典型的图算法以及个性化推荐、欺诈检测、供应链优化和预测分析之类的应用。企业选择图来管理、设计和执行 RDBMS 和其他 NoSQL 系统无法解决的复杂算法。因此，除了声明式语义，图查询语言还需要为开发人员提供非常精细的图遍历与计算的算法控制。实际上，如果语言提供条件分支（“IF…THEN”）、循环（“REPEAT”）和临时数据存储变量，就属于图灵完备的语言，能够实现通用计算机的一切功能。

4 内置高性能并行语义

图算法通常代价高昂，因为每一步都可能使数据复杂度呈指数级上升。我们来思考一下这个过程：从一个点开始，第一步可产生 n 个邻点，从 n 个直接邻点开始走下一步，可再产生 n^2 个邻点，以此类推。正是因为这种指数级增长，许多图数据库无法处理超过两步的查询。为解决这个难题，图中的每一步遍历都应有内置的并行处理语义，以确保高性能。为充分发挥作用，并行遍历需要与无痛的合并和聚合方式相结合。

5 事务性图数据更新

一种功能完备的图查询语言需要不仅支持高效的图读取，还要支持图写入（数据插入、更新和删除）。也就是说，它应该不仅支持联机分析处理 (OLAP)，还要支持联机事务处理 (OLTP)。一个事务数据库要保证，当用户将一个操作块定义为一个事务时，那么该事务要全部完成（或完全不完成）而且不会与可能同时正在处理的任何其他事务交互。语言必须提供一种方法来隐式或显式定义事务以及插入、更新和删除操作。

6 过程式查询调用查询

随着任务变得越来越复杂，优秀的开发人员会希望将最佳的编程思路之一运用到图查询中：过程编程。过程是可保存并在需要时“调用”的一个指令块。过程可提供模块化设计、层次设计、抽象化、代码重用和更好的可读性。例如，请想象一个从源点的 K 步半径内寻找最有影响力的实体的查询。通过过程式查询，这个模块可在多个较大的查询中调用和重复使用：以用于推荐、欺诈调查、优化等。算法设计中最巧妙的一种思路是递归，让过程可以调用自己。深度优先的调查可通过递归巧妙实现。

7 对 SQL 用户友好

大多数企业都有熟悉 SQL 的开发人员。因此，图查询语言应该尽可能接近 SQL 语法和概念，以便 SQL 开发人员用最少的上手时间学习和实现图应用。

8 图的访问控制

企业客户热衷于用图数据促进协作。一方面，他们需要一个可以在多个部门甚至合作伙伴之间共享部分选定数据的图模型。同时，他们还希望保持某些敏感数据的隐私性，仅允许特定角色、部门和合作伙伴根据业务需求而访问。在大型企业的现实环境下，成功的图查询语言需要有可识别图模式且基于角色的访问控制。

TigerGraph GSQL 如何满足这八项前提条件?

基于五年以来的客户反馈，TigerGraph 与我们的客户几经迭代，共同发明了一种名为 GSQL 的图灵完备的图查询语言。它包括用于定义图模式的 DDL（数据定义语言），用于高性能加载结构化和半结构化数据的加载语言，以及有内在并行性的类似 SQL 的图遍历和计算语言。

基于模式并可以动态更改模式

在 GSQL 中，图定义为点类型和边类型的集合。一个 TigerGraph 数据库可包含多个（可能重叠）图，每个图都有自己的一组用户权限和查询。例如

```
CREATE VERTEX person (PRIMARY_ID ssn STRING, age INT, name STRING)
CREATE UNDIRECTED EDGE friendship (FROM person, TO person)
CREATE DIRECTED EDGE teacher_student (FROM person, TO person)
CREATE GRAPH School (person, friendship, teacher_student)
```

另外，GSQL 的 DDL 支持用类似 SQL 的语法动态更改模式。模式更改作业的处理类似其他事务，只不过增加了一个功能来卸载与新模式不一致的任何查询或加载作业。

高级（声明式）图遍历

GSQL 的构成块是类似 SQL 的 SELECT 语句，用以描述一步遍历。要描述一个路径或模式，用户可以直接在一个查询中包含若干个相关的 SELECT 语句。正如在 SQL 中一样，GSQL SELECT 语句是高级别和声明式的。

图遍历的精细控制

GSQL 引入了一组内置累加器，充当点的运行时属性（也叫做特性）。累加器可以附加到遍历的每个点，因此用户可以在运行时遍历期间在点上存储标签或中间结果，为欺诈检测和个性化推荐等复杂应用收集证据。GSQL 还有流控制（WHILE 和 FOREACH 循环）和条件分支（IF—ELSE/CASE—WHEN）以支持复杂的算法。这些特性共同确保了用户掌握控制权。GSQL 是图灵完备的，因此它本身可以实现任何算法，而无需在图数据库和客户端应用之间来回交换数据，否则查询速度会减慢几个数量级。

内置高性能并行语义

一对创新的 ACCUM 和 POST—ACCUM 子句再结合内置的累加器变量类型通过编码实现了用于高级图遍历的相同块内点集或边集的并行处理。这意味着 GSQL 使用户能够非常轻松地实现图数据的并行处理。ACCUM 范式让 GSQL 用户无需担心底层细节即可实现并行算法。并非每个图查询都需要这一强大的功能，但从计算总数到 PageRank 在内的算法都受益于 GSQL 的内置并行语义。

事务性图数据更新

每个 GSQL 查询、加载作业或 REST 请求都自动视为一个事务，满足 ACID 特性。该语言提供读取 (SELECT)、更新、插入和删除语句，这些语句全部都是实时执行的。支持分布式和并发事务。

过程式查询调用查询

一个 GSQL 查询可以在多个地方调用查询：ACCUM 和 POST—ACCUM 子句中或在语句级别。这为真实图遍历问题的解决提供了最大的灵活性和简便性。查询也可以递归调用自身。此特性实现了分而治之，极大简化了查询逻辑和管理。

对 SQL 用户友好

GSQL 再利用了大部分 SQL 语法、语义和关键词 — SELECT 和 FROM、WHERE、HAVING、ORDER BY 与 LIMIT 子句，以及 INSERT、UPDATE 和 DELETE。

图的访问控制

业界首个基于角色控制访问的多图模型 MultiGraph 是 GSQL 不可或缺的一部分，帮助 TigerGraph 的客户同时实现数据共享和保密。

第 8 章

GSQL — 新型的图查询语言

根据咨询公司 DB—Engines.com[1] 的调查，图数据库技术是所有数据管理系统中增长最快的一类。Forrester[2] 最近的一项调查显示，如今超过一半的全球数据和分析技术决策者采用图数据库。

当今企业使用图技术作为应对客户分析、欺诈检测、风险评估和其他复杂数据挑战的竞争优势。该技术能够快速高效地探索、发现和预测关系，以揭示支持业务目标的关键模式和见解。

采用新型图查询语言的时机已经成熟

随着图数据库的广泛采用，推出现代图查询语言的时机已然成熟。在图数据库的早期，若干查询语言已经出现，例如 Neo4j 的 Cypher 和 Apache TinkerPop Gremlin，但这些语言都有各自的局限性。

Cypher 虽然是高级别的而且对用户友好，但主要适用于模式匹配而非分析，它必须与 Java 编程相结合才能达到图灵完备。很多图算法和业务逻辑规则使用 Cypher 无法实现。例如 PageRank/ 标签传播式算法，其中差异对于电力调度和优化、风险分析等垂直市场很重要。

Gremlin 是图灵完备的，但太底层。该语言很适合非常简单的查询，但是当面临现实的业务问题时，需要具备高级编程技能，而且生成的查询难以理解和维护。下一章中将介绍图查询语言的技术对比。

随着数据和用例的增长和复杂度的上升，组织显然需要能够横向扩展，同时能够以实时操作速度对机器学习和 AI 等任务执行复杂的分析。

现代标准化语言也有助于解决图市场中普遍存在的挑战，包括难以找到熟练的图开发人员以及加快企业采用的需求。降低学习和实现图应用的难度（通过高级别图查询语言）将让更多开发人员更轻松地从提出笼统现实问题过渡到轻松制定基于图的解决方案。

现代图查询语言以及支持它的超快速可扩展系统也是帮助企业实现大规模数字化转型的关键。

现代图查询语言应该是怎样的?

我们退后一步，了解一下为什么用户选择图数据库，而不选择 RDBMS、键值、文档存储和其他类型的数据库。这一般是因为他们希望以满意的性能解决复杂的问题。非图数据库有着如下的短板：

1 https://db—engines.com/en/ranking_categories

2 https://www.forrester.com/report/Vendor+Landscape+Graph+Databases/—/E—RES121473

- 极难表达现实业务问题。例如，帐户或人等实体如何以各种（通常是以前未知的）方式关联起来，特别是如何关联到已知的不良行为者？
- 当访问数千万到数亿实体及其关系时，实时性能低下。对于实时个性化推荐、欺诈检测等应用，速度是非常重要的。

图数据库提供了解决这些问题的平台，但用户仍然需要合适的查询语言实现以下目标：定义图模式以便对复杂的实体和关系建模，轻松地将各种数据源映射到图，快速地将数据加载到图，并以充分的表达（图灵完备）对各行各业的现实业务问题建模并解决它们。这些目标是促使图技术跨越鸿沟在企业内得到更广泛使用的关键动力。

有鉴于此，专家们列出了现代图查询语言必须具备的八个重要特征：1) 基于模式并可以动态更改模式；2) 高级（声明式）图遍历；3) 图遍历的精细控制；4) 内置并行语义以保证高性能；5) 保证图更新事务性；6) 过程式查询调用查询（递归）；7) 对 SQL 用户友好；8) 图的访问控制。

GSQL 简介

GSQL 是一种用户友好、表达充分且图灵完备的图查询语言。与其他语言不同，GSQL 支持组织已存在的现实业务需求，并且完全是为满足上述标准而设计的。

事实上，人们曾探索过可充分满足实际业务需求的现代图查询语言，失败之后激发了 GSQL 的开发。六年前成立 TigerGraph 时，我们决定投入时间（几年）创建 GSQL 以缓解这一痛点。我们从头开始设计，以支持实时事务和超快速分析的并行加载、查询和更新。

通过 GSQL，客户可获得以前认为无法获取的数据洞察力。也就是说，GSQL 让原本不可能的事情成为了可能。GSQL 已得到一些巨头公司的应用，并支持世界上最大规模的图部署，每天的实时事务数量超过 20 亿。

用户还表示，GSQL 很容易用于创建基于大数据的查询。另一个优势是速度，因为 GSQL 查询几秒就能给出结果，相比之下，使用 Cypher 或 Gremlin 等解决方案却需要几个小时（如果查询可行的话）。简而言之，GSQL 满足了图查询语言最重要的特性：性能、可表达性和易用性。

MapReduce、Gremlin、Cypher、SPARQL 和 SQL 的概念衍生物

GSQL 不是在真空环境下开发的。它从之前的图查询语言中，当然还从 SQL 中，吸收了许多精华。正因如此，GSQL 通常可支持任何前代语言的功能，而且通常直接将较早期语言的语法映射到 GSQL 中。

由于关系型数据库从 20 世纪 70 年代开始直到 80 年代一直是主流的数据库范例，因此 SQL 成为了标准查询语言。SQL 基于严密的逻辑模型：关系代数。正确性和声明语法是最重要的，而非性能。SPARQL 用于查询 RDF 数据；版本 1.0 在 2008 年 1 月受到 W3C 的推荐。Gremlin、Cypher 和 GSQL 都以属性图为目标。Gremlin 最初以 Apache TinkerPop 项目的身份出现于 2009 年 10 月。Neo4j 在 2011 年推出了 Cypher，TigerGraph 则在 2014 年开发了 GSQL。

设计 GSQL 时，主要受到客户现实需求的激励。其他任何图查询语言（例如 Cypher、Gremlin）都不能满足 TigerGraph 早期采用者的需求。GSQL 在满足执行复杂分析的用户需求和尽可能与 SQL 和其他常见编程语言保留相似之间谨慎达成了平衡。由于 GSQL 是在上述其他语言发布初始版本之后出现的，因此有取其精华、去其糟粕的后见之明。我们下面将逐一详细说明。

SQL 加强

从一开始，GSQL 便使用批量同步并行 (BSP) 模型并行处理图查询。我们需要一种方法来描述图中的一个迭代或一步，因此选择使用 SQL SELECT—FROM—WHERE 块结构作为基础构造块，并尽可能使用 HAVING、ORDER BY、LIMIT、INSERT 等其他关键词。另外，GSQL 还有 DDL 语句，以声明点（点）、边和图类型，例如 CREATE VERTEX person(primary_id name string, age int)。该语法与 SQL 的 CREATE TABLE 非常相似。Cypher 也继承了部分 SQL 关键词（但没有 SQL 中最常见的语句类型 SELECT），因此 GSQL 与 Cypher 间接相似。

继承 Map—Reduce

第二，GSQL 引入了 ACCUM 和 POST—ACCUM 子句以支持 SELECT 块中原有的 Map—Reduce 语义。ACCUM 子句充当 Map 步骤；并行处理所有匹配的边。POST—ACCUM 子句充当 Reduce 步骤；合并处理从 ACCUM 子句发出的源或目标点的消息。

Gremlin 运行时变量

Gremlin 是一种函数链式的查询语言。它有一个遍历指令集和一组遍历器。查询结果是暂停的遍历器的位置。Gremlin 中的一个重要概念是**副作用**（除了遍历本身之外发生的任何事情，包括运行时变量）。副作用有全局性和局部性的。局部副作用实现为 sack()。每个遍历器可保持名为 sack 的局部数据结构，可使用 sack—step 读取或写入 sack 变量。全局副作用 withSideEffect() 可用于存放全局变量。整个查询的副作用与遍历器相关联，因此数据结构可能变得非常错综复杂（对应于运行时遍历树）。

GSQL 有相似的概念，但概念的复杂度低很多。用户可使用 GSQL 定义任意数量的局部累加器（名字以 @ 打头）或全局累加器（名字以 @@ 打头）。语法非常简单明了。要读取累加器，可像对点属性那样处理（例如，v.@sum）。要写入，则使用累加器运算符 +=。与 Gremlin 不同，局部累加器没有绑定到每个遍历器，而是绑定到每个访问过的点。全局累加器可从查询主体的任意位置访问。

SPARQL 三元模式匹配

GSQL 将 1 步遍历描述为与 FROM 子句中的三元匹配的边集。

```
source_vertex_set —(edge_types)—> target_vertex_types
```

这个三元与 SPARQL 的三元模式匹配样式几乎相同。但是，GSQL 不会像 SPARQL 那样将模式放在 WHERE 子句中。GSQL 将模式放在 FROM 子句中，以作为“一等公民”区分对待。

Cypher 匹配模式作为一等公民 — 点模式、边模式或路径模式是 Cypher 的主要特征。在 GSQL 中，边模式和点模式同样是一等公民，位于 FROM 子句中。GSQL 近期将添加对路径模式的支持

融会贯通

GSQL 从先前的图查询语言和 SQL 中继承并借用了许多有用的功能。巧妙之处在于取其精华、去其糟粕。更重要的是，GSQL 添加了流控制以支持高级迭代（WHILE、FOREACH）和条件分支（IF...ELSE、CASE），为用户提供他们最熟悉的语法词汇，以实现图灵完备。

此外，GSQL 查询可以包含任意数量的 SELECT 块（单步）。数据流不必形成连续的链。只要定义了每个 FROM 子句的起始点，每个 SELECT 块可以从图的新部分或先前访问过的部分开始。通常，多步形成 DAG（有向无环图）。

这种一次一步的方法，加上每一步都可用的 Map—Reduce 聚合和累加器（附加到点和全局），以及以几乎任何方式组合多步的能力，让用户能够将代码分解为易于管理的部分，然后组合成非常强大的算法。GSQL 查询语言非常类似于 Oracle PL/SQL 或 Microsoft SQL Server 存储过程语法。

GSQL 101 — 基础教程

为了让您体验 GSQL 的简单性和强大功能，本章的剩余部分将提供一个教程，指导您定义图模式、加载数据并使用简单的内置查询。下一章将介绍更复杂任务的查询，并比较 GSQL 与其他图查询语言的功能。

注意：GSQL 语言使用关键词 vertex 来指代图中的顶点。

数据集

在本例中，我们将创建和查询图 1 中所示的简单好友社交图。此图的数据包含 csv（逗号分隔值）格式的两个文件。要学习本教程，请将 person.csv 和 friendship.csv 这两个文件保存到您的 TigerGraph 本地磁盘。在我们运行的例子中，我们使用 /home/tigergraph/ 文件夹存储这两个 csv 文件。

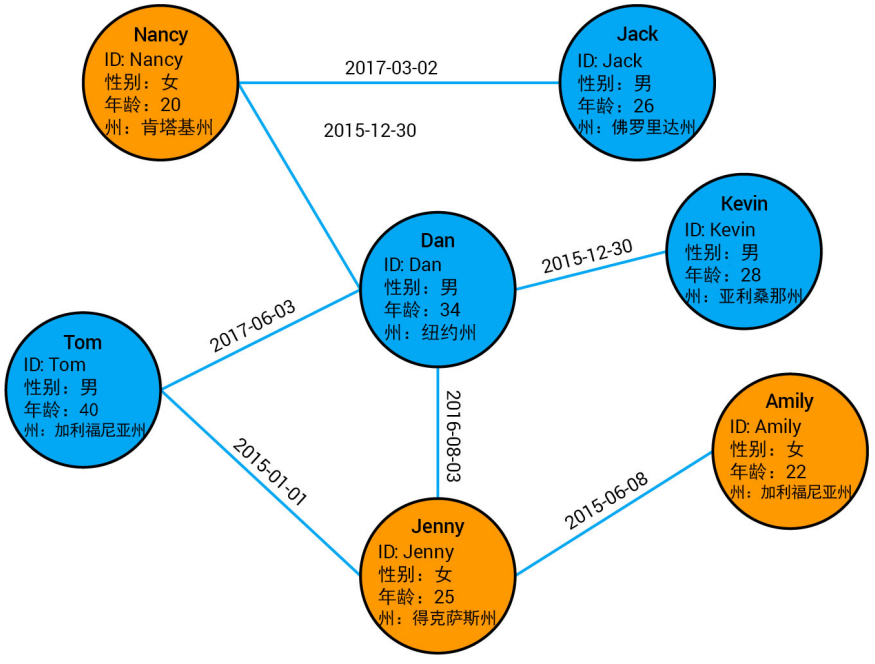


图 1. 好友社交图

person.csv

```
name,gender,age,state
Tom,male,40,ca
Dan,male,34,ny
Jenny,female,25,tx
Kevin,male,28,az
Amily,female,22,ca
Nancy,female,20,ky
Jack,male,26,fl
```

friendship.csv

```
person1,person2,date
Tom,Dan,2017-06-03
Tom,Jenny,2015-01-01
Dan,Jenny,2016-08-03
Jenny,Amily,2015-06-08
Dan,Nancy,2016-01-03
Nancy,Jack,2017-03-02
Dan,Kevin,2015-12-30
```

准备您的 TigerGraph 环境

首先，我们检查您是否可访问 GSQL。

1. 打开 Linux shell。
2. 如下键入 gsql。GSQL shell 提示符应如下显示。
3. Linux shell
4. ```
   $ gsql
   GSQL >
   ```
5. 如果 GSQL shell 没有运行，尝试用“gadmin restart all”重置系统。如果需要更多帮助，请查看 TigerGraph 知识库和常见问题 [3]。

如需全部重置，命令 DROP ALL 将删除所有数据库数据、其模式以及所有相关定义。此命令的运行时间约为一分钟。

[3] 我们省略了一些管理命令和步骤，如果您在实际系统上运行，可能需要应用这些命令和步骤。有关详细说明，请参阅 http://docs.tigergraph.com.cn/
   ```

GSQL shell — DROP ALL

```
GSQL > drop all
Dropping all, about 1 minute ...
Abort all active loading jobs
[ABORT_SUCCESS] No active Loading Job to abort.
Shutdown restpp gse gpe ...
Graph store /usr/local/tigergraph/gstore/0/ has been cleared!
Everything is dropped.
```

提示：从 Linux shell 运行 GSQL 命令

还可从 Linux shell 运行 GSQL 命令。要运行单个命令，只需使用“gsql”，后跟以单引号括起的命令行。（如果不存在解析歧义，则无需使用引号；使用引号更加安全。）例如，

Linux shell — 从 Linux shell 运行 GSQL 命令

```
# "–g graphname" is need for a given graph
gsql –g social 'ls'
gsql 'drop all'
gsql 'ls'
```

还可以执行存储在文件中的一系列命令，只需调用“gsql”，后跟文件名即可。

完成后，可使用命令“quit”（不含引号）来退出 GSQL shell。

定义模式

在本教程中，我们主要在 GSQL shell 的交互模式下工作。有一些命令将来自 Linux shell。创建图的第一步是定义其模式。GSQL 提供一组 DDL（数据定义语言）命令，与 SQL DLL 命令类似，用于建模点类型、边类型和图。

创建点类型

使用 CREATE VERTEX 命令定义名为 **person** 的点类型。此处需要 PRIMARY_ID：每个人都必须具有唯一标识符。其他的是一系列可选属性，用于描述每个 person 点的特性，格式为 *attribute_name data_type, attribute_name data_type, ...*

GSQL 命令

```
CREATE VERTEX person (PRIMARY_ID name STRING, name STRING, age INT, gender
STRING, state STRING)
```

为突出显示，GSQL 关键词全部用大写字母表示，但实际上它们不区分大小写。

GSQL 将确认创建点类型。

GSQL shell

```
GSQL > CREATE VERTEX person (PRIMARY_ID name STRING, name STRING, age INT,
gender STRING, state STRING)
The vertex type person is created.
GSQL >
```

可以创建您所需的多个点类型。

创建边类型

接下来，使用 CREATE ...EDGE 命令创建名为 **friendship** 的边类型。关键词 UNDIRECTED 指示此边为双向边，这意味着信息可从任一点开始流动。如果您想建立单向关联，使信息仅从 FROM 点开始流动，请使用 DIRECTED 关键词而非 UNDIRECTED。此处需要 FROM 和 TO 以指定边类型连接的两个点类型。要指定单条边，需提供其源 (FROM) 点和目标 (TO) 点的 primary_id，后跟一系列可选属性，就像在点定义中一样。

GSQL 命令

```
CREATE UNDIRECTED EDGE friendship (FROM person, TO person, connect_day
DATETIME)
```

GSQL 将确认创建边类型。

GSQL shell

```
GSQL > CREATE UNDIRECTED EDGE friendship (FROM person, TO person, connect_day
DATETIME)
The edge type friendship is created.
GSQL >
```

可以创建您所需的多个边类型。

创建图

接下来，使用 CREATE GRAPH 命令创建名为 **social** 的图。在此处，我们只列出了需包括在此图中的点类型和边类型。

GSQL 命令

```
CREATE GRAPH social (person, friendship)
```

GSQL 将在几秒后确认创建第一个图，在此期间它会将目录信息推送到所有服务，如 GSE、GPE 和 RESTPP。

GSQL shell

```
GSQL > CREATE GRAPH social (person, friendship)

Restarting gse gpe restpp ...
Finish restarting services in 16.554 seconds!
The graph social is created.
```

此时我们已创建 **person** 点类型、**friendship** 边类型和包括它们的 **social** 图。您已构建第一个图模式！我们在 GSQL shell 中键入“ls”命令来查看目录中的内容。

GSQL shell

```
GSQL > ls
——— Global vertices, edges, and all graphs
Vertex Types:
  — VERTEX person(PRIMARY_ID name STRING, name STRING, age INT, gender STRING,
state STRING) WITH STATS=”OUTDEGREE_BY_EDGETYPE”
Edge Types:
  — UNDIRECTED EDGE friendship(FROM person, TO person, connect_day DATETIME)

Graphs:
  — Graph social(person:v, friendship:e)
Jobs:


Json API version:v2
```

加载数据

创建图模式后，下一步是向其加载数据。此时的任务是向 GSQL 加载程序指示如何将一组数据文件中的字段关联（“映射”）至我们刚才定义的图模式中点类型和边类型中的属性。

本地磁盘上应有两个数据文件 person.csv 和 friendship.csv。它们无需位于当前文件夹。

如需因任何原因退出 GSQL shell，可键入“quit”（不含引号）。键入 gsql 可再次进入。

定义加载作业

以下加载作业假定您的数据文件位于文件夹 /home/tigergraph 中。如果数据文件位于别处，请在以下加载作业脚本中，将“/home/tigergraph/person.csv”和“/home/tigergraph/friendship.csv”替换为其各自相应的文件路径。假定您在（回到）GSQL shell 中，请输入以下一组命令。

用于定义加载作业的 GSQL 命令

```
USE GRAPH social
BEGIN
CREATE LOADING JOB load_social FOR GRAPH social {
   DEFINE FILENAME file1="/home/tigergraph/person.csv";
   DEFINE FILENAME file2="/home/tigergraph/friendship.csv";

   LOAD file1 TO VERTEX person VALUES ($"name", $"name", $"age", $"gender",
$"state") USING header="true", separator=",";
   LOAD file2 TO EDGE friendship VALUES ($0, $1, $2) USING header="true",
separator=",";
}
END
```

我们讲解一下命令：

- USE GRAPH social：
  告知 GSQL 您要使用的图。
- BEGIN ...END：
  表示多行模式。GSQL shell 会将这些标记之间的所有内容视为单个语句。只有交互模式需要此命令。如果运行存储在命令文件中的 GSQL 语句，命令解释程序将读取整个文件，因此无需 BEGIN 和 END 提示。
- 创建加载作业：
  一个加载作业可描述从多个文件到多个图对象的映射。必须将每个文件分配至文件名变量。可以按名称或按位置为字段添加标签。名称标签需要源文件中的标题行。逐位标签使用整数来表示源列位置 0、1、... 在以上示例中，第一个 LOAD 语句按名称指示源文件列，而第二个 LOAD 语句按位置指示源文件列。请注意以下细节：
  - file1 中的列“name”映射至两个字段，即 PRIMARY_ID 和 person 点的“name”属性。
  - 在 file1 中，性别先于年龄。在 person 点中，性别在年龄之后。加载时，请按目标对象（本例为 person 点）所需的顺序来声明属性。
  - 每个 LOAD 语句均具有 USING 子句。它用于告知 GSQL 每个文件均包含标题（无论是否选择了使用名称，GSQL 仍然需要知道是否将第一行视为数据）。它还指示列分隔符为逗号。除逗号之外，GSQL 还可处理任何单字符分隔符。

运行 CREATE LOADING JOB 语句时，GSQL 将检查语法错误，并检查指定位置是否有数据文件。如果未检测到错误，它将进行编译并保存作业。

GSQL shell

```
GSQL > USE GRAPH social
Using graph 'social'
GSQL > BEGIN
GSQL > CREATE LOADING JOB load_social FOR GRAPH social {
GSQL >    DEFINE FILENAME file1="/home/tigergraph/person.csv";
GSQL >    DEFINE FILENAME file2="/home/tigergraph/friendship.csv";
GSQL >
GSQL >    LOAD file1 TO VERTEX person VALUES ($"name", $"name", $"age", $"gender",
$"state") USING header="true", separator=",";
GSQL >    LOAD file2 TO EDGE friendship VALUES ($0, $1, $2) USING header="true",
separator=",";
GSQL > }
GSQL > END
The job load_social is created.
```

运行加载作业

现在可运行加载作业以将数据加载到图中：

GSQL 命令

```
RUN LOADING JOB load_social
```

结果如下所示。

GSQL shell

```
GSQL > run loading job load_social


[Tip:Use "CTRL + C" to stop displaying the loading status update, then use
"SHOW LOADING STATUS jobid" to track the loading progress again]
[Tip:Manage loading jobs with "ABORT/RESUME LOADING JOB jobid"]
Starting the following job, i.e.
  JobName:load_social, jobid:social_m1.1528095850854
  Loading log:'/home/tigergraph/tigergraph/logs/restpp/restpp_loader_logs/
social/social_m1.1528095850854.log'

Job "social_m1.1528095850854" loading status
[FINISHED] m1 ( Finished:2 / Total:2 )
```

```
  [LOADED]
  +-------------------------------------------------------------------------------+
  |                        FILENAME | LOADED LINES | AVG SPEED |   DURATION|
  |/home/tigergraph/friendship.csv |            8 |     8 l/s |     1.00 s|
  |    /home/tigergraph/person.csv |            8 |     7 l/s |     1.00 s|
  +-------------------------------------------------------------------------------+
```

注意加载日志文件的位置。此示例假定您已将 TigerGraph 安装到默认位置 /home/tigergraph/。在安装文件夹下是主产品文件夹 tigergraph。在 tigergraph 文件夹中有若干子文件夹，如 logs、document、config、bin 和 gstore。如果安装到了其他位置，如 /usr/local/，则可在 /usr/local/tigergraph 找到产品文件夹。

使用内置 SELECT 查询进行查询

你现在有了一份带数据的图！可运行一些简单的内置查询来检查数据。

选择点

以下 GSQL 命令报告 person 点的总数。person.csv 数据文件的标题之后有 7 行。

GSQL 命令

```
SELECT count() FROM person
```

类似地，以下 GSQL 命令报告 friendship 边的总数。friendship.csv 文件的标题之后也有 7 行。

GSQL 命令

```
SELECT count() FROM person-(friendship)->person
```

结果如下所示。

GSQL shell

```
GSQL > SELECT count() FROM person
[{
  "count":7,
  "v_type":"person"
}]
GSQL > SELECT count() FROM person-(friendship)->person
[{
  "count":14,
  "e_type":"friendship"
}]
GSQL >
```

边数

为什么有 14 条边？对于无向边，GSQL 实际上将创建两条边，每个方向一条。

如果要查看有关一组特定点的详细信息，可使用“SELECT *”和 WHERE 子句来指定谓词条件。可尝试以下一些语句：

GSQL 命令

```
SELECT * FROM person WHERE primary_id==”Tom”
SELECT name FROM person WHERE state==”ca”
SELECT name, age FROM person WHERE age > 30
```

结果是 JSON 格式，如下所示。

GSQL shell

```
GSQL > SELECT * FROM person WHERE primary_id==”Tom”
[{
  “v_id”:“Tom”,
  “attributes”:{
    “gender”:“male”,
    “name”:“Tom”,
    “state”:“ca”,

  “age”:40
  },

  “v_type”:“person”
}]
GSQL > SELECT name FROM person WHERE state==”ca”

[
  {
    “v_id”:“Amily”,
    “attributes”:{“name”:“Amily”},
    “v_type”:“person”
  },
  {
    “v_id”:“Tom”,
    “attributes”:{“name”:“Tom”},
    “v_type”:“person”
```

```
  }
]

GSQL > SELECT name, age FROM person WHERE age > 30
[
  {
    "v_id":"Tom",
    "attributes":{
      "name":"Tom",
      "age":40
    },
    "v_type":"person"
  },
  {
    "v_id":"Dan",
    "attributes":{
      "name":"Dan",
      "age":34
    },
    "v_type":"person"
  }
]
```

选择边

类似地，我们可查看有关边的详细信息。要描述边，请在三个部分中指定点和边的类型，并添加标点符号以表示遍历方向：

GSQL 语法

```
source_type -(edge_type)-> target_type
```

请注意，无论是无向边还是有向边，始终使用箭头 -> 。这是因为我们描述的是查询在图中的遍历（搜索）方向，而非边本身的方向。

我们可在 WHERE 子句中使用 from_id 谓词来选择从“from_id”标识的点出发的所有 friendship 边。允许使用关键词 ANY 来指示任何边类型或任何目标点类型。以下两个查询的结果相同

GSQL 命令

```
SELECT * FROM person—(friendship)—>person WHERE from_id ==”Tom”
SELECT * FROM person—(ANY)—>ANY WHERE from_id ==”Tom”
```

内置边选择查询的限制

为避免查询返回过多输出项，内置边查询具有以下限制：

1. 必须指定源点类型。
2. 必须指定 from_id 条件。

用户定义查询没有此类限制。

结果如下所示。

GSQL

```
GSQL > SELECT * FROM person—(friendship)—>person WHERE from_id ==”Tom”
[
  {
    “from_type”:“person”,
    “to_type”:“person”,
    “directed”:false,
    “from_id”:“Tom”,
    “to_id”:“Dan”,
    “attributes”:{“connect_day”:“2017—06—03 00:00:00”},

  “e_type”:“friendship”
  },
  {
    “from_type”:“person”,
    “to_type”:“person”,
    “directed”:false,
    “from_id”:“Tom”,
    “to_id”:“Jenny”,

“attributes”:{“connect_day”:“2015—01—01 00:00:00”},
    “e_type”:“friendship”
  }
]
```

检查图大小的另一种方法是使用管理员工具的选项之一 gadmin。在 Linux shell 中，输入命令

gadmin status graph -v

Linux shell

```
[tigergraph@localhost ~]$ gadmin status graph —v
verbose is ON
=== graph ===
[m1     ][GRAPH][MSG ] Graph was loaded (/usr/local/tigergraph/gstore/0/
part/):partition size is 4.00KiB, SchemaVersion:0, VertexCount:7,
NumOfSkippedVertices:0, NumOfDeletedVertices:0, EdgeCount:14
[m1     ][GRAPH][INIT] True
[INFO   ][GRAPH][MSG ] Above vertex and edge counts are for internal use
which show approximate topology size of the local graph partition.Use DML to
get the correct graph topology information
[SUMMARY][GRAPH] graph is ready
```

第 9 章

GSQL 功能分析

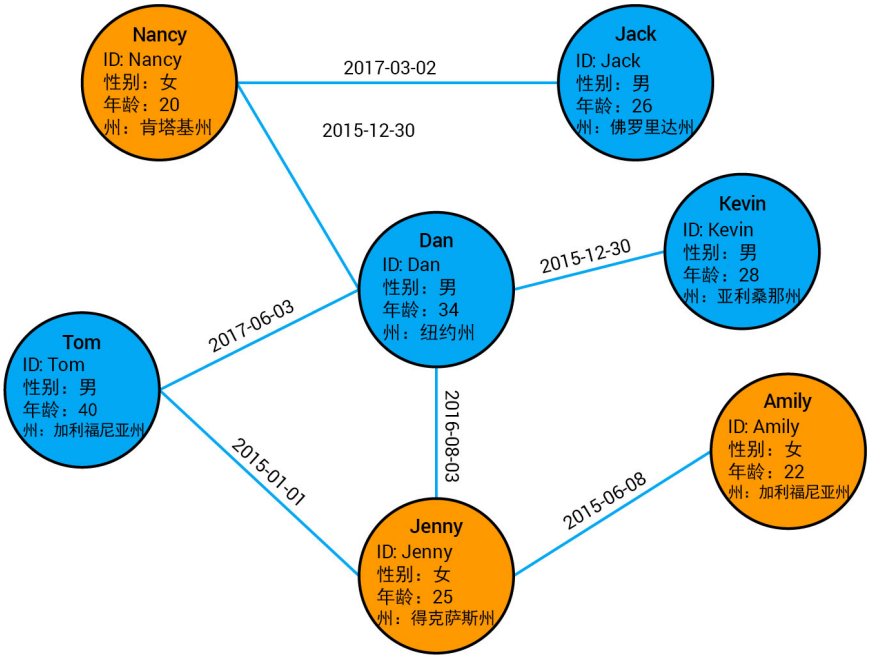


图 1. 好友社交图

在上一章中，我们说明了 GSQL 是现代化图查询语言（尤其对于实时深度关联分析）的主要原因。我们还讲解了基础教程，帮助您直接熟悉 GSQL。

在本章中，我们将以一些实际查询为例来分析 GSQL 的技术功能。随后介绍其他三种图查询语言 Gremlin、Cypher 和 SPARQL 并进行比较。

注意：GSQL 和 Gremlin 语言均使用关键词 vertex（点）来指代图点，因此在本章中，即便有时较为抽象，我们仍使用点而非图点。

示例 1：业务分析 — 基本

假设我们想要针对每一个客户 c，找出向 c 销售“toy”类产品所赚取的总收入。我们的模式包含 Customer 和 Product 点以及 Bought 边。每个产品都有价格，每笔购买都有数量和折扣。

```
SumAccum<float> @revenue;                                    #1
Start = {Customer.*};                                        #2
Cust = SELECT c                                              #3
```

```
    FROM Start:c -(Bought:b)-> Product:p                                    #4
    WHERE p.category == "toy"                                               #5
    ACCUM c.@revenue += p.price*(1-b.discount/100.0)*b.quantity;            #6
    PRINT Cust.name, Cust.@revenue;                                         #7
```

可以明显看到，第 (6) 行计算客户 c 对产品 p 的购买（Bought 边）b 所产生的收入。不太明显的是，ACCUM 子句和 SumAccum 累加器“@revenue”（第 1 行）自动对购买了“toy”的客户迭代（并行）所有 Bought 边，每个客户均具有自己的运行时“@revenue”，虽然可同时向同一累加器添加多个进程，累加器将自动确保没有冲突。使用 MapReduce 的开发人员应熟悉并行处理范式。

示例 2：图算法

PageRank 可在 GSQL 中轻松实施。PageRank 计算点的相对权威度，这对许多需要找到重要或有影响力的实体的应用场景来说非常重要（目标营销、影响预测、有限资源的最佳使用等）。其他图算法（如社区发现和最短路径发现）与此类似，也具有许多实际应用。对于此类用例，需要能够完全使用高级图查询语言来实施和自定义图算法，这非常重要。相反，硬编码 PageRank 算法不向用户提供可解决实际问题的表达能力。

GSQL 查询可被视为参数化程序，如以下示例 2 中的初始 CREATE QUERY 语句所示。（简单起见，我们省略了示例 1 中的 CREATE 语句。）

```
CREATE QUERY pageRank(float maxChange, int maxIteration, float damping)
  FOR GRAPH gsql_demo {
  MaxAccum<float> @@maxDiff=9999; #max score change in an iteration
  SumAccum<float> @recvd_score=0; #sum(scores received from neighbors)
  SumAccum<float> @score=1;       #scores initialized to 1

  V = {Page.*};                       #Start with all Page vertices
  WHILE @@maxDiff > maxChange LIMIT maxIteration DO
    @@maxDiff = 0;
    S = SELECT s
        FROM V:s-(Linkto)->:t
        ACCUM t.@recvd_score += s.@score/s.outdegree()
        POST-ACCUM s.@score = (1-damping) + damping * s.@recvd_score,
                   s.@recvd_score = 0,
                   @@maxDiff += abs(s.@score - s.@score');
  END; #end while loop

  PRINT V;
}#end query
```

高级累加和数据结构

到目前为止，我们已经展示了累加器如何将数字数据聚合为单一的标量结果值。GSQL 的表达能力之所以十分强大，得益于其对于复值累加器的支持，复值累加器包含元组的实例集合，它们可以被划分成组，并且可以选择按组聚合。

示例 3

我们修改了收入聚合部分的代码，使之可以显示对于每个客户 c，按产品类别划分的收入以及按折扣划分的收入。

```
GroupByAccum<string categ, SumAccum<float> revenue> @byCateg;                #1
GroupByAccum<int disc, SumAccum<float> revenue> @byDiscount;                 #2
Start = {Customer.*};                                                        #3
Cust = SELECT c                                                              #4
  FROM Start:c —(Bought:b)—> Product:p                                       #5
  ACCUM c.@byCateg += (p.category—>p.price*(1—
b.discount/100.0)*b.quantity),                                               #6
     c.@byDiscount += (b.discount—>p.price*(1—
b.discount/100.0)*b.quantity);                                               #7
```

注意名为“byCategory”的点累加器的声明（第 1 行）。这些按组划分的累加器将字符串属性“categ”定义为组键，并将和数累加器 “revenue” 关联到每个组，用于聚合组的值。

第 (6) 行显示的语法将值 p.price*(1—b.discount/100.0)*b.quantity 写入和数累加器 "revenue" 之中，此和数累加器与位于点 c 的 byCateg 累加器中的键 p.category 组相对应。在第 2 行中，根据折扣值对组销售额定义了一个类似的 group-by 累加器，用于汇总每个组的销售额。为了操作集合类型的累加器的值，GSQL 还提供了一个 foreach 原语，它可以迭代集合内的所有元素，并向这些元素绑定一个变量。

包含中间结果流的多步遍历

GSQL 支持分析规范，这些分析涉及在图中遍历多步路径，同时沿着路径传播计算出的中间结果。以下示例说明了如何在 GSQL 中准确表达一个简单的推荐系统。

示例 4

以下 GSQL 查询将为客户 1 生成玩具推荐列表。这些推荐按照经典的推荐系统方式进行排名：每个推荐玩具的排名都是其他顾客喜欢程度的加权总和。客户 c 的每一个喜欢的加权值由客户 c 与客户 1 的相似度决定。此相似度为标准的对数—余弦值相似度表示，它反映了客户 1 与客户 c 共同喜欢的玩具有多少。更正式地讲，对于每个客户 c，我们定义一个比特向量 likes_c，当且仅当客户 c 喜欢玩具 i 时，才会设置比特 i。两个客户 x 和 y 之间的对数—余弦值相似度被定义为

```
lc(x, y) = log(1 + likes_x ⊙ likes_y )
```

其中 ⊙ 表示两个向量的点积。请注意，likes_x ⊙ likes_y 简明地计算出客户 x 和 y 共同喜欢的玩具数。在下文中，我们假定客户 c 喜欢产品 p 这一事实被建模为一条连接 c 和 p 的唯一边，标记为“Likes”。

```
SumAccum<float> @rank, @lc;                                          #1
SumAccum<int> @inCommon;                                             #2
Start = {Customer.1};                                                #3
ToysILike =                                                          #4
    SELECT p                                                         #5
    FROM Start:c -(Likes)-> Product:p                                #6
    WHERE p.category == "toy";                                       #7
OthersWhoLikeThem =                                                  #8
    SELECT o                                                         #9
    FROM ToysILike:t <-(Likes)- Customer:o                          #10
    WHERE o.id != 1                                                 #11
    ACCUM o.@inCommon += 1                                          #12
    POST-ACCUM o.@lc = log (1 + o.@inCommon);                       #13
ToysTheyLike =                                                      #14
    SELECT t                                                        #15
    FROM OthersWhoLikeThem:o -(Likes)-> Product:t                   #16
    WHERE t.category == "toy"                                       #17
    ACCUM t.@rank += o.@lc;                                         #18
RecommendedToys = ToysTheyLike MINUS ToysILike;                     #19
PRINT RecommendedToys.name, RecommendedToys.@rank;                  #20
```

该查询指定从客户 1 开始遍历，然后在第一个分支语句中沿着边“Likes”找到她喜欢的玩具（第 6 行），并将结果存储在 ToysILike 点集（第 4 行）中。在第二个分支语句中，遍历继续从这些玩具开始，沿着边“Likes”的反向前进，找到其他也喜欢这些玩具的客户（第 10 行）。这些被找到的客户存储在点集“OthersWhoLikeThem”中。接着，在第三个分支语句中，遍历继续从这些客户出发，沿着边“Likes”找到他们喜欢的玩具（第 16 行），并将结果存储在点集“ToysTheyLike”中。请注意一下我们是如何使用某个分支语句的输出点集作为后继分支语句的输入点集，从而将多个分支语句联系在一起的。最终推荐玩具是由点集“ToysTheyLike”和“ToysILike”的差值得到，这样就能避免向客户 1 推荐她已经喜欢的玩具。

伴随这一遍历过程，计算了以下聚合。第二个分支语句计算其他每个客户 o 与客户 1 共同喜欢的玩具数，它的实现方式为，对于客户 o 喜欢的每个玩具，将数值 1 写入 o 的点累加器 o.@inCommon 之中（第 12 行）。这些单位值通过点累加器求和来进行聚合，最终数值表示期望计数。在累加后阶段，计数值为最终值，第 13 行计算客户 o 与客户 1 的对数—余弦相似度。然后，第三个分支语句将每个喜欢玩具 t 的客户 o 的相似度值 o.@lc 写入到每个玩具 t 之中（第 18 行）。在累加阶段最后，t.@rank 包含上述数值的总和，非常精确地表示了最终玩具推荐的排名。

请注意，一个分支语句中的中间计算结果可通过累加器由后续分支语句访问（例如，客户 o 对于客户 1 的对数—余弦相似度由第二个分支语句存储到点累加器 @lc 中，随后由第三个分支语句读取以计算玩具排名）。

控制流

GSQL 的全局控制流原语包括 if—then—else 形式的分支（也包括从 SQL 借用的标准 case 语句）和 while 循环，两者都可由涉及全局累加器、全局变量和查询参数的布尔条件控制。这尤其适用于编写迭代计算，其中每个迭代都会更新全局的中间计算结果，循环控制根据该中间计算结果决定是否退出循环。一个典型的例子是 PageRank 算法变体，该算法持续迭代直至达到最大迭代数，或者直至所有点上的得分计算最大误差（在全局 Max 累加器中聚合）低于一个给定阈值（例如作为参数提供给查询的值）。

循环可自由嵌套在彼此之前，最内层循环的代码是分支语句的有向无环图（循环过程允许循环数据在图上传递）。

查询—调用—查询

GSQL 支持某一查询调用其他具有名称和参数的查询。GSQL 支持类似 SQL 语法的点级、边级和属性级修改（插入、删除和更新）。

图灵完备性

如前所述，GSQL 支持条件分支、循环和存储变量。很明显，GSQL 是图灵完备的。

GSQL 与其他图语言的比较

我们可以将 GSQL 与当今主流的其他图查询语言进行比较。忽略掉特定语法和定义语义的方式，聚焦于按以下几个关键维度分类的表达能力：

1. **累加：**该语言如何存储查询计算得到的数据（包括数据集合和数据聚合）？
2. **多步路径遍历：**该语言是否支持多步路径遍历链，并能够在遍历过程中收集数据？
3. **中间结果流：**该语言是否支持遍历过程中的中间结果在路径上传递？
4. **控制流：**该语言支持哪些控制流原函数？
5. **查询—调用—查询：**该语言是否支持一个查询调用另一个查询？
6. **SQL 完备性：**该语言是否 SQL 完备？即对于任何关系型数据库 D 的基于图的表示 G 来说，在 D 上的任何 SQL 查询是否都能在 G 上用 GSQL 查询来表达？
7. **图灵完备性：**该语言是否图灵完备？

Gremlin

Gremlin 是一种图查询语言，它将图分析设定为作业的管道，管道内流动的内容是对象（称为“遍历者”）流，它们沿着遍历路径演进并收集信息。例如，某次作业可以（在遍历者之中）将一个点替换成它的内邻居或外邻居、内边或外边及其属性等；同时，由遍历者内容计算出来的值也会与变量绑定，帮助“遍历者”延展遍历范围。Gremlin 的这种语义以作业为核心，并具有功能性的编程风格。

Gremlin 是图灵完备的语言，其表达能力与 GSQL 一样棒。我们在下面详细说明了它如何涵盖我们所阐述的表达维度，具体请参阅 Apache TinkerPop3 v3.2.7 文档。

1. 累加

   Gremlin 支持数据的收集（到列表或表中），以及使用用户定义的聚合函数或通常的 SQL 内置聚合进行聚合。与 SQL 中一样，通过定义关系表、指定其分组属性和聚合列，支持 Group-by 聚合。

   - Gremlin 从 SQL 中继承了一个限制：像 SQL 查询一样，Gremlin 查询可以通过一个按组划分属性列表对一个表进行分组，但是不能同时通过两个按组划分属性列表进行分组。如果要对同样的收集数据同时进行两个不同的按组划分聚合，则需要运行详细查询，该详细查询将同一张变量表绑定两次，然后分别对每张表进行分组。与此相比，在 GSQL 中实现上述功能则更加简单，因为 GSQL 会指定对两组同样的数据进行同步计算，只需将每个值分别写入两个不同的分组累加器中即可实现。
   - 遍历过程中计算的中间结果可以通过“sideeffect”操作符（类似于 GSQL 的全局累加器）存储在全局变量中，或是存储在点 / 边的属性（又称为特性）中。

2. 多步路径遍历

   Gremlin 的设计初衷是希望将多步线性路径简洁地指定为包含许多单个遍历步骤作业的管道。

3. 中间结果流

   中间结果可以通过变量绑定方式存储到遍历者对象，然后沿着管道流转（GSQL 也有类似的方式，只不过 GSQL 通过累加器传送数据）。

4. 控制流

   Gremlin 具有 if-then-else 样式的分支和循环，其控制条件不仅可指定全局中间结果，还可指定遍历者的本地数据。

5. 查询—调用—查询

   Gremlin 支持用户定义函数，用于相互调用以及递归。

6. SQL 和图灵完备性

   Gremlin 是图灵完备，因此也是 SQL 完备。

编程风格

Gremlin 有一个特点，即点和边的属性（特性）建模为一张属性图，并且需要导航以达到属性值。若要访问多个属性，需在属性图中执行分支导航（即从一个给定点或边开始，我们需要沿着一个导航分支前往其每一个属性值）。

另一个特点是，经优化的语法可以简洁地表达线性（非分支）路径导航。分支导航通过详细方式实现，首先将变量绑定在分支点处的点上，然后沿着辅分支进行另一次线性导航，最后跳转回分支点并回到主分支上。

这两个特性的交互会导致在需要同时访问同一边 / 顶点的各种属性的查询中，出现令人费解的导航和冗长的查询表达式。我们通过将示例 1 翻译成 Gremlin 来进行说明。

```
toys =
    V().hasLabel('Customer').as('c')                                    (1)
    .values('name').as('name')                                          (2)
    .outE('Bought').as('b')                                             (3)
    .values('discount').as('disc')                                      (4)
    .select('b')                                                        (5)
    .values('quantity').as('quant')                                     (6)
    .select('b')                                                        (7)
    .outV()                                                             (8)
    .has('Product', 'category', eq('toy'))                              (9)
    .as('p')                                                           (10)
    .values('price').as('price')                                       (11)
.select('price', 'disc', 'quant')                                      (12)
.map{it.get().price*(1—it.get().disc/100.0)*it.get().quant}            (13)
    .as('sale_price')                                                  (14)
.select('name','sale_price')                                           (15)
.group().by('name').by(sum())                                          (16)
```

上述程序执行了以下步骤。(1) 检查点是否具有标签“Customer”，如果是，则将变量“c”绑定到此点。(2) 前往它的“name”属性，将变量“name”绑定到该属性。(3) 前往出发边“Bought”，将变量“b”绑定到该边。(4) 前往“b”的“discount”属性，绑定变量“disc”。(5) 回到边“b”。(6) 前往 b 属性“quantity”的值，绑定变量“quant”。(7) 回到边“b”。(8) 前往边“b”的目标点。(9) 检查它是否有标签“Product”以及值为“toy”的属性“category”。(10) 将变量“p”绑定到这个“Product”点。(11) 前往“price”属性的值，将其与变量“price”绑定。(12) 输出变量绑定元组，组件分别为变量“price”、“disc”和“quant”。(13) 对于每个元组，将其绑定到内置变量“it”，并按如下方法计算每笔销售的价格：执行给定的函数，该函数作为参数传递到 map。(14) 将变量“sale_price”绑定到第 13 步计算得出的值。(15) 输出变量绑定元组，其组件分别为变量“name”和“sales_price”。(16) 将这些元组按“name”进行分组，并在“sales_price”栏中计算总额。

请注意，到“price”、“discount”和“quantity”属性值的导航是分支导航，每次都需要返回至边 b（如第 (6) 行和第 (8) 行）[4]。

4 引自 TinkerPop3 文档（2018 年 3 月）：“在早期版本的 Gremlin—Groovy 中，有许多语法糖，用户可以依靠它们使其遍历更加简洁。这些规则中，有很多都使用了 Java 反射，因此并不追求高性能。而在 TinkerPop3 中，这些语法糖都已经被取消，使其支持标准的 Gremlin—Groovy 语法，从而既符合 Gremlin—Java8 语法，又始终保持最高性能。然而，对于那些愿意以性能损失为代价使用语法糖的用户，还有 SugarGremlinPlugin（又称为 Gremlin—Groovy—Sugar）插件可以使用。”在 TinkerPop3 中取消的语法糖中，有一个语法糖可方便程序员在需要点 v 的 name 属性值的位置简单写入 v.name，而非 v.values('name')。

还要注意，在 Gremlin 中，能够绑定变量对于表达此查询（以及我们运行示例中的所有其他查询）至关重要。**目前的一些系统，例如 Amazon 的 Neptune，仅支持无变量的 Gremlin 表达式（请参阅用户指南）而并不包含上述查询的规范。**

最后请注意，若要将示例 3 也重写成 Gremlin 的表达方式，会遇到一个额外难点：像 SQL 查询一样，Gremlin 查询可以通过一个按组划分属性列表对一个表进行分组，但是不能同时通过两个按组划分属性列表进行分组。因此，像示例 3 那样，既按类别分组又按折扣分组，就需要用到一个冗长的详细查询，该详细查询将同一个变量绑定表定义两次，每个变量绑定表都有它自己的分组，最后把两者拼接或者合并（合并的运算结果比较精简）。与此相比，在 GSQL 中实现上述功能则更加简单，因为 GSQL 会指定对两组同样的数据进行同步计算，只需将每个值分别写入两个不同的分组累加器中即可实现（示例 3 第 6 行和第 7 行）。

Cypher

Cypher 是一种基于变量模式的声明性图查询语言，每种模式匹配都会产生一个变量绑定元组。这些元组的集合是一个关系表，可以像在 SQL 中一样操作。

1. 累加

   Cypher 对累加的支持是有限制的。它支持收集数据（收集到列表或表中），并且与 Gremlin 一样，通过定义关系表并指定其分组属性和聚合列，以 SQL 方式计算按组划分聚合。在遍历期间计算的中间结果可以存储到点或边的属性（又称为特性）中，但它们只能是标量或者标量列表类型，因为这是 Cypher 中仅有的属性类型。相比之下，GSQL 可以使用累加器在点存储复杂值的数据（例如元组集合，包括 set、bag、list、heap 或 map）。

   特别地，示例 3 中的查询需要在客户点存储多个元组（每个类别一个，每个折扣值一个），但这在 Cypher 中无法完全实现。对于这两个分组中的任何一个，Cypher 查询最接近的做法不是在每个客户顶点存储累积的复杂值，而是构造一个表，将客户顶点的 ID 与每个组关联起来。Cypher 的这一近似做法继承了来自 SQL 的另一个限制：像 SQL 查询一样，Cypher 查询可以通过一个按组划分属性列表对一个表进行分组，但是不能同时通过两个按组划分属性列表进行分组。因此，按类别分组和按折扣分组需要将同一个变量绑定表定义两次，确保每个分组都是恰当的，最后组合在一起：

```
MATCH (c:Customer) -[b:Bought]-> (p:Product)
WITH c, p.category as cat,
     sum(p.price*(1-b.discount/100.0)*b.quantity) as rev
WITH c, collect({category:cat, revenue:rev}) as byCateg
MATCH (c) -[b:Bought]-> (p:Product)
WITH c, byCateg, b.discount as disc,
     sum(p.price*(1-b.discount/100.0)*b.quantity) as rev
RETURN c, byCateg, collect({discount:disc, revenue:rev}) as byDisc
```

与此相比，在 GSQL 中实现上述功能则更加简单，因为 GSQL 可以指定对两组同样的数据进行同步计算，只需将每个值分别写入两个不同的分组累加器中即可实现。请注意，为了进行高效评估，Cypher 的优化器必须确定两个 Match 子句可重复使用匹配，而不是冗余地重新计算它们。

2. 多步路径遍历

   Cypher 可指定具有变量的多步模式。

3. 中间结果流

   通过引用绑定到中间结果的变量，可沿遍历路径访问中间结果。

4. 控制流

   Cypher 的控制流包括 if-then-else 样式的分支。循环控制是有限的，仅支持在迭代开始之前计算的元素列表上的循环。这可用于模拟给定迭代次数的循环，首先创建相应的整数列表：展开范围 (1,30)，因为 iter 使用迭代变量 iter 指定 30 个迭代循环，并且展开 L 作为 E 指定在列表 L 的元素上进行迭代，将变量 E 绑定到每个元素。

   上文已经提及，在 PageRank 版本中，迭代持续运行至最大误差值低于某个阈值，需注意的是，这个过程依赖于循环，而循环的控制条件取决于迭代期间更新的中间结果。这种循环在 Cypher 中不受支持，并且相应的算法类也无法表达（例如，PageRank 是作为 Java 实施的内置原语提供的）。

5. 查询—调用—查询

   Cypher 允许用户定义函数，但必须参照 Neo4j 开发者手册在 Java 中实施。这种设计有两个问题。首先，程序员必须同时熟练使用 Cypher 和 Java。其次，用户定义的函数对于 neo4j 优化器来说是黑盒，因此无法在调用查询的过程前后进行优化。

6. SQL 和图灵完备性

   Cypher 是 SQL 完备，但不是图灵完备。上面提到的循环控制限制是该语言不能达到图灵完备的原因之一。这里忽略了 Neo4j 支持的任意用户定义函数。由于用户定义函数是 Java 实施的，而 Java 是图灵完备的，所以这个问题对于 Java 来说就不是问题。

SPARQL

SPARQL 是声明性查询语言，针对语义图数据而设计，其中互联数据以资源描述框架 (RDF https://www.w3.org/RDF/) 形式表示。RDF 的主要目标是使机器理解网页内容。SPARQL 支持对语义 Web 的更高级别查询。RDF 和 SPARQL 随时间进行了扩展，可管理更大范围的互联数据。也就是说，SPARQL 并非针对属性图模型（点实体和边实体，每个实体均具有特征属性）而设计。

RDF 以三元方式定义数据：< 主语 >< 谓语 >< 宾语 >。一个三元组描述两个实体之间的关系或者实体与其一个属性之间的关系。每个三元组可被视为图中的一条边：< 源 _ 点 >< 边 >< 目标 _ 点 >，因此 RDF 数据集是一个图。考虑以下三元组：<John Smith>< 教课给 ><Calculus>。“John Smith”和“Calculus”分别为主语和宾语。“教课给”是指示他们之间关系的谓语。考虑另一个三元组 <John Smith>< 性别 >< 男性 >。“John Smith”是主语，“Male”是属性。谓语“性别”说明主语与其属性之间的关系。

SPARQL 语法借用了 SQL 的许多关键词和设计理念。例如，

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title>
?title .
}
```

就像 SQL 一样，SELECT 从基础数据集（三元存储）投影一系列表达式。WHERE 子句以三元或路径形式保存图模式列表。用户可在主语或宾语位置使用 ? 或 $ 以寻找匹配并将变量绑定到这些位置。

1. 累加

与 SQL 一样，SPARQL 提供 COUNT、SUM 等聚合运算符以及 GROUP BY 子句。查询可选择一系列三元组并将其聚合。一种方法是定义命名图以存储中间结果。这与 Cypher 的 WITH 子句和 SQL 中的 VIEW 概念类似。然后，用户可从命名图的 FROM 子句中进行选择。相应限制以及与 GSQL 的差异与在 Cypher 中所述的相同。示例：

```
CREATE GRAPH <urn:example/g/g1>
INSERT {

        GRAPH <urn:example/g/g1>
        {?s  <urn:example/predicate/p1>  ?tot}
}

WHERE {
        SELECT ?s (count(?t) AS ?tot
       WHERE {  ?s <urn:example/predicate/p0> ?t .}
}
```

2. 多步路径遍历

SPARQL 可通过声明一系列三元组来指定多步模式，这些三元组通过通用变量而互相关联。以下示例提供了 2 步路径 ?title—<title>—?book—<price>—?price。它选择具有价格的所有书籍的书名。使用多个三元组与具有多个 SELECT 分支语句的 GSQL 查询类似，其中每个分支语句描述一步。SPARQL 语法更加简洁，但 GSQL 结构可更轻松地自定义每一步的操作。

```
SELECT ?title
WHERE
{
  ?book  <http://purl.org/dc/elements/1.1/title> ?title .
  ?book <http://purl.org/dc/elements/1.1/price> ?price .
}
```

3. 中间结果流

   中间结果可通过以下方式访问：(1) 命名图（类似于关系型数据库中的实体化视图）中的 FROM 子句，引用绑定的变量，或 (2) 子查询。由于 SPARQL 查询评估具有自下而上的性质，因此将先以逻辑方式评估子查询，评估结果将投影到母查询。

4. 控制流

   SPARQL（从 1.1 版开始）不支持迭代流控制。用户必须依靠外部主语言（如 JavaScript）来构建循环。

5. 查询—调用—查询

   SPARQL 语法允许用户定义函数，并将这些函数插入 WHERE 子句模式。但是，它本身不支持可被另一个查询调用的命名存储过程。

6. SQL 和图灵完备性

   SPARQL 是 SQL 完备，但不是图灵完备[5]。上面提到的循环控制限制是该语言不能达到图灵完备的原因之一。

结语

GSQL 还有一个区别于其他图查询语言的功能，这一功能可增强查询优化和安全性。

GSQL 假定一个预定义模式，并将其作为元数据存储在数据库目录中。这可提高存储和访问效率，因为模式元数据可从点 / 边 / 属性表示中分解出来。这使性能得以提高，不仅因为节省了空间，还因为在预先更深入了解数据模型后，可进一步优化查询：更多筛选、更好的排序、更多捷径等。在优化时间方面运用预定义的模式是一种久经考验的数据库技术，能够一次又一次地提高性能。

此外，为了安全和隐私的目的，运用这种预定义的模式可提供图模式认知访问控制。GSQL 的多图功能允许在全局模式中定义无限个子图模式，此外，它使用子图名称作为授予权限的额外因素，从而增强传统的基于角色的访问控制。此功能在实际应用中非常重要，因为它实现了多用户功能（例如公司中的多个部门），即针对同一个图数据库，不同用户拥有不同访问权限。这是目前业内首个且唯一一个提供此类支持的语言。

GSQL 拥有达到甚至超越了其他图查询语言的表达能力。虽然基本编程范式只是一个偏好问题，但 GSQL 经过精心设计，可兼容 SQL 和 NoSQL 的 Map—Reduce 编程范式，使其对于上述两个程序员社区都具备吸引力。

5 “SPARQL 的表达能力” https://dl.acm.org/citation.cfm?id=1483165

第 10 章

实时深度关联分析的实际运用

作为全球首个且唯一一个提供实时深度关联分析功能的原生并行图数据库，TigerGraph 拥有独特优势，能够解决当今一些业务挑战。在本章中，我们将介绍一些实际用例，并说明实时深度关联分析在目前如何提供实际解决方案和实际业务价值。

通过图分析打击欺诈和洗钱

自从货币存在以来，就出现了非法钱财和洗钱犯罪。据联合国估计，每年全球的洗钱金额达 20 亿美元[6]。如今的犯罪极其复杂，犯罪团体不断变换技巧来逃过传统反欺诈系统的监管。虽然企业有时能拥有足够数据来发现非法活动，但更多情况下，他们无法开展分析来揭示非法活动。

6 https://www.unodc.org/unodc/en/money—laundering/globalization.html

随着打击洗钱活动的深入开展，AML（反洗钱）合规变得极为重要。WealthInsight 调查表明，全球仅反洗钱支出就超过了 80 亿美元[7]。鉴于任何提供财务交易的组织也属于反洗钱法律范围，这一数字将继续增长。

但打击犯罪绝非易事。尤其是为了避免监管费用，组织现在迫切需要降低成本并加快实现反洗钱合规。传统监控系统经证明在调整、验证和维护方面既繁琐又昂贵。传统系统常常涉及手动流程，通常无法分析海量的客户、机构和交易数据。然而，这种数据分析对于反洗钱成功至关重要。

现在出现了新的方式来解决反洗钱挑战。这包括半受控学习方法、基于深入学习的方法以及基于网络 / 图的解决方案。此类方法必须能够实时工作并处理海量数据，尤其是在每时每刻都有新数据生成之时。因此，最好运用全面的数据策略来打击金融犯罪，尤其是采用机器学习 (ML) 和人工智能，以帮助关联并分析数据关联。

用于反洗钱的图分析

图分析作为支持反洗钱的理想技术出现在前沿领域。图克服了挑战，能够在海量、复杂且互联的数据中发现关系。图模型是完完全全的图，将关系视为“一等公民”。这提供了一种原生支持和映射数据关系的结构，即使在高度互联的海量数据中也是如此。图分析基于此类互联数据开展，可充分洞察数据关联和关系。

例如，“度中心性”提供进出每个实体的关联数。此指标可统计每个实体与网络中的其他实体之间存在多少直接关联。这尤其有助于寻找关联十分广泛的客户或实体，他们可能会充当中枢的角色，并与更广泛的网络相连。

另一个是“中介”(Betweenness)，可以给出实体落在其他实体间最短路径上的次数。此指标可显示哪一个实体充当其他实体之间的桥梁。中介可以成为侦查洗钱或可疑活动的起点。

当今的组织工作需要实时图分析能力，以便探究、发现和预测非常复杂的关系。这代表着实时深度关联分析，它通过利用 3 到 10 步以上的大图遍历，以及快速遍历和数据更新来实现。

我们来看一下实时深度关联分析如何通过辨识高风险交易来打击金融犯罪。我们将从一笔转入信用卡交易开始，演示如何辨识这笔交易与其他实体之间的关系：

```
New Transaction → Credit Card → Cardholder → (other) Credit Cards → (other)
Bad Transactions
```

该查询使用 4 步来寻找与转入交易只有一卡之隔的关联。现今的欺诈者试图将自己与已知不良行为或不良行为者之间建立迂回关联，藉以掩盖自己的活动。路径中的任何个人都可能看上去清白无辜，但如果能够在 A 到 B 之间发现多条路径，则存在欺诈的可能性就会增加。

有鉴于此，成功的反洗钱需要具备多步遍历的能力。这种遍历模式适用于许多其他使用情形，只需将交易替换为网页点击事件、电话记录或汇款即可。借助实时深度关联分析，可以揭示多个隐藏的联系，从而最大限度减少欺诈。

通过将数据关联在一起，实时深度关联分析可以实时支持基于规则的机器学习方法，以便自动执行反洗钱程序，减少误报。企业使用图引擎将诸如自动化数据流分析、社交网络分析和机器学习等复杂的数据科技融合到他们的反洗钱程序中，可以利用更好的数据更加迅速地提高洗钱侦查率。他们还可以远离繁琐的事务性流程，转向更具战略性和更高效的反洗钱方法。

7 https://www.marketresearch.com/product/sample–7717318.pdf

示例：电子支付公司

为了举例说明用于支持反洗钱的图分析技术，我们可以将目光投向全球最大的电子支付公司。这家组织目前拥有超过 1 亿日活跃用户，并且采用图分析技术实现了调查方法的现代化。

此前，该公司的反洗钱实践是一项非常依靠人工的工作，因为从检查数据到识别可疑的资金运动行为，需要调查人员全面参与。运营支出很高，整个过程也非常容易出错。

通过实施图分析平台，该公司成功地使用基于机器学习的实时响应源，实现了智能反洗钱查询的自动化开发。最终，他们通过使用更有效的反洗钱程序、减少误报和转化为更高的侦查率，获得了高经济收益。

示例：信用卡公司

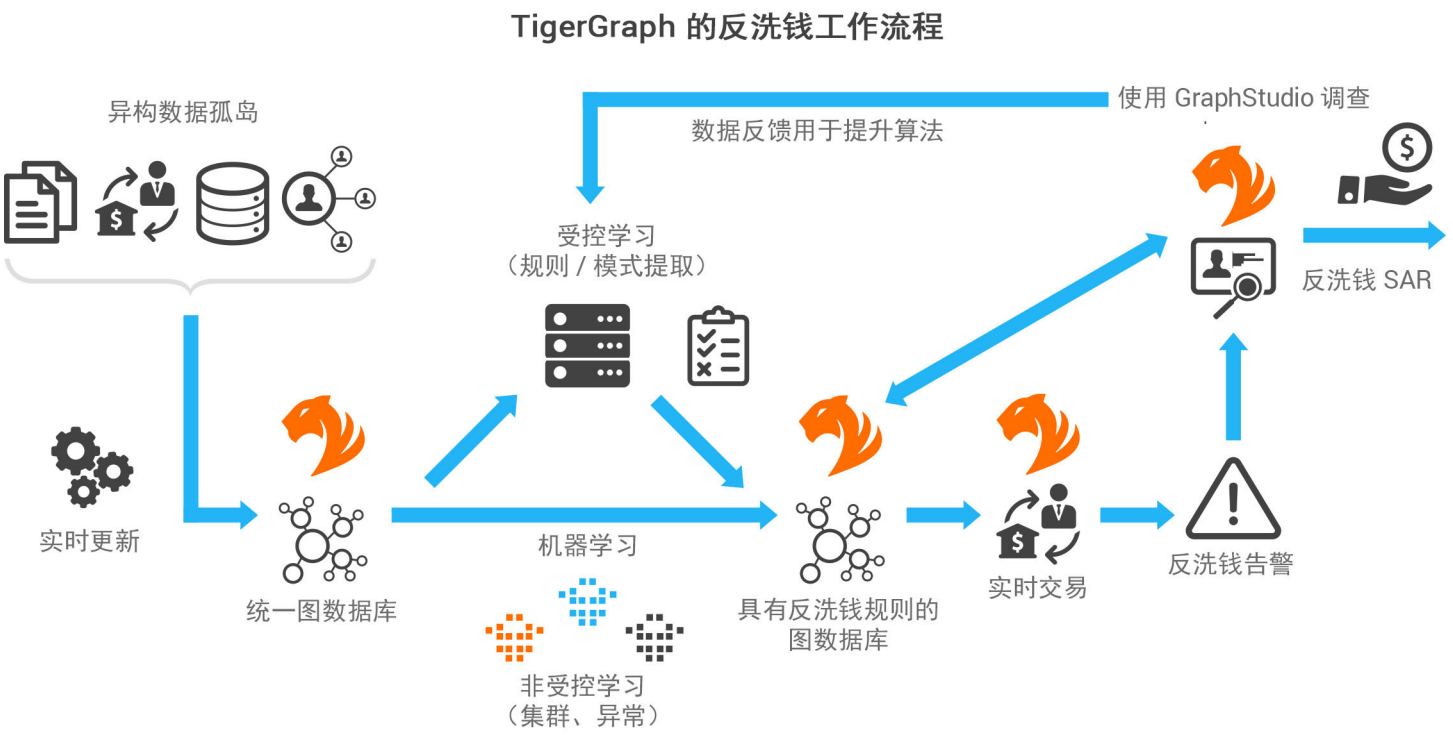
同样，一家排名前五的支付服务提供商也寻求提高他们的反洗钱能力。主要痛点包括成本高企，以及因无法遵守美国联邦反洗钱法规而招致罚款。该组织过去依赖一个机器学习团队来执行人工调查程序，该团队由数百名调查人员组成，结果导致整个过程速度缓慢、成本高昂且效率低下，误报率超过 90%。

这家公司现在利用图引擎来实现调查程序的现代化。他们不再依赖机器学习团队来拼凑琐碎的调查程序，转而将图分析技术的力量与机器学习相结合，以便洞察个人、帐户、公司和位置之间的关联。

通过合并更多数据维度并融合额外数据点（如与客户有关的外部信息），他们能够自动实时监测潜在的洗钱活动，使调查人员腾出手来将更趋巨大的数据用于更加具有战略意义的用途。最终，他们可以对其庞大的数据量进行全面而富有洞察力的观察，产生较少的误报警报。

随着我们进入数据爆炸的时代，对于组织而言，实时、充分地分析他们的海量数据以用于反洗钱目的变得越来越重要。在降低成本、迅速确保反洗钱合规，以及阻止洗钱欺诈者的能力方面，图分析技术可以为组织带来巨大的潜力。

图分析技术如何助力电子商务



我们已经进入了电子商务统治零售业的时代。考虑一下行业报告中所做的如下预测：线上销售规模到 2020 年将超过 4 万亿美元，占全球零售总支出的 14.6%（来源：eMarketer[8]）。过去数年间，线上购物转变了我们的购买方式，带来了新的“消费者时代”。今天，购物者围绕产品和品牌可以获得比以往更多的信息，全都有助于他们做出合理的购买决策。他们自己的线上购物体验也领先潮流，全球市场尽在指尖。

相应地，电子商务也在拥抱人工智能助手、推荐引擎，甚至自动化平台，来帮助消费者考虑应该估量和购买什么。人工智能平台生成的洞察可以提供巨大的价值，还可能为公司带来进一步的收益，以及为购物者提供更好、更定制化的客户体验。

传统实体店依靠（并且将继续依靠）与客户建立关联来进行销售，表现为识别和评估他们的需求、向他们推荐商品以及带领他们完成销售过程。就线上销售而言，关联的作用更大，因为当今最成功的零售商都会提供以客户为中心的电子商务平台。这其中涉及统一的多路径购买体验，为此需要以 360 度全景视角看待每一位客户。通过将客户数据和活动转化为行动情报，在客户购物期间为其提供产品推荐等支持，可以带来更高的转化量和客单量。

成功利用这些数据的关键在于，首先在客户的实时购物会话期间获得数据，并且能够分析数据，更不必说实时做到这一切了。大多数电子商务零售商能够收集数据，但同时也发现传统分析解决方案无法做到这一点。在过去，解决方案过于慢速或昂贵，而且通常无法从海量客户、交易和外部数据中得出复杂的洞察。

实时优惠（不论是库存产品、有条件促销，还是为促成销售而提供的包邮服务）需要了解客户，而且要清楚地了解。这意味着电子商务网站需要尽可能快速地收集和利用消费者细分数据，提供针对性的推荐和客户服务。最终，他们能够提供更个性化的购物体验，使顾客享受其中、乐于交互，并且不断回来购买商品以满足需求。

8 https://www.emarketer.com/Article/Worldwide-Retail-Ecommerce-Sales-Will-Reach-1915-Trillion-This-Year/1014369

带来更好的线上购物体验 ... 以及促进收益增长

实现这一目标并不容易，使用通常以表来存储数据的传统数据库时尤其如此。因为此原因，越来越多的零售商开始借助图数据库技术的力量来支持实时分析等。在零售领域，对于关联有着至关重要意义的行业来说，图数据库是理想的选择。这是因为图数据库是从头开始设计的，它将关系当做“一等公民”来对待，可以提供一种原生拥抱数据间关系的结构。这可以确保比以往任何时候都更好地存储、处理和查询关联。

例如，请考虑一种简单的个性化推荐，如“与您喜好相同的顾客还购买了以下商品。”从一个人开始，图数据库中的查询第一步先辨识查看 / 喜欢 / 购买过的商品。第二，寻找其他查看 / 喜欢 / 购买过这些商品的人。第三，找出这些人购买过的其他商品。

```
Person → Product → (other)Persons → (other) Products
```

为了支持这类查询，在过去数年中，图数据库在企业中的采用率大幅上升，特别是那些运营电子商务网站的零售商。虽然第一代图数据库解决方案有助于解决电子商务数据问题，但它们并非没有局限性。数据的查询速度和分析相对较慢，而且图引擎只能在一定程度上支持遍历（访问或检查并 / 或更新数据点的过程）。更深度的遍历可以让您从数据中得出更好的洞察。

实时深度关联分析助力电子商务

近来，我们看到图数据库进化到了下一个阶段，新技术通过提供深度关联分析来满足电子商务的需求。这可以实时获取客户情报并实现强大的关系分析。借助这些实时能力，电子商务网站可以快速综合并理解客户的行为。最终，他们可以捕获关键的“商业时刻”(Business Moment)，商业时刻是指一些转瞬即逝的机会，其中人、业务、数据和“物”动态地共同创造价值，可用来实现顾客体验的个性化，最终带来更多交易。

这是通过支持超过 10 步（或上文所述的数据点遍历）的新图数据库技术实现的，而不会像第一代解决方案一样受限于 2 步。不论新顾客还是回头客，每多一步都会增加更多有关各个用户购物历史记录的情报。分析数据 / 产品 / 客户位置 / 基于位置的天气 / 推荐的产品时需要拥有遍历数据的能力，才能实时给出推荐。

上文的查询需要实时三步计算，因此超过了现有图技术在处理大型数据集时的两步局限性。加入其他关系后，可以轻松将查询扩展到四步或更多步。这些数据源源不断供给人工智能和机器学习算法，帮助带来更好的线上购物体验。

深度关联分析使电子商务公司能够遍历这些数据，打造更智能的人工智能和机器学习，以便形成竞争优势、达成更多交易和建立客户忠诚度。这一切都可以归结为，能够通过考虑数据之间的更多关联得出更好的洞察，以便充分考虑购物者的行为和偏好。

深度关联分析用于推荐产品

示例：移动电子商务公司

我们来考虑一个当今的零售商使用深度关联分析的示例。一家大型的全球性电子商务平台使用深度关联分析来为他们庞大的产品目录建模、对消费者实体数据绘图，并对他们复杂海量的数据执行实时分析。最终，这实现了为每位购物者提供个性化的实时推荐，促进了销售和收益的增加。

在技术层面，相比公司之前的内部解决方案，该解决方案将查询速度提高了 100 倍，查询响应时间在 100 毫秒以内。此外，通过基于图的高效编码和压缩，还将内存用量节省了 10 倍。由于所需的计算器数量减少，因此还帮助降低了硬件管理和维护成本。

数据科学家和机器学习专家能够在一个大图中综合公司的所有关联数据，因此可以更快完成更多工作。企业可以从其海量数据中得出洞察，用以支持做出更好的决策和缩短上市时间。图数据库平台还可以提供所需要的速度和扩展性，以便充分利用数据关系来获得竞争优势。

客户关联

零售商发现，不论是与实体店内购物的消费者当面建立关联，还是与线上的消费者建立关联，二者同样重要。为了实现这一目的，对于电子商务来说，设法扫描海量的数据并充分理解它们正变得越来越重要。

今天，这比以往任何时候都更有可能做到，特别是使用图数据库时，而数据之间的关联正是图数据库的前沿领域。随着图数据库的成熟，它们支持对关联更广泛的数据进行查询，帮助得出更深度的洞察。客户将会享受更好的购物体验，而零售商则会受益于更开心、更投入的购物者，以及更多的销售和收益。

电力系统现代化

创建超实时能源管理系统 (EMS) 是电力行业面临的巨大挑战。超实时意味着 EMS 必须能够在一个数据采集与监视控制 (SCADA) 采样周期内完成执行，该周期通常为 5 秒。

超实时 EMS 能力是时刻察觉电力系统事件的关键所在，有助于防止电力系统发生大停电。十多年来，电力系统工程师一直无法成功地为大规模电力系统开发速度更快的 EMS 应用。如果成功，将会帮助控制中心的操作人员，使电力系统的运行变得更加安全和符合成本效益。截至目前，尚未出现能够以超实时水平运行的商用 EMS。

在技术上，妨碍 EMS 应用计算效率的挑战包括：A) 电力系统为了满足日益增长的需求而不断扩大规模，B) 高可再生性能源的渗透、FACTS 设备和 HVDC 输电线增加了电力系统模型的复杂性，以及 C) 电力需求和供应的急剧波动与快速响应设备相结合，导致需要更加频繁和密集地进行重新计算。

并行计算是可以加快 EMS 应用速度的突破性解决方案。电力系统工程师研究了基于关系型数据库结构的不同的并行计算方法，以求改善 EMS 应用的计算效率，但迄今仍无法实现超实时 EMS。

并行计算是可以加快 EMS 应用速度的突破性解决方案。电力系统工程师研究了基于关系型数据库结构的不同的并行计算方法，以求改善 EMS 应用的计算效率，但迄今仍无法实现超实时 EMS。

示例：国家公用事业公司

通过使用 TigerGraph 的原生并行图数据库，以及“点做计算引擎”的设计，中国国家电网 (SGCC) 实现了超实时 EMS 原型，并且得到了省级电力系统使用案例的验证。

传统上，电力系统是使用关系型数据库在相互关联的表集合中建模的。由于电力系统的不同组成部分存储在单独的表中，因此需要使用共享键值关联在一起，以便对电力系统的关联性和拓扑建模。根据案例研究的结果，联系或关联不同的表（数据库联接）大约需要电力潮流计算总处理时间的 25%，以及电网状态估计总处理时间的 35%。

大型线性方程的标准解法需要庞大、耗时的矩阵运算。采用图（而不是矩阵）的形式来对电力系统建模可以更自然地表示关联和拓扑。无需准备数据，可以将电力潮流计算和状态估计通常需要的时间减少 25—35%。母线编序和导纳图的生成是采用点并行（所有点同时计算，确保编序和导纳图并行不悖）方式执行的。符号分解和数值分解以及向前 / 向后替换是采用分层并行方式执行的，其中将点分成不同的层次。同一层次的点以并行方式进行计算，整个计算过程从最低层次开始。核心计算全部在图上进行，而且解出的值作为图点和边的属性来存储，而不是向量或矩阵中的未知变量。

在传统方法中，电力系统问题作为未知变量进行解析，并分配给 X 向量。为了使解出的值可视化，需要通过映射过程将未知变量链接到显示屏。

通过使用 TigerGraph，解出的值作为图上点和边的属性来存储，无需映射过程。根据测试案例的结果，在使用传统方法时，输出可视化大约需要电力潮流计算总用时的 70%，以及状态估计总用时的 28%。使用 TigerGraph 图数据库和图计算时，这部分时间可以消除。

此表概括了超实时 EMS 与 D5000 的测试结果比较，后者是广泛用于中国及其他许多国家 / 地区大多数控制中心的商用 EMS。

基于实际使用的省级 2650 母线系统测试			
测试系统	状态估计	电力潮流	静态安全分析
商用 EMS	4488 毫秒	3817 毫秒	18000 毫秒
基于 TigerGraph 的 EMS 原型	172 毫秒	79 毫秒	772 毫秒

EMS 三大应用（状态估计、电力潮流和静态安全分析）的执行时间**总和**略高于 1 秒，远低于标准 SCADA 采样周期 5 秒。在 TigerGraph 的帮助下，中国电网实现了第一个可行的商业用途超实时 EMS 解决方案。

超实时能源管理系统 (EMS) 已经成为电力管理的“圣杯”，特别是当任何停电事故都会直接影响一国的生产力和经济产出 /GDP（国内生产总值）时。这样一种系统可以发现电力需求和供应之间的错配，降低电网非关键部分的电力消费，将电力转移到高优先级的工业产出和国家安全领域，而且这一切都可以在 1 秒内完成。

第 11 章

图和机器学习

> ***从许多方面来说，欺诈侦查如同大海捞针。您必须整理并理解海量的数据，才能找到那根“针”，在本例中是指欺诈者。***

下面我们来看一家电话公司的示例，这家公司的网络上每周都会发生数十亿计的电话呼叫。他们如何能够从堆积如山或浩如烟海的通话记录中发现欺诈活动的迹象？这便是机器学习的用武之地，它可以提供一块巨大的“磁石”来帮助寻找那根“针”，在本例中，这块“磁石”就是发现疑似欺诈者的行为和模式的能力。通过使用图模型，计算机将更加熟练地识别可疑的电话模式，而且能够从数十亿计海量正常通话数据中将其分离出来。

事实上，越来越多的组织利用机器学习及图技术来防止各种类型的欺诈，包括电话诈骗、信用卡退单、广告、洗钱等。在进一步探讨机器学习与图技术这一强大组合的价值之前，我们先看一下当前基于机器学习的欺诈者识别法是如何错失目标的。

训练数据的质量决定了机器学习算法的上限

为了侦查某一具体的情况，如从事诈骗的电话或涉嫌洗钱的付款交易，机器学习系统需要足够数量的欺诈电话或可能与洗钱相关的支付交易。下面我们以电话欺诈为例深入分析。

除可能属于欺诈的电话数量外，机器学习算法还需要与电话欺诈行为高度相关的特征或属性。

由于欺诈（与洗钱非常相似）在交易总量中所占的比重不到 0.01% 或万分之一，因此，存在确认欺诈活动的训练数据体量非常小。相应地，数量如此之少的训练数据将导致机器学习算法的准确度不佳。

选择与欺诈相关的一些特征或属性十分简单。就电话欺诈来说，这些特征或属性包括某些电话呼叫其他网内网外电话的历史记录、预付费 SIM 卡的卡龄、单向呼叫（即被呼叫方未回电）所占的百分比，以及被拒呼叫所占的百分比。同样，为了查找涉嫌洗钱的付款交易，需要为机器学习系统提供诸如付款交易的规模和频率等特征。

但是，由于依赖仅侧重于各个点的特征，导致误报率居高不下。例如，频繁进行单向呼叫的电话可能属于销售代表所有，他们需要致电潜在客户寻找销售线索或销售商品和服务。这种呼叫也可能涉嫌骚扰，是一方对另一方的恶作剧。大量的误报会造成浪费精力去调查非欺诈电话，最终降低对欺诈侦查机器学习解决方案的信心。

算法好不如数据多

在机器学习领域有一个很流行的说法：“算法好不如数据多”。很多机器学习就是因为缺乏充足的训练数据而失败的。简单来说，样本大小直接影响着预测的质量。与海量的交易相比（订单、付款、电话呼叫和计算机访问日志），诸如欺诈、洗钱或网络安全违规等异常检测事件的确认量很低。

从 Uber、支付宝到 Wish.com 和中国移动，很多大型客户使用 TigerGraph 来计算机器学习领域所谓的基于图的属性或特征。就中国移动来说，TigerGraph 为其 6 亿个号码分别生成 118 项新特征。这将创造超过 700 亿项新特征，用于将存在疑似欺诈活动的“坏号码”与其余属于普通用户的“好号码”区分开来。以中国移动为例，将会有更多训练数据，即可以生成 700 亿项特征，供机器学习解决方案提高欺诈侦查的准确性。

为电话欺诈打造更好的“磁石”

很多现实生活中的示例不断证明着图技术和机器学习在打击欺诈方面的价值。目前，一家大型移动运营商正使用具备实时深度关联分析的新一代图数据库，解决现有机器学习算法训练方法的缺陷。该解决方案分析了 6 亿部手机的超过 150 亿通呼叫，最终为每个手机生成了 118 项特征。这些特征基于对通话记录的深度分析，范围不限于直接被呼叫方。

下图说明了图数据库如何识别“好”号码或“坏”号码。图数据库解决方案还可以识别疑似欺诈的类型（例如，垃圾邮件广告、诈骗销售等），并且在被呼叫人的手机上显示警告消息，这一切全部都在手机接通之前完成。

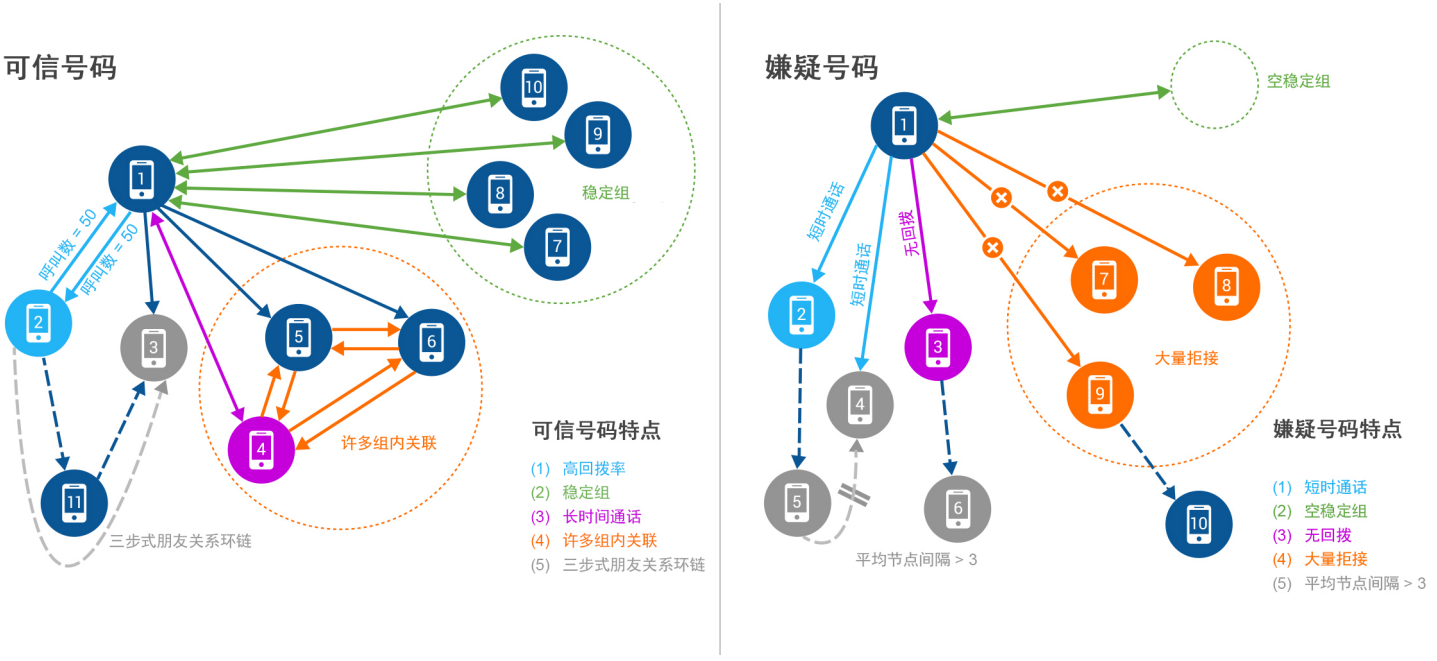


图 1 — 通过分析网络或图关系特征来检测电话欺诈

拥有好号码的用户致电其他用户，大多数人都回复他们的电话。这有助于指示用户之间的熟悉度或信任关系。好号码还会定期拨打一组其他号码（比如，每天或每月），这一号码组在一段时间内非常稳定（“稳定组”）。

指示好号码行为的另一个特征是，当呼叫已经入网数月或数年的号码时得到回电。我们还看到，在好号码、长期联系号码及网内与二者频繁联系的其他号码之间有着大量呼叫。这表明我们的好号码具有很多组内关联。

最后，“好号码”通常会参与三步式朋友关联，意思是我们的好号码会呼叫另一号码，即号码 2，后者将呼叫号码 3。好号码还会通过直接呼叫与号码 3 联系。这表示着一种三步式朋友关联，形成信任和相互关联性圆环。

通过分析号码之间的这类呼叫模式，我们的图解决方案可以轻松识别坏号码，即可能涉嫌诈骗的号码。这些号码会短暂呼叫多个好号码，但不会收到回电。此外，它们也没有定期呼叫的稳定号码组（即“空稳定组”）。当坏号码呼叫长期网内用户时，对方不会回电。坏号码的很多呼叫还会被拒绝，而且缺乏三步式朋友关系。

图数据库平台利用超过 100 项图特征（如稳定组），它们与我们使用案例中的 6 亿移动号码各自的好坏号码行为高度相关。相应地，它可以生成 700 亿项新的训练数据特征，供机器学习算法使用。最终提高了欺诈侦查机器学习的准确率，同时减少误报（即非欺诈号码被标记为潜在欺诈者号码）和漏报（即未标记出参与欺诈的号码）。

为了了解基于图的特征如何提高机器学习的准确率，我们来看一个示例（图 2），其中使用了以下四位移动用户的侧写：Tim、Sarah、Fred 和 John

基于图的特性提高机器学习的准确性

SIM 卡使用时间	2 周	4 周	3 周	2 周
单向通话比例	50%	10%	55%	60%
被拒接比例	40%	5%	28%	25%
机器学习根据历史记录所预测的结果	疑似欺诈者	普通客户	疑似欺诈者	疑似欺诈者
稳定组	是	是	否	否
许多组内关联	否	是	否	是
三步式朋友关系环链	否	是	否	是
机器学习根据深度链路图特性所预测的结果	疑似恶作剧者	普通客户	疑似欺诈者	疑似销售人员

图 2 — 使用图特征提高机器学习的准确率

按照传统的通话记录特征，如 SIM 卡龄、单向呼叫的百分比以及被拒绝的呼叫总量百分比），四人中的三人（Tim、Fred 和 John）将被标记为疑似或潜在欺诈者，因为从这些特征来看，他们非常相似。经过分析基于图的特征，以及号码和用户之间的深度关联或多步关系，最终帮助机器学习将 Tim 归类为爱恶作剧者、John 为销售人员，而 Fred 则被标记为疑似欺诈者。我们来思考一下这个过程。

就 Tim 来说，他有一个“稳定组”，这意味着他不太可能是销售人员，因为销售人员每周都会拨打不同的电话号码。Tim 没有很多组内关联，这意味着他可能经常给陌生人打电话。他也没有任何三步式朋友关联，用于确认他所呼叫的陌生人不存在关联。根据这些特征判断，Tim 很可能是爱恶作剧者。

我们来思考一下 John 的情况，他没有稳定组，这意味着他每天都通过电话寻找新的潜在销售线索。他会给具有很多组内关联的人打电话。当 John 介绍产品或服务时，如果接听方对它们感兴趣或认为与自己相关，则其中一些人很可能会将 John 介绍给其他联系人。John 还通过三步式朋友关系与他人产生关联，这表明他作为优秀的销售人员将整个环链闭合，通过在同一组内第一次联系的人的朋友或同事当中遴选，找到最终的买家来购买他的产品或服务。依据这些特征的组合，最终将 John 归类为销售人员。

就 Fred 来说，他既没有稳定组，也不与具有很多组内关联的群体交流。此外，他与所呼叫的人之间也没有三步式朋友关系。这使得他非常容易成为电话诈骗或欺诈的调查对象。

回到我们最初海底捞针的比喻，在本例中，我们可以利用图分析改善机器学习，进而提高准确率，最终找到那根“针”，即潜在的欺诈者 Fred。为此，需要使用图数据库框架对数据进行建模，以便能够识别和考虑更多特征，用于进一步分析我们的海量数据。相应地，计算机将利用越来越准确的数据进行训练，使自己不断变得聪明，更加成功地识别潜在的诈骗分子和欺诈者。

为反洗钱打造更好的“磁石”

除了识别电话诈骗外，图技术和机器学习还用于大量欺诈侦查案例。机器学习算法正在接受训练，以求侦查各种其他类型的异常行为，如识别潜在的洗钱活动。

根据普华永道的报告，全球洗钱交易估计占全球 GDP 的 2%—5%，每年大约达到 1—2 万亿美元。洗钱风险存在于整个金融服务生态系统，包括银行、支付服务提供商，以及诸如比特币和瑞波币等新型数字加密货币。考虑到每时每刻都在发生大量的金融活动，那么，如何才能做到大海捞针（即欺诈者）呢?

基于规则的传统方法侧重于各个点（如存疑的支付或用户）的属性或特征，但这往往会导致大量的误报。同样的数据可能会供机器学习算法使用，结果导致机器学习系统对未来欺诈预测的准确率不佳 — 输入的数据差，输出的结果也会差!

可以预料，欺诈者会掩盖自己的活动，在自己与已知不良行为或不良行为者之间建立迂回关联。这就是图技术的价值所在，它可以在数据关联和关系中识别和寻找特征，以便为机器学习提供更完善的信息，对其进行更好地训练。这些特征可能包括支付交易的规模和频率，或者，它们可能会基于数据之间的关系而更加抽象。

例如，基于图的方法可以揭示点之间在语义上有意义的关联路径。下面我们通过一笔转入的信用卡交易，演示一下如何能够识别它与其他实体的关系：

```
New Transaction → Credit Card → Cardholder → (other)Credit Cards → (other)Bad
Transactions
```

该查询使用 4 步来寻找与转入交易只有一卡之隔的关联。路径中的任何个人都可能看上去清白无辜，但如果能够在 A 到 B 之间发现多条路径，则存在欺诈的可能性就会增加。有鉴于此，成功的反洗钱需要具备多步遍历的能力。通过这种方式，实时深度关联分析可以在揭示多个隐藏关联，以便最大限度减少洗钱方面发挥价值。

第二，利用图技术支持的方法，可以使用基于图的统计来衡量点、关联和路径的全局相关性。例如，中介特征可以给出实体落在其他实体间最短路径上的次数。此指标可显示哪一个实体充当其他实体之间的桥梁。中介可以成为侦查洗钱或可疑活动的起点。中介分数高可能表明某人或事物是欺诈集团或洗钱掩藏的中间人。

同样，*度中心性* 和*社区检测*等基于图的分析可以为原本不显著的数据点加入必要的着色。度中心性提供进出每个实体的关联数，可以统计每个实体与网络中的其他实体之间存在多少直接关联。这尤其有助于寻找关联十分广泛的客户或实体，他们可能会充当中枢的角色，并与更广泛的网络相连。社区发现通过比较组内关联与组间关联的相对密度，来寻找同一网络内的自然分组。与已知犯罪分子同属一个小型社区的人也是犯罪分子的可能性更大。通过将数据关联在一起，图分析技术可以实时支持基于规则的机器学习方法，从而自动执行反洗钱 (AML) 程序，减少误报。企业使用图引擎将诸如自动化数据流分析、社交网络分析和机器学习等复杂的数据科技融合到他们的反洗钱程序中，可以利用更好的数据更加迅速地提高洗钱侦查率。他们还可以远离繁琐的事务性流程，转向更具战略性和更高效的反洗钱方法。

示例：电子支付公司

为了举例说明用于支持反洗钱的图技术和机器学习，我们可以将目光投向全球最大的电子支付公司。这家组织目前拥有超过 1 亿日活跃用户，并且采用图分析技术实现了调查方法的现代化。

此前，该公司的反洗钱实践是一项非常依靠人工的工作，因为从检查数据到识别可疑的资金运动行为，需要调查人员全面参与。运营支出很高，整个过程也非常容易出错。

通过实施图分析技术，该公司成功地使用基于机器学习的实时响应源，实现了智能反洗钱查询的自动化开发。最终，他们通过使用更有效的反洗钱程序、减少误报和转化为更高的侦查率，获得了高经济收益。

示例：信用卡公司

同样，一家排名前五的支付服务提供商也寻求提高他们的反洗钱能力。主要痛点包括成本高企，以及因无法遵守美国联邦反洗钱法规而招致罚款。该组织过去依赖一个机器学习团队来执行人工调查程序，该团队由数百名调查人员组成，结果导致整个过程速度缓慢、成本高昂且效率低下，误报率超过 90%。

这家公司利用图引擎实现了调查程序的现代化。他们不再依赖机器学习团队来拼凑琐碎的调查程序，转而将图分析技术的力量与机器学习相结合，以便洞察个人、帐户、公司和位置之间的关联。

通过合并更多数据维度并融合额外数据点（如与客户有关的外部信息），他们能够自动实时监测潜在的洗钱活动，使调查人员腾出手来将更趋巨大的数据用于更加具有战略意义的用途。最终，他们可以全面分析自己的海量数据，从中获得深刻的内部信息，减少错误警报的数量。

结语

在当今数据爆炸的时代，对于组织而言，实时、充分地分析他们的海量数据以进行欺诈侦查正变得越来越重要。图技术与机器学习的强大组合为确保机器学习获得优质的数据带来了巨大的价值。随着机器学习变得更加有效，越来越多正在发生的欺诈活动得以被发现。图技术有助于确保识别更优质、更复杂的特征，以便支持为“大海捞针”而设计的准确机器学习，从这个方面来说，它是一种强大的资产。

www.tigergraph.com.cn