TigerGraph

# 图分析系统
# 基准测试报告

# 概　　要

作为世界上第一个原生、实时和大规模并行处理（MPP）的图数据库，TigerGraph ™ 系统加载、存储和查询数据的速度比其他图数据库快。

- 单台服务器上，TigerGraph 的图遍历和查询响应时间比其他图数据库要快 2 倍到 8000 倍。随着访问层数的增加， TigerGraph 的优势会更加明显 。
- TigerGraph 的查询性能与机器的数量几乎是线性增长的。即使是像 PageRank 这样需要大量的点到点通信的查询，TigerGraph 仍然可以使用 8 倍的机器数量实现 6.7 倍的加速。
- TigerGraph 与生俱来的并行性使它能够比 Neo4j 的离线加载数据速度快 1.8 至 3 倍，比其他数据库快 12 至 58 倍，即使在使用一台机器时也是如此。
- TigerGraph 的高数据压缩能力可以将一个大图存储在比其他测试的图数据库少 5 倍到 13 倍的磁盘空间中。

这个基准测试研究了 TigerGraph、Neo4j、Amazon Neptune、JanusGraph 和 ArangoDB 的数据加载和查询性能。基准测试包括以下内容：

1. 数据加载
    - 数据加载时间
    - 加载后数据占用的磁盘容量
2. 数据查询
    - K 步子图的查询响应时间
    - 弱联通子图和 PageRank 的查询响应时间
    - 集群上查询响应时间的可扩展性

# 基准设置

本节描述测试的图系统、使用的硬件平台和数据集。

### 图数据库 / 分析系统

- TigerGraph Developer Edition 2.1.4
- Neo4j 3.4.4 Community Edition
- Amazon Neptune 1.0.1.0.200233.0
- JanusGraph 0.2.1, 使用 Cassandra 作为后端存储
- ArangoDB 3.3.13, 分别使用 MMFiles 和 RocksDB 作为存储引擎[1]

**1 ArangoDB 提供两个存储引擎选项 :MMFiles( 内存映射文件 ) 和 RocksDB。MMFiles 适用于主要内存的数据集 ;RocksDB 对更大的数据集更好。https://www.arangodb.com/why-arangodb/comparing-rocksdb-mmfiles-storageengines/**

### 硬件平台

所有的测试都是在相同的 Amazon EC2 硬件上运行的，除了 Neptune 使用了两个顶级强大的数据库型实例。

***Table 1A – 每个数据库的云硬件和操作系统***

| Database | EC2 Machine | vCPUs | Memory (GiB) | Provisioned IOPS | Max Bandwidth (Gbps) | OS |
|---|---|---|---|---|---|---|
| TigerGraph | r4.8xlarge[2] | 32 | 244 | 25,600 | 7 | Ubuntu 14.04.5 LTS |
| Neo4j | r4.8xlarge | 32 | 244 | 25,600 | 7 | Ubuntu 14.04.5 LTS |
| Neptune1 | db.r4.4xlarge | 16 | 122 | 160,000[3] | 3.5 | Amazon Linux AMI 2018.03 |
| Neptune2 | db.r4.8xlarge | 32 | 244 | 323,000[2] | 7 | Amazon Linux AMI 2018.03 |
| JanusGraph | r4.8xlarge | 32 | 244 | 25,600 | 7 | Amazon Linux AMI 2018.03 |
| ArangoDB | r4.8xlarge | 32 | 244 | 25,600 | 7 | Ubuntu 14.04 LTS |

此外，Neptune 需要一个客户端机器来提交数据加载作业。

*Table 1B – Neptune 客户端机器*

| EC2 Machine | vCPUs | Memory (GiB) | OS | Maximum Bandwidth |
|---|---|---|---|---|
| r4.xlarge | 4 | 30.5 | Amazon Linux AMI 2018.03 | 850 Mbps |

### 数据集

该基准测试使用两个数据集，一个是合成的，一个是真实的。

*Table 2 — 数据集*

| Name | Description and Source | Vertices | Edges |
|---|---|---|---|
| graph500 | Synthetic Kronecker graph<br>http://graph500.org | 240万 | 6700万 |
| twitter | Twitter user-follower directed graph<br>http://an.kaist.ac.kr/traces/WWW2010.html | 4160万 | 14.7亿 |

**2 对于 r4.8 xlarge，我们附加了一个 512 GB 的 EBS-optimized Provisioned IOPS SSD(IO1)，为所有的 I/O 流量提供了 50 IOPS/GB 的流量。**

**3 Amazon Neptune IOPS 的值是从 Amazon Cloudwatch 控制台获取的。**

对于每一个图，原始数据都被格式化为一个以单个制表符（tab）分隔的边的列表：

```
U1	U3
U1	U4
U2	U3
```

顶点没有属性，所以不需要一个单独的顶点列表。

## 数据加载测试

数据加载测试检查了这两个方面：

1. 加载耗时和速度
2. 加载后数据占用的磁盘容量

### 数据加载方法

对于每个图数据库，我们选择了最有利的初始数据批量加载方法：

*Table 3 — 每个数据库的数据加载方法*

| Name | Loading API or Method |
|---|---|
| TigerGraph | GSQL声明加载作业 |
| Neo4j | neo4j-import （脱机加载器），然后在顶点上构建索引以加快查询速度 |
| Neptune | 将外部文件（存储在S3中的）直接加载到Neptune DB实例的Neptune加载器 |
| JanusGraph | 使用TinkerPop API 添加顶点和边的Java程序 |
| ArangoDB | arangoimp 批量导入工具 |

我们发现，一些图数据库要么需要一些前置处理、要么需要一些后处理工作。

我们的数据集只包含边列表，但是其他系统要么必须 (Neo4j，Neptune，ArangoDB) 在加载边之前加载顶点、要么 (JanusGraph) 能从加载边之前加载顶点的方式获得益处。

因此，我们编写了一个简单的脚本来生成一个顶点文件，它包含了边列表中所有惟一的顶点 id。

Neptune 的数据格式要求比其他的要严格。为了避免在使用 JanusGraph 时的内存问题，我们将数据文件分割成几个小的数据集，然后按顺序加载它们。

对于 Neo4j，用户需要在加载后，额外构建一个索引以获得良好的查询性能。我们将整个数据加载分为四个阶段，并测量每个阶段的耗时。

- **顶点文件准备：**生成顶点文件的时间。
- **顶点加载：**将顶点加载到图的时间。
- **边的加载：**将边 ( 也许有顶点 ) 加载到图的时间。
- **索引构建：**大多数图数据库在加载期间构建一个主 ID 索引。Neo4j 要求这是作为一个单独的步骤来完成的。如果没有索引，查询性能会显著降低。

### 加载耗时和速度

下面的表格显示了两个数据集的总加载耗时和每个数据库的每个阶段耗时。

*Table 4A — graph500 的数据加载耗时 ( 单位 : 秒 )*

| | TigerGraph | Neo4j | Neptune1 | Neptune2 | JanusGraph | ArangoDB.m | ArangoDB.r |
|---|---|---|---|---|---|---|---|
| edge | 56.0 | 37.3 | 1,571.0 | 1,052.0 | 574.6 | 599.6 | 944.9 |
| vertex | - | - | 45.0 | 30.0 | 43.9 | 11.5 | 44.5 |
| vertex file prep | - | 54.4 | 93.2 | 93.2 | 54.4 | 54.4 | 54.4 |
| index | - | 7.99 | - | - | - | - | - |
| total | 56.0 | 99.7 | 1,709.2 | 1,175.2 | 672.9 | 665.5 | 1,043.8 |
| **relative** | **1** | **1.8** | **30.5** | **21.0** | **12.0** | **11.9** | **18.6** |

*Table 4B — Twitter 数据加载耗时 ( 单位 : 秒 )*

| | TigerGraph | Neo4j | Neptune1 | Neptune2 | JanusGraph | ArangoDB.m | ArangoDB.r |
|---|---|---|---|---|---|---|---|
| edge | 785 | 685 | 42248 | 21700 | 12192 | N/A[4] | 20137 |
| vertex | - | - | 733 | 503 | 839 | 206 | 272 |
| vertex file prep | - | 1270 | 2259 | 2259 | 1270 | - | 1270 |
| index | - | 126 | - | - | - | - | - |
| total | 785 | 2081 | 45240 | 24462 | 14301 | N/A | 21679 |
| **relative** | **1** | **2.7** | **57.6** | **31.2** | **18.2** | **N/A** | **27.6** |

**4 ArangoDB.m 的边加载时间为 24.4 小时，加载了 54% 的数据，然后耗尽了内存。**

下面以图方式说明了这些加载耗时。

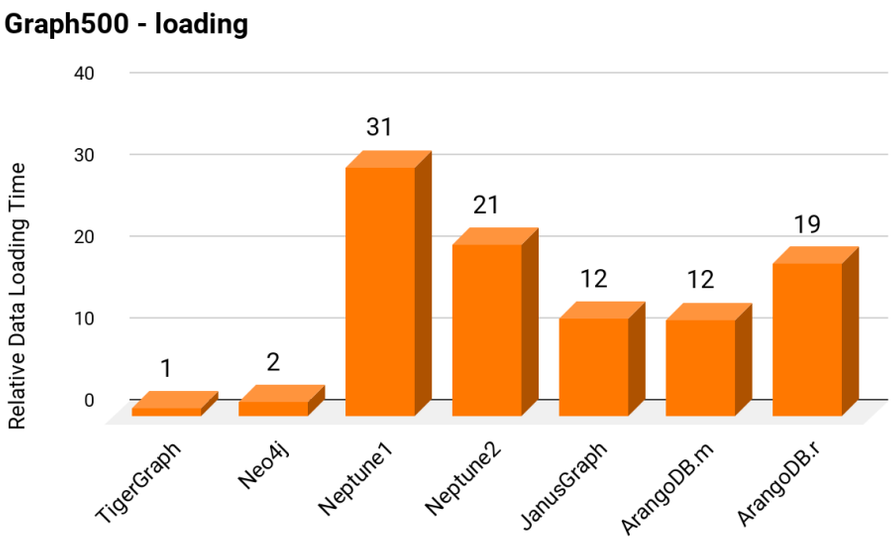


*图 1A – graph500 数据加载耗时*

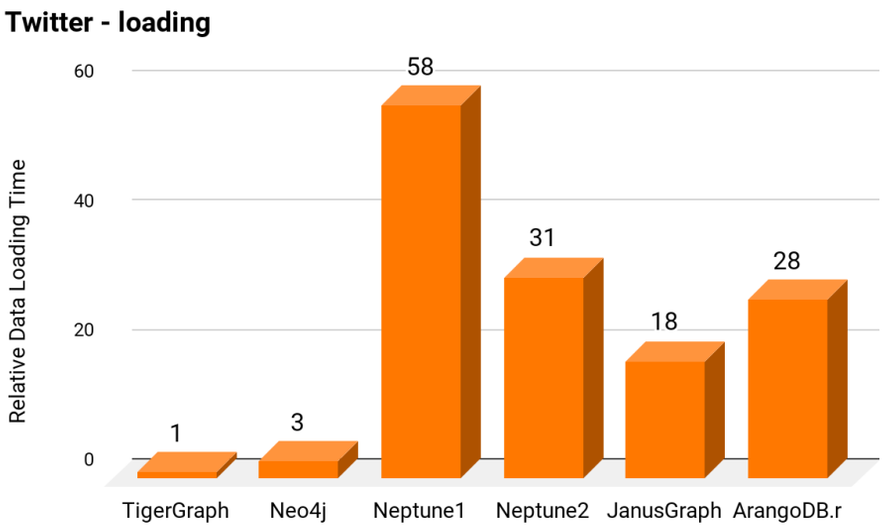


*图 1B – Twitter 数据加载耗时*

- *只有 Neo4j 的加载时间与 TigerGraph 的加载时间是可比的（比 TigerGraph 慢 1.8 倍到 2.7 倍）。我们使用 Neo4j 的离线加载方法（不允许并发数据库操作）；它的在线加载方法要慢得多 [5]。*
- *除了 Neo4j 之外，TigerGraph 比其他图数据库加载小数据集时快 12 到 31 倍，加载大数据集时快 18 到 58 倍。*
- *TigerGraph 具有最简单的加载过程，不需要前置处理或后处理。*

**5 我们在 2017 年对 Neo4j 的离线和在线加载方法进行了基准测试：**

https://doc.tigergraph.com/report/WP_NPGbenchmark_Sep17_web.pdf

### 加载后数据占用的磁盘容量

加载后数据占用的磁盘容量是系统成本和性能的重要考虑因素。在其他条件相同的情况下，一个紧凑的数据库存储可以在给定的机器上存储更多的数据和拥有更快的访问时间，因为它可以获得更多的缓存和页面命中率。

TigerGraph 自动编码和压缩数据，将加载后数据的大小减少到小于原始数据量的一半。相反，所有其他数据库都放大了输入数据量。我们在加载后测量了数据库存储数据的大小。对于 Neptune，我们使用 Cloudwatch 控制台来获得加载后数据的存储量大小。亚马逊用户指南说他们的系统自动复制 6 次数据，所以我们把测量到的数据量数字除以 6。

*Table 5 — 加载数据的存储量 ( 单位 : MB)*

| Dataset | Raw Data | Tiger Graph | Neo4j | Neptune | JanusGraph | ArangoDB |
|---|---|---|---|---|---|---|
| graph500 | 967 | 482 | 2,300 | 3,850 | 2,764 | 6,657 |
| twitter | 24,375 | 9,500 | 49,000 | 91,140 | 55,296 | 126,970 |

下面的图表比较了图存储量大小，比较基准为原始数据量大小。

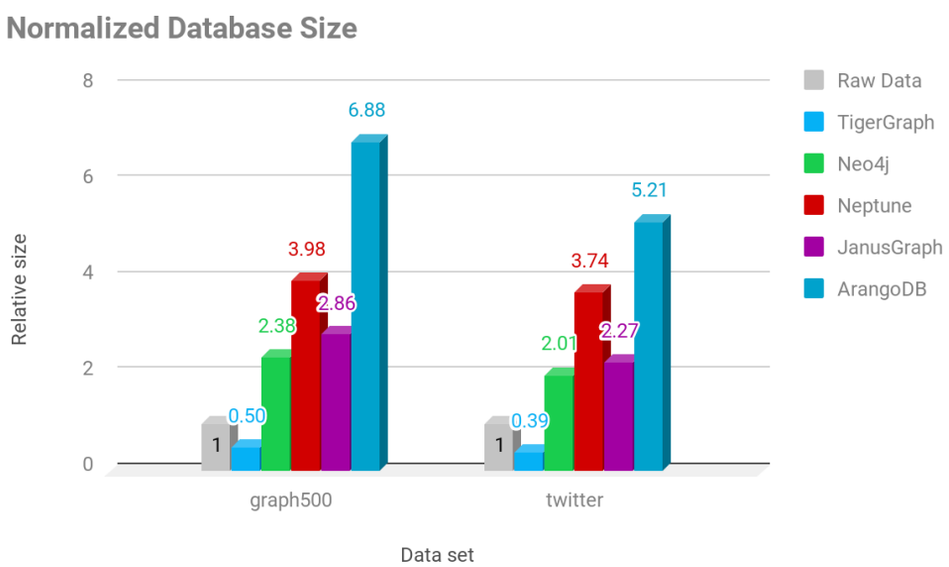


*图 2 — 加载数据后数据库数据容量比较*

### 数据存储量综述

- *原始数据被压缩并存储在 TigerGraph 数据库中，从而为 Graph 500 数据集节省了 50% 的存储空间，并为 Twitter 数据集节省了 61% 的存储空间。*
- *其他的图数据库创建的图数据量都比原始数据量大，需要比 TigerGraph 多 5 倍到 13 倍的存储空间存储数据。*

## 查询性能测试

数据检索或查询是任何数据库系统的基本功能之一。数据分析系统必须能够执行用户应用程序所需的查询类型，并且必须快速执行。

查询性能测试检查了这三个方面 :

- K 度顶点计数查询的查询响应时间
- 弱联通子图和 PageRank( 全图查询 ) 的查询响应时间
- 查询响应时间的可扩展性

### K 度查询

K 度路径顶点计数查询，它计算从起始顶点有长度为 K 的路径的顶点的总计数，这是一个很好的图遍历性能的度量。对于每个数据集，我们测量下列查询的查询响应时间 :

- 计算 300 个固定随机种子的所有一度的顶点，并将超时设置为 3 分钟 / 查询。
- 计算 300 个固定随机种子的所有两度的顶点，并将超时设置为 3 分钟 / 查询。
- 计算 10 个固定随机种子的三度的顶点，并将超时设置为 2.5 小时 / 查询。
- 计算 10 个固定随机种子的六度的顶点，并将超时设置为 2.5 小时 / 查询。

“固定随机种子”，意思是我们从图中一次性随机选择 N 个顶点，并将这个列表保存为测试的可重复输入条件。每条路径查询都从一个顶点开始，并找到所有位于长度为 K 路径末端的顶点。

N 个查询是按顺序运行的。每个起始顶点都有不同的查询子图大小 ; 我们报告中取平均值。为了将注意力集中在图遍历和最小化网络输出时间上，我们只输出 K 度路径查询子图的大小，而不是完整的顶点列表。我们在参与测试的图数据库中，对所有测试查询结果进行交叉验证。

***Table 6 — 平均 K 度查询的大小***

| dataset ↓ \ hops→ | 1 | 2 | 3 | 6 | Total Vertices |
|---|---|---|---|---|---|
| graph500 | 5.13E+03 | 4.89E+05 | 1.43E+06 | 1.58E+06 | 2.40E+06 |
| twitter | 1.06E+05 | 3.25E+06 | 2.07E+07 | 3.50E+07 | 4.16E+07 |

我们在每个数据库的原生查询语言中实现查询 : 用于 TigerGraph 的 GSQL、Neo4 的 Cypher、Neptune 和 JanusGraph 的 Gremlin，以及 ArangoDB 的 AQL。下面的表格报告了所有数据库的实际和基准的查询时间。如之前所说，ArangoDB.m 是无法加载 twitter 数据的。

*Table 7 – 一度路径查询时间*

| Dataset | Measure | TigerGraph | Neo4j | Neptune1 | Neptune2 | JanusGraph | ArangoDB.r | ArangoDB.m |
|---|---|---|---|---|---|---|---|---|
| graph500 | time (ms) | 6.3 | 21.0 | 15.8 | 13.5 | 27.2 | 91.9 | 23.0 |
| | normalized | 1 | 3.3 | 2.5 | 2.1 | 4.3 | 14.6 | 3.7 |
| twitter | time (ms) | 24.1 | 205.0 | 335.8 | 289.2 | 394.8 | 1674.7 | N/A |
| | normalized | 1 | 9 | 14 | 12 | 16 | 69 | N/A |

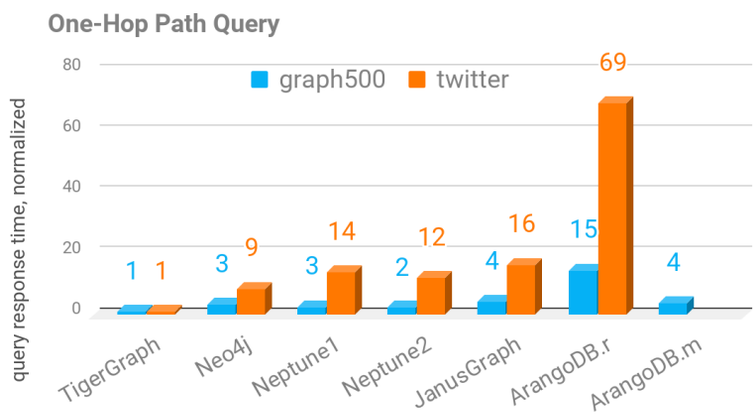


*图 3 – 一度路径查询响应时间*

当路径长度增加到 2 度时，在 300 次尝试中一些数据库会超时。( 当 k=1 和 2 时，我们为每颗种 子设定了 3 分钟的时间限制 )。在下面的表格中，我们报告了成功案例的平均查询响应时间、相应的基准查询时间以及超时的尝试次数。

*Table 8 – 两度路径查询时间*

| Dataset | Measure | TigerGraph | Neo4j | Neptune1 | Neptune2 | JanusGraph | ArangoDB.r | ArangoDB.m |
|---|---|---|---|---|---|---|---|---|
| graph500 | time (sec) | 0.071 | 4.18 | 7.77 | 8.25 | 15.39 | 24.05 | 16.65 |
| | normalized | 1 | 59 | 109 | 116 | 216 | 337 | 234 |
| | timeouts | 0 | 0 | 1 | 0 | 3 | 49 | 20 |
| twitter | time (sec) | 0.46 | 18.34 | 26.17 | 27.40 | 27.78 | 28.98 | N/A |
| | normalized | 1 | 40 | 57 | 60 | 60 | 63 | N/A |
| | timeouts | 0 | 0 | 63 | 60 | 48 | 101 | N/A |

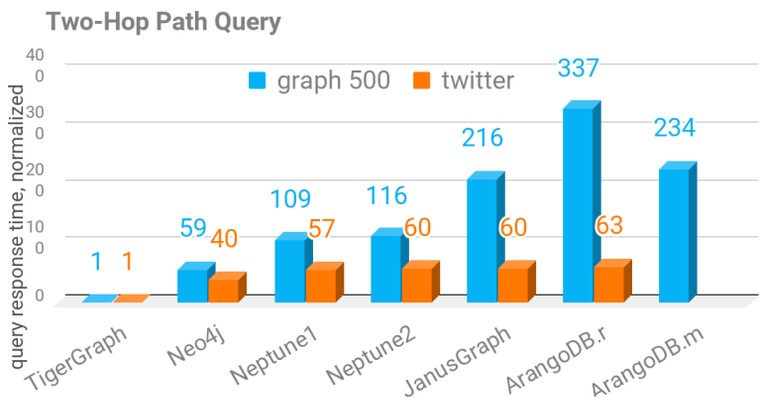


*图 4 – 两度路径查询响应时间*

对于大多数数据库来说三度路径查询更具挑战性。我们将每个查询的超时阈值从 3 分钟提高到 2.5 个小时，但我们还是看到了一些重要的问题。有些系统内存不足。为了使总测试保持可管理，我们将试验总数从 300 次减少到 10 次。除了查询时间之外，我们还报告了数据库多久耗尽内存（OOM, Out Of Memory）或超时的频率。TigerGraph 是唯一没有出现 OOM 或超时情况的数据库。

*Table 9 – 三度路径查询时间*

| Dataset | Measure | TigerGraph | Neo4j | Neptune1 | Neptune2 | JanusGraph | ArangoDB.r | ArangoDB.m |
|---|---|---|---|---|---|---|---|---|
| graph500 | time (sec) | 0.41 | 51.38 | 2.12 | 2.27 | 1846.71 | 3461.34 | 1854.87 |
| | normalized | 1 | 125 | 5 | 5 | 4477 | 8391 | 4497 |
| | % OOM | 0 | 0 | 80% | 80% | 0 | 0 | 0 |
| | % timeouts | 0 | 0 | 0 | 0 | 20% | 50% | 30% |
| twitter | time (sec) | 6.73 | 298.0 | 38.2 | 38.7 | 4324.0 | 3901.6 | N/A |
| | normalized | 1 | 44 | 6 | 6 | 642 | 580 | N/A |
| | % OOM | 0 | 0 | 90% | 90% | 0 | 0 | N/A |
| | % timeouts | 0 | 60% | 0 | 0 | 50% | 80% | N/A |

因为查询时间的范围变化了 3 个数量级，下面图中的垂直轴使用的是对数坐标轴。

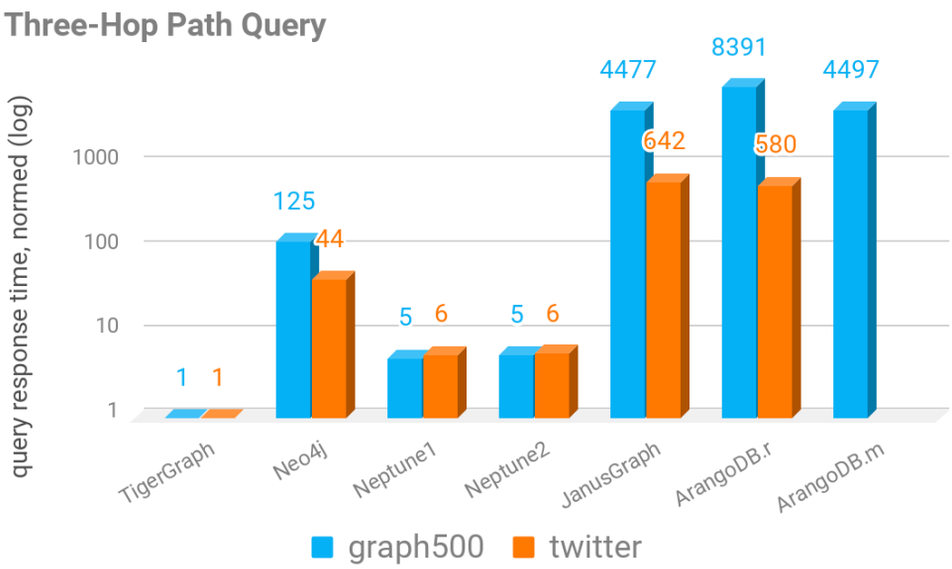


*Figure 5 – 三度路径查询响应时间*

为了真正了解数据库的限制，我们将路径长度增加到 6。TigerGraph 在两个数据集上运行六度路径查询没有任何问题。两个 Neptunes 系统的内存耗尽。对 JanusGraph 和 ArangoDB 而言，查询由于超时而失败。

*Table 10 – 六度路径查询时间*

| Dataset | Measure | TigerGraph | Neo4j | Neptune1 | Neptune2 | JanusGraph | ArangoDB.r | ArangoDB.m |
|---|---|---|---|---|---|---|---|---|
| graph500 | time (sec) | 1.78 | 1312.7 | N/A | N/A | N/A | N/A | N/A |
| | normalized | 1 | 737 | N/A | N/A | N/A | N/A | N/A |
| | % OOM | 0 | 0 | 100% | 100% | 0 | 0 | 0 |
| | % timeouts | 0 | 80% | 0 | 0 | 100% | 100% | 100% |
| twitter | time (sec) | 63.06 | N/A | N/A | N/A | N/A | N/A | N/A |
| | normalized | 1 | N/A | N/A | N/A | N/A | N/A | N/A |
| | % OOM | 0 | 0 | 100% | 100% | 0 | 0 | N/A |
| | % timeouts | 0 | 100% | 0 | 0 | 100% | 100% | N/A |

- *TigerGraph 的查询速度在一度路径查询上比其他图数据库要快 2 倍至 60 倍。*
- *在两度路径查询中，Neptune、JanusGraph 和 ArangoDB 有时会因为超时而失败。只看执行成功实验的数据，TigerGraph 的速度是其他图数据库的 40 倍至 300 倍。*
- *在三度路径查询中，只有 TigerGraph 和 Neo4j 可以在较小的数据集上完成所有的试验。只有 TigerGraph 可以在更大的数据集上完成所有的试验。只看完成了至少一半试验的数据库，TigerGraph 的速度比其他数据库快 125 倍到 4000 倍。*
- *在 20 个测试用例中，有 17 个测试用的是三度路径查询的测试用例，Amazon Neptune 出现内存耗尽现象。而在六度路径查询测试用例中，Amazon Neptune 所有测试用例都出现内存耗尽现象。*
- *只有 TigerGraph 可以完成六度路径查询（在小图上用时 1.8 秒、大图上用时 63 秒）。*

**弱联通子图 (Weakly Connected Component) 和 PageRank 查询**

全图查询是指查询检查整个图，并计算出描述图特征的结果。全图查询的两个例子是弱联通子图和 PageRank。弱联通子图（WCC）是所有可以互相连通的顶点、以及顶点之间的边的集合，如果是有向边则忽略其方向。WCC 查询在图中找到并标记所有的弱联通子图。这个查询要求遍历每个顶点和每条边。

PageRank 是一种迭代算法，它在每次迭代中遍历每条边，并为每个顶点计算一个得分值。经过多次迭代后，这些分值将收敛于稳定状态的分值。我们的测试是运行 10 次迭代。

对于 TigerGraph，我们在 GSQL 语言中实现了每个算法。

对于 Neo4j 来说，我们使用 Neo4j 原生的、在 Neo4j AOPC 过程中提供的 (release 3.4.0.1)WCC 和 PageRank 实现方式。

对于 Amazon Neptune，我们没有找到任何方法来运行像 WCC 和 PageRank 这样的查询算法，因为 Neptune 目前只支持声明性查询。具体来说，Neptune 不支持 Gremlin OLAP API 的 VertexProgram。因此，如果没有内置的图函数库或支持过程计算的语言，那么 Neptune 就不能被包含在基准测试的这一部分中。

JanusGraph 有一个原生的 PageRank 实现，但没有 WCC 的实现。因此，我们使用 Gremlin OLAP API 来编写 WCC。

对于 ArangoDB，我们使用其内置的 WCC 和 PageRank 查询。

*Table 11 – Weakly connected component 查询时间*

| Dataset | Measure | TigerGraph | Neo4j | JanusGraph | ArangoDB.m | ArangoDB.r |
|---|---|---|---|---|---|---|
| graph500 | time (sec) | 3.08 | 49.97 | 2246.54 | 87.77 | 442.89 |
| | relative time | 1 | 16.2 | 729.8 | 28.5 | 143.9 |
| twitter | time (sec) | 74.06 | 1093.26 | too slow[6] | no data[7] | too slow[8] |
| | relative time | 1 | 14.8 | | | |

6 第一次迭代在 24 小时内没有完成。

7 twitter 的图数据不能被加载。

8 在 24 小时内没有完成。

*Table 12 – PageRank 查询响应时间*

| Dataset | | TigerGraph | Neo4j | JanusGraph | ArangoDB.m | ArangoDB.r |
|---|---|---|---|---|---|---|
| graph500 | time (sec) | 12.52 | 30.27 | 2610.72 | 119.43 | 582.62 |
| | relative time | 1 | 2.4 | 208.6 | 9.5 | 46.6 |
| twitter | time (sec) | 265.36 | 614.96 | too slow[9] | no data[10] | too slow[11] |
| | relative time | 1 | 2.3 | | | |

9 第一次迭代在 24 小时内没有完成。

10 twitter 的图数据不能被加载。

11 在 24 小时内没有完成。

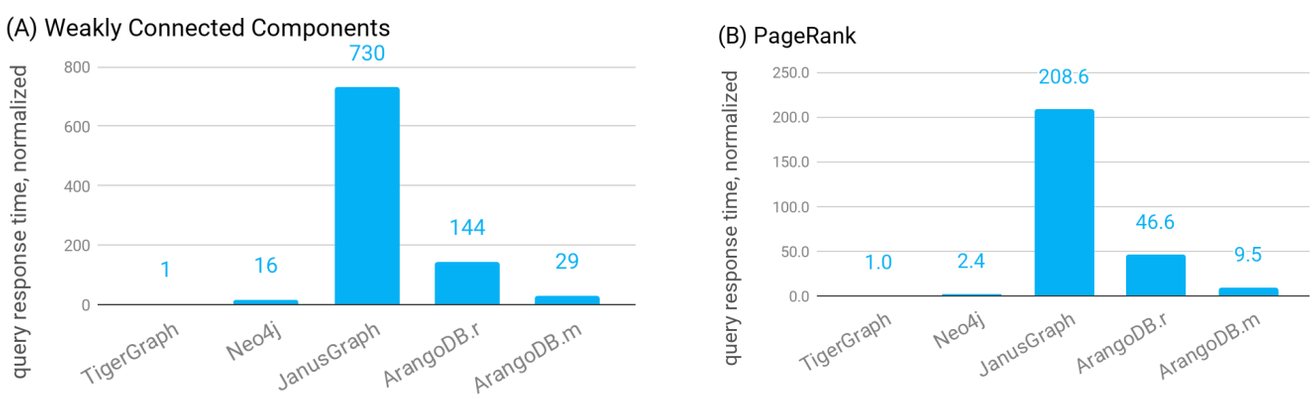


*图 5 –(A) WCC and (B) PageRank 查询响应时*

- *Amazon Neptune 不提供原生的运行分析查询的能力。*
- *JanusGraph 和 ArangoDB 在大图上不能在 24 小时内完成 WCC 或 PageRank。*
- *TigerGraph 的速度比 Neo4j 的 WCC 快 15 倍左右。如果其他图数据库能够完成同样查询的话，那么 TigerGraph 的速度将比那些图库快 28 至 700 倍。*
- *TigerGraph 的速度比 Neo4j 的 PageRank 快 2.3 倍左右。如果其他图数据库能够完成同样查询的话，那么 TigerGraph 的速度将比那些图库快 10 至 200 倍。*

**查询响应时间的可扩展性**

在前面的测试中，我们考虑了单机性能。现在我们来看看 TigerGraph 在数据分布在集群上时性能如何。在这个测试中，我们测试服务器数量的增加是如何影响查询性能的。在这个测试中，我们使用了更经济的 Amazon EC2 实例类型 (r4.2 xlarge: 8 vCPU，61 GiB 内存，以及 200 GB 附加的 GP2 SSD)。

当 twitter 数据集 ( 由 TigerGraph 压缩到 9.5 GB) 分布在 8 台机器上时，每台机器有 60 个 GiB 内存和 8 个内核 (core) 就足够了。对于较大的图或更高的查询吞吐量，更多的 CPU 内核可能会有所帮助 ;TigerGraph 提供了调优内存和计算资源利用率的设置。另外，为了在集群上运行，我们从 TigerGraph Developer Edition(v2.1.4) 切换到等效的企业版 (v2.1.6)。

我们使用了 Twitter 数据集，并运行了 10 次迭代的 PageRank 查询。我们重复了三次，对查询时间进行了平均，并且对包含 1、2、4、6 和 8 台机器的集群反复进行了测试。对于每一个集群，twitter 图被平均切片部署在所有正在使用的机器上。

*Table 13 – TigerGraph 平均查询响应时间的可扩展性*

| No.of machines | 1 | 2 | 4 | 6 | 8 |
|---|---|---|---|---|---|
| average query time (sec) | 969.8 | 535.5 | 263.4 | 209.1 | 144.8 |
| speedup | 1.0 | 1.81 | 3.68 | 4.64 | 6.70 |

PageRank 是一种迭代算法，它在每次迭代中遍历每条边。这意味着机器之间有很多通信，信息从一个分区发送到另一个分区。尽管存在这种通信开销，但是 TigerGraph 的原生 MPP 图数据库架构仍然成功地实现了 8 台机器的 6.7 倍加速。

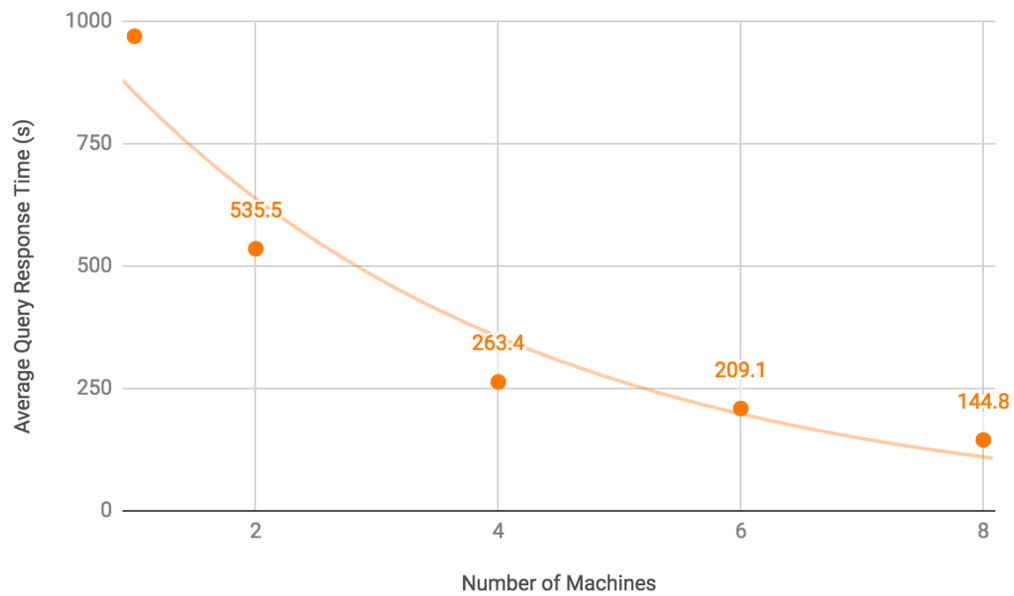


*图 6 – TigerGraph 查询响应时间与机器数量关系*

在 Neo4j 或 Amazon Neptune 上不能执行可扩展性测试。Neo4j 必须在一台服务器上存储完整的图形，并且不能在多台机器上对图进行切片。Amazon Neptune 也不能在多台机器上对图切片，也找不到运行 PageRank 的方法。我们还没有尝试 JanusGraph 或 ArangoDB Enterprise Edition [12] 的可扩展性测试。

**12 ArangoDB 社区版在共享时不会表现良好，关注：** https://www.arangodb.com/why- arangodb/arangodbenterprise/arangodb-enterprise-smart-graphs/

## 可重复性

所有重复基准测试（数据集、查询、脚本、输入参数、结果文件和一般指令）所需的文件都可以在 GitHub 上找到 :https://github.com/tigergraph/ecosys/tree/benchmark/benchmark/

第 1 节给出了机器和系统软件规范。

如果您对这些测试有疑问或反馈，请与我们联系。benchmark@tigergraph.com

# 关于 TigerGraph

**TigerGraph** 特有的原生并行图 (NPG) 技术使其成为世界最快的图分析平台。面对实时处理极度复杂数据的挑战，TigerGraph 的图分析平台依然能够无视数据的海量性与复杂性，为客户达成预期并创造价值。TigerGraph 支持的应用包括物联网，人工智能以及机器学习，它们都通过感知瞬息万变的数据来实现。TigerGraph 的成功案例包括了支付宝、软银集团、中国国家网、Wish 电商平台以及 Elementum 公司。您可以关注 TigerGraph 的微信账号 TigerGraph 或访问我们的网站 www.tigergraph.com.cn 获取更多信息。