

LDBC Benchmark on JanusGraph

0：IO | 大数据系统与大规模数据分析课程报告 | 2018.6.19

张俊阳 | 2017E8018661163 | zhangjunyang@iie.ac.cn

张小洋 | 2017E8018661065 | zhangxiaoyang@iie.ac.cn

王传仁 | 2017E8018661118 | wangchuanren@iie.ac.cn

刘志磊 | 201728018629141 | liuzhilei@iie.ac.cn

冀海川 | 2017E8018661165 | jihaichuang@iie.ac.cn

# LDBC SNB简介

LDBC社交网络图数据Benchmark是一项针对专业图数据库的基准测试工具，它包含一个数据生成器生成社交网络数据，三种工作负载（workloads）：交互式(Interactive)，商业智能(Business Intelligence)和图分析( Graph Analytics)。目前，只有交互式工作负载在初稿阶段发布。商业智能工作负载的只读部分目前能够预览。

主要的SNB主要部件有下列四项

1. SNB benchmark 规范文档 <https://github.com/ldbc/ldbc_snb_docs>
2. SNB数据生成器 <https://github.com/ldbc/ldbc_snb_datagen>
3. LDBC驱动（实现查询的驱动）<https://github.com/ldbc/ldbc_driver>
4. 交互式工作负载实现<https://github.com/ldbc/ldbc_snb_implementations>

# JanusGraph简介

JanusGraph是一个可扩展的图数据库，可以把包含数千亿个顶点和边的图存储在多机集群上。它支持事务，支持数千用户实时、并发访问存储在其中的图。

除此之外JanusGraph还具有下列特点：

* 弹性，线性可扩展
* 针对性能和容错的数据分发和复制机制
* 多数据中心高可用性和热备份
* 支持ACID和eventual consistency
* 支持多个后端存储
* 通过与大数据平台集成支持全局图数据分析，报表和ETL
* 通过集成索引引擎支持地理位置，数字和全文搜索
* 原生集成Apache Tinker Pop图技术栈
* 开源，基于Apache 2 Licenses
* 通过可视化工具展示存储在JanusGraph中的图

基于LDBC 社交网络图实现Benchmark，并比较不同的选项对性能的影响，具体分为以下五个部分：

1. **数据生成：datagen**

利用ldbc\_smb\_datagen工具可以生成用于测试的社交网络数据，生成的数据以文件形式存储。

1. **数据导入：importer**

利用已生成的数据文件，将数据导入到图存储数据库Janusgraph中。

1. **工作负载：Workload**

工作负载（workload）是进行实际测试的方式，我们将介绍用于本测试的工作负载的实现。

1. **Benchmark执行**

完成图数据的导入和工作负载的实现后，可基于这些数据和操作执行benchmark。benchmark在执行时也可对具体的测试方案进行定制。

1. **性能比较及分析**

由benchmark测试结果可反映不同的janusgraph配置条件对其性能的影响。我们将对这些测试结果进行简单的分析。

1. **数据生成：Datagen**
   1. 简介

测试数据的生成是进行benchmark测试的第一步，LDBC为其制定的测试需求设计了一个数据生成器：ldbc\_snb\_datagen ([github仓库地址](https://github.com/ldbc/ldbc_snb_datagen))，本实验使用该数据生成器进行数据生成。

* 1. ldbc\_snb\_datagen
     1. 环境需求

1. **Apache Hadoop v2.6.0**

ldbc\_snb\_datagen (以下简称datagen)使用Apache Hadoop v2.6.0进行数据生成，实验机条件有限，我们按照以下的方式进行数据生成：



1. **Apache Maven**

ldbc\_snb项目主要使用Apache Maven工具进行项目管理和自动构建，在datagen的安装中，需要配置好maven工具以支持安装的进行。

1. **Python 2.7**

datagen通过python实现对生成数据属性的自定义设置，官方给出的版本需求是python 2.7。

* + 1. 参数配置

将datagen文件下载到本地后，根据datagen官方教程配置运行环境。参数配置主要是配置~/\*/ldbc\_snb\_datagen-master/run.sh文件中的路径：

* + - DEFAULT\_HADOOP\_HOME=/usr/local/java/hadoop-2.6n.0
    - $LDBC\_SNB\_DATAGEN\_HOME/test\_params.ini

第一条参数设置对应Hadoop的实际安装路径；

第二条参数设置对应datagen进行数据生成时参数文件位置。

运行命令./run.sh即可通过maven工具自动构建项目生成jar包。我们将利用这个jar包生成所需的测试数据集。

1. datagen参数

datagen通过参数文件的形式对生成数据的属性（图属性、数据规模、部分数据格式等）进行自定义设置，下面就实验中用到的几个重要的参数进行介绍。

* **ldbc.snb.datagen.serializer.personSerializer**
  1. 该参数主要用于设置社交网络中person节点和knows边数据序列化的模式；
  2. 本次实验设置

[*ldbc.snb.datagen.serializer.snb.interactive.CSVPersonSerializer*](https://github.com/ldbc/ldbc_snb_datagen/blob/master/src/main/java/ldbc/snb/datagen/serializer/snb/interactive/CSVPersonSerializer.java)

* **ldbc.snb.datagen.serializer.invariantSerializer**

1. 该参数主要用于设置社交网络中person节点和knows边数据序列化的模式；
2. 本次实验设置

[*ldbc.snb.datagen.serializer.snb.interactive.CSVInvariantSerializer*](https://github.com/ldbc/ldbc_snb_datagen/blob/master/src/main/java/ldbc/snb/datagen/serializer/snb/interactive/CSVInvariantSerializer.java)

* **ldbc.snb.datagen.serializer.personActivitySerializer**

1. 该参数主要用于设置社交网络中person节点和knows边数据序列化的模式；
2. 本次实验设置

*ldbc.snb.datagen.serializer.snb.interactive.CSVPersonActivitySerializer*

* **ldbc.snb.datagen.generator.scaleFactor**

1. 该参数主要用于设置社交网络数据的规模。本实验测试的数据用于snb.interactive测试，该条件下的数据规模从0.1GB~1000GB共有9个可用的参数选项，对于不同的数据规模设置，图属性大致如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Scale Factor | 1 | 3 | 10 | 30 | 100 | 300 | 1000 |
| # of Persons | 11K | 27K | 73K | 182K | 499K | 1.25M | 3.6M |
| # of Years | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Start Year | 2010 | 2010 | 2010 | 2010 | 2010 | 2010 | 2010 |

1. 本次实验设置

*snb.interactive.0.1*

* **ldbc.snb.datagen.serializer.dateFormatter**

1. 本次实验基于Janusgraph实例的基础上完成，该实例实现了importer工具完成图数据的导入，要求数据及文件中日期格式必须为Long型，因此在生成数据时要对日期格式进行设置
2. 本次实验设置

*ldbc.snb.datagen.serializer.formatter.LongDateFormatter*

通过上述的参数设置，本次实验将生成标准的大小为100MB左右的测试数据，基本上每个不同entity的数据内容分别存储在不同的csv文件中。

* 1. 数据

datagen的数据生成利用Hadoop的Mapreduce方法实现，最后生成的数据将构成一张完整的社交网络图。每次执行datagen将生成3类数据文件，主要分布在2个文件夹中。下面将对datagen产生的数据位置、格式和属性进行介绍。

* + 1. 数据分布

datagen生成的与benchmark测试相关的数据分布在2个文件夹中，一个命名为social\_network，一个命名为substitution\_parameters。

1. social\_network

该文件夹存储了datagen生成的社交网络图的数据，数据格式是csv。这些文件由mapreduce任务生成，因此被存储在hdfs上。本次实验需要将文件从虚拟机上传到服务器上进行测试，因此在完成数据生成后，需要将该文件夹通过get方法下载到虚拟机本地上便于之后的文件传输。

1. substitution\_parameters

该文件针对benchmark中设计的25个BI query和14个IC(interactive) query设置了参数，这些参数是在进行查询时必须提供的。参数文件的格式为txt文件。

* + 1. 数据属性

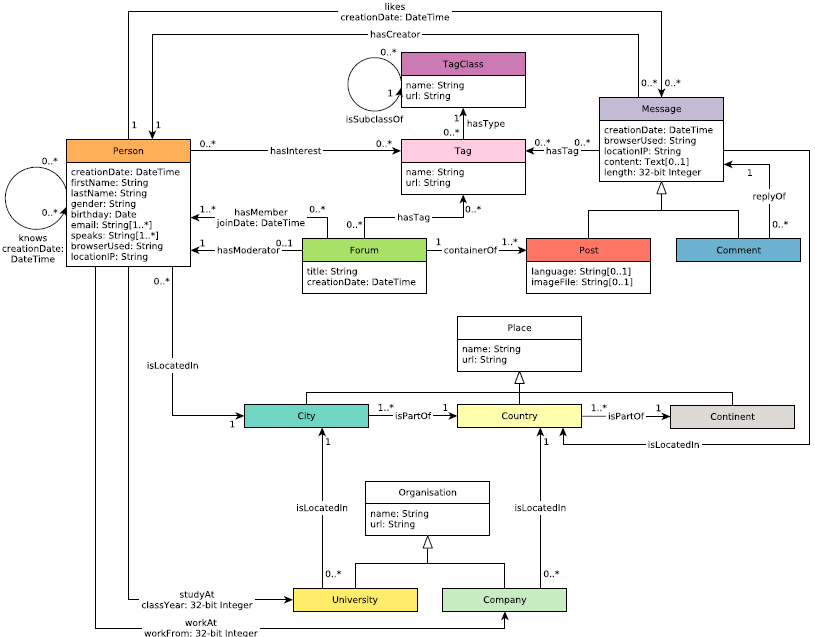
1. 文件属性

datagen将生成三类文件：

* 数据集：用于benchmark测试的主要数据集，大约占生成数据的90%；
* 更新流（update stream）：用于update query的数据，大约占生成数据的10%；
* 参数（substitution parameters）：用于BI query和IC query的参数。

1. data schema

datagen生成的所有文件构成完整的社交网络图。根据官方文件的说明，下图展示了这个图数据的schema：



* 1. 数据使用

本次实验的实验数据存储在~/janusgraph/test-data-100m/下，包括social\_network和substitution\_parameters文件夹，在进行数据测试时，对workload的参数进行如下设置：

* ldbc.snb.interactive.parameters\_dir=/home/user26/test-data-100m/substitution\_parameters
* ldbc.snb.interactive.updates\_dir=/home/user26/test-data-100m/social\_network

# 数据导入：Importer

数据生成后，可使用importer工具进行数据导入。

由于实现具体的benchmark查询(Query)和生成数据高度相关，因此对于数据生成所用到的参数会在下一章节工作负载实现中使用，该部分主要着重阐述将生成的social\_network数据导入到JanusGraph后端存储中，通过执行com.ldbc.snb.janusgraph.importer.Main类来导入，基于LDBC ldbc\_snb\_implementations开源代码修改实现。

1. 启动JansusGraph server（gremlin-server）

导入前需要启动JansusGraph server（gremlin-server），具体命令：

*java -cp target/janusgraphSNBInteractive-0.1-SNAPSHOT-jar-with-dependencies.jar com.ldbc.snb.janusgraph.importer.Main [-n 2] [-s 2000] [-d test-data-100m/social\_network] [-c bdb.conf]*

参数列表：

* *--numThreads/-n*

加载过程中线程数

* *--transactionSize/-s*

读取文件事务的大小（每个读取任务读取的行数）

* --dataset/-d

要导入数据集的文件夹路径（数据集的时间戳必须采用*Long*，使用*ldbc.snb.datagen.serializer.formatter.LongDataFormatter*生成数据 ）

* --backend-config/-c

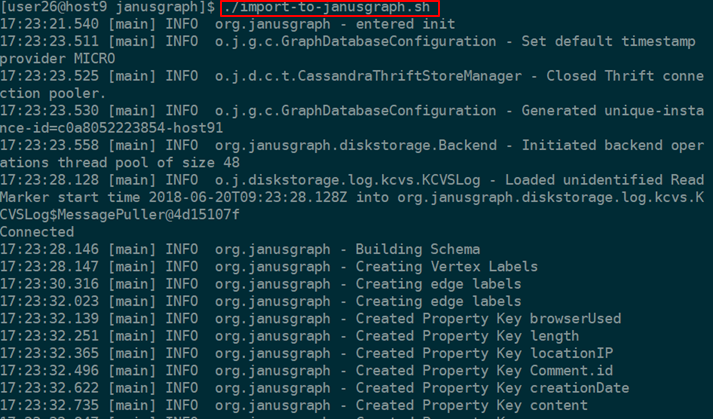
配置后端存储的文件路径。示例文件为resources/bdb.conf

1. 通过shell 脚本import-to-janusgraph.sh 导入数据

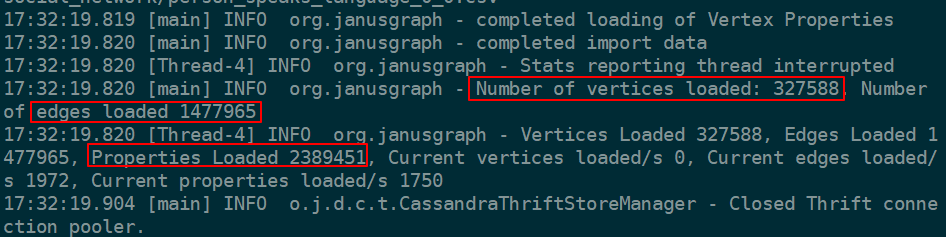
根据Janusgraph不同的存储后端，需用不同的参数进行数据导入。这里分别针对三类存储后端（Cassandra，BerkeleyDB，Hbase）使用脚本进行数据导入：

* *./import-to-janusgraph.sh* （Cassandra）
* *./import-to-janusgraph-berkeley.sh* （BerkeleyDB）
* *./import-to-janusgraph-hbase.sh* （Hbase）

通过上述的两步操作即可完成数据导入，下面以Cassandra导入过程为例介绍整个导入过程：



导入完成



测试100m数据导入了327588个点，1477965条边，2389451条properties。

# 工作负载：workload

该部分基于<https://github.com/ldbc/ldbc_snb_implementations>官方LDBC Interactive代码中针对Titan实现的交互式工作负载benchmark部分实现。

* 1. 主要组件

1. Workload

* ReadOperation
* UpdateOperation

1. DbConnector

* JanusGraphDb

1. Operation Handler

* LdbcQuery[1-14]Handler：14个复杂查询
* LdbcQueryU[1-8]Handler.java：8个更新操作
* LdbcShortQuery[1-7]Handler，7个简单查询

具体的Query规范在ldbc-snb-specification.pdf 文件中有详细描述。具体实现代码请看代码仓库或者附件。

此外，该部分需要LDBC SNB Dirver包，打包前在pom.xml引入如下依赖：

*<dependency>*

*<groupId>com.ldbc.driver</groupId>*

*<artifactId>jeeves</artifactId>*

*<version>0.3-SNAPSHOT</version>*

*</dependency>*

# Benchmark执行

Benchmark的执行主要需要完成两个部分：配置后端和执行benchmark。

* 1. 配置后端janusgraph

JanusGraph支持多种数据存储后端，包括：Apache Cassandra、Apache HBase、Google Cloud Bigtable和Oracle Berkeley DB。本次实践中使用了Apache Cassandra和Orcale Berkeley DB Java Edition分别作为JanusGraph的存储后端进行测试。

1. JanusGraph + Apache Cassandra 集成配置。

Apache Cassandra是一个开源的、分布式、无中心、支持水平拓展、高可用的KEY-VALUE类型的NOSQL数据库[1]。根据JanusGraph的官方文档介绍，Cassandra作为JanusGraph的存储后端时有多种集成模式[2]，这里使用了Remote Server模式，Cassandra以集群方式存在， 运行在其他主机上的JanusGraph基于Socket的读/写来访问Cassandra集群。我们通过控制Cassandra集群的节点数量，运行LDBC的社交网络图Benchmark对JanusGraph的性能进行测试。测试中使用的Cassandra版本是3.11.2。

1. 单节点的Cassandra连接配置

在本例中JanusGraph与单个节点的Cassandra进行连接，Cassandra运行在实验主机host5上，其主要配置文件conf/cassandra.yaml中的关键配置如下：

|  |
| --- |
| # 集群名称  cluster\_name**:** 'IO Cassandra Cluster'  # 设置种子节点 种子节点为host9 ip地址为192.168.5.34  seed\_provider**:**  -class\_name**:** org.apache.cassandra.locator.SimpleSeedProvider  parameters**:**  -seeds**:** "192.168.5.34"  # 设置监听地址  listen\_address**:** 192.168.5.34  # 启用thrift rpc server，否则JanusGraph无法与Cassandra建立连接  start\_rpc**:** true  rpc\_address**:** 192.168.5.34 |

JanusGraph的相关配置文件（主要是gremlin-server.sh加载的配置文件和运行benchmark的程序加载的配置文件）中添加如下配置：

|  |
| --- |
| storage.backend**=**cassandrathrift  storage.hostname**=**192.168.5.34 |

1. 两节点的Cassandra连接配置

在本例中我们部署了两个节点的Cassandra集群，使用host9作为集群的seed节点，host4作为普通节点，每个节点的Cassandra配置同上例中的单节点配置类似：seeds设置为host9的ip地址192.168.5.34，listen\_address和rpc\_address设置为各自对应的ip地址，启动时先启动seed节点上的Cassandra进程，之后普通节点启动会自动加入到集群中。集群启动后使用Cassandra提供的nodetool工具查看各节点的状态，如图1.1所示，我们向JanusGraph导入了约100M的数据，这些数据被分配存储在了Cassandra集群的节点中。

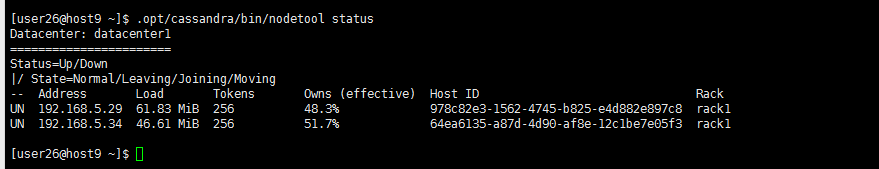


图 1. 1 两节点Cassandra集群状态

1. 四节点的Cassandra连接配置

为了进一步验证Cassandra集群节点数量对测试性能的影响，在本例中我们使用host5、host7、host8、host10部署在四个结点的Cassandra集群，seed结点为host5和host7，host8和host10作为普通节点加入集群。由于在这些机器上有其他用户运行的Cassandra进程造成了端口占用，除了前面例子中的配置外还需要修改Cassandra的端口，cassandra.yaml中关键配置如下：

|  |
| --- |
| # 修改节点通信端口 默认为7000，ssl默认端口为7001  storage\_port**:** 7200  ssl\_storage\_port**:** 7201  # 修改CQL客户端通信端口 默认为9042  native\_transport\_port**:** 9052  # 修改rpc端口 即JanusGraph与Cassandra的通信端口 默认端口为9160  rpc\_port**:** 9260 |

Cassandra通过JMX监测结点，所以需要修改conf/cassandra-env.sh中相关的端口：

|  |
| --- |
| # 修改 JMX 连接端口 默认为9042  JMX\_PORT**=**"8012" |

除此之外，JanusGraph相关的配置中除了指定存储后端Cassandra的地址，还要指定端口：

|  |
| --- |
| # 修改后端通信端口  storage.port=9260 |

做出如上修改后，就可以分别启动Cassandra集群和JanusGraph的gremlin-server进行连接了。Cassandra集群按照seed节点和普通节点的顺序依次启动，启动完成后使用nodetool查看集群节点状态，如图1.2所示，同上例中一样，导入JanusGraph的数据被分配到了所有的节点进行存储。

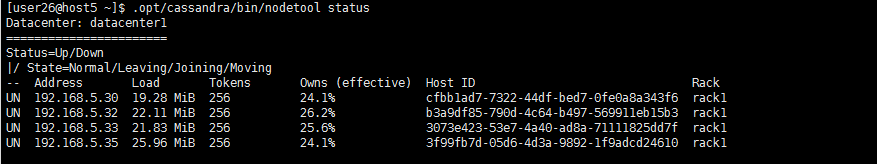


图 1. 2 四节点Cassandra集群状态

1. JanusGraph + Orcale Berkeley DB Java Edition 集成配置。

Berkeley DB是一个是一个开源的文件数据库，介于关系数据库与内存数据库之间，使用方式与内存数据库类似，它提供的是一系列直接访问数据库的函数，而不是像关系数据库那样需要网络通讯、SQL解析等步骤[3]。

JanusGraph使用的相应存储后端为Orcale Berkeley DB Java Edition，它与JanusGraph运行在同一个机器的JVM中，导入JanusGraph的图数据全部都会保存在本地磁盘上。这就将图数据的大小限制在了硬盘空间和内存空间的范围内(大约10-100万个顶点的图)。但是，对于这种规模的图，使用Berkeley DB作为存储后端具有高于其他分布式数据库的性能。

JanusGraph使用Berkeley DB作为存储后端集成的方法比较简单，无需安装Berkeley DB，Orcale Berkeley DB Java Edition 以jar包的形式存在于JanusGraph的lib目录下，所以只需在JanusGraph相关的配置文件（gremlin-server和LDBC Benchmark）中指定后端名称和存储路径即可：

|  |
| --- |
| # 指定后端名称为berkeleyje  storage.backend**=**berkeleyje  # 指定berkeley db 的数据存储路径  storage.hostname**=**../data/graph |

* 1. 交互式工作负载benchmark

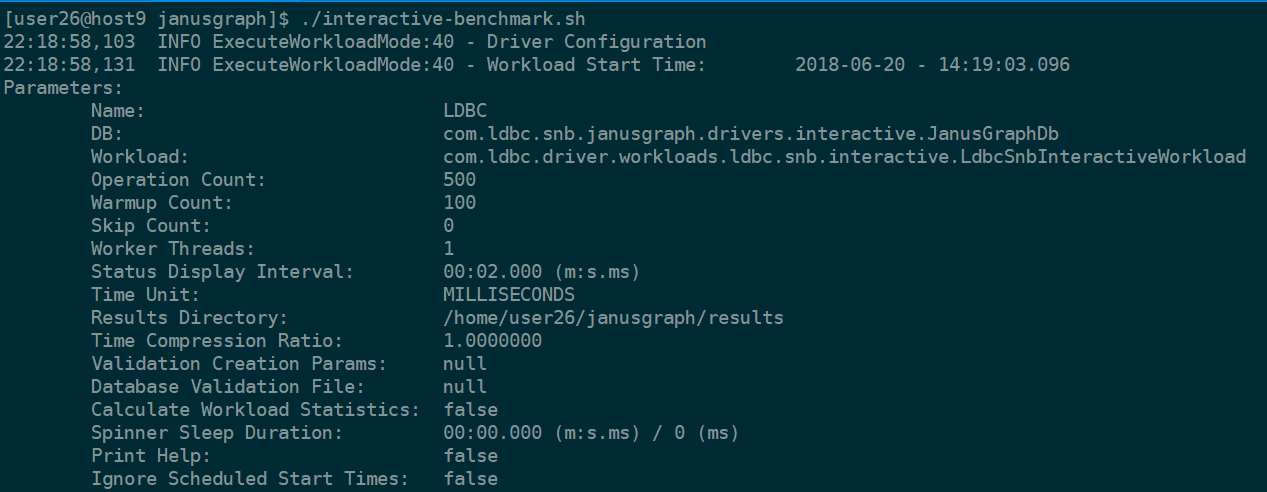
首先获取<https://github.com/ldbc/ldbc_snb_driver>中的源码，然后使用maven编译安装driver包

*mvn clean install -DskipTests*

接着进入项目janusgraph目录下生成实现benchmark的JAR包

*mvn clean assembly:assembly -DskipTests*

对于我们的Janusgraph 交互式工作负载benchmark实现，首先根据系统配置修改ldbc\_snb\_interactive.properties文件，接着使用脚本interactive-benchmark.sh执行驱动程序运行基准测试。



# 性能比较及分析

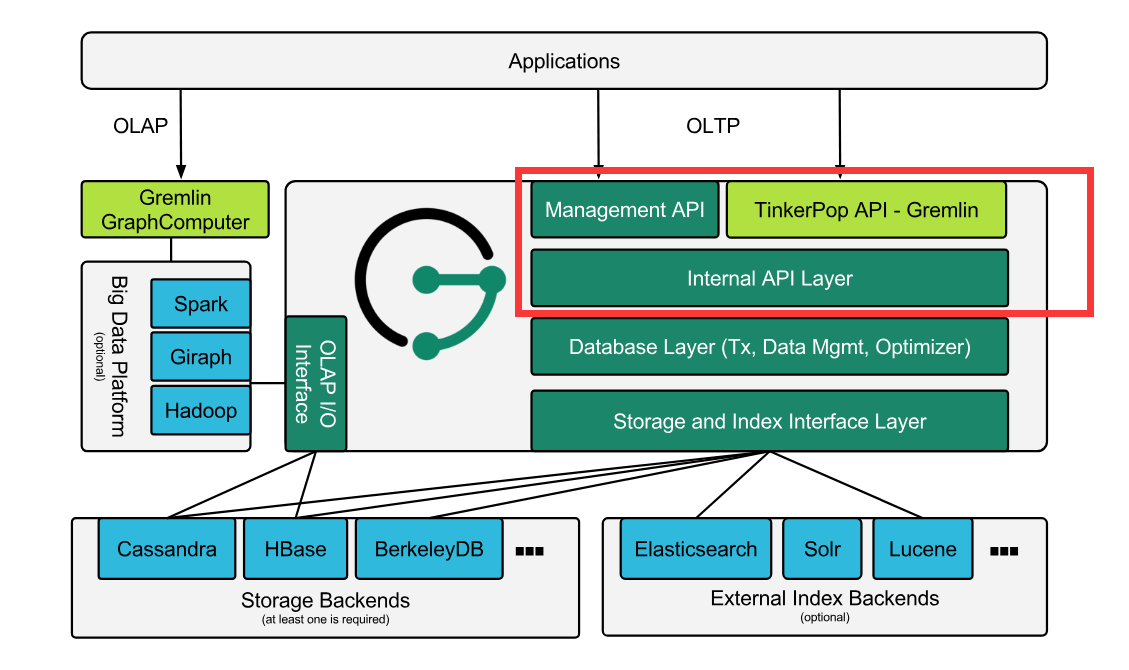
性能的比较分析主要从以下几个方面进行讨论：janusgraph不同存储后端间的性能比较，benchmark性能测试中对query的不同API实现方式的比较，以及janusgraph存储图时不同的划分方式之间的比较。

在这三类性能比较中，将简单介绍不同API实现方式和不同图划分方式，重点以不同存储后端的性能比较为例来进行分析。

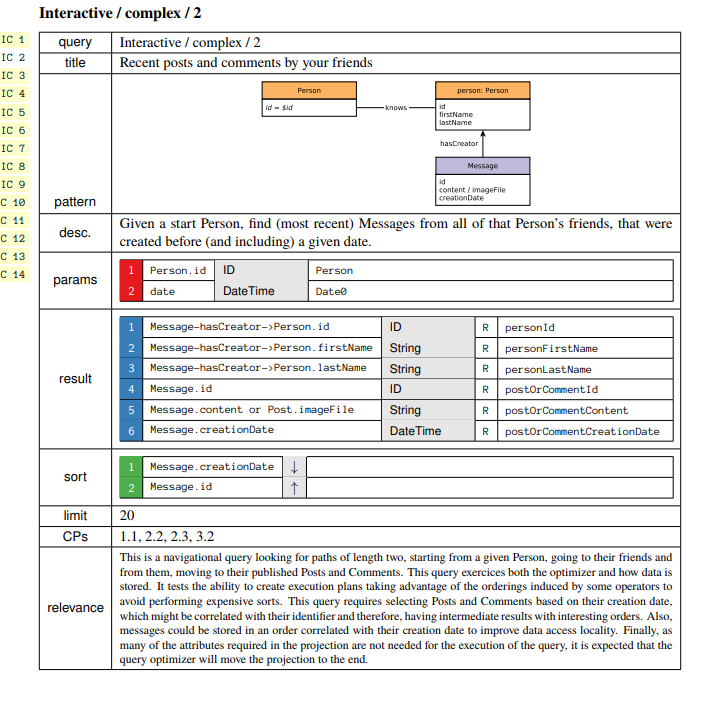
* 1. API实现方式：Gremlin API

janusgraph 中query的实现主要依靠两类API：**janusgraph本身提供的API**和**Gremlin API**。本次实验主要是

JanusGraph发行版包含一个Gremlin控制台命令行，可以轻松上手并与JanusGraph交互。调用bin/gremlin.sh即可启动控制台，然后使用JanusGraphFactory打开JanusGraph图形。JanusGraphFactory也可用于在基于JVM的用户应用程序中打开嵌入式JanusGraph图形实例。在这种情况下，应用程序可以直接通过其公共API调用JanusGraph。



如上图所示，Gremlin API 位于Internal API Layer上层，直接面向应用程序。Gremlin是Apache TinkerPop的一个组件，它独立于JanusGraph开发，得到大多数数据库的支持。JanusGraph的发行版会自带一个Gremlin。Gremlin是一种面向路径的语言，能够简洁地表达复杂的图形遍历和变异操作。Gremlin是一种功能性语言，便利运算符连接在一起形成类似路径的表达式。ldbc\_snb\_driver中定义了对于interactive workload的基本的查询与更新操作。基于给定的查询操作，我们基于gremlin java api 实现了相关的handler。以其中的中的一个query2为例，如下图所示：



根据描述，此条查询的目的是给定一个起始的people，寻找其朋友在给定的日期前发布的消息，返回的结果为personId，personFirstName，personLastName，postOrCommentId，personOrCommentContent和postOrCommentCreationDate。ldbc\_snb\_driver对每一个查询的返回结果定义了一个接收类，query2的数据结构定义在LdbcQuery2Result中。查询语句如下：

*query="g.V().has('Person.id', $id)."+*

*"out('knows').as('friend').valueMap().as('x').in('hasCreator').has('creationDate',P.lte($maxDate))."+*

*"order().by('creationDate',decr).by('messageId',incr)."+*

*"limit($limit).as('post').valueMap().as('y')"+*

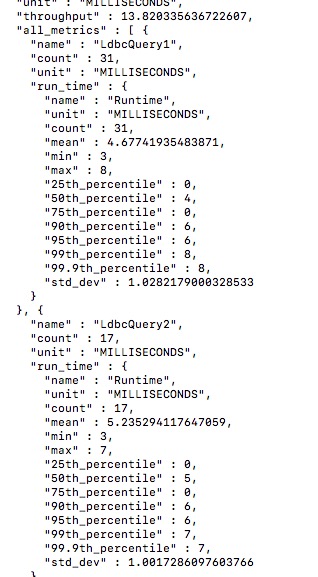
*".select('x','y')\n";*

通过

*ResultSet resultSet = dbConnectionState.runQuery(query, parameters);*

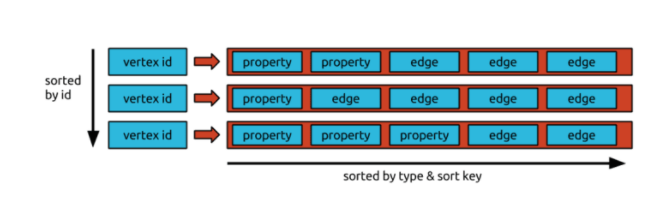
即可获得结果集，使用到的接口在*org.apache.tinkerpop.gremlin*中，具体实现可参考程序代码。

以下为benchmark给出的测试结果：



* 1. 不同图划分模式

JanusGraph以邻接表形式存储图数据，顶点的分配到机器上的方法就决定了图的分区。



图的分区主要有两种策略：默认策略和显示分区策略。

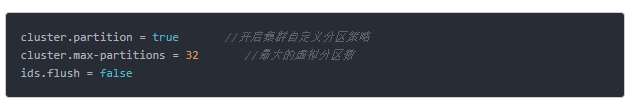
1. 默认策略

随机分区策略。随机安排顶点到所有机器上。缺点：查询效率慢，因为存在大量的跨实例的通信。

1. 显示分区策略

具有强关联性和经常访问的子图存储在相同的机器上，这样可以减少跨机器的通信成本。

* 配置参数



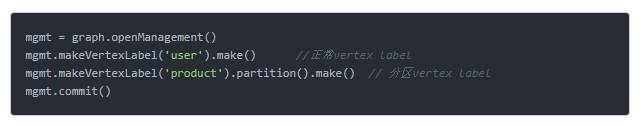
* *max-partitions*：最大虚拟分区数，建议配置为存储数据个数的两倍。
* 精确分区只有支持key排序的存储后端
  + Hbase：支持图自定义分区
  + Cassandra：需要配置ByteOrderedPartitioner来致辞图分区
* 在图分区下有edge cut和vertex cut两方面可以单独控制。

1. Edge Cut（默认）

* 目的：对于频繁遍历的边，应该减少cut edge的存在，从而减少跨设备间的通信，提高查询效率。即把进行遍历的相邻顶点放在相同的分区，减少通信消耗。
* 为顶点确定分区：JanusGraph通过配置好的 placement strategy来控制vertex-to-partition的分配。
* 默认策略：在相同事务中创建的顶点分配在相同分区上。
* 缺点：如果在一个事务中加载大量数据，会导致分配不平衡。
* 定制分配策略：实现IDPlacementStragegy接口，并在通过配置文件的ids.placement项进行注册。

1. Vertex Cut

* Vertex Cut：顶点切割，即把一个顶点进行切割，把一个顶点的邻接表分成多个子邻接表存储在图中各个分区上。
* 目的：一个拥有大量边的顶点，在加载或者访问时会造成热点问题。Vertex Cut通过分散压力到集群中所有实例从而缓解单顶点产生的负载。
* 方法：JanusGraph通过label来切割顶点，通过定义vertex label成partition，那么具有该label的所有顶点将被分配在集群中所有机器上。
* 案例：对于product和user顶点，product顶点应该被定义为partition，因为用户和商品有购买记录（edge），热销商品就会产生大量的购买记录，从而会造成热点问题。



* 1. 不同存储后端的性能比较

同样参数情况下对于不同的Cassandra集群测试结果如下：

Operation Count: 482

1. embedded Cassandra

Throughput: 65.78 (op/s)

Throughput: 65.92 (op/s)

Throughput: 65.92 (op/s)

Throughput: 65.59 (op/s)

1. Cassandra cluster 2 节点

Throughput: 24.34 (op/s)

Throughput: 66.05 (op/s)

Throughput: 65.89 (op/s)

Throughput: 65.96 (op/s)

1. Cassandra cluster 4节点

Throughput: 21.35 (op/s)

Throughput: 45.25 (op/s)

Throughput: 66.05 (op/s)

Throughput: 48.00 (op/s)

由于测试图数据集较小结果差距不大，但多节点Cassandra集群benchmark稳定性较弱，这可能和集群之间需要较多通信有关。