项 目 申 请 书

项目名称	图数据库 NebulaGraph 对接数据访问层 OpenDAL
项目导师	Suyan
申 请 人	陈昱辰
邮　　箱	feathercyc@163.com
日　　期	2024.06.03

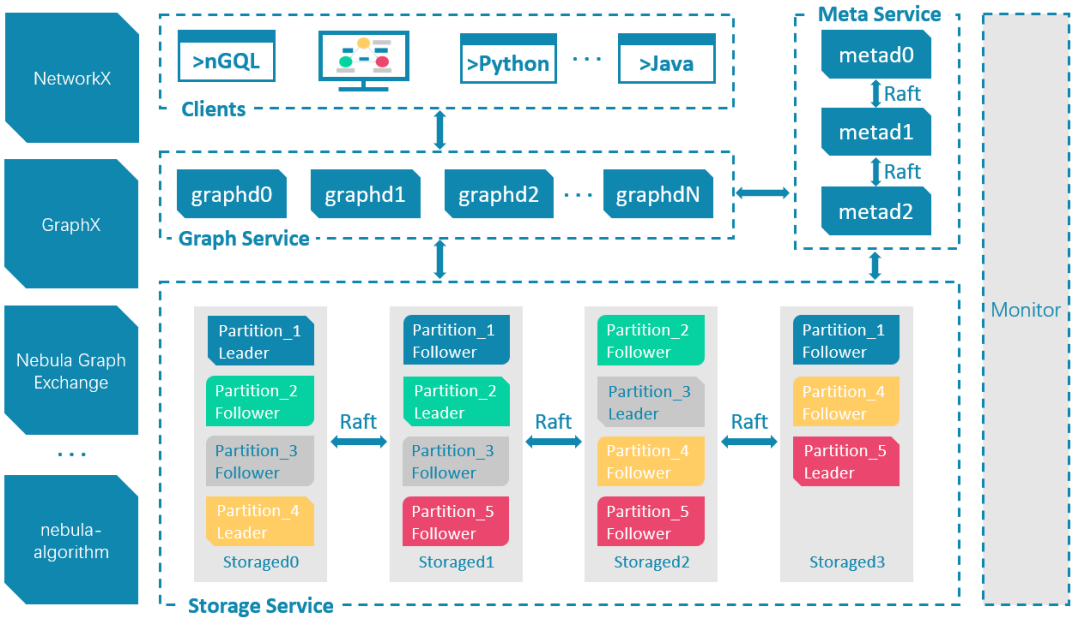
项目介绍

项目背景

NebulaGraph 简介

NebulaGraph 是图数据库，其以 vertex，edge 和 tag 的形式存储数据。其中 vertex 为点，在一般项目中可以是一个人、一篇帖子、一个组织等任意实体；edge 则是点之间的关系，如引用、属于、包含等各种关系；而 tag 则是用来修饰 vertex 的东西，如个人信息，帖子发布信息，组织信息等。从 tag 的角度看，vertex 可以视作为一堆 tag 的集合。

图数据库这样做的优点在于灵活性高，支持复杂的图形算法，可用于构建复杂的关系图谱。它可以看作是特化了传统关系型数据库的 JOIN 操作，简化了用户查询实体之间的关系的操作，换言之，图数据库是比关系型数据库更注重关系的数据库。



上图是官方的架构图，协议与生态细节略去不谈，易知 NebulaGraph 由三部分——graphd, metad, storaged 组成。其中 graphd 算是查询引擎，metad 存有服务地址和 Schema 等各类元信息，而 storaged 存储具体的数据。

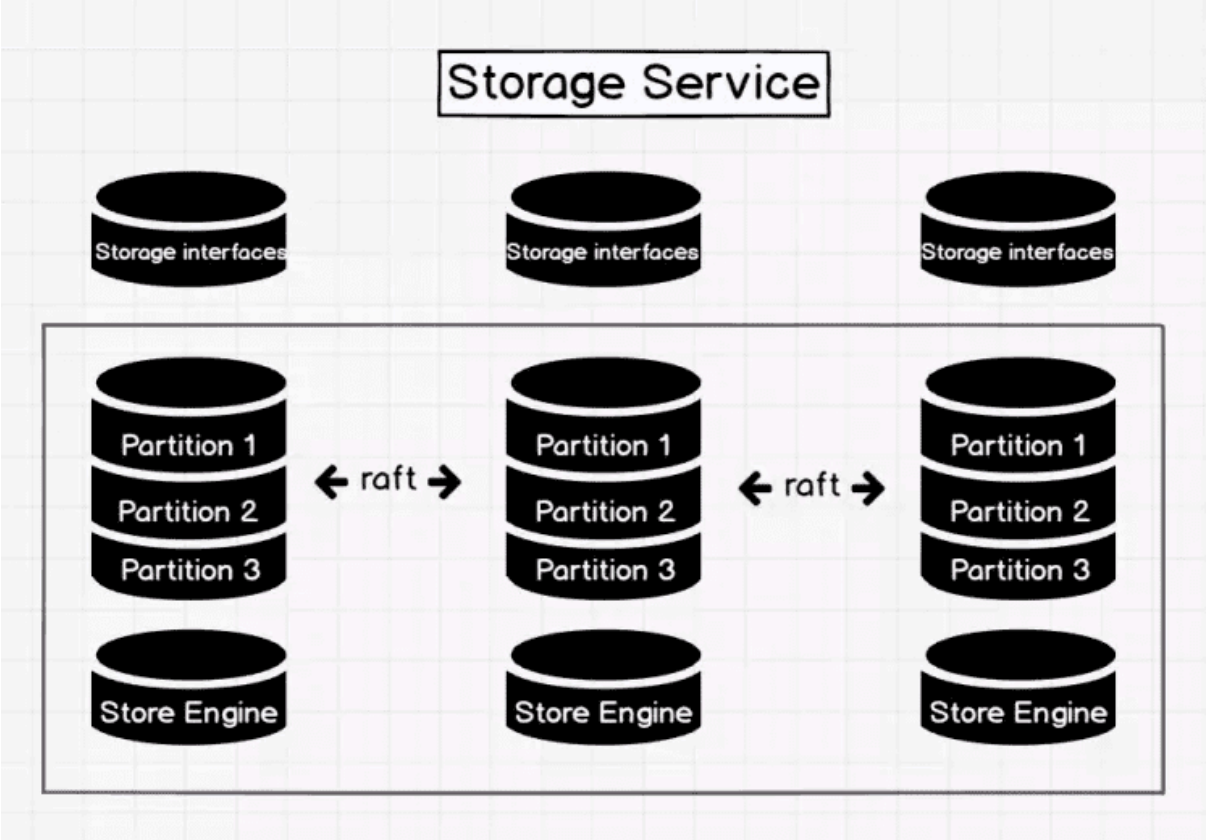
graphd

NebulaGraph 结合 GQL 自研了 nGQL(NebulaGraph GQL)。正如各类 SQL 引擎所做的那样，graphd 负责将 nGQL 解析成对底层存储引擎相应的操作，算子合并、下推等优化操作也在这里进行，与本任务关系不大。

metad 与 storaged

> Nebula 的 Storage 包含两个部分，一是 meta 相关的存储，我们称之为 Meta Service，另一个是 data 相关的存储，我们称之为 Storage Service。这两个服务是两个独立的进程，数据也完全隔离，当然部署也是分别部署，不过两者整体架构相差不大。
>
> https://www.nebula-graph.com.cn/posts/nebula-graph-storage-engine-overview

参考官方介绍，metad 与 storaged 在原理架构上并无太大区别，因此可以合并介绍，接下来只介绍 storaged。



> Storage interface 层
>
> Storage 服务的最上层，定义了一系列和图相关的 API。API 请求会在这一层被翻译成一组针对分片的 KV 操作，例如：
>
> - getNeighbors：查询一批点的出边或者入边，返回边以及对应的属性，并且支持条件过滤。
> - insert vertex/edge：插入一条点或者边及其属性。
> - getProps：获取一个点或者一条边的属性。
>
> 正是这一层的存在，使得 **Storage** 服务变成了真正的图存储，否则 **Storage** 服务只是一个 **KV**

> 存储服务。
>
> https://docs.nebula-graph.com.cn/3.8.0/1.introduction/3.nebula-graph-architecture/4.storage-service/

同样参考官方介绍，storaged 为基于 RocksDB 的分布式存储服务，这点与 TiKV 很像。OpenDAL 已经支持了 TiKV，OpenDAL 自然也可以用相似的方式访问一个独立的 NebulaGraph storaged，这并不困难。

nebula-rust 客户端简介

既然 NebulaGraph 提供了 3 个独立部署的 server，那么 Client 直连这 3 种 server 中任意一个也应是可行的。

官方提供了各种语言版本的 Client，不同语言 Client 的功能完善度都不同。其中 nebula-python 与 nebula-go 是官方支持最好的，毕竟官方的 nebula-console 是用 Golang 写的，而 python 主要用于数据科学方面，所以也得到了较好的支持。这两个项目都能独立连接 metad、graphd、storaged。

而 nebula-rust 客户端仓库目前目录与包含的项目就比较杂乱了，梳理介绍如下：

- nebula-fbthrift，这是由 fbthrift 自动生成的接口文件，没什么好更改的
- deserialize-nebula-fbthrift，这是用来解析 graphd、metad、storaged 返回的结果请求的 crate，目前只能解析 graphd 的请求和 storaged 的 scan vertex/edge 请求。根据 nebula-go 来看，这个 crate 有大量的工作可以做，主要集中在 storaged 和 metad 上，例如 insert/delete vertex/edge 等等。
- nebula-client，客户端 crate，本仓库的重点内容，功能依赖 deserialize-nebula-fbthrift，所以自然也只支持给 graphd 发请求和给 storaged 发 scan vertex/edge 请求。
- bb8-nebula，客户端连接复用池 crate，目前只实现了 graphd 的连接复用。
- demos/tokio，其中：
    - `v3_graph_client.rs` 为使用 nebula-client 访问 graphd 的示例
    - `v3_scan_vertex_edge.rs` 为使用 nebula-client 访问 storaged 的示例
    - `v3_bb8_graph_pool.rs` 为使用 bb8-nebula 访问 graphd 的示例

    这三个文件都实现了简陋的命令行参数解析功能，看文件夹名字能推测出来作者似乎还想多写几个不同的运行时示例。不过 Golang 版本的 nebula-console 已经推出了，这个 demos 更像是 examples 了。

该仓库基本处于放养状态，上次 commit 是在 7 个月前。我初步尝试了使用 nebula-client 连接 graphd 来执行自定义的 nGQL 并获取查询结果，过程还算顺利，至少按其 README 所言的方法连接并没有出什么问题。

总结一下，使用 nebula-rust Client 连接 NebulaGraph 目前有两种方式可选：

- 使用 nGQL 操作 NebulaGraph。这是最简单的方法，这种方式先请求 graphd 解析 nGQL，然后由 graphd 去请求 metad 和 storaged 获取数据，再返回给用户
- 直连 storaged。考虑到一些简单的、大批量的操作并没有必要经过 graphd 这个中转站再传给 storaged 去做，用户也可以直连 storaged 获取 vertex 与 edge

这两种方法各有优劣：

- 使用 nGQL 操作固然方便，但要是 graphd 和 metad、storaged 不在一个服务器上，再简单的操作也要多出一段时延。
- 直连 storaged 固然能减少时延，但对于 NebulaGraph 用户而言，metad 与 storaged 很多时候都是不对外暴露服务的，参考 Should I use graphd-client or storaged-client?，而且 nGQL 能实现的各种基于图的算法肯定是用不了的

OpenDAL 简介

OpenDAL 是一个数据访问层，其支持以相同的方式访问多种存储后端，旨在减轻开发人员要为每个服务重新实现访问各种存储后端的工作量。

要给 OpenDAL 新增 Service，最低要求是为新增的 Service 实现位于 `core/src/raw` 下的 `Access trait` 和 `Builder trait`。OpenDAL 主要支持的云存储后端都实现了这两个 trait。

而要新增 DB Service 则简单不少，因为 OpenDAL 定义了 `Adapter trait`，实现这个 `trait` 就可以让 OpenDAL 使用任何 KV 工作。因此，只要将各类 DB 的操作映射为 KV DB 的操作——`set`, `get`, `delete`, `scan`，OpenDAL 就可以使用这个 DB 了。

实际上，直接为 DB 实现 `Access trait` 也是可行的，OpenDAL 能访问实现了 `Adapter trait` 的 DB 是因为 OpenDAL 自己为 `Adapter trait` 实现了 `Access trait`。这在简化了添加 DB 服务难度的同时，也减弱了 DB 能提供的功能。

项目目标

参考 Add support for NebulaGraph，该项目希望为 OpenDAL 添加对 NebulaGraph 的支持以便用户可以通过 OpenDAL 访问 NebulaGraph 并存储数据。

使用方法会像这样：

```
let op = Operator::via_map(Scheme::Nebula, map)?;
let bs = op.read("path/to/file").await?;
```

本提案计划为新增的 NebulaGraph Service 直接实现 `Access trait`，它将支持以下操作：

- `write`
- `read`
- `delete`
- `rename`

同时，本提案计划为 NebulaGraph Service 提供两种访问方式，一种是通过 graphd 访问，一种是通过 storaged 访问。

用户可以通过指定 space，tag 与 tag 中的属性名来轻松访问 NebulaGraph 中的数据。

提案方案设计

由上可知只需将图数据库映射为 KV DB 就够了。

graphd 连接

考虑这样的 space 设计：

```
CREATE SPACE test_kv(PARTITION_NUM = 20, VID_TYPE = INT);
CREATE TAG file(name string, value string);
```

以 INT 作为 space 的 VID 类型，VID 可以是雪花算法或者其他算法产生的 INT，唯一即可。TAG 可以简单地看作是 SQL DB 中的表，其中 name 记录文件名，value 记录文件内容。

而要模拟 KV DB 操作只需按下列命令操作：

```
// insert key-value pair, randomInt could be generated by snowflake
algorithm or others
INSERT VERTEX kv VALUES randomInt:("test", "23");

// get value by key
MATCH v(file{name:"test"}) RETURN v.value;
+-------+
| value |
+-------+
| "23"  |
+-------+

// delete key-value pair
MATCH v(file{name:"test"}) RETURN v; | DELETE VERTEX $-.id;

// rename key
MATCH (v:file{name:"test"}) RETURN v;
UPDATE VERTEX ON file v SET name = "begin";
```

这样 NebulaGraph 将支持 `write`、`read`、`delete`、`rename` 操作。

这是通过 graphd 访问 NebulaGraph 的方案，NebulaGraph 的 graphd 与 storaged 之间正如 TiDB 与 TiKV 之间的关系，若是想单独连接 storaged 操作数据也是可行的。

storaged 连接

正如上文所说，storaged 除了包装一层图语义接口外与分布式 KV 差异不大，而 nebula-go 已经支持了直接对 storaged 插入/删除 vertex 操作，这说明在原理上来说 storaged 完全有能力作为 OpenDAL 的后端。

映射逻辑与上文无异，要实现这个的主要工作只需补全 nebula-rust storaged Client 这方面的操作，因为目前 nebula-rust 支持读取 storaged 但不支持写入 storaged。只要补全这一功能，那么 OpenDAL 也可以支持直连 NebulaGraph storaged 作为后端。

计划表

参照官方日程表，如果我被选中，我将在 07/01-09/30 期间进行开发。我预计每天为这个项目投入 4-6 小时时间，周六日以及暑假期间全天候开发。

初定计划如下：

1. 7月份初 - 7月份末，主要测试方案可行性，确保 nebula-rust 客户端可用，并实现 OpenDAL 对 NebulaGraph 基础的连接功能：
    - 07/01 - 07/07

      使用 nebula-console 测试上述方案的 nGQL 性能与可行性，这一阶段主要是验证可行性并测试性能，如固定 255 长度的 VID 对性能的影响，一定数量文件下两种 space 的性能对比。

    - 07/08 - 07/21

      测试 Nebula rust 客户端，若是上述 nGQL 方案可行且性能也能接受，那么这一阶段主要为上述的 nGQL 写 Rust 结构体，以及找出 nebula-rust 潜在的 bug 并予以修复。

    - 07/22 - 07/28

      为 NebulaGraph Service 实现 `Builder trait`，这一阶段主要为 OpenDAL 添加连接 NebulaGraph 的支持。

2. 7月末 - 9月中上旬，这一阶段开始正式实现项目的主要功能：
    - 07/29 - 08/11

      为 NebulaGraph Service 实现 `Access trait` 中的 `write`、`read`、`delete`，这三个功能是最基础的功能，实现了这三个操作能保证 NebulaGraph 基础的可用性。

    - 08/12 - 09/01

      尝试为 nebula-rust 完善 storaged Client 的相关功能，进展顺利则后续流程与 graphd 方案无异。

    - 09/02 - 09/08

      测试 Nebula rust 连接复用客户端，这里主要是优化 OpenDAL 对 NebulaGraph 的连接性能。连接复用是锦上添花的事，这项工作的耗时长短取

决于 bb8-nebula 的完善程度。同样的，我也会在使用 bb8-nebula 的过程中去寻找可能的 bug 并尝试修复。

3. 9月中上旬 - 结项日期，这一阶段是预留阶段，前面阶段进展都顺利的话，这几周我会跑一跑 OpenDAL 自己的 benchmark，对比一下其他的服务；撰写结项报告；帮 OpenDAL 修几个力所能及的 bug；还可以为 bb8-nebula 实现 storaged 的连接复用。要是前面阶段进展不顺利，这一阶段就是缓冲段，可以继续推进。