项目名称

Fly to M78

项目简介

针对目前Nebula在OLTP和OLAP下的查询和数据更新性能有待提高的问题，我们目标通过对Nebula的整体架构以及不同层（layer）进行优化来提升Nebula的TP和AP性能，包括Nebula存储引擎的优化，caching等。

项目背景

目前Nebula无论在OLTP还是OLAP场景下都存在比较明显的性能问题。其原因存在于多个方面，包含底层的存储引擎，整体架构，以及Nebula不同layer。

从存储引擎来讲，存储引擎直接影响了每一次数据的读写，也因此直接影响了TP和AP的所有query。Nebula目前使用了RocksDB作为存储引擎。我们知道图workload最主要的特征是对于关系的多层次遍历。一次图数据库的查询往往需要多次对存储引擎的读取。因此对RocksDB一次读取的性能劣势在图查询中会被多次放大。这也是为什么多跳图查询延迟巨大。反过来说，对于单次RocksDB查询的性能提升也将在Nebula查询中被多次放大。因此如何优化RocksDB，成为为优化存储端性能的重中之重。

目前Nebula采用了存算分离的架构。存算分离可以带来非常优异的可扩展性（scalability），但同时也带来了额外的计算服务和存储服务之间的通信开销。首先，由于数据是shard到不同的存储节点上，一个query往往需要访问多个存储节点才能完成（比如取一个节点的所有邻接点属性）。不仅如此，如果一个query需要深度查询图，例如从一个节点出发访问多跳之后的节点，那么需要计算节点和存储节点之间多次通信才能完成查询。考虑到存算分离的这个tradeoff，我们将考虑如何在保持scalability的同时，减少计算服务对于存储服务的访问。

通过以上两个部分的优化，我们期待可以对Nebula OLTP和OLAP都带来巨大的性能提升。

项目价值

对Nebula OLTP和OLAP场景都带来巨大的查询和写入性能提升。

项目设计

1. 存储引擎优化

Nebula目前使用RocksDB。RocksDB的核心数据结构是LSM Tree。所有key和value都存于LSM Tree中。但这种设计有一个缺点。由于value往往比key大，LSM Tree中大部分的空间是用来存value的。一旦value特别大，那么LSM Tree会需要更深的level来存储数据。而LSM Tree的读写性能是直接跟层数负相关的。层数越深，可能带来的读写放大就越多，性能也就越差。因此我们提出使用KV分离来存储图数据库：将值较小的数据存在LSM tree中，而将值较大的数据存在log中。这样由于大value并不存在LSM tree中，LSM tree的高度会降低。考虑到RocksDB相邻两层的size是10倍关系。因此哪怕减少一层LSM tree高度，也能大大增加cache的可能性，从而提高读性能。哪怕不考虑cache，KV分离带来的读写放大的减少也能加快读写速度。

1. 架构优化

为了保持Nebula的高可扩展性，我们必须保持当前的存算分离结构。如何在此基础上，减少计算节点和存储节点的通信是我们的优化目标。为此，我们提出了计算节点也就是graph server上的cache。在此hackathon中，我们主要研究邻接点信息，也就是edge的cache。当一个点的所有边都存在于cache中时，graph节点就不需要向存有此节点的存储节点发送请求。甚至如果该点的所有邻接点的边也都存在于cache中，就可以省去计算节点向多个存储节点发送请求。其省去的通信和存储开销将随着遍历深度提升而指数型增加。

但graph server cache也存在着一致性问题。由于计算节点是无状态的，增加cache后，每一个计算节点都可能存有同一份数据的cache。一旦有数据更新，就需要访问所有计算节点来淘汰cache或者更新cache。但由于graph

server之间并无通信机制，一旦某个graph server收到了数据更新请求，只能通过meta server和其他graph server之间通信。目前，meta server和graph server之间的通信是有延迟的。因此graph server cache目前做到强一致性比较困难。我们在此Hackathon中主要关注graph server cache对于OLAP场景的提升。因为OLAP中，数据往往是集中更新和集中查询。在更新后，我们可以集中销毁graph server中的cache，甚至可以直接等待其自己expire（e.g, 60s）。这个等待时间和AP查询所需时间是可以忽略的。

项目测试

单机benchmark
分布式benchmark

真实query测试

真实OLAP测试