TGraph:时态图数据管理系统

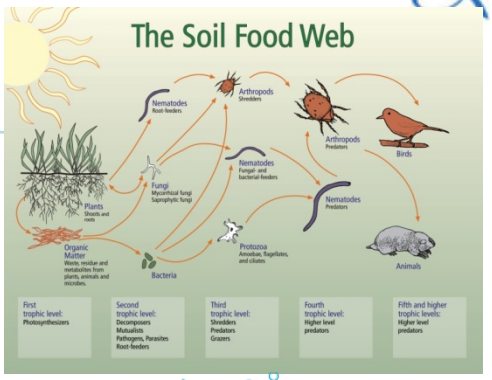

马 帅

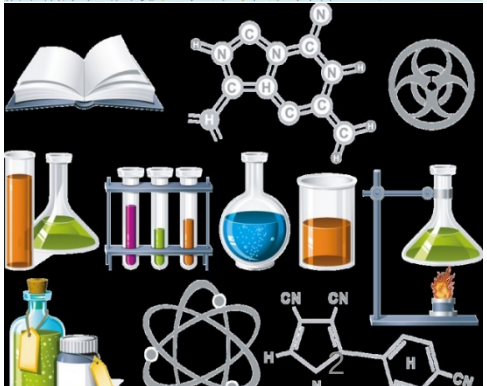
BIG DATA
BRAIN COMPUTING
大数据科学与脑机智能高精尖创新中心

北京航空航天大學
BEIHANG UNIVERSITY

图 (Graphs, Networks)

- 现实世界中有大量数据可以通过图来建模，例如
  - 生态网络
  - 社交网络
  - 道路交通网络
  - 生物医药
- 用图来对这些数据建模，可以帮助我们更好的理解这些系统中的元素及其相互关系。

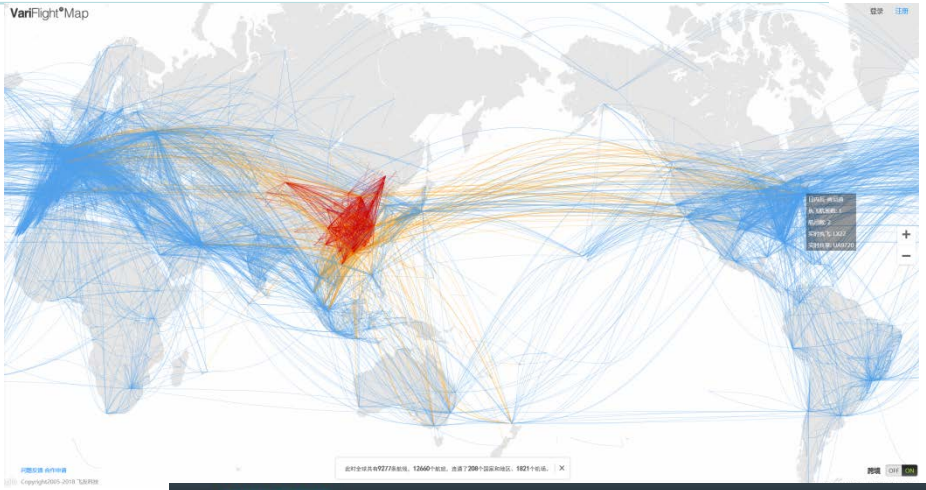




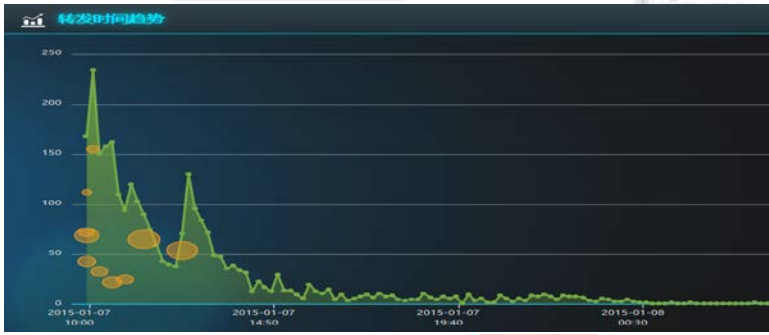
然而很多情况下，图的结构及点边上的属性是随时间动态变化的，研究这种变化可以更好的理解系统的动态演化过程、开发更多应用。

时态图 (Temporal Graphs)

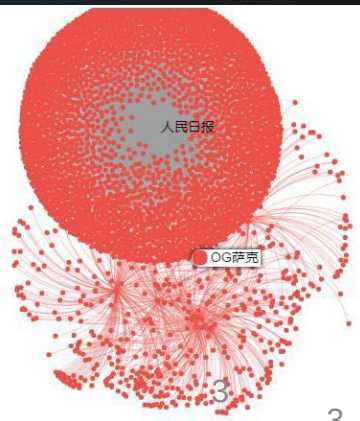
时态图是一类特殊的图，其结构或点、边上属性的值会随时间动态变化。



- 例1（航班运行Graph）
  - 节点：机场
  - 有向边：航线
    - ✓ 边仅在某个航班运行时存在。
    - ✓ 跨机场中转的路径规划需考虑边的有效时间。



- 例2（微博传播Graph）
  - 节点：微博用户
  - 边：转发某一条微博
    - ✓ 对某个热门微博的转发是在一段时间内完成的。
    - ✓ 整个传播Graph的动态形成过程需要结合时间进行分析。
      - 单位时间的转发次数会很快达到峰值，并迅速下降
      - 某些大V的二次转发会形成新的转发高峰。

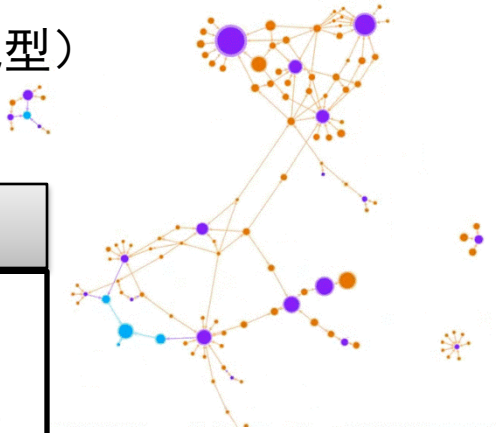


时态图通过引入时间维度来建模图的演化

3

时态图：分类 & 建模

- 时态图的两种变化：结构变化、属性值变化
- 按照哪种类型的变化更主要，可以大致将时态图数据分为两类
  - 图结构变化频繁变化，点/边属性值变化较少（结构演化型）
  - 点/边属性值频繁变化，图结构变化较少（属性演化型）

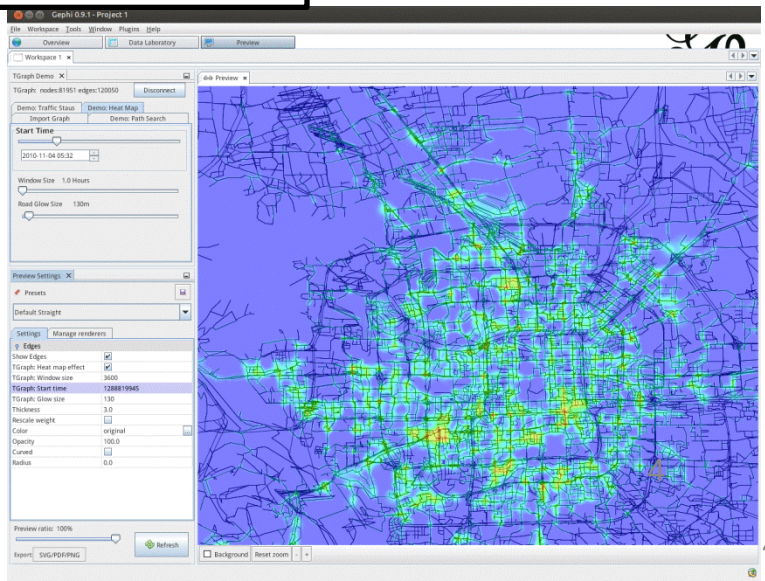


典型的属性演化型时态图数据：物联网系统产生的数据

- ◆ 物联网传感器之间是有逻辑联系的（图结构）
- ◆ 物联网传感器可以产生不同类型的数据（点/边属性）
- ◆ 传感器产生数据的速度远高于传感器之间关系的变化的速度

- 为何要分类
  - 对两类数据分别建立更直观高效的数据模型
  - 两类数据的常见读写模式和查询不同

TGraph主要针对属性演化型数据优化



应用场景简介

城市道路交通路况信息管理

- 某城市的道路（约10万条）上安装有探测装置，每台装置每隔5分钟左右自动上报一次道路的交通状况。上报信息包括
  - ✓ 拥堵情况（status，三个状态）：jam、slow、smooth
  - ✓ 通行时间（ travel-time，单位秒 ）
  - ✓ 道路上的车辆数（ vehicle-count，单位辆 ）
  - ✓ 拥堵路段数（ jam-count ，单位个）
- 每条道路有一些静态信息如道路长度、道路类型等
- 支持路况信息的实时更新和查询
  - ✓ 道路在某时刻的历史状态
  - ✓ 统计道路在某段时间内的状态
  - ✓ 路径规划（最快到达、最晚出发、最短时间）
  - ✓ 区域可达性分析

现有时态图系统

- AT&T
  - Nepal [Sigmod'16'17'18] VNF/SDN网络服务管理及trouble shooting
- 微软研究院、微软亚洲研究院
  - Kineograph [EuroSys'12] 实时edge-stream分析挖掘
  - Chronons [EuroSys'14] Immortal Graph [TOS'15]
  - 为时态图在内存中进行的iterative computation进行优化
- 马里兰大学帕克分校、IBM
  - DeltaGraph [ICDE'13] 管理与分析动态图
  - Historical Graph Store [EDBT'16] 存储分析大规模图变化历史
- 纽约州立大学、浦项科技大学
  - G* [DPD'15][ICDE'13] 存储分析大规模时态图数据

TGraph vs 现有时态图系统

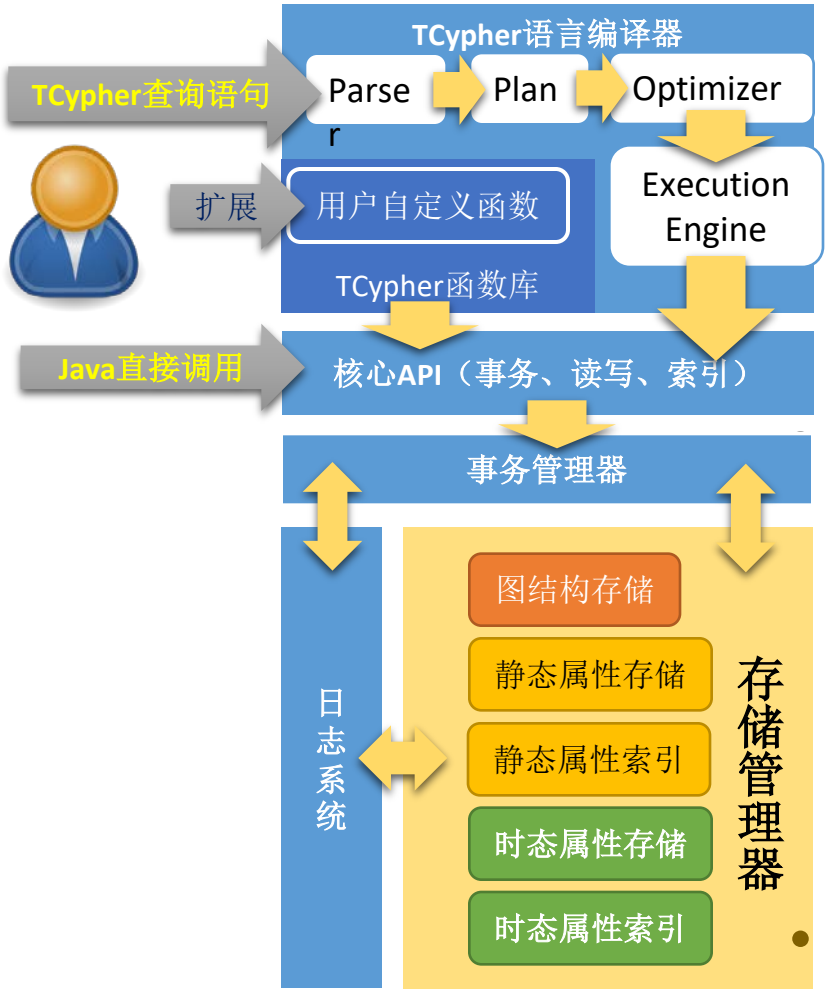
	Nepal	Kineograph & Chronos & Immortal Graph	Delta Graph & Historical Graph Store	G*	TGraph
Type	OLAP/OLTP	OLAP	OLAP	OLAP	OLTP
ACID Support	√	-	-	-	√
Model (impl.)	Event Log	Event Log	Event Log	Snapshot List	Temporal Property Graph
Based on	PostgreSQL	-	Spark & Cassandra	-	Neo4j
Query Language	Nepal	API	API	API	API & TCypher
Distributed	-	√	√	√	-

查询功能接近Nepal（更丰富）

TGraph时态图管理系统

- 时态图数据管理系统（数据库）
  - 针对属性演化时态图数据优化的存储
    - ✓ 高速实时写入（ ？ MB/s）
  - 支持事务 (ACID)
  - 丰富的时态查询功能
    - ✓ 时态图查询语言TCypher
    - ✓ 支持Temporal Predicate查询
    - ✓ 支持时态最短路径查询
  - 时态索引
    - ✓ 加速Temporal Predicate查询
    - ✓ 加速Temporal Aggregation查询
  - Native Graph Storage（fast!）
  - 向后兼容Neo4j（到2.3.12版）

TGraph时态图管理系统：架构



主要模块

- TCypher查询语言编译器
- 时态图核心操作API
- 事务管理、日志管理、存储管理、索引管理

使用方式

- 发送TCypher查询语句到TGraph Server
  - ✓ 查询→编译→执行计划→调用核心API运行
  - ✓ TCypher包含一个时态查询相关的函数库。用户可以自定义函数来扩展该函数库，自定义函数通过Java语言编写，通过调用核心API完成相关功能。
- 直接调用TGraph系统核心API
  - ✓ 核心API是一组Java类。提供了事务操作，基本的时态图读写操作，及建立/删除索引操作。

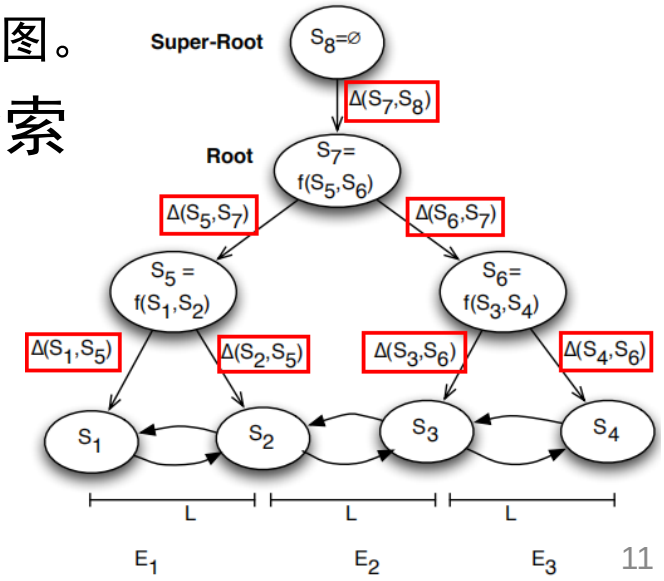
所有操作都在事务内进行。系统采用redo日志，更新存储和索引前会先写入日志。

TGraph时态图管理系统：架构

- 系统基于图数据库Neo4j构建
  - 支持事务
  - Native Graph存储
  - 有较成熟的查询语言Cypher
  - 可以更好的利用Neo4j生态系统
- 对Neo4j的改动
  - 新增时态属性存储/索引模块
  - 修改事务、日志、语言等模块

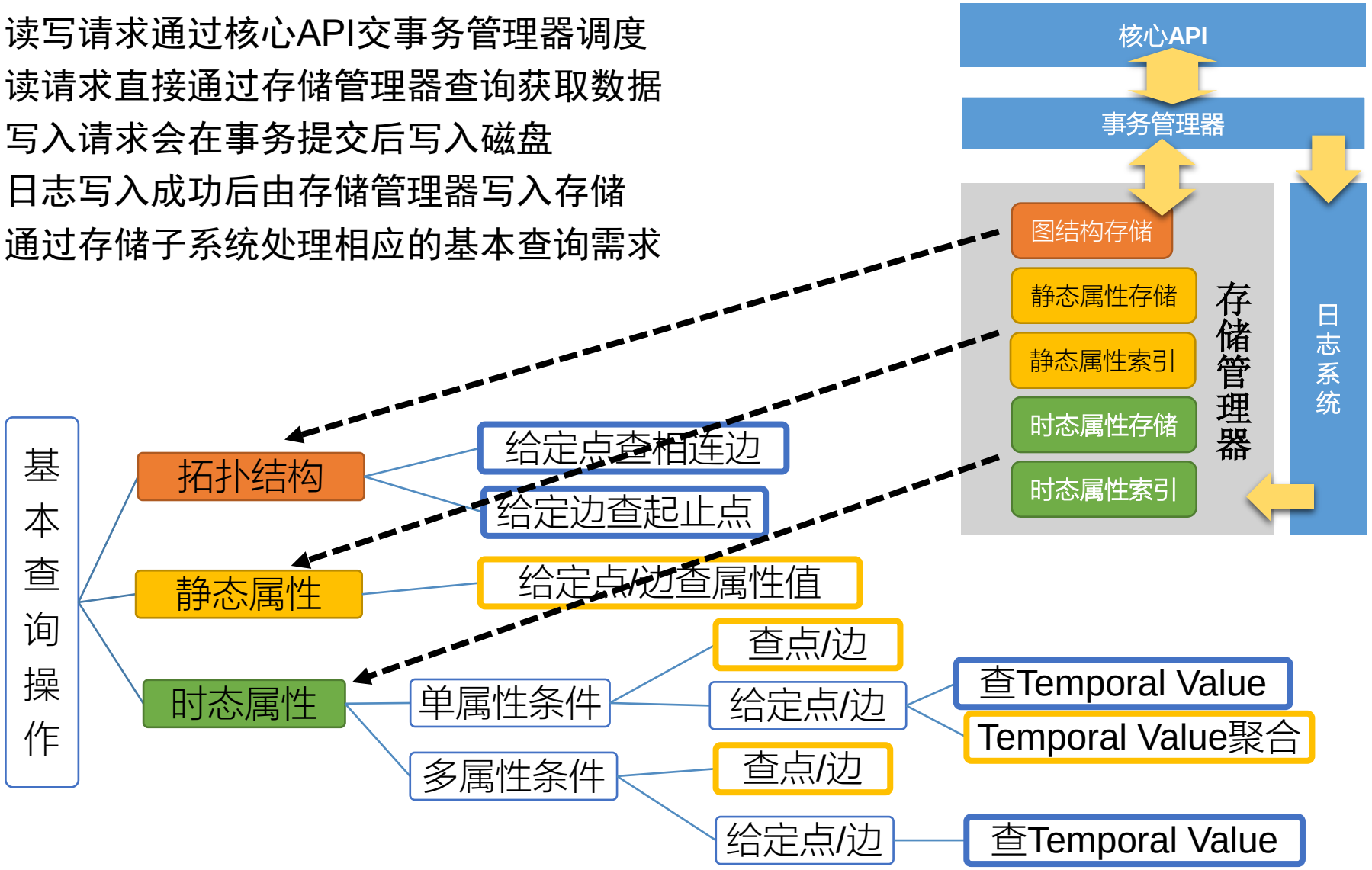
TGraph存储架构

- G*将时态图按子图切分到不同的机器上，每个机器保存这个子图的历史变化数据。
  - 基本数据项是（图id、点id、属性列表、相邻点id列表）
  - 如果点在不同图中属性邻边无变化则压缩存储
  - 在内存中建立点id到磁盘位置的B+树索引（快速按ID查询）
- Chronos是为(内存)图迭代计算设计的计算引擎
  - 计算时首先把需要的图数据全加载到内存中
  - (内存中)同一个点不同时间的数据放在一起减少cache miss
  - (磁盘上)通过event日志加checkpoint存储时态图。
- Delta Graph的目标是快速snapshot检索
  - 基本存储是event log
  - 构造由不同snapshot之差(delta)组成的树(图)
- Nepal是基于PostgreSQL数据库封装



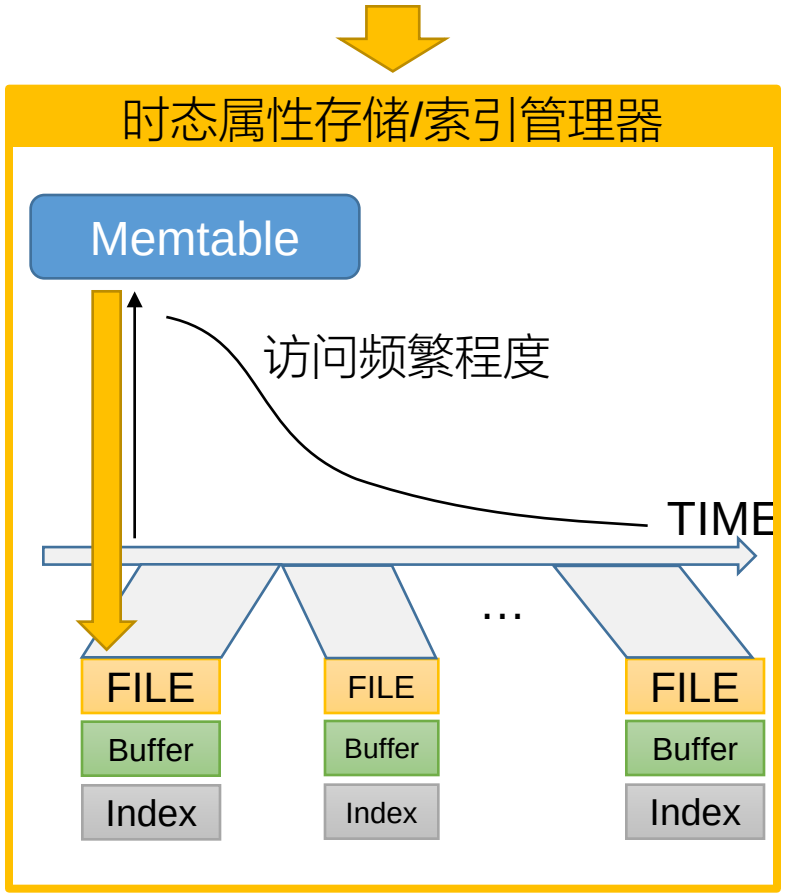
TGraph存储架构

读写请求通过核心API交事务管理器调度
读请求直接通过存储管理器查询获取数据
写入请求会在事务提交后写入磁盘
日志写入成功后由存储管理器写入存储
通过存储子系统处理相应的基本查询需求



时态属性存储架构

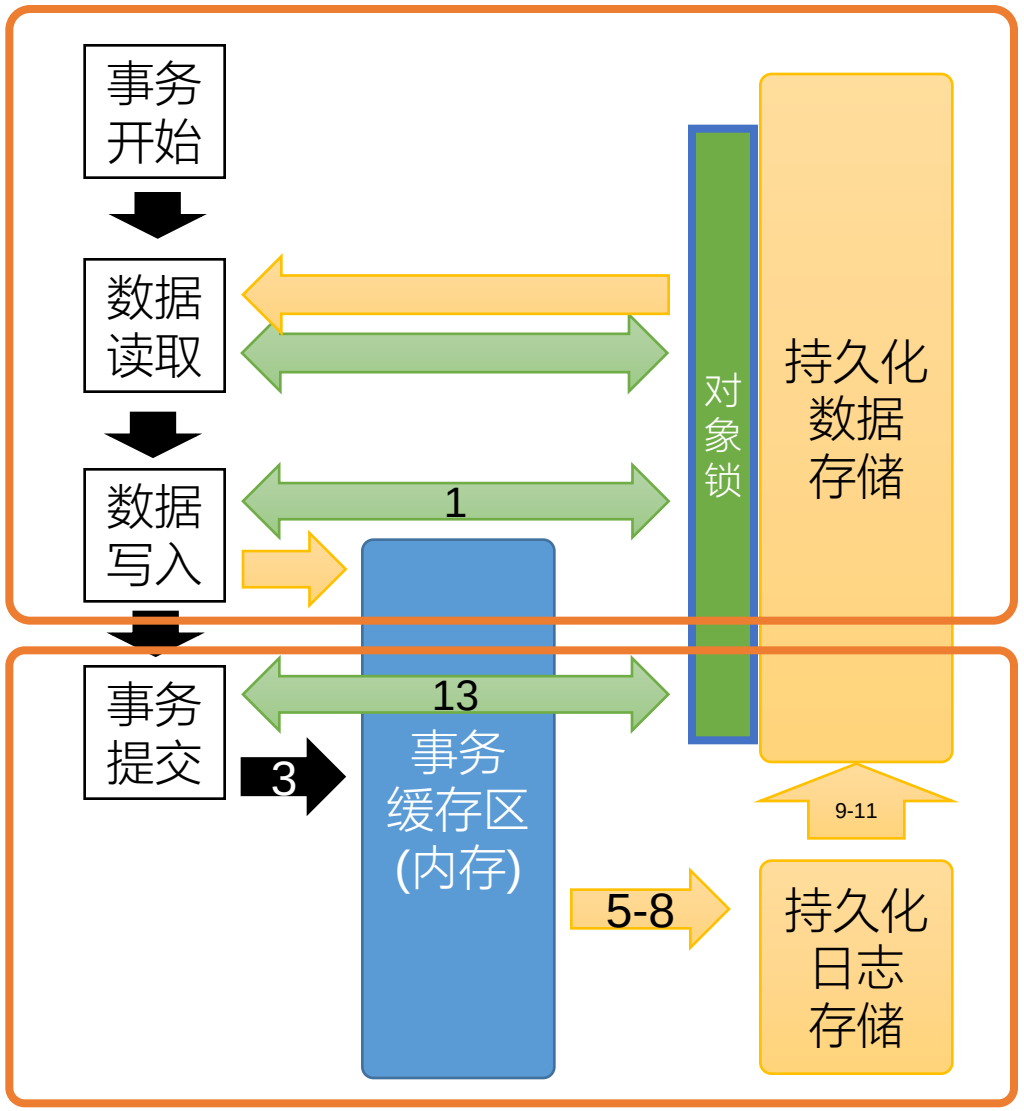
- **设计目标(考虑特定数据写入/访问模式)**
  - 减少写入和查询时的磁盘（随机）IO
  - 减少存储大小
- 考虑Access pattern
  - 文件按时间排序，且时间不相交
- 减少IO
  - 数据先写内存，到一定大小后批量写磁盘
  - 不修改磁盘上的已有数据文件
  - FILE的修改全部写入Buffer（类似于日志）
  - Buffer达到一定大小后与FILE合并
  - Memtable写磁盘及File合并均由后台线程完成
- AS限制：FILE合并、打Checkpoint
- 其他
  - 查询时需要联合查FILE、Buffer、Memtable
  - Index可选，存储Time Value或Temporal Aggr 索引
    - ✓ Time Value索引内容和FILE相同，但是另一种AS
  - 点和边的时态属性分别放在两个时态属性存储instance



时态属性存储实例

借鉴了Level DB、Cassandra等存储系统的特点

事务运行和提交流程



事务的ACID特性通过锁、日志、缓存区等机制实现

每个事务在创建时都会在内存中分配一块缓存区

```
1.  Lock_X (entity_id)
2.      将数据更新写入事务缓存区
3.      //开始提交
4.      Lock_X (journal)
5.          写入事务的start到日志
6.          所有数据更新写入日志
7.          写入事务end标志
8.          将日志flush到磁盘
9.          Lock_X (Storage)
10.             将数据更新写到存储系统
11.         Unlock_X (Storage)
12.     Unlock_X (journal)
13. Unlock_X (entity_id)
```

日志 & 检查点 & 恢复

Redo日志（格式见下）

- 重复apply到存储多次的结果一样。

Check Point

- 有一些数据没有及时写磁盘
- 不用每次从日志最开始恢复
- 系统正常关闭时会写一个check point

打checkpoint流程

1. Lock_X (journal)
2. Lock_X (内存数据)
3. 将所有内存数据flush到磁盘
4. 在日志中打checkpoint并flush日志到磁盘
5. Unlock_X (内存数据)
6. Unlock_X (journal)

故障恢复流程（系统启动时）

- 反向读取日志，找到最新的检查点
- 正向读取日志，将已提交事务（有end标记的）redo
- 所有内存数据输出到磁盘并打一个checkpoint

<entity_id, property_id, new_value> //静态属性更新
<entity_id, property_id, time_start, time_end, new_value> //时态属性更新
<property id list，索引类型，时间区间> //时态属性索引创建/更新
<Tx, start> //事务start
<Tx, commit> //事务end 标志
<Tx, checkpoint> //检查点

部分操作的日志记录示意

锁 & 死锁检测

	脏读	不可重复读	幻读
读未提交	√	√	√
读提交	×	√	√
可重复读	×	×	√
序列化	×	×	×

事务隔离级别

- TGraph默认为“读提交（Read Committed）”
- 可以手动提升为“可重复读”
- 使用共享锁和排他锁机制实现

死锁检测（资源分配图）

- 两类节点：申请资源锁的事务，资源本身
- 申请资源锁
  - ✓创建一条由事务指向资源的边。
  - ✓检查资源图中是否存在含有上述边的环。，存在则说明会死锁，抛出异常，并标记本次事务失败并结束事务
- 得到资源锁
  - ✓删除刚才创建的边
  - ✓新建一条由资源指向事务的边。
- 释放锁
  - ✓删除资源-事务边。
- 事务结束
  - ✓删除事务顶点。
- 资源无事务使用时
  - ✓删除资源顶点。

资源上已经放置的锁	第二个事务进行读操作	第二个事务进行更新操作
无	立即获得共享锁	立即获得独占锁
共享锁	立即获得共享锁	等待第一个事务解除共享锁
排它锁	等待第一个事务解除排它锁	等待第一个事务解除排它锁

Cypher in Neo4j

Cypher

```
(a) -[:LIKES]-> (b)
```

- Cypher是Neo4j为属性图查询设计的类SQL查询语言
- Data Model [Francis2018sigmod]
  - ✓Value
  - ✓Graph
  - ✓Table

```
MATCH (node:Label) RETURN node.property

MATCH (node1:Label1)-->(node2:Label2)
WHERE node1.propertyA = {value}
RETURN node2.propertyA, node2.propertyB
```

- DQL（流程）

1.Path匹配：MATCH path_pattern

2.投影查询结果为表：WITH/UNWIND/RETURN

3.表上的查询：WHERE、GROUP、HAVING、COUNT/MIN/MAX

- DML

创建/删除点边：CREATE node/rel、DELETE node/rel

创建/删除标签、属性：SET/REMOVE label/property

- DDL

创建/删除索引、约束：CREATE/DROP INDEX/CONSTRAINT

- DCL：无

Cypher: An Evolving Query Language for Property Graphs. SIGMOD 2018.
Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Platnikow, Mats Rydberg, Petra Selmer, Andres Taylor.

TCypher语言对Cypher的扩展

- 新增2种基本数据类型
  - Time Interval用于描述时间区间
    - ✓ ‘2011-11-01 08:53:00’ ~ 1533201391
    - ✓ INIT ~ NOW
  - Temporal Value用于描述时态属性的值
    - ✓ 例(常量)：TV(1 ~ 2: ‘A’, 6~8:’B’, 10:’C’, 14~NOW:’D’)
    - ✓ 例(创建节点时指定时态属性)
      - CREATE ({name:'my', tp:TV(1~4:16, 5~8:9, 12~2010:3)})
    - ✓ 例(已有节点创建/修改时态属性)
      - MATCH (n{name:'my'}) SET n.tp = TV(1~4:16, 5~8:9, 12~2010:3)
- 新增时态相关函数
  - 构造时态属性查询条件
  - 操作Temporal Value
  - 创建时态属性索引

谢谢！