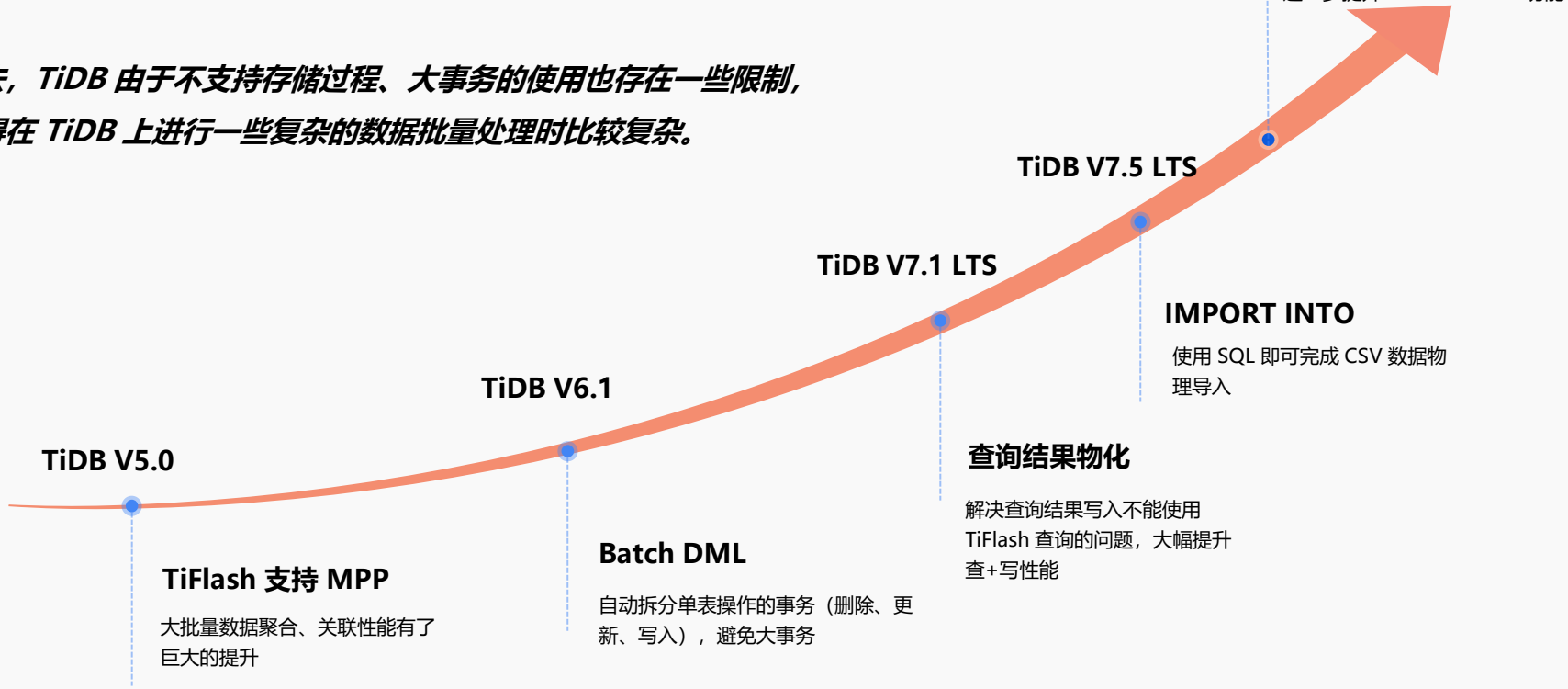
TiDB 7.5 LTS 高性能批处理方案

PingCAP | 售前顾问 | 汤博文

背景介绍

TiDB 批处理能力演进

过去，TiDB 由于不支持存储过程、大事务的使用也存在一些限制，使得在 TiDB 上进行一些复杂的数据批量处理时比较复杂。



过去 TiDB 中批处理场景有哪些

单表批量删除/更新场景

- **update ... where ：** 简单；大事务、消耗内存高，写入是单并发
- **delete ... where ：** 简单；需拆分文件大小+多并发来获得高性能（可能会遇到热点问题）
- **使用 limit 循环处理：** 比较简单；性能差
- **根据 PK 范围分批处理：** 复杂；多线程操作性能高

批量数据写入场景

- **INSERT INTO ... SELECT：** 简单；大事务、消耗内存高，写入是单并发
- **LOAD DATA：** 比较简单；需拆分文件大小+多并发来获得高性能（可能会遇到热点问题）
- **JAVA：** 需要一定编码；使用 Batch 方法+多并发可获得高性能（可能会遇到热点问题）
- **ETL 平台：** 简单；ETL 平台多线程写入难以控制

查询结果导出场景

- **循环 limit m,n 处理：** 简单；大数据量场景，性能极低
- **根据 PK 范围分批处理：** 复杂；拆分多个 range 后，性能较高

以上是我接触的客户中，比较常见的一些用法。

单表批量写入/更新/删除场景优化方案

优化方案 - 单表批量写入/更新/删除场景

Batch DML（TiDB 6.1/6.5 引入）：

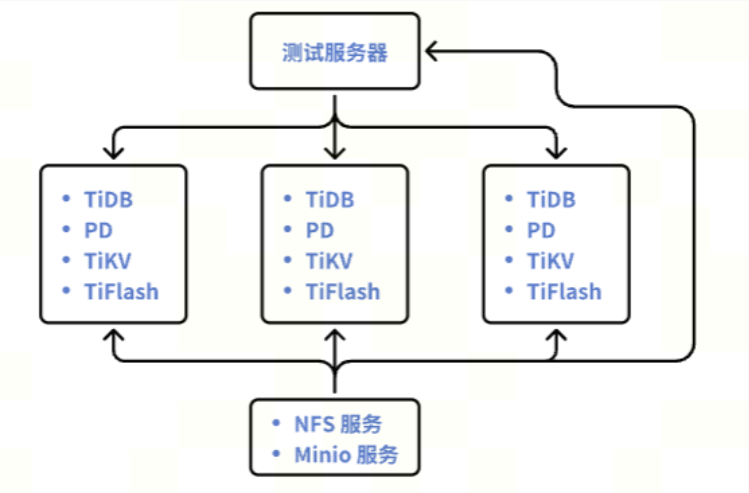
自动拆分单表操作的事务（删除、更新、写入），避免大事务

- 快照表加工
  - 之前：INSERT INTO inventory_snapshot SELECT … FROM inventory;（风险：大事务，性能低）
  - 现在： BATCH (ON id) LIMIT 10000 INSERT INTO inventory_snapshot SELECT … FROM inventory;
- 批量数据刷新
  - 之前： UPDATE order_dtl SET status = 7 where status = 0;（风险：大事务，性能低）
  - 现在：BATCH (ON id) LIMIT 10000 UPDATE order_dtl SET status = 7 WHERE status = 0;
- 历史数据删除
  - 之前： DELETE FROM order_dtl WHERE create_time < ‘2020-01-01’ ;（风险：大事务，性能低）
  - 现在：BATCH (ON id) LIMIT 10000 DELETE FROM order_dtl WHERE create_time < ‘2020-01-01’ ;

在单表操作场景，可解决大事务、内存消耗高、甚至性能不佳问题。

复杂场景如何优化?

测试环境



TiDB 集群资源

- 3 台 16VC/64GB 虚拟机 + SSD 云盘（3500 IOPS + 250MB/S 读写带宽）
- TiDB 版本：TiDB V7.5.0 LTS
- TiDB 组件：TiDB/PD/TiKV/TiFlash（混合部署）

存储资源

- 1 台 8C/64GB 虚拟机 + SSD 云盘（3500 IOPS + 250MB/S 读写带宽）
- 存储服务：NFS 服务、Minio 对象存储

测试资源

- 1 台 8C/64GB 虚拟机 + SSD 云盘（3500 IOPS + 250MB/S 读写带宽）
- 测试服务：datax + Dolphin调度/java 程序/dumpling、tidb-lightning 工具以及 MySQL 客户端

测试场景：将大批量查询快速写入到结果表中

查询 SQL

```sql
select  c_custkey,c_name,sum(l_extendedprice * (1 - l_discount)) as revenue,
        c_acctbal,n_name,c_address,c_phone,c_comment,min(C_MKTSEGMENT),min(L_PARTKEY),
        min(L_SUPPKEY,min(L_LINENUMBER),min(L_QUANTITY), max(L_TAX), max(L_LINESTATUS),
        min(L_SHIPDATE), min(L_COMMITDATE), min(L_RECEIPTDATE), min(L_SHIPINSTRUCT),
        max(L_SHIPMODE), max(O_ORDERSTATUS), min(O_TOTALPRICE), min(O_ORDERDATE),
        max(O_ORDERPRIORITY), min(O_CLERK), max(O_SHIPPRIORITY),
        @@hostname as etl_host,current_user() as etl_user,current_date() as etl_date
from
        tpch.customer,tpch.orders,tpch.lineitem,tpch.nation
where
        c_custkey = o_custkey and l_orderkey = o_orderkey
        and o_orderdate >= date '1993-10-01' and o_orderdate < date '1994-10-01'
        and l_returnflag = 'R' and c_nationkey = n_nationkey
group by
        c_custkey,c_name,c_acctbal,c_phone,n_name,c_address,c_comment
order by c_custkey;
```

```
+------------+-----------+
| table_name | cnt_rows  |
+------------+-----------+
| customer   |  15000000 |
| orders     | 150000000 |
| lineitem   | 600037902 |
| nation     |        25 |
+------------+-----------+
```

基于 TPCH 100GB 中的 Q10 查询，
扩展了查询字段以及查询范围

写入目标表

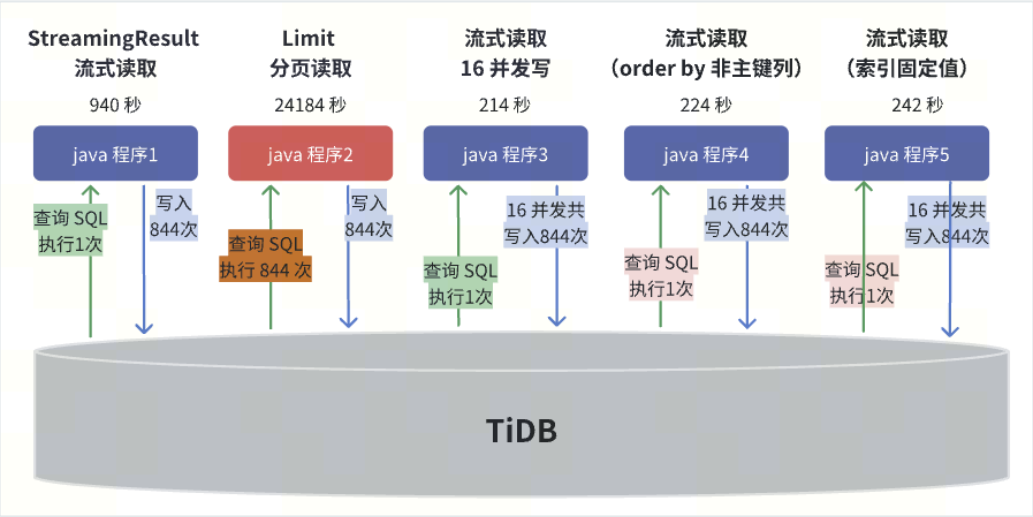
```sql
CREATE TABLE `tpch_q10` (
  `c_custkey` bigint(20) NOT NULL,
  `c_name` varchar(25) DEFAULT NULL,
  `revenue` decimal(15,4) DEFAULT NULL,
   ...
  `etl_host` varchar(64) DEFAULT NULL,
  `etl_user` varchar(64) DEFAULT NULL,
  `etl_date` date DEFAULT NULL,
  PRIMARY KEY (`c_custkey`) /*T![clustered_index] CLUSTERED */,
  KEY `idx_orderdate` (`o_orderdate`),
  KEY `idx_phone` (`c_phone`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

29 个字段+2 个索引

批处理测试场景及结果（1）

场景	SQL/工具	耗时（秒）	TiDB 节点最大内存	TiDB 节点最大 CPU	说明
查询+写入一体化处理	INSERT INTO ... SELECT ...	586	23GB	1C	一条 SQL 完成查询+写入
	Java 1 读 + 1 写（使用 streaming Resultset）	940	15GB	1C	-
	Java 1 读 + 1 写（使用 limit 分批）	24184	25GB	1C	错误示例：每次 limit 10000 行
	Java 1 读 + 16 写	214	15GB	7C	16 个 queue
	Java 1 读 + 16 写（模拟写放大）	224			查询时 order by 非主键字段
	Java 1 读 + 16 写（模拟写入热点）	242			c_phone 字段使用固定值
	datax mysqlreader+mysqlwriter	1011	15.5GB	1C	多表 query 时，无法并发
仅查询，导出 CSV	SQL：SELECT ... INTO OUTFILE	145	13.5GB	1C	导出到 NFS，单文件 3.2GB
	工具：dumpling	133	13.5GB	1C	导出到 NFS，12 个文件
仅写入，导入 CSV	工具：tidb-lightning 单个文件	184	-	-	消耗的是应用节点资源（其中 analyze 18s）
	工具：tidb-lightning 多个文件	181	-	-	
	SQL：LOAD DATA 单个文件	509	11.6GB	1C	单个 3.2GB CSV
	SQL：LOAD DATA 多个文件+8并发	183	-	3C	拆分成 2 万行一个，手动并行
	SQL：IMPORT INTO 单个文件	99	-	7C	具备自动切分文件功能
	SQL：IMPORT INTO 单个文件 16T	88	-	15C	thread=16
	SQL：IMPORT INTO 多个文件	97	-	7C	可导入dumpling生成的多个文件
	datax：txtfilereader 单个文件	1011	-	1C	单个 csv，1 channel
	datax：txtfilereader 多个文件	231	-	6C	12 个 csv，8 channel 并行

JAVA 批处理结果分析



StreamingResult 流式读取还有哪些用途?

- 使用 JAVA 处理时，**StreamingResult 流式读取**+多并发写入能够获得非常好的性能。**强烈不建议使用 limit 分页这种形式拆批**，这种逻辑数据库将执行 844 条查询 SQL，效率极低，消耗的资源极高。
- 在程序 4 中，将原本查询 SQL 里的 order by c_custkey 换成了 order by revenue desc 后，对性能也有一定影响，原因主要是多线程写入时 RPC 开销严重放大。
- 在程序 5 中，将原本查询 SQL 中的 c_phone 换成 '132-0399-0111' as c_phone，模拟索引热点。

使用 ETL + 调度平台实现批处理结果分析

不同处理方式对比

1. **调度类型：datax(mysqlreader + mysqlwriter)，简单，效率一般**
   - Dolphin 调度 datax 作业：使用 mysqlreader 方式读取时，默认就使用流式读取，但是对于多表查询的 query 时，写入阶段无法并发
2. **调度类型：shell + datax(txtfileread + mysqlwriter)，十分复杂，效率较高**
   - Dolphin 调度 shell：使用 dumpling 导出成多个 csv 文件
   - 再调度 datax 作业：使用 txtfilereader + mysqlwriter，此时可以多线程并发写入，效率较高
3. **调度类型：SQL，简单高效**
   - Dolphin 调度 SQL：SELECT … INTO OUTFILE
   - Dolphin 调度 SQL：IMPORT INTO …

SELECT … INTO OUTFILE 还有哪些用途?

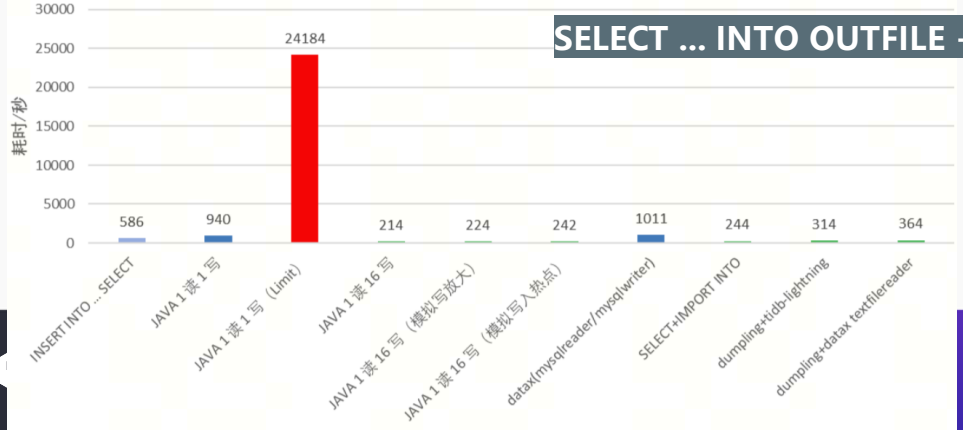
IMPORT INTO … 使用示例

```
IMPORT INTO test.tpch_q10 FROM '/mnt/nfs/test.tpch_q10.csv' with FIELDS_TERMINATED_BY='\t',split_file,thread=8;
```

注意：IMPORT INTO 导入过程中，不会产生日志，所以针对需要 CDC 同步或 Kafka 分发的场景，该方案不适用。

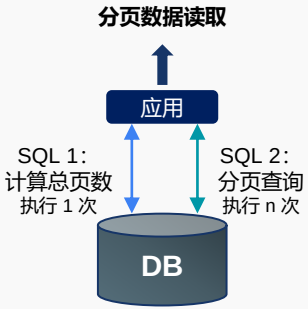
批处理测试场景及总结

批处理方案	性能	内存消耗	使用复杂度
INSERT INTO ... SELECT	大数据量：性能一般	高	**最简单**（注意当需要使用 TiFlash 时，请将 sql_mode 指定为非严格模式）
JAVA 程序方式	高	中	需要一定编码工作、以及规避热点
datax 方式	性能一般	中	比较简单，一般需要配合调度平台使用
导出：SELECT ... INTO OUTFILE	高	低	**简单**，SQL 可编码到程序 需挂载 NFS/对象存储到 TiDB 文件系统
导出：dumpling	高	低	复杂，难以编码到程序中，参数比较多**（不建议）**
导入：LOAD DATA	高	取决于单个 CSV 大小	应用需要对 CSV 大小进行切分，同时应用需要多线程写入，以及规避热点问题
导入：IMPORT INTO	高	极低	**简单**，SQL 可编码到程序，不消耗应用节点 CPU 需挂载 NFS，或通过 s3 api 使用对象存储
导入：tidb-lightning	高	极低	复杂，难以编码到程序中，消耗应用节点 CPU 较高**（不建议）**

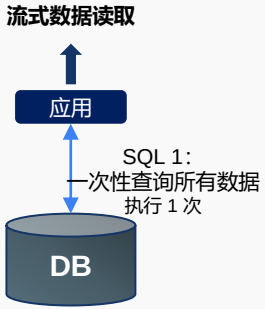


SELECT ... INTO OUTFILE + IMPORT INTO 是一种简单且高效的处理方式

数据批量导出场景优化方案

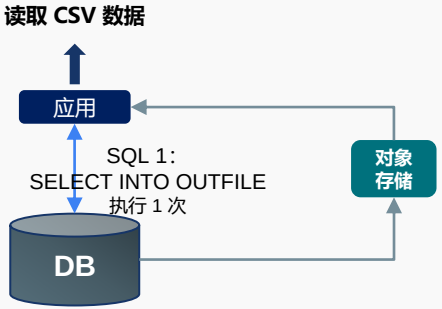


原始 Limit 分页导出逻辑



StreamingResult 流式读取逻辑

落地场景多，优化效果明显



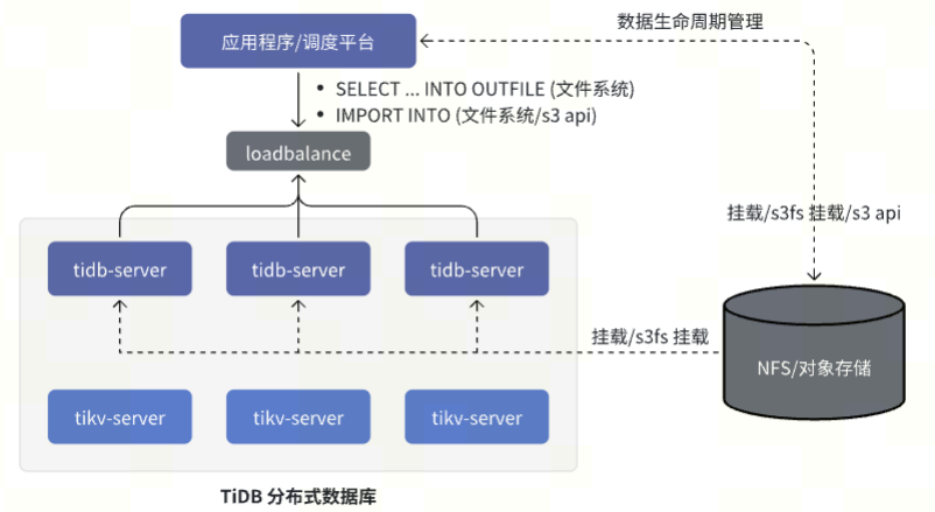
SELECT .. INTO OUTFILE 方案

数据共享场景

总结与展望

总结

TiDB 7.5 引入的 IMPORT INTO，结合 SELECT ... INTO OUTFILE、以及 NFS/对象存储，让 TiDB 上增加了一种相对简单且非常高效的批处理方案，JAVA 程序处理时更加简单，ETL 调度也更简单。以下是架构示例：



注意：

1. 由于 TiDB 是一个分布式系统，而应用往往通过 LB 连接到 TiDB，所以需要确保所有计算节点、应用节点都能访问共享的存储（推荐使用对象存储）
2. SELECT ... INTO OUTFILE 当前只支持写入到文件系统

展望：未来持续增强 IMPORT INTO 功能

当前 IMPORT INTO 功能仅支持 CSV 数据源，未来 TiDB 8.x 版本中，还支持 SELECT STATEMENT 这种数据源方式，使用实例如下：

```
# 一条 SQL 完成数据的查询与导入
IMPORT INTO table[column, column...] FROM select-statement [WITH thread=xx]
# 同时支持快照查询结果导入
IMPORT INTO table[column, column...] FROM select-statement as of timestamp ‘2024-02-20 23:59:59’ [WITH thread=xx]
```

实际测试效果：189 秒

```
MySQL [test]> truncate table test.tpch_q10;
Query OK, 0 rows affected (1.04 sec)

MySQL [test]> IMPORT INTO test.tpch_q10 FROM  select          c_custkey,         c_name,         sum(l_extendedprice * (1 - l_discount)) as revenue,         c_acctbal,
      n_name,          c_address,          c_phone,         c_comment,         min(C_MKTSEGMENT),         min(L_PARTKEY),         min(L_SUPPKEY),          min(L_LINENU
MBER),          min(L_QUANTITY),          max(L_TAX),          max(L_LINESTATUS),          min(L_SHIPDATE),          min(L_COMMITDATE),          min(L_RECEIPTDATE),
    min(L_SHIPINSTRUCT),          max(L_SHIPMODE),          max(O_ORDERSTATUS),          min(O_TOTALPRICE),          min(O_ORDERDATE),          max(O_ORDERPRIORITY),
     min(O_CLERK),          max(O_SHIPPRIORITY),          @@hostname,          current_user(),          current_date() from          tpch.customer,          tpch.orders,
    tpch.lineitem,          tpch.nation where          c_custkey = o_custkey          and l_orderkey = o_orderkey          and o_orderdate ≥ date '1993-10-01'          and o_
orderdate < date '1994-10-01'          and l_returnflag = 'R'          and c_nationkey = n_nationkey group by          c_custkey,          c_name,          c_acctbal,
c_phone,          n_name,          c_address,          c_comment order by          c_custkey  WITH thread=8;
Query OK, 8344700 rows affected (3 min 8.89 sec)
Records: 8344700, ID: 63f5f9c6-11d3-4cd2-b127-9ae884111519

MySQL [test]> select count(*) from test.tpch_q10;
+----------+
| count(*) |
+----------+
|  8344700 |
+----------+
1 row in set (1.24 sec)
```

敬请期待

TiDB

THANK YOU.