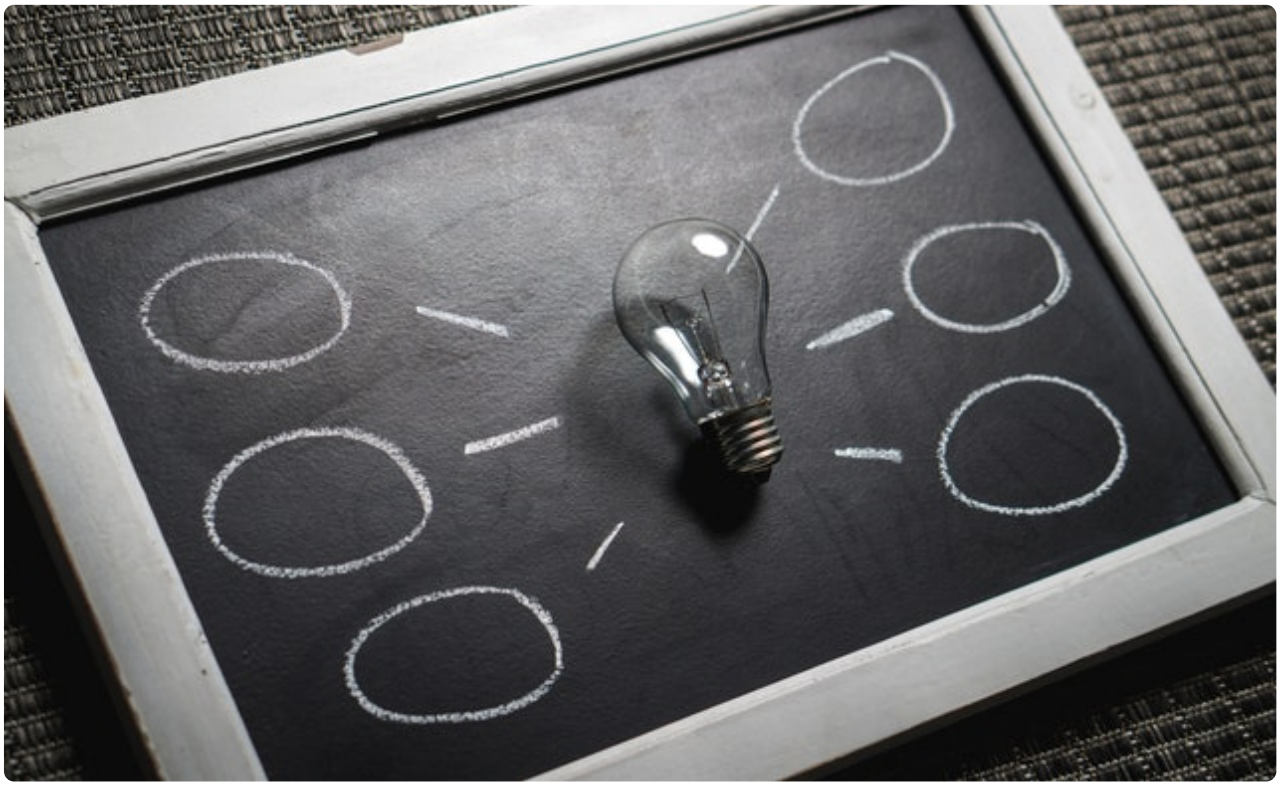
24 大数据量插入遇到瓶颈，我该怎样做性能优化呢？

更新时间：2020-05-11 09:27:46



> 如果说我比别人看得要远一点，那是因为我站在巨人的肩上。——牛顿

数据迁移、数据恢复往往都需要做大数据量的插入操作，但是，不同的插入方法对性能的影响也是非常大的。这一节里，我将会分析数据插入耗时的因素，以及常见的数据插入方法。通过比对不同插入方法的性能以及它们各自的优缺点，才能确定哪一种才是你所需要的。

1. 插入数据分析

我们在客户端执行一条数据插入命令就可以实现 MySQL 服务器的插入操作，但是，这背后其实是做了很多工作的。这里，我们先来看一看插入数据影响耗时的因素有哪些。之后，再去介绍下 MySQL 支持的数据插入方法。

1.1 插入数据的过程

在 MySQL 中，能够插入数据是很多个组件（或流程）共同配合的成果，且每个组件所耗时的比例也是不相同的。我们先来看一看插入数据都会涉及哪些过程：

- 服务器与客户端建立连接，耗时占比最多，大约 30%
- 发送插入数据到服务器，需要通过网络连接，耗时大约 20%
- 查询分析，包括对 SQL 语法、数据、权限等校验，耗时大约 20%
- 插入记录到数据表中，耗时大约 10%
- 更新索引，耗时大约 10%
- 关闭服务器与客户端的连接，耗时大约 10%

但是，需要注意，这里所说的是插入一条数据记录，对于大数据量的插入操作来说，插入记录和更新索引肯定是最为耗时的地方。

1.2 插入数据有哪些常用方法

MySQL 提供了多种方式用于插入数据，下面，我来总结一些常见的方法：

- **顺序 INSERT 插入数据**：一次执行一条数据记录的插入
- **批量 INSERT 插入数据**：一次执行多条数据记录的插入
- **LOAD DATA INFILE 插入数据**：从文本文件中执行数据记录的插入

接下来，我们就来看一看这几种方法怎样去应用，以及对它们插入耗时的分析。以此，来确定哪一种方法更好，哪一种方法更适合你的场景。

2. 顺序 INSERT 插入数据

为了更好的分析插入性能，我这里以 Java 语言去编写并执行插入过程，对于其他语言也都是类似的。这里，我以 worker 表来做演示，首先看一看它的结构：

```
mysql> DESC worker;
+---------+---------------------+------+-----+---------+----------------+
| Field   | Type                | Null | Key | Default | Extra          |
+---------+---------------------+------+-----+---------+----------------+
| id      | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| type    | char(64)            | NO   | MUL |         |                |
| name    | char(64)            | NO   |     | NULL    |                |
| salary  | bigint(20) unsigned | YES  |     | NULL    |                |
| version | bigint(20)          | NO   |     | 0       |                |
+---------+---------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

由于插入数据量较大，所以，可以忽略字段取值的复杂度。同时，不仅仅对于“顺序 INSERT”，其他的几种数据插入方法也一样会使用 worker 表来做对比测试。

2.1 插入过程详解

想要获取更加有代表性的数据，就需要插入更多的数据量，那么，直接在 MySQL 客户端中执行 INSERT 语句是不合适的。我这里直接以测试用例（SpringBoot、JUnit）的形式完成代码，当然，你也可以写成任意的形式。首先，我们需要去定义一些常量：

```
// MySQL 连接信息、用户名、密码定义
private static final String URL = "jdbc:mysql://localhost:3306/imooc_mysql";
private static final String USERNAME = "root";
private static final String PASSWORD = "root";

// 插入语句格式化定义
private static final String INSERT_SQL = "INSERT INTO `worker`(type, name, salary, version) " +
        "VALUES ('%s', '%s', %d, %d)";

// worker 表各字段取值定义
private static final List<String> TYPES = Arrays.asList("A", "B", "C", "D", "E", "F");
private static final List<String> NAMES = Arrays.asList("G", "H", "I", "J", "K", "L");
private static final List<Integer> SALARYS = Arrays.asList(1000, 2000, 3000, 4000, 5000, 6000);
private static final List<Integer> VERSIONS = Arrays.asList(0, 1, 2, 3, 4, 5, 6);

// 随机数对象定义
private static Random random = new Random();
```

接下来，编写插入数据的逻辑。很显然，对于顺序 INSERT 操作，我们使用一个 for 循环就可以完成。代码及其注释如下所示：

```
/**
 * <h2>顺序插入数据逻辑</h2>
 * @param count 插入条数
 * */
private void insertToMySQL(int count) {

    try {
        // JDBC 与 MySQL 建立连接
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        Statement statement = conn.createStatement();

        // 计时器定义, 并开始计时
        StopWatch stopWatch = new StopWatch();
        stopWatch.start();

        // for 循环构造 INSERT 语句, 并执行语句, 执行的次数由参数 count 控制
        for (int i = 0; i < count; i++) {
            String sql = String.format(
                    INSERT_SQL,
                    TYPES.get(random.nextInt(TYPES.size())),
                    NAMES.get(random.nextInt(NAMES.size())),
                    SALARYS.get(random.nextInt(SALARYS.size())),
                    VERSIONS.get(random.nextInt(VERSIONS.size()))
            );

            statement.executeUpdate(sql);
        }

        statement.close();

        // 计时器停止计时
        stopWatch.stop();

        // 打印执行耗时日志
        log.info("time elapsed for {} inserts: {}s", count, stopWatch.getTotalTimeSeconds());

        conn.close();
    } catch(SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

很明显，我们只需要调用 insertToMySQL 方法，并传递想要插入的 count 参数就可以了。下面，我们依次执行 100、1000、10000 条（这些数字不具有特殊含义，当然，你可以任意指定。但是，为了方便对比，接下来的插入方法也要插入相同体量的数据）INSERT 操作。代码如下：

```
@Test
public void test100Insert() {

    insertToMySQL(100);
}

@Test
public void test1000Insert() {

    insertToMySQL(1000);
}

@Test
public void test10000Insert() {

    insertToMySQL(10000);
}
```

2.2 插入耗时分析

执行上述的插入方法，我们可以得到每一次插入数据的打印日志，也就得到了每一次的操作耗时，总结如下表所示：

插入条数	耗时
100	1.195s
1000	12.091s
10000	124.163s

可以看到，顺序 INSERT 的耗时是线性增长的，随着插入数据量的增大，操作过程是非常缓慢的。这种插入方式无疑是比较慢的（对于软件工程来说，线性和指数增长都是低效的），它所耗时的地方在于：

- 每一次 INSERT 都需要一次网络 IO
- 每一次 INSERT 都需要一次 SQL 语句解析
- 每一次 INSERT 都需要一次数据写入（先写入缓冲区，再刷写到磁盘）
- 每一次 INSERT 都需要一次索引的更新
- 每一次 INSERT 都可能需要多次日志记录过程（查询日志、Binlog 日志等等）
- 每一次 INSERT 都需要一次事务的创建与提交

综上所述，正是由于每一次 INSERT 都需要“经历”很多个阶段，所以才导致了大数据量插入性能很低。在实际的企业级开发中，涉及到大数据量插入问题时，选择这种方式是极不明智的做法。

3. 批量 INSERT 插入数据

INSERT 语句除了可以一次插入一条数据之外，还可以一次插入多条数据，且它们的语法也是相似的。例如，通过 INSERT 语句一次性往 worker 表中插入三条数据，可以执行命令：

```
INSERT INTO `worker` (`type`, `name`, `salary`, `version`)
VALUES ('B', 'H', 2000, 1), ('C', 'I', 3000, 0), ('C', 'L', 6000, 0);
```

但是，如果想要通过 INSERT 一次性插入大批量的数据，就需要去考虑 MySQL 中的 max_allowed_packet 参数。这个参数会限制 MySQL 服务器接受的数据包大小，如果超过这个值时会导致大批量数据写入或更新失败。我们可以把它设置为一个比较大的值，例如：

```
-- 设置服务器最大接受的数据包是 200M，且生效范围是 GLOBAL
mysql> SET GLOBAL max_allowed_packet = 2 * 100 * 1024 * 1024;
Query OK, 0 rows affected (0.00 sec)

-- 检验变量的值是否符合预期（需要退出当前 session，重新登录）
mysql> SHOW VARIABLES LIKE 'max_allowed_packet';
+--------------------+-----------+
| Variable_name      | Value     |
+--------------------+-----------+
| max_allowed_packet | 209715200 |
+--------------------+-----------+
1 row in set (0.00 sec)
```

3.1 插入过程详解

由于基本的常量（MySQL Url、用户名、密码等等）都已经定义了，所以，我们只需要去完成插入逻辑就可以了。其实，聪明的你一定可以想到，插入逻辑的核心一定是构造批量 INSERT 语句。代码和注释如下：

```
// 批量插入语句前缀
private static final String BATCH_INSERT_SQL_PREFIX = "INSERT INTO `worker`(type, name, salary, version) " +
        "VALUES";
// 批量插入语句后缀
private static final String BATCH_INSERT_SQL_SUFFIX = " ('%s', '%s', %d, %d)";

/**
 * <h2>批量插入数据逻辑</h2>
 * @param count 插入条数
 * */
private void batchInsertToMySQL(int count) {

    try {
        // JDBC 与 MySQL 建立连接
        Class.forName("com.mysql.jdbc.Driver");
        Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        Statement statement = conn.createStatement();

        StringBuilder sb = new StringBuilder();

        // for 循环构造插入列值语句, 执行的次数由参数 count 控制
        for (int i = 0; i < count; i++) {
            sb.append(
                    String.format(
                            BATCH_INSERT_SQL_SUFFIX,
                            TYPES.get(random.nextInt(TYPES.size())),
                            NAMES.get(random.nextInt(NAMES.size())),
                            SALARYS.get(random.nextInt(SALARYS.size())),
                            VERSIONS.get(random.nextInt(VERSIONS.size()))
                    )
            );
            // 最后一个列值不需要加分号
            if (i + 1 < count) {
                sb.append(",");
            }
        }

        // 计时器定义, 并开始计时
        StopWatch stopWatch = new StopWatch();
        stopWatch.start();

        // 执行批量插入语句
        statement.executeUpdate(BATCH_INSERT_SQL_PREFIX + sb.toString());

        statement.close();

        // 计时器停止计时
        stopWatch.stop();

        // 打印执行耗时日志
        log.info("time elapsed for {} inserts: {}s", count, stopWatch.getTotalTimeSeconds());

        conn.close();
    } catch(SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

同样，类似于顺序插入，我们也只需要调用 batchInsertToMySQL 方法，并传递想要插入的 count 参数就可以了。下面，我们依次执行100、1000、10000条数据的批量 INSERT 操作。代码如下：

```
@Test
public void test100BatchInsert() {

    batchInsertToMySQL(100);
}

@Test
public void test1000BatchInsert() {

    batchInsertToMySQL(1000);
}

@Test
public void test10000BatchInsert() {

    batchInsertToMySQL(10000);
}
```

3.2 插入耗时分析

执行以上三个测试用例，我们可以惊讶的发现它们的耗时（可以从打印的日志中得到）都是非常低的。总结如下表所示：

插入条数	耗时
100	0.033s
1000	0.051s
10000	0.504s

其实，只要对照顺序插入而言，批量插入还是非常好理解的。这种方式效率高的原因主要是合并了日志、写入次数、网络往返 IO 以及事务，它将多次的执行过程变成了一次执行过程。主要的耗时是客户端到服务器的大数据量传输以及数据写入（表数据以及表索引）。正是由于批量执行的高效性，它也成为企业级开发中普遍采用的方式。

4. LOAD DATA INFILE 插入数据

MySQL 官方对 `LOAD DATA INFILE` 的描述是：用于高速的从一个文本文件中读取行，并把它写入数据表中。如果我们不想编写代码，且恰好手边有这样的一份文本文件，可以尝试考虑这种方式去完成大批量的数据插入。接下来，我们去看一看它的使用方法以及相关分析。

4.1 插入过程详解

由于它是 MySQL 提供的一种功能，我们先来看一看它的语法：

```
LOAD DATA
  [LOW_PRIORITY | CONCURRENT] [LOCAL]
  INFILE 'file_name'
  [REPLACE | IGNORE]
  INTO TABLE tbl_name
  [PARTITION (partition_name [, partition_name] ...)]
  [CHARACTER SET charset_name]
  [{FIELDS | COLUMNS}
    [TERMINATED BY 'string']
    [[OPTIONALLY] ENCLOSED BY 'char']
    [ESCAPED BY 'char']
  ]
  [LINES
    [STARTING BY 'string']
    [TERMINATED BY 'string']
  ]
  [IGNORE number {LINES | ROWS}]
  [(col_name_or_user_var
    [, col_name_or_user_var] ...)]
  [SET col_name={expr | DEFAULT},
    [, col_name={expr | DEFAULT}] ...]
```

确实，想要把它的每一个参数或选项搞清楚是件不容易的事。不过，类似于其他工具（就像是惯例一样），重要的“部分”也并不多。下面，我将讲解 `LOAD DATA INFILE` 命令最常用的选项或参数的含义：

- **LOW_PRIORITY | CONCURRENT**：这两个关键字控制写表过程中存在读表的行为
  - **LOW_PRIORITY**：在写入过程中，如果有客户端读表，写入将会被延后，直至没有任何客户端读表再继续写入。但是，这个选项仅适用于表锁存储引擎，例如 MyISAM、MEMORY、MERGE
  - **CONCURRENT**：允许在写入过程中其他客户端读取表内容
- **LOCAL**：标识从客户端所在主机读取文件，否则，文件必须位于 MySQL 服务器上
- **REPLACE** | IGNORE：控制对现有唯一键记录重复处理的方式
  - **REPLACE**：新纪录替换重复的原唯一键记录
  - **IGNORE**：跳过唯一键重复的行
  - 如果不指定任何一个选项，遇到重复唯一键时，报错并停止数据插入
- **FIELDS 子句**：指定文件记录列值的分割格式，如果使用这个子句，则至少要提供以下选项中的一个
  - **TERMINATED BY**：分隔符，行记录中每个列值的分隔符
  - **ENCLOSED BY**：控制哪些字段应该包裹在引号里面
  - **ESCAPED BY**：转义字符
- **LINES 子句**：指定文件记录行的分割格式，默认是换行符 '\n'
  - **STARTING BY**：所有的行都包含相同的前缀，使用这个选项可以忽略前缀。特别地，如果某行不包含前缀，则整行都会被跳过
  - **TERMINATED BY**：行分隔符
- **IGNORE number**：忽略文件开始的 number 行，例如表头
- **col_name_or_user_var**：如果想导入表的某些列，可以显示的指定列名

可以知道，`LOAD DATA INFILE` 数据插入方式最核心的工作是构造文本文件。我这里同样使用 Java 语言去完成这件事，代码如下：

```
// 数据目录及文件名格式定义（目录 /tmp/load_data_file 需要存在）
private static final String LOAD_DATA_FILE = "/tmp/load_data_file/%d_worker.txt";

/**
 * <h2>构造 LOAD DATA INFILE 所需的数据文件</h2>
 * */
@Test
public void buildLoadDataFile() throws IOException {

    // 定义需要写入的记录数
    List<Integer> workerCount = Arrays.asList(100, 1000, 10000);

    for (Integer integer : workerCount) {

        FileWriter fw = new FileWriter(String.format(LOAD_DATA_FILE, integer), true);

        for (int j = 0; j < integer; j++) {

            BufferedWriter bw = new BufferedWriter(fw);
            // 构造随机插入的数据，列值之间以逗号分隔
            String line = String.format("%s,%s,%s,%s",
                    TYPES.get(random.nextInt(TYPES.size())),
                    NAMES.get(random.nextInt(NAMES.size())),
                    SALARYS.get(random.nextInt(SALARYS.size())),
                    VERSIONS.get(random.nextInt(VERSIONS.size())));

            bw.write(line);
            bw.newLine();
            bw.flush();
        }

        fw.close();
    }
}
```

文本文件构造完成之后，先别着急去执行 `LOAD DATA INFILE` 命令，因为你大概率会遇到 “ERROR 1290 (HY000): The MySQL server is running with the --secure-file-priv option so it cannot execute this statement” 错误（可以去尝试执行下看看）。这其实是受到 secure_file_priv 参数的影响，它用于限制数据表导入导出。其取值和含义如下：

- **NULL**：默认值，限制 MySQL 表数据不允许导入导出
- **/tmp**：限制 MySQL 表数据只能在 /tmp 目录中执行导入导出，其他目录不能执行
- **空值**：没有限制

另外，secure_file_priv 是一个只读参数，我们只能通过修改 MySQL 的配置文件，并重启 MySQL 服务来解决问题。打开 my.cnf，并加入如下配置：

```
secure_file_priv = ''
```

最后，就可以通过 MySQL 客户端完成数据导入工作，如下所示：

```
mysql> LOAD DATA INFILE '/tmp/load_data_file/100_worker.txt' INTO TABLE `imooc_mysql`.`worker`
    -> FIELDS TERMINATED BY ','
    -> LINES TERMINATED BY '\n'
    -> (type, name, salary, version);
Query OK, 100 rows affected (0.01 sec)
Records: 100  Deleted: 0  Skipped: 0  Warnings: 0
```

同样的方式，我们也可以完成1000和10000条数据的插入工作（命令的选项之前都有说明，这里不再赘述）。

4.2 插入耗时分析

通过执行三个数据导入（插入）命令，我们可以收集客户端中打印的执行时间，总结如下表所示：

插入条数	耗时
100	0.01s
1000	0.06s
10000	0.28s

经过对比可以发现，`LOAD DATA INFILE` 与批量 INSERT 的耗时是差不多的。这其实也很好理解，它们的原理都是类似的：一次性将数据发给服务器，经过一次校验、写日志、写数据、更新索引的过程即可。同样，也正是由于这种方式的插入效率很高，自然也就成为企业级开发中的可选方案。

5. 总结

通过对比三种大批量数据插入的方式，我们能够确定每一种方式的优劣性，当然也就能够在需要的时候做出合适的选择。对于大数据量插入需求来说，顺序插入不仅会影响客户端的性能，同样会给服务器带来很大的压力；批量插入和 `LOAD DATA INFILE` 都具有很高的性能，使用它们其中的任何一种都是可行的，只是需要考虑工作重心的问题。最后，还要好好把握每一种数据插入方式性能损耗的地方，也就是即使很慢，也要知道慢的原因。

6. 问题

你做过大批量数据插入吗？是使用哪一种方式完成的呢？

对于批量 INSERT 过程，可以考虑使用多线程去完成吗？你觉得性能会有提升吗？

你能总结三种数据插入方式耗时的地方吗？

如果你遇到了大数据量插入的需求，你会选择使用什么方法呢？为什么？

7. 参考资料

《高性能 MySQL（第三版）》

MySQL 官方文档：INSERT Statement

MySQL 官方文档：LOAD DATA Statement

MySQL 官方文档：Optimizing INSERT Statements

MySQL 官方文档：Concurrent Inserts

MySQL 官方文档：The InnoDB Storage Engine

}

23 关于 SQL 查询语句，有什么好的建议吗？

25 加速 order by 查询，可以从哪些方面做优化呢？