文章编号: 1000-5641(2019)05-0190-13

Woodpecker+: 基于数据特征的自定义负载性能评测

张 涛, 张小磊, 李宇明, 张春熙, 张 蓉

(华东师范大学 数据科学与工程学院, 上海 200062)

摘要: 数据库的性能评测随着复杂多样的应用出现变得更加重要. 在很多情况下, 研究、开发人员的性能评估工作受限于负载的缺乏. 虽然 OLTP-Bench 通用数据库性能测试框架在一定程度上提高了性能测试的效率, 但由于标准 Benchmark 负载固定且针对用户多样的应用场景的代表性差, 故无法精准地满足不同应用的系统性能; 此外, 大部分测试框架使用高级编程语言支持为应用编写测试负载, 不仅增加测试负担还会引入大量重复性工作, 导致测试效率低下. 本文设计并实现了一个用户自定义性能测试负载的工具: Woodpecker+. 该工具的主要贡献为: 易于使用和可拓展、提供了可高效构造测试案例、做测试安排的测试定义语言(TDL)、可灵活控制事务执行模式及数据访问分布、实现了轻量级的细粒度统计信息收集与分析、支持多种主流数据库系统 DBMS 及提供数据库访问接口的其他数据库. 通过一组详细的运行在主流 DBMS 上的自定义负载实验来验证 Woodpecker+ 的特性.

关键词: 数据库; 性能测试; 自定义负载

中图分类号: TP311.56 **文献标志码**: A **DOI:** 10.3969/j.issn.1000-5641.2019.05.016

Woodpecker+: Customized workload performance evaluation based on data characteristics

ZHANG Tao, ZHANG Xiao-lei, LI Yu-ming, ZHANG Chun-xi, ZHANG Rong

(*School of Data Science and Engineering*, *East China Normal Univeristy*, *Shanghai* 200062, *China*)

Abstract: There are a number of performance testing tools, like Sysbench and OLTP-Bench, that can be used to benchmark the testing of database performance. However, because the standard benchmark workload is fixed and application scenarios for users are not always representative, it is impossible to accurately determine system performance. Moreover, if users are required to use a high-level programming language to implement a test workload separately for each application, this will undoubtedly introduce a substantial amount of repetitive work, resulting in inefficient testing. To address these issues, this paper designs and implements a user-defined performance test workload tool. The main benefits of this tool can be summarized as follows: It is easy to use and expandable;

收稿日期: 2019-07-28

基金项目: 国家重点研发计划(2018YFB1003404); 国家自然科学基金(61432006)

第一作者: 张 涛, 男, 硕士研究生. 研究方向为数据库基准测试. E-mail: zhangtao1906@163.com.

通信作者: 张 蓉, 女, 教授, 研究方向为分布式数据管理. E-mail: rzhang@dase.ecnu.edu.cn.

it provides a test definition language (TDL) for efficient construction of test cases; and it offers flexible control for mixed execution of transactions, data access distribution, lightweight and granular statistical information collection and analysis, and support for multiple mainstream DBMSs and other databases that provide database access interfaces. We highlight the tool's features through a detailed set of customized workload experiments running on the mainstream DBMS.

Keywords: database;　performance test;　customized workload

0　引　　言

自 1970 年提出关系数据模型至今, 数据库的发展依次经历了传统的关系型数据库系统, 具有高可用和良好拓展性的 NoSQL 系统(如HBase[1])以及同时兼顾可拓展性和事务支持的 NewSQL(如 VoltDB[2])新型数据库系统 3 个阶段. 目前针对新硬件(如NVM[3])下的数据库优化技术也在如火如荼地发展中. 随着云端应用的急速发展, 为了满足不同的目的数据库系统爆炸式增长, 由此有效的数据库测试是提高数据库开发效率的重要手段.

按照测试目的分类, 数据库测试可主要分为功能测试、性能测试和数据库系统测试三大类. 功能测试即验证数据库所提供的功能接口是否正确即是否符合设计文档, 如验证基本 SQL 操作, 二级索引和多行事务操作等; 性能测试即指数据库在一定硬件配置和系统配置下运行一类负载的性能, 常见的性能指标如 TPS、QPS 和百分比延迟等; 系统测试则是验证数据库系统对外提供的系统服务接口, 如分布式数据库中对集群切换和节点上下线控制等.

大多数测试工具对功能测试支持地较好, 但在性能测试方面, 它们在功能支持、负载模式、可拓展性上仍存在较大限制. 如 MySQL Test Framework[4]测试框架只支持基本的 SQL 功能测试, 不支持性能测试; 多线程性能测试工具 Sysbench[5]使用 Lua 脚本语言编写测试负载, 但仅可对主键列的访问分布进行简单的控制, 缺乏多样性与灵活性; 开源的通用性能测试框架 OLTP-Bench[6]实现了大多数负载模式固定的标准 Benchmark, 但不支持自定义性能测试负载, 故无法准确反应特定应用下的性能; Woodpecker[7]通用测试框架虽然提供了灵活的测试定义语言描述测试任务, 在性能测试模块中实现了增删改查五类负载, 但负载与逻辑代码紧密耦合, 可拓展性上存在不足.

由此可见, 现有框架或工具要么不支持自定义性能测试负载, 要么在支持程度上较弱, 且均未考虑对性能测试结果影响重大的因素——数据特征. 主要表现为:

(1) 非主键列上非重复值的个数, 当数据表有非常大的规模, 且在某些非主键列上建有二索引, 当在那些列上进行查询时, 非重复值的个数直接影响查询结果集的大小以及遍历结果集的时间, 从而影响二级索引的效率;

(2) 字符串的平均长度和最大长度, 不同的字符串长度导致不同的查询结果集大小, 它将影响磁盘 I/O、网络传输时间等, 这都会影响性能测试的结果;

(3) 不同的访问分布, 高 Skew 的访问分布相对于均匀的访问分布缓存效果会更好, 因为 Skew 分布涉及的数据量相对于均匀分布集中, 所以内存缓存效果好. 另外, 对于具有高冲突的负载来说, 高 Skew 增加了并发控制的代价, 故高 Skew 的性能将低于平均访问分布.

这篇文章中呈现的工作就是去试图解决上述问题, 并集成到 Woodpecker 框架下. 本文提供了一个基于数据特征的自定义负载性能评测工具——Woodpecker+, 主要贡献点为:

(1) 设计并实现了一套描述数据特征的关键字;

(2) 支持表、事务和操作中对数据特征定义;
(3) 支持灵活的性能测试负载定义和组织;
(4) 保证负载的生成具有良好的可拓展性.

1 相关工作

目前开源的数据库系统测试工具较少, 流行的测试工具主要有以几种: MySQL Test Framework、Sysbench、OLTP-Bench, 小众的测试框架如 Woodpecker.

MySQL Test Framework[4]是 MySQL 数据库开源的一套数据库测试框架. 主要针对 MySQL 系统进行语法和功能适配, 它有着强大的 SQL 功能测试、简洁高效的测试语法和测试成本低等优点, 但该工具不支持性能测试, MySQL 的性能测试需借助第三方测试工具. 目前官方维护了 1 200 多个涉及不同功能模块的可回归的测试案例.

Sysbench[5]是一个开源的、模块化的、可跨平台的多线程性能测试工具, 不仅支持 CPU、内存、磁盘 I/O、操作系统线程开销的测试, 还支持数据库的性能测试. 针对性能测试, 测试人员使用 Lua 脚本语言编写测试逻辑, 可指定多组测试线程数、测试时长、统计信息间隔等参数. 该工具在主键列上创建了若干个分布函数, 也即自定义负载中只支持主键上访问分布的自定义, 数据特征定义缺乏灵活性.

OLTP-Bench[6]是卡耐基梅隆大学开源的性能测试框架, 该框架融合并实现了 19 个 OLTP 类型的 Benchmark(TPC-C[8]、YCSB[9]、SmallBank[10]等), 针对不同的 DBMS, 该工具将 SQL 和数据库进行适配, 使数据库之间的语法差异透明化. 该工具支持测试过程中动态的负载变化和细粒度的统计信息收集, 但该工具不支持自定义性能测试负载.

对于小众的测试框架——Woodpecker[7], 是由华东师范大学数据学院自主研发的一套通用的数据库系统测试框架, 有着良好的测试 case 复用性和支持高压环境下的功能测试等优点. 但其性能测试仅支持增、删、改、查等固定负载且与逻辑代码耦合, 可拓展性较弱.

2 Woodpecker 测试框架[7]

Woodpecker 测试框架是第一个同时支持数据库功能测试、性能测试和系统功能测试的通用数据库测试框架, 相比于功能单一的测试工具, 该框架测试覆盖面更广; 此外, 它还支持简单快速的自动化回归测试, 减少测试人员的干预; 以及能够自动地收集测试结果并生成综合报告, 对于分析测试案例的执行过程和发现系统瓶颈提供有效信息.

Woodpecker 设计并实现了一套基于关键字的语义丰富、易用、可高效编写测试任务的测试定义语言(TDL). 考虑到当前的许多测试工具使用高级编程语言来描述测试任务, 这其中存在大量重复性工作, 以及不同的数据库系统在所支持的 SQL 语法上存在差异, 测试案例在不同的数据库上运行需要进行额外的适配工作, 这都将影响测试效率. 所以关键字旨在降低测试人员构造测试案例的成本, 相比于测试任务与具体编程语言绑定的方式, 关键字拥有更好的透明性, 测试人员无需关注不同数据库系统之间的差异, 工具本身支持测试案例与多种数据库系统的适配, 使得测试案例做到“一次编写, 到处运行”.

本文的性能测试描述关键字基于 Woodpecker 的测试定义语言并进行了大量拓展.

3 性能测试关键字定义

Woodpecker+ 的性能测试关键字是要抽象出影响数据库性能的数据特征, 表 1 中的关键字支持细粒度控制数据特征的数据生成与导入.

表 1　数据生成与导入关键字

Tab. 1　Data generation and importing of keywords

关键字	说明
TABLE[*tbl_name*; *tbl_size*; col_name, col_type; ⋯; *PK*(*col_name*, ⋯, *auto_increment*); *FK*(*col_name*, *ref_tbl_name*, *ref_col_name*); ⋯; *INDEX*(*index_name*, *col_name*, ⋯); ⋯]	定义表模式, PK为主键(支持复合主键), FK为外键, INDEX为表中的索引, 其中外键和索引可存在多个
Column[col_name, null_ratio, cardinality, range1, range2];	指定某属性的数据特征
IMPORT_TBL[*tbl_name*; ⋯]	创建表并按照数据特征导入数据
CLEAR_TBL[*tbl_name*; ⋯]	删除已经创建的表和导入的数据

Table 用于详细描述表模式. *Column* 用来指定某属性的数据特征, 数据特征包括: 空值比例、非重复值个数(基数)、最大值和最小值(对于数值型)、平均长度和最大长度(对于字符串型). *IMPORT_TBL* 负责完成表的创建和数据生成, 可一次操作多张表. *CLEAR_TBL*v用来删除已创建的表和清空表数据.

表 2 展示了事务描述关键字, 这些关键字可以很好的满足测试人员对不同性能测试场景的描述需求. 在给出示例之前, 将讨论关键字中重要的参数——操作的数据访问分布(distribution_type).

表 2　负载操作关键字

Tab. 2　Load operation keywords

关键字	说明
TXN[*ratio*; *txnName*] transaction body; END_TXN	定义事务, ratio 为该类事务在负载中所占的比例, txnName 为事务名
TXN_LOADING[*thread_number*; *thread_run_times*; *load_machine_number*; *sync_or_async*]	加载事务, 参数分别为总数据库链接数、每个链接的执行次数、负载机个数和执行方式(同步或异步).
MULTIPLE[*min*, *max*] operations; **END_MULTIPLE**	模拟真实业务 while 逻辑, 执行次数为 min 和 max 之间的随机值
BRANCH[*ratio*; ⋯] operations; **END_ BRANCH**	模拟真实业务分支逻辑, ratio 为每个分支的执行比例
BRANCH_DELIMITER	BRANCH 中分支间的分隔符
INSERT/REPLACE[*tbl_name*; *is_prepared*; *distribution_type*]	插入(替换)操作, 参数分别为表名、是否预编译执行、操作访问分布
SELECT[*tbl_name*, ⋯; *is_prepared*; *distribution_type*, *select_expression*, ⋯; *FILTER*; *APPEND (expression)*]	选择操作, FILTER 为 where 子句, APPEND 为 having、order by、group by 等子句
DELETE[*tbl_name*, ⋯; *is_prepared*; *distribution_type*; *FILTER*]	删除操作
UPDATE[*tbl_name*, ⋯; *is_prepared*; *distribution_type*; *col_name operator updated_value*, ⋯; *FILTER*]	更新操作, 参数示例: c1 += 7 => c1 = c1 + 7, operator 可为 =、+=、++ 等
SELECT_FOR_UPDATE[*tbl_name*, ⋯; *is_prepared*; *distribution_type*; *select_expression*, ⋯; *FILTER*; *APPEND(statement)*]	查询操作, 除了会对涉及的数据加锁, 功能基本与 SELECT 操作相同
FILTER(*col_name operator*, *&/*, ⋯)	where 子句后的过滤条件, 参数示例为: (c1 =, &, c2 =) 表示同时满足 c1 和 c2 两个等值条件, 等于的值由指定操作分布生成, operator 可为 =,>,<,!= 等
APPEND(*statement*)	group by、order by、limit、having 条件

针对 OLTP 应用, 操作的不同访问分布对性能结果有很大影响. Woodpecker+ 的关键字支持多种访问分布来模拟事务的读写比例分布和冲突强度. 目前工具支持 4 种访问分布类型: unique(min, max), uniform(min, max), normal(min, max, sigma), zipfian (min, max, size, skew). 4 种分布中的 min 和 max 确定生成数据的范围(阈值).

(1) unique 为唯一值分布, 它顺序生成 min 和 max 间的所有值.

(2) uniform 为随机分布, 它在 min 和 max 间随机产生数据.

(3) normal 为正态分布(高斯分布), $X\sim\mathrm{N}(\mu,\ \sigma^2)$ 中的参数 μ (均数)默认为 0, σ(标准差)需要由测试人员指定.

(4) zipfian 分布为齐夫分布, size 为给定的元素数量, skew 为指数, 该值越大说明访问分布越倾斜, 实际使用中 size 和 skew 一般取较小的值(如 size = 10, skew = 2).

表 3 展示了如何利用给定的关键字定义表数据特征的案例. Student 表为学生基本信息, 该表有 100 条数据记录、4 个属性列、其中主键为 STU_ID. 为了简化, score 为某门课程的全体学生成绩, 主键为 STU_ID. COLUMN 为指定列的数据特征, 对于 COLUMN 的前 4 个参数, 分别为表名、属性名、空值比例和基数(非重复值个数). 最后两个参数为属性范围, 对于字符型属性, 分别为最短长度和最大长度; 对于数值型属性, 分别为最小值和最大值.

表 3 自定义表 Schma 导入数据案例

Tab. 3 Customized table schema example

行号	操 作
1	**TABLE**[student; 100; STU_ D int, STU_ NAME varchar(20), STU_ AGE int, STU_ DEPT int; PK(STU_ ID)];
2	**COLUMN**[student; STU NAME; 0.0; 100; 4; 16];
3	**COLUMN**[student; STU_ AGE; 0.0; 20; 18; 38];
4	**COLUMN**[student; STU_ DEPT;0.0; 10; 1; 10];
5	**TABLE**[score; 2000; STU_ D int, SC_ AVG Decimal(4,2), SC_ GRADE char; PK(STU_ ID)];
6	**COLUMN**[score; SC AVG; 0.0; 30; 50; 100];
7	**IMPORT TBL**[student; score];

表 4 包含两个事务, 每个事务均指定了执行比例和事务名, 案例中涉及的表为 score. 行 2—4 将随机执行*SELECT_FOR_UPDATE* 若干次(*MULTIPLE*), 次数为 1∼100 之间的随机数. *SELECT_FOR_UPDATE* 操作以 STU_ID 为过滤条件, 按照正态分布(*NORMAL* 最小值为 1, 最大值 100, μ 默认为 0, σ 取 1)访问 score 表, 并将查询结果按照 SC_AVG 降序排列. 行 9 将对 score 表所有 SC_AVG 大于 90 分的学生等级设置为 A, 访问分布为*UNIQUE* 的唯一值分布. 行 10—13 的分布结构包含两个分支(*BRANCH*), 每个分支的执行概率分别为 0.8 和 0.2, 行 15 以异步的方式加载事务(*TXN_LOADING*), 在一台负载机上开启 10 个数据库链接, 每个链接执行 100 次.

4 Woodpecker+ 架构

如图 1 所示, Woodpecker+ 总体上可以分为两个部分, 客户端和数据库服务器端. 其中客户端模块有 5 个核心组件: 解析器、数据库适配器、负载分发器、负载机和统计信息收集, 数据库服务器端即为待测的 DBMS 服务器和资源监控器.

测试流程大致如下: 解析器首先读取测试环境的配置文件和测试案例, 生成负载任务(可能有多个)传递给负载分发器. 解析的过程中数据库适配器将测试案例中的某些数据类型或语法适配为目标 DBMS 所支持的类型. 接着负载机(可能有多个)接收负载分发器发送的负载任务并以

多线程的方式将数据库操作发送给 DBMS 执行. 执行过程中, 统计信息收集模块实时收集负载机端计算的性能测试结果和数据库服务器端的硬件资源消耗指标, 并在测试结束时生成综合性能报告. 负载管理器在发送新的负载任务前, 需确保上一个负载任务全部执行完成, 统计信息收集模块给通知负载管理器上一个负载任务是否已完成, 负载分发器再决定是否发送新负载.

表 4　自定义数据特征事务案例

Tab. 4　Customized data characteristics transactions example

行号	操　作
1	**TXN**[0.4; "AddScores"];
2	**MULTIPLE**[1, 100];
3	**REPLACE**[score; true; NULL;];
4	**END_MULTIPLE**;
5	**SELECT_FOR_UPDATE**[score; true; NORMAL(1, 100, 1)
6	SC_AVG, SC_GRADE; FILTER(STU_ID =); APPEND(ORDER BY SC_AVG)];
7	**TXN_END;**
8	**TXN**[0.6; "OperateScores"];
9	**UPDATE**[score; true; UNIQUE(1, 2000); SC_GRADE = 'A'; filter(SC_AVG > 90)];
10	**BRANCH**[0.8; 0.2];
11	**SELECT**[score; true; ZIPFIAN(1, 100, 10, 3) "*"; filter(STU_ID =); append(GROUP BY SC_GRADE)];
12	**BRANCH_DELIMITER**
13	**DELETE**[score; true; UNIFORM[1, 2000]; FILTER(S_AVG = 0)];
14	**TXN_END;**
15	**TXN_LOADING**[10, 100,1, asyn]

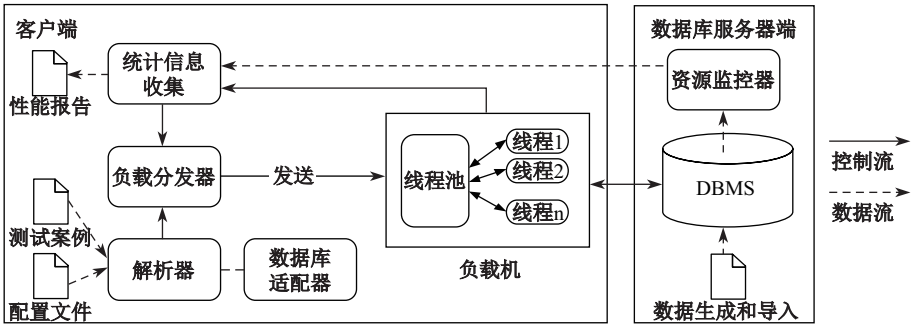

图 1　Woodpecker+ 架构

Fig. 1　Architecture of Woodpecker+

4.1　解释器和负载分发器

解析器和负载分发器是客户端中重要的模块. 解析器读取测试人员定义的测试案例(包括表模式、具体事务操作、加载事务的线程数、每个线程执行次数、负载个数等)和测试环境配置文件(包括 DBMS 端 IP 地址和端口号、负载机个数及 IP 地址集合、Netty 通信端口号等), 使用 JavaCC[11]完成词法、语法解析生成负载任务并传递给负载分发器. 负载分发器负载任务逐个分发给负载机, 由于负载机可能为多个, 且不同负载机的完成情况是异步的, 所以统计信息收集模块采用同步操作来保证所有负载机均成功反馈上一个负载任务的执行结果, 并通知负载分发器上一个负载任务已完成, 这时负载分发器才发送新的负载任务. 最后, 当所有的负载任务均执行结束, 负载分发器通知关闭所有负载机上的监听进程, 结束本地客户端进程. 若发生异常,

则提前结束.

4.2 负载机实体

性能测试的目的是在高压环境下测试系统的性能峰值. 负载机是连接负载分发器和 DBMS 系统的中间模块, 用来完成高并发的数据请求操作. 为了不让发送操作请求的速度成为测试瓶颈, 选择开启多台负载机, 每台负载机以多线程的方式同时发送数据库操作请求. 在实现时每个线程对应一个数据库链接. 并不意味着更多的数据库链接就可以测试出更高数据库性能. 实验发现, 一旦数据库链接数超过数据库固有的最大链接数(尽管可手动将该值调大), 系统的吞吐量下降, 延迟大幅度增加. 原因为当某一时刻的请求速率使得数据库达到最佳性能后, 之后大量的请求将阻塞在请求队列中, 这时数据库处理能力已达到上限, 大量未处理的操作增加了操作延迟, 导致系统的吞吐量下降. 此外, 负载机还负责计算操作执行的性能指标(如吞吐量和百分比延迟), 并实时反馈给统计信息收集模块.

4.3 统计信息收集模块

由于性能测试可能耗时较长, 如果在整个测试任务的执行过程中, 测试人员无法知道测试进展, 这对测试人员是难以忍受的. 所以统计信息收集模块负责汇总各个负载机计算的性能指标, 如当前正在执行的事务 ID、负载完成进度(100% 为完成)、瞬时 TPS、测试时长、50%、90%、95%、99% 延迟、每个操作的平均延迟等; 该模块还收集数据库服务器端的硬件资源使用情况, 由于测试任务可能包含多个负载, 而每个负载由所有负载机同时执行, 这将导致各个负载机的统计信息到达是异步的, 所以统计信息模块收集到所有负载机的统计信息才通知负载管理器发送下一个负载.

4.4 数据库适配器

由于数据库产品的多样性, 许多 DBMS 在支持的数据类型和数据定义语言(DDL)上存在很大差异, 为了使关键字编写的测试案例能够以尽可能小的代价复用于多种数据库系统, 工具提供了数据库适配器模块, 该模块自动将测试案例中的一些数据类型和 DDL 操作转换为目标 DBMS 支持的类型和正确的语法格式. 这样测试人员只需专注于构造测试案例, 而无需考虑底层数据库系统某些细节上的差异. 这极大地提高了测试效率.

4.5 资源监控器

对于 DBMS 所在的服务器, 负载任务的高并发执行导致系统硬件资源处于紧张占用状态, Woodpecker+ 使用轻量级的开源系统监控工具——nmon[12], 对系统的 CPU、内存、磁盘读写速度、磁盘 I/O 和网络带宽等资源进行监控. 这些系统指标能够更好地帮助测试人员理解负载的执行过程并为发现系统瓶颈提供有效信息. 资源监控器在测试结束时将所有的统计信息发送给客户端的统计信息收集模块.

5 实 验

5.1 实验设置

首先, 以几种常用的标准 OLTP 类型的 Benchmark 作为实验对象, 比较 Woodpecker+ 与 OLTP-Bench 工具[6]对这些 Benchmark 实现的代码行数, 展示本文工作的高效性和用户友好性. 其次, 通过比较 Woodpecker+ 与 Sysbench 工具[5]对测试案例的组织方法, 展现工具的灵活性. 再者, 展示 Woodpecker+ 对分布式数据库系统测试方面支持, 表现其测试功能支持的全面性. 最后, 展示 Woodpecker+ 在负载生成上的可扩展性.

实验使用了 3 种数据库产品, 分别是开源集中式数据库 MySQL 5.7.18 和 PostgreSQL 9.6 和分布式开源数据库 CBase 1.2[13]. 它们部署在同一集群环境下的 4 台服务器上, 每台机器 CPU 为两个 Intel Xeon(R) E5-2620, 共 12 核 24 线程, 内存 128 GB, 1 000 Mbps 以太网, 操作系统 CentOS 6.8. 其中 MySQL 和 PostgreSQL 部署在同一台服务器, CBase 由于是分布式系

统, 且为了降低控制节点(RootServer)和事务处理节点(UpdateServer)的压力, 它们分别部署在两台服务器上, 基线数据存储节点(ChunkServer)和接口服务节点(ChunkServer)部署于同一台服务器.

5.2 TDL 测试案例构造代价对比

以几种 OLTP 类型的 Benchmark 作为实验对象, 比较 OLTP-Bench 中 Java 实现这些负载所需要的代码行数和 Woodpecker+ 中 TDL 实现所需的代码行数, 发现 Java 需要上千行代码控制负载的执行与信息统计, 而 TDL 编写的测试案例只需几十行代码便可清楚描述负载任务, 实验结果对比见表 5, 大约降低了**两个数量级**的工作量. 相比于 Java 任务描述于代码绑定的方式, Woodpecker+ 中 TDL 的测试案例具有更好的复用性和灵活性. 这极大地降低了测试人员构造测试案例的代价.

表 5　TDL 测试案例构造代价对比

Tab. 5　Comparison of the cost of generating a test case with TDL

Benchmark	表数	事务数	Java代码行数	TDL代码行数
TPC-C	9	5	6223	84
SmallBank	3	6	4575	39
SEATS	10	6	7364	48
Voter	3	3	4278	13
YSCB	1	6	4368	12

5.3 负载组织案例

该组实验对比了 Woodpecker+ 中的 TDL 与 Sysbench 提供的方法对读密集型负载组织的差别, 见表 6 和表 7. SmallBank 共有 6 组事务, 其中有 1 组纯读事务, 另外 5 组事务仅包含 Select 和 Update 操作, 其中有 4 组事务仅有一条 Update 操作且均为最后一条操作. 为了设计读密集型负载, 该组实验移除了 SmallBank 中所有事务中的 Update 操作, 并保持原有事务的执行比例, 通过 Woodpecker+ 定义的关键字组织 SmallBank 中的负载实现对读密集型任务的定义, 如表 6. 其中所有的事务均以预编译的方式执行, 3 张表共导入 300 万条记录, 每条操作的数据访问方式为 unique(0∼999 999), 过滤条件为账户表主键.

在表 6 的测试案例中, 定义了 3 张表和 6 组事务, 表模式和事务操作均可自定义, 表模式可设定表大小、属性名、数据类型和主外键列等信息. 事务可设定执行比例, 事务中操作可包含表名、是否预编译执行、访问分布类型和过滤条件.

表 7 为相同负载针对 MySQL 语法在 Sysbench 中的实现的部分代码, 行 2—23 创建账户表, 剩下的两张表和创建账户表相似. 行 25—36 执行第一组事务, 事务的 3 条操作对应 3 个独立实现的函数. Sysbench 默认只实现了 Uniform 随机分布, 其他的访问分布需额外实现. 该方式测试案例的语法针对 DBMS 定制, 比如在表 7 中, *CREATE TABLE ACCOUNT*(*custid INTEGER NOT NULL, name VARCHAR DEFAULT '0' NOT NULL, PRIMARY KEY*(*custid*))中的数据类型适用于 MySQL 数据库. 在多 DBMS 的适配上存在缺陷, 而且功能代码冗余, 难以复用在其他负载上. Woodpecker+ 提供的案例组织手段则可方便地直接迁移到其他数据库上进行测试, 提供了负载任务定义灵活性的同时降低了测试代价.

5.4 分布式系统性能测试表达能力

分布式事务是分布式数据库设计和实现的难点之一. 目前许多的 NewSQL 系统声称支持分布式事务[14], 但分布式事务的复杂性使得简单的 SQL 功能测试不一定能完成全面的分布式 DBMS 的测试. Woodpecker+ 同样提供了丰富的关键字以支持分布式系统的分布式事务性能测试, 通过在创建表模式时使用 *PARTITION_BY* 关键字与哈希分区函数绑定来实现数据分布,

通过向分区规则表和 Paxos 系统表插入分区函数和 Paxos 组信息来实现事务分组, 通过设置 INSERT、UPDATE 和 REPLACE 操作的访问分布来提高分布式事务的发生率.

表 6 SmallBank 读密集型负载组织-TDL 实现

Tab. 6 Read intensive workload organization of SmallBank with TDL

行号	操 作
1	// **创建三张表, 每张表大小100万**
2	**TABLE**[ACCOUNTS; 1000000; custid int, name varchar(64); PK(custid)];
3	**TABLE**[SAVINGS; 1000000; custid int, bal float; PK(custid)];
4	**TABLE**[CHECKING; 1000000; custid int, bal float; PK(custid)];
5	// **导入创建的三张表**
6	**IMPORT_TBL**[ACCOUNTS; SAVINGS; CHECKING];
7	// **第一组事务, 执行比例**15%
8	**TXN**[0.15; “txn1”];
9	SELECT[ACCOUNTS; true; unique(0,999999); "*"; filter(custid =)];
10	SELECT[SAVINGS; true; unique(0,999999); "bal"; filter(custid =)];
11	SELECT[CHECKING; true; unique(0,999999); "bal"; filter(custid =)];
12	**END_TXN;**
13	// **第二组事务, 执行比例**15%
14	**TXN**[0.15; “txn2”];
15	SELECT[ACCOUNTS; true; unique(0,999999); "*"; filter(custid =)];
16	SELECT[SAVINGS; true; unique(0,999999); "bal"; filter(custid =)];
17	SELECT[CHECKING; true; unique(0,999999); "bal"; filter(custid =)];
18	**END_TXN;**
19	// **第三组事务, 执行比例**15%
20	**TXN**[0.15; “txn3”];
21	SELECT[ACCOUNTS; true; unique(0,999999); "*"; filter(custid =)];
22	**END_TXN;**
23	// **第四组事务, 执行比例**15%
24	**TXN**[0.25; “txn4”];
25	SELECT[ACCOUNTS; true; unique(0,999999); "*"; filter(custid =)];
26	SELECT[CHECKING; true; unique(0,999999); "bal"; filter(custid =)];
27	**END_TXN;**
28	// **第五组事务, 执行比例**15%
29	**TXN**[0.15; “txn5”];
30	SELECT[ACCOUNTS; true; unique(0,999999); "*"; filter(custid =)];
31	SELECT[SAVINGS; true; unique(0,999999); "bal"; filter(custid =)];
32	**END_TXN;**
33	// **第六组事务, 执行比例**15%
34	**TXN**[0.15; “txn6”];
35	SELECT[ACCOUNTS; true; unique(0,999999); "*"; filter(custid =)];
36	SELECT[SAVINGS; true; unique(0,999999); "bal"; filter(custid =)];
37	SELECT[CHECKING; true; unique(0,999999); "bal"; filter(custid =)];
38	**END_TXN;**
39	// **导入创建的三张表**
40	**TXN_LOADING**[100; 10000; 1; sync]

该组实验对象为 CBase1.2 分布式数据库系统, 该版本拓展单事务处理架构(单 Paxos 组)为多事务处理结点架构(多 Paxos 组), 本文关注多事务处理结点架构下的分布式事务测试. 分布式事务测试往往需要将数据和负载进行映射[15], 原始测试方法需手工准备测试环境, 包括分区集群部署、分区规则创建、表模式创建和数据导入, 然后才运行负载, 测试过程难以自动化和回归. Woodpecker+ 可自动化分布式事务测试, 测试案例中可同时兼容 SQL 功能性语句和事务操作, SQL 功能性语句的支持能解决测试人员的介入问题, 提高了测试灵活性.

表 7　SmallBank 读密集型负载组织-Sysbench 实现

Tab. 7　Read intensive workload organization of SmallBank with Sysbench

行号	操　作
1	//创建账户表的函数
2	function create_account_table(table_num)
3	local query
4	query = string.format(CREATE TABLE ACCOUNT(custid INTEGER NOT NULL ,
5	name VARCHAR DEFAULT '0' NOT NULL, PRIMARY KEY(custid)))
6	con:query(query)
7	if sysbench.opt.auto_inc then
8	query = "INSERT INTO ACCOUNT" .. table_num .. "name) VALUES"
9	else
10	query = "INSERT INTO ACCOUNT" .. table_num .. "(custid, name) VALUES"
11	end
12	con:bulk_insert_init(query)
13	for i = 1, sysbench.opt.table_size do
14	name_val = get_name_value()
15	if (sysbench.opt.auto_inc) then
16	query = string.format("('%s')",sysbench.rand.default(1, sysbench.opt.table_size),name_val)
17	else
18	query = string.format("(%d,'%s')",i,sysbench.rand.default(1,sysbench.opt.table_size),name_val)
19	end
20	con:bulk_insert_next(query)
21	end
22	con:bulk_insert_done()
23	end
24	//执行第一组事务的函数
25	function execute_trx1()
26	if not sysbench.opt.skip_trx then
27	begin()
28	end
29	account_point_selects()
30	savings_point_selects()
31	checking_point_selects()
32	if not sysbench.opt.skip_trx then
33	commit()
34	end
35	check_reconnect()
36	end

为了验证 CBase 系统的分布式事务处理性能, 该实验构造了一个混合负载, 负载包括一组事务, 事务包含 4 条操作: 主键的单点查询、二级索引列上的范围查询、主键的单点更新、随机主键的替换操作. 负载分别在双 Paxos 组和三 Paxos 组情况下测试, 为了提高分布式事务发生的概率, 数据均匀的分布在 Paxos 组中, 事务中操作的访问分布与 Paxos 组数据的分区保持一致.

表 8 为双 Paxos 组情况下的分布式事务, 行 1 创建一个数据库链接, 行 3 向Paxos系统表插入 Paxos 组信息, 行 7 向分区规则表插入哈希函数. 行 11 中 *PARTITION_BY* 关键字使 USERTABLE 表的数据按照相应的哈希函数进行分区, 行 13 导入数据, 行 15—20 执行事务操作, 行 18—19 中操作的访问分布均分了 0~200 万行的数据, 即 UPDATE 和 REPLACE 两条操作分别发送到两个 Paxos 组的主系统上处理, CBase 使用两阶段提交协议处理分布式事务[16], 分布式事务的比例达到 100%. 若同样的分布式事务在 Sysbench 或者 OLTP-Bench 上执行, 需要先手动创建哈希规则、向 Paxos 组系统表中插入分区记录和导入数据(这一步骤对应表 8 的行 1—13), 最后再执行测试负载, 不支持事务操作和基本 SQL 操作的混合, 测试过程无法自动化.

表 8 双 Paxos 组分布式事务测试负载

Tab. 8 Distributed transaction workload of Double Paxos group

行号	操 作
1	**CREATE_CLIENT** clt1;
2	// 三Paxos**组需多插入一条**(‘par1#2’,2,2)**记录**
3	REPLACE INTO __all_all_group(group_name, start_version, paxos_id) VALUES ('par1#0', 2, 0), ('par1#1', 2, 1);
4	//三Paxos**组修改哈希函数**’x%2’为’x%3’
5	INSERT INTO __all_partition_rules(rule_name, rule_par_num, rule_par_list, rule_body, type) values ('func1', 1, 'x', 'x%2', 0);
6	//三Paxos**组插入的数据为**300**万**
7	TABLE[USERTABLE; 2000000; YCSB_KEY int, FIELD1 varchar(100), FIELD2 varchar(100), FIELD3 varchar(100)，
8	**PARTITION_BY**(par1, func1, YCSB_KEY)];];
9	**IMPORT_TBL**[USERTABLE;];
10	//三Paxos**组**UPDATE**和**REPLACE**的唯一值访问分布范围为**100~200**万和**200~300**万**
11	**TXN**[1.0; “distributed_txn”];
12	SELECT[USERTABLE; true; unique(0, 9999999); "*"; filter(YCSB_KEY =)];
13	SELECT[USERTABLE; true; unique(0, 9999999); "*"; filter(FIELD1 >, &, FIELD1 <)];
14	**UPDATE**[USERTABLE; true; unique(0, 999999); FIELD2 = “tee”; filter(YCSB_KEY =)];
15	**REPLACE**[USERTABLE; true; unique(1000000, 1999999); FIELD2 =”tee”; filter(YCSB_KEY =)];
16	**END_TXN**
17	**TXN_LOADING**[100; 12000; 1; sync];

5.5 负载可拓展定义

为了验证负载的可拓展性, 该组实验使用 YCSB 读写混合负载作为实验负载, 在 MySQL、PostgreSQL 和 CBase1.2 上测试. 为了使负载机所能创建的最大链接数不影响测试结果, 该实验使用多台负载机, 每台负载机均分总的链接数并限制每台最多建立 40 个链接. 通过不断的增加负载机和 DBMS 之间的链接数, 统计负载在稳定后所能达到的最大 TPS, 以及此时 DBMS 服务器端 CPU 的使用率. 测试案例定义如表 9, 性能结果如图 2.

表 9 为负载可拓展定义测试案例, 行 1 创建表模式、行 5 导入数据, 行 6—11 执行事务操作, 行 14 是该组测试的重点, 当测试人员指定了上述 3 个参数后, Wood-pecker+ 从最小线程数 10 开始, 逐次增加 10 个线程(该间隔可修改)直到等于最大线程数为止运行多次测试负载, 线程

数指定后, 线程的执行次数等于表总数除以线程数, 负载机个数则根据线程数动态调整. 每一次的运行结果将单独打印出来, 该组实验打印 12 组测试结果. Sysbench 虽然支持多组线程指定, 但是在负载机上均分线程时易出现异常, 即负载机上建立的链接数容易达到链接上从而限导致测试结果不准确, 该现象可能由 Sysbench 的线程池策略导致. OLTP-Bench 不支持多组线程指定.

表 9　负载可拓展定义案例

Tab. 9　Extensible workload test case

行号	操　作
1	**TABLE**[USERTABLE; 1200000; YCSB_KEY int, FIELD1 varchar(100), FIELD2 varchar(100),
2	FIELD3 varchar(100), FIELD4 varchar(100), FIELD5 varchar(100), FIELD6 varchar(100),
3	FIELD7 varchar(100), FIELD8 varchar(100), FIELD9 varchar(100), FIELD10 varchar(100);
4	PK(YCSB_KEY)];
5	**IMPORT_TBL**[USERTABLE;];
6	**TXN**[0.8; “read_txn”];
7	SELECT[USERTABLE; true; zipfian(0,1199999, 10, 3); "*"; filter(YCSB_KEY =)];
8	**END_TXN** ;
9	**TXN**[0.2; “write_txn”];
10	DELETE[USERTABLE; true; zipfian(0, 1199999, 10, 3); filter(YCSB_KEY =)];
11	**END_TXN** ;
12	// **三个参数的意义分别为: 最大线程数、每个线程的执行次数、负载机个数和负载执行方式**
13	// **可通过调整最大线程数和负载机个数来控制负载的执行次数。**
14	**TXN_LOADING[120; 10000; 3; sync];**

图 2 左为吞吐量和链接数的关系, CBase1.2 和 MySQL 在使用 50 个链接数时, TPS 基本已达到上限, 之后再增加链接数, 性能均无明显变化. PostgreSQL 在链接数增加的过程中, TPS 逐渐上升, 在链接数为 90 时, 达到最大吞吐量. 总之集中式数据的性能要高于分布式系统. 这是因为分布式数据库除了要保证数据的正确性外, 还要保证数据的可靠性, 如多日志同步机制, 多副本备份等, 这都增加了系统的开销.

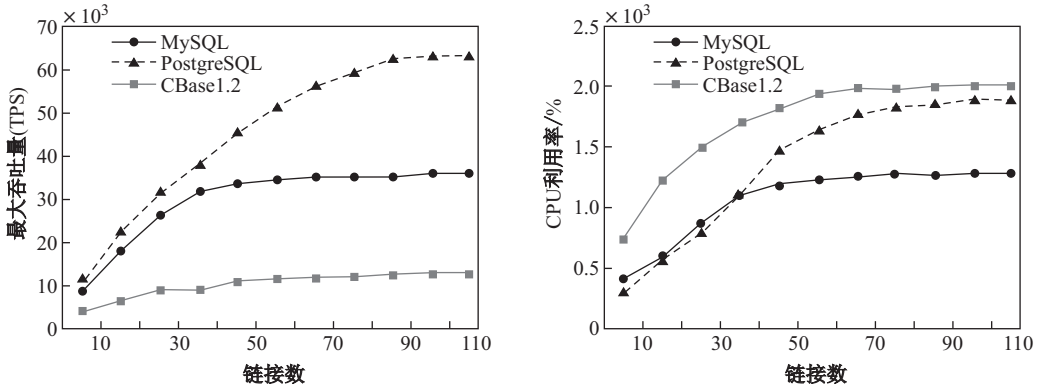

图 2　不同链接数时的吞吐量和 CPU 利用率

Fig. 2　Throughput and usage of CPU with varing number of database links

图 2 右为 CPU 利用率和链接数的关系, 结合图 2 左可知, 随着链接数的增加数据库的吞吐

量逐渐提升, 该过程 CPU 的利用率变化符合预期, 即 CPU 利用率与链接数成正比, 并在数据库达到最大吞吐时维持稳定.

吞吐量和 CPU 利用率是衡量数据库性能的重要指标, 相比于手动的统计信息收集, 自动化的统计信息收集显得尤为重要, 该组实验所有的统计结果均来自 Woodpecker+ 的统计信息收集模块, 该模块生成结构化日志, 对分析负载执行过程及发现系统瓶颈提供帮助.

6 总 结

本文提出了可基于数据特征的自定义负载评测工具——Woodpecker+, 提供了语义丰富, 简便易于的测试关键字, 以及支持灵活负载组织方式, 测试人员编写的测试案例复用性高, 工具拓展性良好. 通过实验验证了 TDL 测试案例的构造代价对比和读密集型负载组织和分布式事务性能测试以及验证了两种类型数据库的可拓展性.

[参 考 文 献]

[1] GEORGE L. HBase: The Definitive Guide: Random Access to Your Planet-Size Data[M]. CA: O'Reilly Media, 2011.

[2] STONEBRAKER M, WEISBERG A. The VoltDB Main Memory DBMS[J]. IEEE Data Eng Bull, 2013, 36(2): 21-27.

[3] NVM [EB/OL]. [2019-01-14]. https://en.wikipedia.org/wiki/NVM.

[4] ORACLE. The MySQL Test Framework[EB/OL]. [2017-07-12]. https://dev.mysql.com/doc/mysqlt est/2.0/en/.

[5] SysBench: A system performance benchmark[EB/OL]. [2019-01-14]. https://sysbench.sourceforge.net.

[6] DIFALLAH D E, PAVLO A, CURINO C, et al. OLTP-Bench: An extensible testbed for benchmarking relational databases[J]. Proceedings of the VLDB Endowment, 2013, 7(4): 277-288.

[7] ECNU [EB/OL]. Woodpecker. [2019-01-14]. https://github.com/Gizing/Woodpecker.

[8] Transaction Processing Performance Council (TPC) [EB/OL]. [2019-01-14]. http://www.tpc.org. 2011.

[9] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems with YCSB[C]//Proceedings of the 1st ACM symposium on Cloud computing, New York: ACM, 2010: 143-154.

[10] CAHILL M J , ROHM U , FEKETE A D . Serializable isolation for snapshot databases[J]. ACM Transactions on Database Systems, 2009, 34(4): 1-42.

[11] KODAGANALLUR V. Incorporating language processing into Java applications: A JavaCC tutorial[J]. IEEE Software, 2004, 21(4): 70-77.

[12] Nmon for Linux [EB/OL]. [2019-01-14]. http://nmon.sourceforge.net/pmwiki.php.

[13] CBASE. Bank of Communications [EB/OL]. [2019-01-14]. https://github.com/BankOfCommunications/CBASE.

[14] YAN C, CHEUNG A. Leveraging lock contention to improve OLTP application performance[J]. Proceedings of the Vldb Endowment, 2016, 9(5): 444-455.

[15] 杨传辉. 大规模分布式存储系统原理解析与架构实现[M]. 北京: 机械工业出版社, 2013.

[16] Two-phase commit protocol [EB/OL]. [2019-01-14]. https://en.wikipedia.org/wiki/Two-phase_commit_protocol.

(责任编辑: 张 晶)