文章编号:1000-5641(2014)05-0320-10

如何客观评测内存数据库的性能

康强强[1], 金澈清[1], 张 召[1], 胡华梁[1,2], 周傲英[1]
(1. 华东师范大学 软件学院 数据科学与工程研究院,上海 200062;
2. 浙江理工大学 经济管理学院,杭州 310033)

摘要: 在过去的 10 年间,随着硬件技术不断发展,内存价格越来越低,许多计算机系统均布置了大容量内存.数据库系统开发商和研究人员认识到这一趋势,并开发出多款内存数据库产品,其特点在于先将数据装载到内存之中,再执行相应的数据管理任务.随着内存数据库的出现,如何客观、公正地评测它的性能显得愈发重要.尽管当前不乏关于关系型数据库系统的评测基准,例如威斯康星测试基准和 TPC-X 系列等,但是这些基准并未充分考虑内存数据库的重要特性,因此不适合评测内存数据库.本文提出了一种面向内存数据库的评测基准(InMemBench),与传统的关系数据库基准显著不同,它综合考虑了内存数据库特有的数据预取过程、物理组织方式和压缩能力等方面的重要特点.最后,本文还通过新基准比较了 4 款内存数据库的性能.

关键词: 内存数据库; 评测基准; 工作负载; 度量
中图分类号: TP333.1 **文献标识码**: A **DOI**:10.3969/j.issn.1000-5641.2014.05.029

How to evaluate in-memory database objectively

KANG Qiang-qiang[1], JIN Che-qing[1], ZHANG Zhao[1],
HU Hua-liang[1,2], ZHOU Ao-ying[1]
(1. *Institute for Data Science and Engineering, Software Engineering Institute, East China Normal University, Shanghai* 200062, *China*;
2. *School of Economics and Management, Zhejiang Sci-tech University, Hangzhou* 310033, *China*)

Abstract: The hardware technology continues to develop in the past decade, and the price of memory gets lower so that many computer systems tend to deployhuge-size memory. To fulfill this benefit, the researchers also developed several in-memory databases (IMDB) that execute workloads after pre-loading the whole data into memory. The bloom of various in-memory databases shows the necessity of testing and evaluating their performance objectively and fairly. Although the existing database benchmarks have shown great success during the development of the database technologies, including Wisconsin benchmark, TPC-X series, and so on, such work cannot be applied straightforwardly due to the lack of consideration of several unique characteristics of in-memory databases. In this article, we propose a novel benchmark, called InMemBench,

收稿日期:2014-06
基金项目:国家自然科学基金(61370101);上海市教委创新项目(14ZZ045);上海市自然科学基金(14ZR1412600)
通信作者:金澈清,男,研究方向为基于位置的服务、不确定数据管理、数据流管理和数据基准评测. E-mail: cqjin@sei.ecnu.edu.cn.

to test and evaluate the performance of an in-memory database objectively and fairly. Different from traditional database benchmarks, we take special consideration of startup, data organization, and data compression. Moreover, we conduct extensive experiments to illustrate the effectiveness and efficiency of our proposal.

Key words: in-memory database; benchmark; workload; measurement

0　引　　言

在过去的 10 年间，随着硬件迅速发展，内存价格显著降低，一个典型的计算机系统往往会布置大容量内存. 数据库产品提供商于是投入大量资源来开发内存数据库，以提高数据处理吞吐量，其中包括 SAP 公司的 HANA[1]，Oracle 公司的 TimesTen[2]，还有开源的 MonetDB[3] 等等. 内存数据库需要预先将数据装载到内存中，从而避免了后续的数据管理阶段的 I/O 开销. 随着诸多内存数据库产品的问世，如何公平、客观地评测这些内存数据库的性能也变得愈发重要.

数据库基准旨在客观、公正地评测数据库产品的性能，以激励数据库厂商优化其产品. 自从威斯康星评测基准被提出以来，数据库评测基准已在过去 30 年里取得了巨大的成功[18]. 事务处理性能委员会(Transaction Processing Performance Council，TPC)提出了一系列基准来评测关系型数据库，包括 TPC-C、TPC-H、TPC-E、TPC-DS 等，得到学术界和工业界的广泛认可. 其他典型基准包括混合了 TPC-C 和 TPC-H 工作负载的 CH-Benchmark[4] 和针对决策支持系统的星型模式评测基准(Star Schema Benchmark，SSB)[5]. 此外，针对非关系型数据库的评测基准包括针对 NoSQL 数据库的 YCSB 测试基准[6] 和针对面向对象数据库的 Bucky 测试基准[7] 等. 传统数据库系统将数据存储于磁盘之中，再通过大量 I/O 操作执行数据管理任务，现有基准均力求反映这一重要特性. 然而，与传统数据库相比，内存数据库具有以下重要性质：(1)启动时加载：内存数据库需要将数据预先加载到内存中. (2)数据组织：内存数据库并不局限于采用行存储方式，也可采用列存储方式，或者同时支持两种数据存储方式. (3)数据压缩：内存数据库系统广泛采用数据压缩技术，以在内存之中存储更多数据. 总之，现有数据库基准并不适合评测内存数据库，亟需设计新基准进行评测，以充分反映以上若干特性.

本文设计了一个新的数据库基准(InMemBench)来评测内存数据库系统. 该评测标准的数据库模式包含 8 张表：2 张大表和 6 张小表. 大表的字段较多，分别超过 100 列和 200 列；小表的列数较少，不超过 30 个. 该基准还包括一个并行数据生成工具 COLGen，以产生模拟数据. 该基准拥有 4 个新颖的度量：启动消耗(Startup Cost)、压缩率(Compression Ratio)、每小时执行查询个数(Query per hour)，和列扩展性(Column Scalability). 在工作负载方面，该基准包括 2 种类型，分别聚焦读操作和写操作.

本文第 1 节描述相关工作，第 2 节介绍数据库模式，第 3 节介绍度量指标，第 4 节介绍工作负载和执行计划，第 5 节报告实验结果，并在最后一节总结全文.

1　相关工作

随着数据库技术不断发展，数据库评测基准也获得较大提高. 从 20 世纪 80 年代开始，

学术界和工业界已经研发出多款数据库评测基准,包括针对关系型数据库的威斯康星测试基准(Wisconsin)和 TPC 系列,针对面向对象数据库的 OO7[8] 和 bucky[7] 评测基准,针对 XML 数据的 XMark[9] 和 EXRT[10] 评测基准,还有针对于大数据应用的 YCSB[6] 和 BigBench[11] 评测基准等.

近期,内存数据库技术得到了较大发展,典型代表包括 SAP 的 HANA、Oracle 的 TimesTen,微软的 Hekaton,还有开源的 HyperSql、SQLite、MemSql 和 MonetDB 等,这些数据库均采用关系数据模型. 威斯康星基准是一款较早的评测基准,其数据库模式较为简单,较好地评测了当时的数据库产品. TPC 开发了一系列数据库基准来评测数据库,包括针对于在线事务处理(Online Transaction Processing,OLTP)的 TPC-C 和 TPC-E,针对在线分析处理(Online Analytical Processing,OLAP)的 TPC-H 和 TPC-DS. CH-Benchmark 则混合 TPC-C 和 TPC-H 的特点,可同时评测数据库 OLTP 和 OLAP 功能. Set query 基准[12] 使用一组基本的查询语句,从文档查询、市场决策支持的角度来评估数据库的性能. 星型测试基准 SSB 使用星型数据模型来测试数据库的性能,这样做主要是为了支持传统的数据仓库的应用.

然而,以上评测基准并未充分考虑内存数据库的特性,因此并不适合内存数据库. 研究人员尝试采用现评测基准来评估内存数据库的性能,但是在度量选取等方面并未全面地契合内存数据库的特性[13-17]. 本文所提出的新基准则尝试更加全面地解决了以上问题.

2 数据模型

本节先描述数据库模式及其特点,再介绍数据生成器 COLGen.

2.1 数据库模式

图 1 描述了 InMemBench 的数据库模式. 该模式使用了 8 张表来描述一个零售产品供应的应用场景. PART 表描述销售的零件,SUPPLIER 表和 CUSTOMER 表描述零件的提供商和所有客户,NATION 表和 REGION 表描述供应商和客户来自的国家和地区. 由于一个零件可能有不止一个提供商,因此使用 PARTSUPP 表来描述此依赖关系. 最后,ORDERS 表描述客户的所有订单,每个订单的详细信息存储在 LINEITEM 表中. 这个模式源于 TPC-H 数据库模式,但为了适应内存数据库而进行了若干调整.

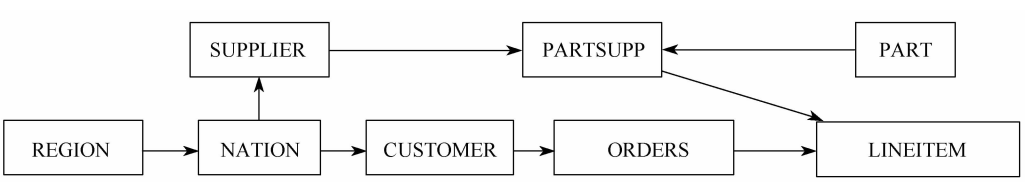

图 1 数据库模式

Fig. 1 Database schema

主要的扩展包括两处. 第一,向 LINEITEM 表中添加 200 个字段,名为:COLUMN1,…,COLUMN200,分属 3 种数据类型:字符型 char、数值型 decimal(13,2)和日期类型 date. 在这新添的 200 个字段中,规定 20%的字段使用数值型,剩下的 20%和 60%分别使用日期类型和字符类型. 第二,向 ORDERS 这张表中添加 100 个字段,这 100 个字段也是有数值型、字符型和日期型 3 种类型组成,分别命名为抽象字段 COLUMN1,…,COLUMN100. 在

这新添的 100 个字段中，他们的类型分布和 LINEITEM 这张表是一样的. 这样设计的想法最初来自于真实的销售场景中，有些大型企业，例如淘宝和亚马逊，它们的销售表往往是超过 16 个字段的(TPC-H 最大的表字段个数是 16). 因此，这需要对维护销售明细的字段进行一个扩展. 其实，这两张表中所有新加的字段在实际的销售场景中也有明确的含义，例如，LINEITEM 这张表就有表示物品毛重和净重的新增字段. 考虑到空间的限制，这里没有列出其他 198 个字段的介绍. 这样设计的另外一个想法是，足够多的字段给我们评测行存储数据库和列存储数据库提供了一个可能途径.

本数据库模式的一个主要特点是“动态可适应性”：即 LINEITEM 和 ORDERS 表中的字段个数会改变. 该动态模型主要包含 6 张小表(SUPPLIER，PARTSUPP，PART，REGION，NATION 和 CUSTOMER)和 2 张大表(LINEITEM 和 ORDERS). 模型的动态变化主要是 2 张大表中新增字段个数的变化. 例如，在模型的 5 次动态变化过程中，每一个状态下 2 张大表新增的字段数目分别占新增总字段的 20%、40%、60%、80%和 100%. 这样设计是为了评测行存储和列存储 2 种不同存储方式的列可扩展性.

2.2　数据生成器

通过扩展 DataGen(TPC-H 的数据生成工具)，我们研发了一个并行数据生成工具(COLGen)来产生模拟数据. 该数据生成器需要使用数据字典和产生规则(如图 2 所示). 数据字典包括了 3 类单词，这些单词均可通过 RNG 的随机选取填充到对应字段中；数据产生规则定义了各字段之间的依赖关系、单个字段值的填充形式和取值范围等.

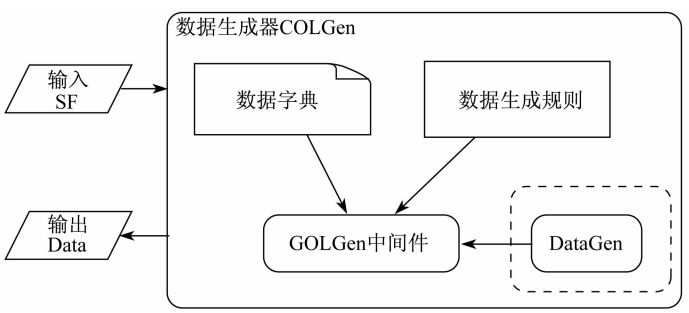

图 2　COLGen 的架构

Fig. 2　The architecture of COLGen

此外，还定义了扩展因子(Scale Factor，SF)来控制模拟数据库的规模. 不同 SF 值对应不同的数据库规模. 例如，当 $SF = 1$ 时，COLGen 会生成 10 GB 数据；若 $SF = 2$，则生成 20 GB数据；依次类推.

3　度量指标

本文定义了 4 个度量指标来评测内存数据库的性能，即：启动消耗(Startup Cost，$SC@SF$)、压缩率(Compression Ratio，$CR@SF$)、每小时查询个数(Query per hour，$Qph@SF$)和列处理能力(Column Scalability，$CS@SF$).

• 启动消耗 $SC@SF$. 启动消耗描述将数据从磁盘装载到内存所耗费的时间. 各内存数据库都需要预先将数据装载入内存中以进行管理.

• 压缩率 $CR@SF$. 数据的压缩率描述内存数据库压缩数据的能力. 如前所述，内存数

据库会对数据进行压缩之后再存储到内存之中,以提升空间利用率.令 S_{disk} 表示数据在磁盘上的空间容量,S_{mem} 表示数据在内存中的占用量,以下公式计算压缩率:

$$CR@SF = S_{mem}/S_{disk} * 100\% \tag{1}$$

• 每小时查询个数 $Qph@SF$.每小时查询个数描述评测任务的执行性能,执行时间包括:数据从磁盘文件装载到数据库的时间(T_L)、数据从磁盘加载到内存中的时间(T_S)、执行两次查询负载的时间(T_{QR1} 和 T_{QR2})和执行更新任务的时间(T_R),具体流程如图 3 所示:

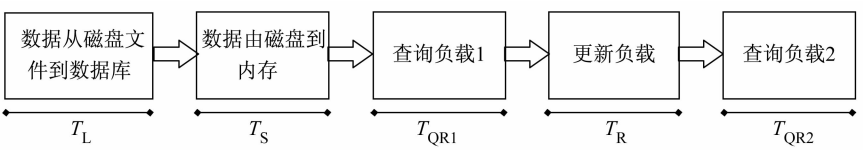

图 3 度量指标 $Qph@SF$ 的执行过程

Fig. 3 The execution of Metric $Qph@SF$

从图 3 中可以看出,整个流程包括 2 次查询负载,首次执行查询负载的目的是评测数据库在装载数据之后该系统执行查询的能力,再次执行查询负载的目的是评测数据库在执行完更新操作之后该系统执行查询的能力.为了提高可读性,T_L、T_S、T_{QR1}、T_{QR2} 和 T_R 的具体执行过程会放在第 4 节执行计划中介绍.

公式(2)定义了 $Qph@SF$ 的计算方法,其中 S 是执行测试计划的并发用户个数,共有 16 个查询任务(执行 2 次查询负载,每个查询负载种包括 8 个查询),0.01 是装载时间可并入总的消耗时间的比例,指的是进行一次该执行过程可算入在内的有效装载时间.该权重值较小的原因是数据装载仅在数据库启动时运行一次,而查询则会经常执行.因为上述过程都是以秒(s)为单位进行记时,因此需要在公式中乘以 3600 进行单位换算.文献[19-20]使用一个相似公式计算每小时的查询个数.

$$Qph@SF = 3600 \times \frac{16 \times S}{T_{QR1} + T_R + T_{QR2} + 0.01 \times S \times (T_L + T_S)} \tag{2}$$

• 列扩展性 $CS@SF$.这个指标描述内存数据库系统的列扩张能力.如前所述,内存数据库中存在 2 种数据组织方式:列存储和行存储.一般说来,当需要执行某个分析任务时,只有一张表中的少部分字段是需要处理的[17].$CS@SF$ 的设计目标就是测量某个内存数据库在字段个数不断变化情况下的性能.正如第 3 节数据模型介绍的那样,2 张大表 LINEITEM 和 ORDERS 的字段个数不是固定的,会随着数据库模式动态适应而不断改变.那么就记录了不同数据模式下数据库系统完成图 3 中的执行过程所需要的时间.公式(3)形式化地定义了 $CS@SF$.其中 T_1,…, T_5 分别表示不同模型下数据库的执行时间.具体的测试过程会在第 5 节执行计划中介绍.

$$CS@SF = \frac{T_1 + T_2 + \cdots + T_5}{5} \tag{3}$$

表 1 总结了 InMemBench 和其它现有评测基准的差异.和现有基准类似,InMemBench 也可评测数据库系统的常用特性,例如吞吐量和多用户模式.此外,新基准还能评测内存数据库的若干特性,而现有基准并未考虑到这些因素.

表 1　InMemBench 和其他评测基准的比较

Tab. 1　The comparison of InMemBench and other benchmarks

性质	SSB 测试基准	TPC-H	TPC-DS	CH-Benchmark	InMemBench
吞吐量	支持	支持	支持	支持	支持
多用户模式	支持	支持	支持	支持	支持
动态数据模型	不支持	不支持	不支持	不支持	支持
启动	不支持	不支持	不支持	不支持	支持
压缩率	不支持	不支持	不支持	不支持	支持
列处理能力	不支持	不支持	不支持	不支持	支持

4　工作负载和执行计划

本节首先描述了查询和更新负载，然后提出两个执行计划.

4.1　查询负载

查询负载共包括 3 组查询，每组分别包括 3、3 和 2 个查询. 以下详细描述各查询组.

• 查询组 1：主要是针对表 LINEITEM，有 sum、avg 和 count 等聚集函数. 根据过滤条件(*where* 子句)划分，共有 3 个查询，如查询组 1 所示，其中 DELTA1 和 DELTA2 是 2 个输入参数.

查询组 1 的查询

```
select l_returnflag, l_linestatus, sum(l_quantity) as
    sum_qty, sum(l_extendedprice) as sum_base_price, sum
    (l_extendedprice*(1-l_discount)) as sum_disc_price,
    sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as
    sum_charge, avg(l_quantity) as avg_qty, avg(
    l_extendedprice) as avg_price, avg(l_discount) as
    avg_disc, count(*) as count_order
from lineitem
where l_shipdate >= date 'DELTA1' and l_shipdate <=
    date 'DELTA2'
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

查询 1.1 DELTA1 = 1992-01-01 and DELTA2 = 1992-12-01

查询 1.2 DELTA1 = 1992-12-01 and DELTA2 = 1994-12-01

查询 1.3 DELTA1 = 1994-12-01 and DELTA2 = 1995-12-01

• 查询组 2：主要是针对 LINEITEM 和 ORDERS 2 张表，包括 3 个查询. 因为是对 2 张表进行连接操作，所以这个查询组消耗的内存空间比第一个查询组要大. 如查询组 2 所示，SHIPMODE1、SHIPMODE2 和 DELTA 是输入参数.

查询组 2 的查询

```
select l_shipmode, sum(case when o_orderpriority = '1-
    URGENT' or o_orderpriority = '2-HIGH' then 1 else 0
    end) as high_line_count, sum(case when
    o_orderpriority <> '1-URGENT' and o_orderpriority <>
     '2-HIGH' then 1 else 0 end) as low_line_count
from orders, lineitem
where o_orderkey = l_orderkey and l_shipmode in ('
    SHIPMODE1', 'SHIPMODE2') and l_commitdate <
    l_receiptdate and l_shipdate < l_commitdate and
    l_receiptdate >= date 'DELTA' and l_receiptdate < '
    DELTA' + interval '1' year
group by l_shipmode
order by l_shipmode;
```

查询 2.1 SHIPMODE1 = FOB, SHIPMODE2 = TRUCK and DELTA = 1996-12-01

查询 2.2 SHIPMODE1 = TRUCK, SHIPMODE2 = SHIP and DELTA = 1997-06-01

查询 2.3 SHIPMODE1 = SHIP, SHIPMODE2 = AIR and DELTA = 1997-12-01

• 查询组 3:对所有 8 张表进行连接操作,其内存消耗量最大. 该查询组包括两个查询,由参数 DELTA1 和 DELTA2 决定,如查询组 3 所示.

查询组 3 的查询

```
select n_name, sum(l_extendedprice * (1 - l_discount))
    as revenue
from customer, orders, lineitem, supplier, nation,
    region, partsupp, part
where c_custkey = o_custkey and l_orderkey = o_orderkey
     and l_suppkey = s_suppkey and c_nationkey =
    s_nationkey and s_nationkey = n_nationkey and
    n_regionkey = r_regionkey and s_suppkey = ps_suppkey
     and ps_partkey = p_partkey and o_orderdate >= date
    'DELTA1' and o_orderdate < date 'DELTA2'
group by n_name
order by revenue desc;
```

查询 3.1 DELTA1 = 1994-01-01 and DELTA2 = 1996-01-01

查询 3.2 DELTA1 = 1992-01-01 and DELTA2 = 1998-12-01

在上面查询组中,可以看到查询组 1 的每个查询获取的数据是没有交集的,然而查询组 2 和查询组 3 获取的数据是有重复的. 因此在执行查询组 1 时,缓存(caching)没有起到加速的作用. 当执行查询组 2 和查询组 3 时,缓存会重复利用前面的查询结果,于是会加速执行过程. 同时需要注意的是,查询组 2 中的每个查询获取的数据和查询组 1 是没有交集的,而查询组 3 和查询组 1 是有交集的,于是当查询组 3 放在查询组 1 后面执行的时候,缓存会加速执行过程. 实际上,3 个查询组按照数值生序顺序依次执行,缓存会在数据有重复的两个查询中加速执行过程.

4.2 更新负载

该负载主要用于图 3 中的更新负载执行过程,它主要在查询负载 1 和查询负载 2 之间执行. 它主要包括两个部分:插入操作和删除操作. 首先,测试系统会向表 LINEITEM 和 ORDERS 插入占原始记录为 0.1%的新记录. 随后,所有新增加的记录会被删除. 需要注意的是,所有的更新数据都是由数据生成器 COLGen 预先生成,数据记录大小和 SF 的比例是 1500∶1.

4.3 执行计划

本文前面提出了 4 个度量标准从不同的角度测试内存数据库系统的性能. 然而,这 4 个指标并不能在一次执行过程中全部完成测试. 因此设计了 2 个执行计划来完成这一目标. 第 1 个计划(计划 1)旨在测试启动消耗($SC@SF$),压缩率($CR@SF$)和每小时的查询个数($Qph@SF$),通过执行图 3 的过程,可以计算出上述的 3 个度量指标. 第 2 个测试计划(计划 2)来评估某个数据库的列扩展性($CS@SF$),这个测试需要不断改变数据库模式中的字段个数.

计划 1:测量启动消耗、压缩率和每小时查询个数

执行这个测试计划之前,需要预先设置 SF 的值来确定数据库的大小. 为了支持多用户的模式,S 的值也需要预先指定. 这个值表示查询流的数目,每个查询流表示一个用户. 整个

执行过程如图 3 所示，首先，启动数据库将测试数据从磁盘文件装载到数据库中，然后，数据库将所需的数据从磁盘装载到内存中，依次执行查询负载 1，更新负载和查询负载 2. 查询负载 1 和查询负载 2 是相同的，包括 8 条查询语句.

在上面的执行过程结束之后，可以记录 T_L、T_S、T_{QR1}、T_{QR2} 和 T_R. 接着，启动消耗 $SC@SF$ 和每小时查询数 $Qph@SF$ 可以直接计算得到. 为了测量压缩率，可以分别记录数据在磁盘和内存中的大小，然后用公式(1)计算求出.

计划 2：测量列扩展性

这个测试是一个迭代的过程，因为需要不断改变数据库模式中字段的个数. 通过改变 LINEITEM 和 ORDERS 这两张表新增字段的数目. 例如，在 LINEITEM(ORDERS)这张表中，用所有新增 200(100)个字段中的 20%、40%、60%、80%和 100%来构建每次迭代过程中的不同数据库模式. 对该测试计划来说，需要预先指定 SF 的值来确定数据的大小. 在每次迭代开始前，要重启数据库，执行图 3 中的每个过程，记录相应的时间 T_1，…，T_5，最后用公式(3)计算得出 $CS@SF$.

5　实　　验

这一节通过一系列的实验去评测内存数据库系统. 我们以数据库 X，Y，Z 和 MonetDB 作为实际例子. 内存数据库 X 和 MonetDB 支持列存储组织方式，内存数据库 Y 和 Z 支持行存储组织方式. 被测试平台有 1 TB 的内存容量和 8 个 CPU. 在默认情况下，每个内存数据库可以使用的最大内存容量为 0.8 TB.

(1) 压缩率实验

图 4(a)描述了随着 SF 的变化，4 个系统压缩能力的改变. 可以看到，系统 X 和 MonetDB 有更好的压缩能力，基本可压缩原始文件的 80%，对于系统 Y 和 Z，数据在压缩之前和压缩之后基本没有发生太大的变化. 这是因为 X 和 MonetDB 都是以列存储方式为主，这种存储方式可以把同一个字段的数据在物理上组织到一起.

(2) 启动消耗实验

图 4(b)描述启动消耗随 SF 增大而变化的趋势，从图中可以看到系统 X 和系统 Y 的启动消耗都是随着 SF 不断增大的. 然而，在同等数据量的情况下，系统 X 需要更少的启动消耗，这是因为系统 X 有着更好的压缩能力.

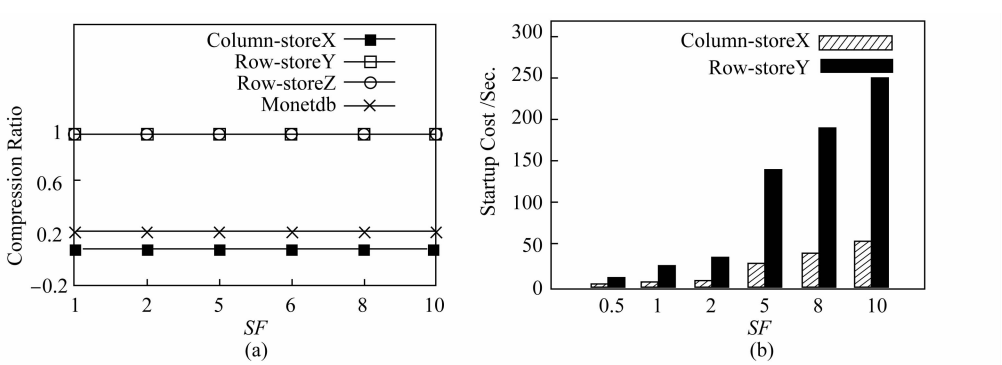

图 4　压缩率和启动消耗的评估

Fig. 4　Testing and evaluation for the compression ratio and startup cost

(3) 每小时查询数实验

图 5(a)描述了当 SF 为 5 时,每个查询和更新操作的执行时间. 在执行前 6 条查询的时候,列存储系统表现的性能更好,这是因为它只需要获取部分字段为将来的操作. 而且前 6 条查询主要是涉及到两张表,进行连接操作中间产生的数据量也比较少. 在执行后面 2 个查询时,行存储系统 Y 表现出了更好的性能,这是因为后面 2 个查询涉及到 8 张表的连接操作,中间会产生很多的数据,而且列存储系统需要执行更多额外的解压缩的操作. 对于更新操作来说,行存储比列存储的性能要好,这是因为记录本身是按照行存储的物理组织方式进行插入的. 图 5(b)显示了随着 SF 的增长,所有查询的时间也随之增长,使用公式(2)计算每小时的查询个数($Qph@SF$).

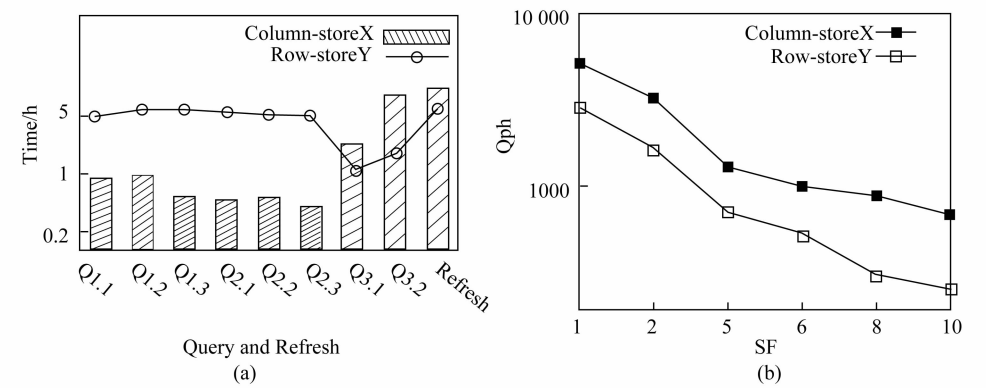

图 5 每小时查询数的评估

Fig. 5 Testing and evaluation for query per hour

(4) 列扩展性实验

在这一部分,图 6(a)描述了当 SF 设置为 5 时每个模型执行图 3 的过程所需要的时间(单位为 s),可以看到系统 X 随着模型的不断改变基本不发生太大的变化,然而系统 Y 却随着模型中字段个数的增加执行时间也不断变大. 这是因为列组织方式的数据库只需要查询将需要处理的字段,而行存储的组织方式则需要读取全部的字段. 随着列数的降低,系统 Y 的性能超过系统 X,这是因为在数据量相差不大的情况下,使用索引可以很好地加快行存储组织的数据库,对于列存储组织的数据库,索引一般没有任何作用. 随后,图 6(b)表明,系统 X 和系统 Y 随着 SF 的增加,整体执行时间也不断增大.

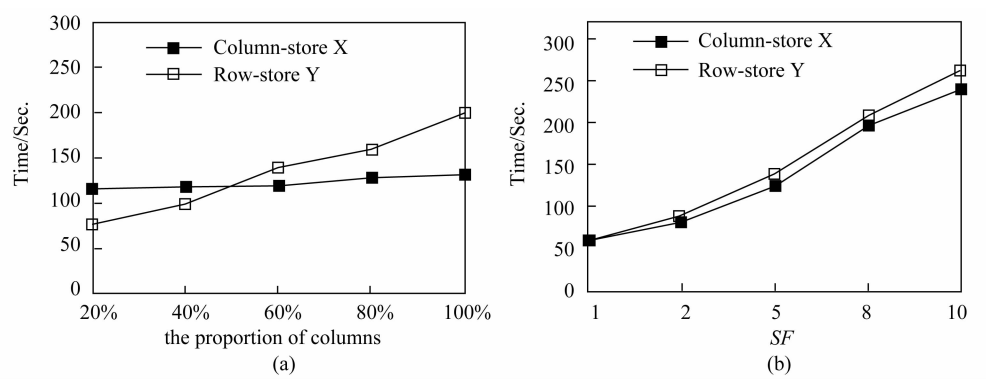

图 6 列扩展性的评估

Fig. 6 Testing and evaluation for column scalability

6 总　　结

本文提出了针对内存数据库的测试基准 InMemBench. 这个测试基准充分考虑了内存数据库的主要特性，与之相应地，提出了 4 个度量去客观公正地评估内存数据库的性能. 除此之外，该测试基准还包括自适应动态数据库模式、测试计划、查询和更新的工作负载. 最后，对 4 个内存数据库进行了大量的实验来验证所提测试基准的有效性和执行效率. 在未来的研究工作中，我们会设计更为复杂的查询以及对更多的内存数据库进行实验验证.

[参　考　文　献]

[1] FÄRBER F, CHA S K, PRIMSCH J, et al. SAP HANA database: data management for modern business applications[J]. SIGMOD Record, 2011,8: 45-51.

[2] lAHIRI T, NEIMAT M A, FOLKMAN S. Oracle TimesTen: an in-memory database for enterprise applications [J]. IEEE Data Eng Bull, 2013: 6-13.

[3] BONCZ P A, KERSTEN M L, MANEGOLD S. Breaking the memory wall in MonetDB[J]. Commun ACM, 2008,51(12): 77-85.

[4] COLE R, FUNKE F, GIAKOUMAKIS L, et al. The mixed workload CH-benCHmark[C]. DBTest, 2011,8:1-8:6.

[5] RABL T, POESS M, JACOBSEN H A, et al. Variations of the star schema benchmark to test the effects of data skew on query performance[C]. ICPE, 2013: 361-372.

[6] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems with YCSB[J]. SoCC, 2010: 143-154.

[7] CAREY M J, DEWITT D J, NAUGHTON J F, et al. The BUCKY object-relational benchmark (Experience Paper)[C]//SIGMOD Conference, 1997: 135-146.

[8] CAREY M J, DEWITT D J, NAUGHTON J F. The 007 Benchmark[C]//SIGMOD Conference, 1993: 12-21.

[9] SCHMIDT A, WAAS F, KERSTEN M, et al. XMark: A benchmark for XML data management[C]//Proceedings of the 28th international conference on Very Large Data Bases, 2002: 974-985.

[10] CAREY M J, LING L, NICOLA M, et al. EXRT: towards a simple benchmark for XML readiness testing[C]// TPCTC, 2010: 93-109.

[11] GHAZAL A, RABL T, HU M, et al. BigBench: towards an industry standard benchmark for big data analytics [C]//SIGMOD Conference, 2013: 1197-1208.

[12] PATRICK E, NEIL O. A set query benchmark for large databases[C]//Int CMG Conference, 1989: 710-721.

[13] LIU D, LUAN H, WANG S, et al. Main memory database TPC-H workload characterization on modern process [J]. Journal of Software, 2008,19(10): 2573-2584.

[15] TÖZÜN P, PANDIS I, KAYNAK C, et al. From A to E: analyzing TPC's OLTP benchmarks: the obsolete, the ubiquitous, the unexplored[C]//EDBT, 2013: 17-28.

[16] PLATTNER H. A common database approach for OLTP and OLAP using an in-memory column database[C]// SIGMOD Conference, 2009: 1-2.

[17] ABADI D J, MADDEN S, HACHEM N. Column-stores vs. row-stores: how different are they really[C]//SIGMOD Conference, 2008: 967-980.

[18] GRAY J. Benchmark Handbook: For Database and Transaction Processing Systems[J]. San Francisco: Morgan Kaufmann Publishers Inc, 1992.

[19] NAMBIAR R O, POESS M. The making of TPC-DS[C]//VLDB, 2006: 1049-1058.

[20] POESS M, NAMBIAR R O, WALRATH D. Why you should run TPC-DS: a workload analysis[C]//VLDB, 2007: 1138-1149.

(责任编辑　赵　伟)