数据库技术和系统测试简介 **V0.9**

嘉迪正信（北京）管理资讯有限公司

配置管理文档编号 6003-RPT1001

夏锋（嘉迪）

feng.xia@jadetrust.com.cn

Abstract

为配合晋能集团利用云计算技术建设可扩展的企业平台的构想，我们就云环境的数据库技术和集团如何测试一个系统做了介绍:

1. 关系型数据库和非关系型数据库
2. 数据库的分布式存储
3. 系统压力测试

我们希望通过这个报告切实的协助晋能集团客观、准确的考量建设和利用云计算企业平台的构想，并为未来更加细化的研究和探讨打下一个坚实的基础。

Contents

List of Figures

1 数据库技术趋势

1.1 数据库的基本概念

数据库是当前任何一个“系统”的核心，因为系统最终需要存储数据，而数据与数据之间的关系也是用户和系统最关心的。在我们进一步讨论数据库的技术之前，先来了解一下几个基本的数据库概念:

RDBMS Relational Database Management System，也就是我们平时常说的关系型数据库。这类数据库的数据模型以列代表数据属性，每一行代表一条完整的数据，因此也称为“以行存储的数据库“。目前的数据库主流依然是关系型的 RDBMS，商业产品中的甲骨文的 Oracle 11g，微软的 SqlServer, IBM 的 DB2，Sybase 的 Sybase，开源的 MySQL, PostgreSql, Sqlite 等等。

SQL Structured Query Language，结构化查询语言，主要就是针对关系型数据使用的，包

括 6 部分：数据查询语言、数据操作语言、事务处理语言、数据控制语言、数据定义语言、指针控制语言。

NoSQL 其实是对非关系型数据库的泛指，即数据为非关系型、查询也不用 SQL。这类数据库由于其设计的目的是高拓展性，所以更适合被称为“分布式数据库”。

ACID 特性 原子性 (Atomicity)、一致性 (Consistency)、隔离性 (Isolation)、持久性 (Durability)。

- 原子性 ：要求每一个操作都是“全部成功或根本没有发生”。如果一个操作的一部分失败了，那么全部操作都被认为失败，数据库的状态也应该没有发生改变，如同“什么都没有发生过一样”。因此对外界来说，对数据库的某一个操作如同是一个不可分割的原子，一旦开始了要么成功要么失败，也不可能中途取消。
- 一致性 ：指任何一个成功的操作都将改变数据库的状态。任何状态的改变都保证满足所有定义的规则，包括 constraints 限制，cascades 级联, triggers 触发等。数据库状态的改变并不保证数据的“正确”，因为是否“正确”是需要在应用层判断的。
- 隔离性 ：也就是常说的并发隔离，即并发的操作对数据库的影响和把这些并发操作串行执行的效果是相同的。
- 持久性 ：指一个操作在被 commit 以后，无论是发生断电、系统崩溃或其他错误，其改变的数据库状态都已经被永久保存。当然，数据库的存储介质的错误依然可能引起数据丢失。

对于单个节点的事务，数据库都是通过并发控制（两阶段封锁，two phase locking 或者多版本，multiversioning）和恢复机制（日志技术）保证事务的 ACID 特性。对于跨多个节点的分布式事务，通过两阶段提交协议（two phase commiting）来保证事务的 ACID。

CAP 定理 一致性（Consistency)、可用性 (Availability)、分区容错性 (Partition Tolerance)，三者只能同时满足两个：

- 一致性 ：指数据被写入数据库后，无论那个节点读取的都是最新的数据。如果不能保证这一点，则成为弱一致性。
- 可用性 ：指每一个请求都保证会有一个回应，无论这个请求是成功的还是不成功。

分区容错性 ：指系统作为一个整体在部分系统出现故障和消息丢失的情况下依然可以正常工作，如部分服务器节点的网络断网。这个通常是通过各种形式的冗余来实现的。

BASE 说法 指系统满足以下三个条件 – 基本可用（Basically Available）, 软状态/柔性事务，即状态可以有一段时间的不同步（Soft state）, 和最终一致性（Eventual consistency）。

1.2 数据库的两种拓展方法

在数据（库）层有两个办法可以拓展：纵向 (Scale Up) 和横向（Scale Out)。第一种也是最常见、最简单的一种，就是把数据库换到更好的硬件上去 – 更快的服务器、更快的存储。这是完全可行的，但可以预见的，随着硬件的成本越来越高，回报率却在逐渐减小，最终将出现得不偿失的瓶颈。对一般的企业系统来说，其实这个问题并不重要，因为数据的增长和用户数的压力都增长缓慢，升级一次也够用一阵的了。但另一个问题是高端硬件的厂家数量有限，走这条路使得企业对某个厂家的依赖性越来越强，有被绑架的嫌疑。

另一种横向的拓展方案通过增大硬件的数量和一些软件的帮助实现总体性能的提升。这个方法同样也存在问题，首先是数量的增加和性能往往没有线性关系，所以投资的增加最终依然会有得不偿失的瓶颈，但由于这个瓶颈之前的容量是如此之大，大多数的应用/企业是不需要为这个担心的。那第二个问题就是这个办法增加了复杂性，无论在数据库本身的设计，还是部署后的管理。

横向的拓展可以再细分为两个层面：按功能分区和按数据分区。按功能分区指在设计数据库表的时候，将库表按其功能分开，如用户表、产品表、交易表、统计表等等等等。然后需要时将不同功能的表部署到不同的数据库服务器上，然后通过在应用层分别访问所需要的库。在功能分区的基础上，还可以将表中的数据按某种规则分区（sharding，碎片），如一个交易表内已经有 10 年的数据，可以将没两年的数据作为一个分区划分出来，使得每个数据库里只存了不超过 2 年的数据。数据分区可以是人工的，也可以通过事先定义的某种规则由数据库自动完成。

当一个数据库被分成多个的时候，如何保证 ACID 呢？数据库厂商们早想到了这个问题，于是出现了 2PC – 两段提交 (Two-Phase Commit)。2PC 其实原理很简单，当需要更改数据库状态时，第一步先由一个操作协调员（往往是一个指定的管理服务器，有时是一个指定的数据库节点）询问所有数据库节点是否操作可行。如果所有节点都表示可以，则进入第二步由管理员要求所有节点都执行操作。如果任一个节点操作失败，则所有节点都被要求回滚 (rollback)。

2PC 既然可以在分区容错性（数据现在可以多点拓展了）的基础上还保证了一致性，那么根据 CAP 定理，必然牺牲了可用性（A）。那让我们来看看可用性是如何下降的呢？由于所有数据库的操作是有原子性的，所以整个部署（包括所有的节点）的可用性是所有节点数据库的可用性的乘积。如果每个数据库都有 99.9% 的可用性的话，那么一个有两个数据库的分布式部署就有 $0.999 * 0.999 = 99.8\%$ 的可用性，即每个月可能多 43 分钟的下线时间，而 5 个数据库的话则有 3.6 个小时的下线时间，CAP 真的是不能兼得。

说到这儿，既然关系型数据库也能分表、也能分布式部署，虽然损失了 A，但也不失为一个保留 ACID 情况下的方案，那么为什么现在出现了这么多 NoSQL 数据库呢？NoSQL 也可以实现 ACID 吗？

1.3 数据库的 SQL vs. NoSQL

大家都耳熟能详的 NoSQL，最容易的理解是“不用 SQL”，抛弃了 SQL 也就意味着抛弃了后台的关系型数据库，但现在更愿意说它们是 Not-Only SQL“不只是 SQL”，因为等会儿我们就会看到，有些非关系型数据库为了兼容性（我认为主要是为了照顾现有应用的迁移和这么多年编程者形成的知识习惯）也提供了类似于 SQL 的能力，可以使用传统的 SQL 查询一个非关系型数据库。NoSQL 的出现有它的时代背景。首先，云计算使部署和存储的成本都大大下降，但前提是数据可以在多个节点分散。对一个依赖于关系型数据库开发的应用来说，很多数据的读取都依赖于多个表的 join，而如何在数据存储分散的情况下实现 join 对数据库是个技术挑战。其次，非结构化的数据大量出现，关系型数据库对这类数据并不擅长。另外，关系型数据库的 schema 由于需要事先定义，和应用层的数据模型耦合紧密，使得改变不能

分层隔离，降低了系统的维护能力。

目前几个流行的 NoSQL 有 HBase, Cassandra, Redis, MongoDB, Voldemort, CouchDB, Dynomite, Hypertable。根据统计有近 150 种 NoSQL 数据库[1]，没想到吧。大多数非关系型数据库都采用了"key-value" 对，即每一条数据都有一个 key（钥匙，但我觉得更合适称作一个身份认证，如同关系型数据库里中一条数据的主键 PK。但主键只在当前表内是唯一的，而这里的 key 在整个库内都是惟一的 ◦)，当用户需要读取这条数据时，只需要给数据库这个身份认证即可。这种简单的设计使开发和横向拓展（无论是在存储还是用户处理的分流上）都变得更加的简单。在 key-value 的基础上出现了文本（或者也叫文件）数据库 (document DB)。一个文本是一些信息的集合，RDBMS 里一个表也是一些数据单元的集合，但每个单元的属性、长度都是固定的；而一个文本内的数据属性、长度都可以不同，对非结构化的数据特别适用，有些甚至可以支持直接存储编程语言中的对象，当然再把对象读回来在内存中还原成一个实例！

对于习惯了关系型数据库的人来说，NoSQL 是个不靠谱的东西，因为它给你的只是一个“意见”，或者说“目前的最佳答案”，而 ACID 给你的永远是个禁得起考验的确切答案。在 ACID 的数据库占上风的年代，无疑任何不满足 ACID 的东西都不配称自己是个数据库。ACID 有它的应用场景，但很多时候它也带来了不必要的开销。很多数据库的起源是在 70、80 年代为单一数据库服务器设计的，所以 P 根本就不需要考虑。一个服务器要么就是在工作，要么就是下线了，不会出现“部分服务器在工作”的情况吧。所以之前的 RDBMS 都侧重在 C 和 A，也就是现在所说的 ACID。而其中的可用性 A 也只是通过多个数据库之间的镜像、或者复制技术等来实现的。

NoSQL 也受到 CAP 的限制，因此也得选择它们的侧重点。由于在离散部署的情况下很难保证 C，它们索性不再纠结一致性的问题，而提出了“最终一致”的概念，也就是在节点之间会不断传播各个节点的状态以保持一致性，但在任意一个单个时间点上节点的状态可以不一致。那么这个“最终”的时间有多长呢？谁也不知道。那么什么样的应用场景适合这样的技术呢？当我们在 ATM 机取出现金时，我们需要立刻知道帐号上已经更新了，这时，我们需

[1] http://nosql-database.org/

要 ACID；而如果我们淘宝时想知道一个宝贝有多少卖家在卖，我们只需要足够的卖家可以选择，而不是确切的所有卖家（不仅统计这样的数字是个不小的工程，只需要想一想淘宝的数据量；而且，这对用户来说没有什么意义。），所以在电商平台上 NoSQL 比 RDBMS 更适合。NoSQL 其实和 ACID 并不是天敌，都只是在 CAP 中只能三选二，所以取舍不同罢了。

RDBMS 并不过时，它们不仅技术成熟，而且已经在各个场景中证明了自己的能力。如果设计和使用正确的话，RDBMS 的效率一点儿也差，除非一些极限的情况外足够用了。我来引用下面这段话结束这个小节吧：

> Non-SQL gives you a very sharp knife to solve a selected set of issues. If you find SQL too hard to use, you should not try Non-SQL.
>
> NoSQL 对某一类问题为你提供了一把非常锋利的刀。但如果你发现 SQL 很难使用的话，你也最好别尝试 NoSQL。
>
> — Monty Widenius, MySQL 的创始人

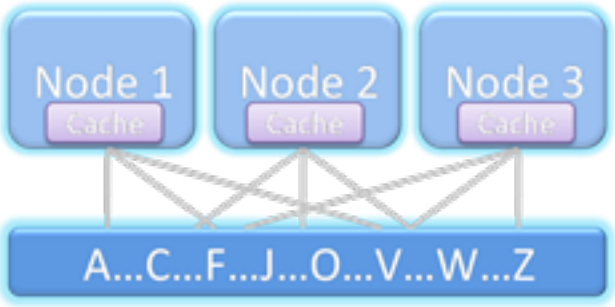
1.4 Shared-Everything vs. Shared-Nothing

数据库模型的另一个重要趋势是在拓展架构中采用那个原则：共享一切（shared everything）还是什么都不共享 (shared nothing)。无论是哪一个，都是希望通过某种机制实现平行处理。共享一切的数据库又可以再细分为共享内存和共享存储两种。共享内存的例子我们最熟悉不过 – 每个人的电脑，无论有多少个 CPU，都共享一组内存和同一组磁盘。每一个 CPU 可以访问所有的内存地址并通过内存传递数据。由于需要通过一个总线才能访问，所以多个 CPU 将通过某种锁机制保证数据的完整，随着 CPU 数目的增加，总线的 I/O 将成为瓶颈。

共享存储的典型代表是 Oracle RAC。在共享存储的架构中，每一个独立的处理节点都有自己的内存，这些节点共用一组存储磁盘（SAN，NAS）。首先，CPU 和磁盘之间的网络联接是个 I/O 瓶颈（当然，高速的光缆通道在这一点有所帮助，但这也决定了存储池不能太大，也不能在地理上太远）。其次，由于没有一个明显的共享空间，在哪里实现锁表和缓冲池是个

问题。如果需要锁，要么用一个中心的锁管理处理器，要么实现一个复杂的分布式锁协议。不管怎样，这都很容易成为瓶颈。

Figure 1.1: Shared-Disk 共享存储机制的示意图

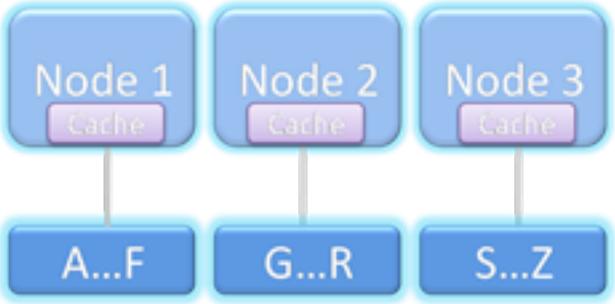


在共享存储的基础上，厂商们一般采用了“分享缓存“(shared-cache) 的方法，也就是在每个节点设计一个缓存区。当一个节点需要读取数据时：

1. 首先在自己的缓存区搜索需要的数据
2. 如果没有，在任一个其它节点的缓存区里寻找
3. 如果还是没有找到，则从磁盘存储中读取

这个机制的问题显而易见。在缓存区里找到想要的数据机率并不大，这取决于应用的类型，OLTP 大多数时候使用的数据范围比 OLAP 的要小，并且重复使用同一组数据的机会也多，所以缓存机制更有效。而 OLAP（例如 BI）做多个大数据表的 join，就数据量而言缓存是远远不够的。

Figure 1.2: Shared-Nothing 什么都不共享机制的示意图



什么都不共享的架构中，每一个处理器节点是独立的一个，有自己的 CPU、内存、存储。数据被水平分割成多组，即一个表中的行被以某种规则分成几份，分别存储在不同的节点上。每个节点由于只负责它自己的数据，所以不需要共享锁机制和缓冲区，这大大简化了系统的瓶颈。节点越多，则系统的数据能力越强，理论上讲两者是线性关系，即 100 个节点的系统在同样条件下是 50 个节点的系统的两倍。由于这个结构靠数量占优，所以常见的是部署在便宜的低配硬件上，听说 Google 的搜索引擎就采用了这个结构，有几十万个节点，每个节点的造价只有 700 美元[2]。这个架构的一个代表就是我们在智慧供应链平台项目中准备采用的 MySQL Cluster。

1.4.1 Oracle RAC

Oracle RAC 就采用了共享存储架构（Shared-disk），整个 RAC 集群是建立在一个共享的存储设备之上的，节点之间采用高速网络互连。在一个应用环境当中，所有的服务器使用和管理同一个数据库，目的是为了分散每一台服务器的工作量，硬件上至少需要两台以上的服务器，而且还需要一个共享存储设备。同时还需要两类软件，一个是集群软件，另外一个就是 Oracle 数据库中的 RAC 组件。同时所有服务器上的 OS 都应该是同一类 OS, 根据负载均衡的配置策略，当一个客户端发送请求到某一台服务的 listener 后，这台服务器根据负载均衡策略，会把请求发送给本机的 RAC 组件处理也可能会发送给另外一台服务器的 RAC 组件处理，处理完请求后，RAC 会通过集群软件来访问我们的共享存储设备. 逻辑结构上看，每一个参加集群的节点有一个独立的 instance，这些 instance 访问同一个数据库。节点之间通过集群软件的通讯层（communication layer）来进行通讯。同时为了减少 IO 的消耗，存在了一个全局缓存服务，因此每一个数据库的 instance，都保留了一份相同的数据库 cache。

Oracle RAC 提供了非常好的高可用特性，比如负载均衡和应用透明切换 (Transparent Application Failover, TAF)，其最大优势在于对应用完全透明，应用无需修改便可以切换到 RAC 集群。但是，RAC 的扩展能力有限，首先因为整个集群都依赖于底层的共享存储，所以共享存储的 IO 能力和可用性决定了整个集群的可以提供的能力，其依然无法摆脱对大型存储设备的依赖。Oracle 显然也意识到了这个问题，在 Oracle 的 MAA(Maximum Availability Architecture) 架构中，采用 ASM 来整合多个存储设备的能力，使得 RAC 底层的共享存储

[2] http://db.csail.mit.edu/madden/high_perf.pdf

也具备线性扩展的能力，整个集群不再依赖于大型存储的处理能力和可用性。

RAC 的另外一个问题是，随着节点数的不断增加，节点间通信的成本也会随之增加，当到达某个限度时，增加节点可能不会再带来性能上的提高，甚至可能造成性能下降。这个问题的主要原因是 Oracle RAC 对应用透明，应用可以连接集群中的任意节点进行处理，当不同节点上的应用争用资源时，RAC 节点间的通信开销会严重影响集群的处理能力。所以使用 Oracle RAC 有两个建议：1. 节点间通信使用高速互联网络；2. 尽可能将不同的应用分布在不同的节点上。基于这个原因，Oracle RAC 通常在 DSS 环境中可以做到很好的扩展性，因为决策支持系统（Decision Support System, DSS) 环境很容易将不同的任务分布在不同的计算节点上，而对于 OLTP 应用，Oracle RAC 更多情况下是用来提高可用性，而不是为了提高扩展性。

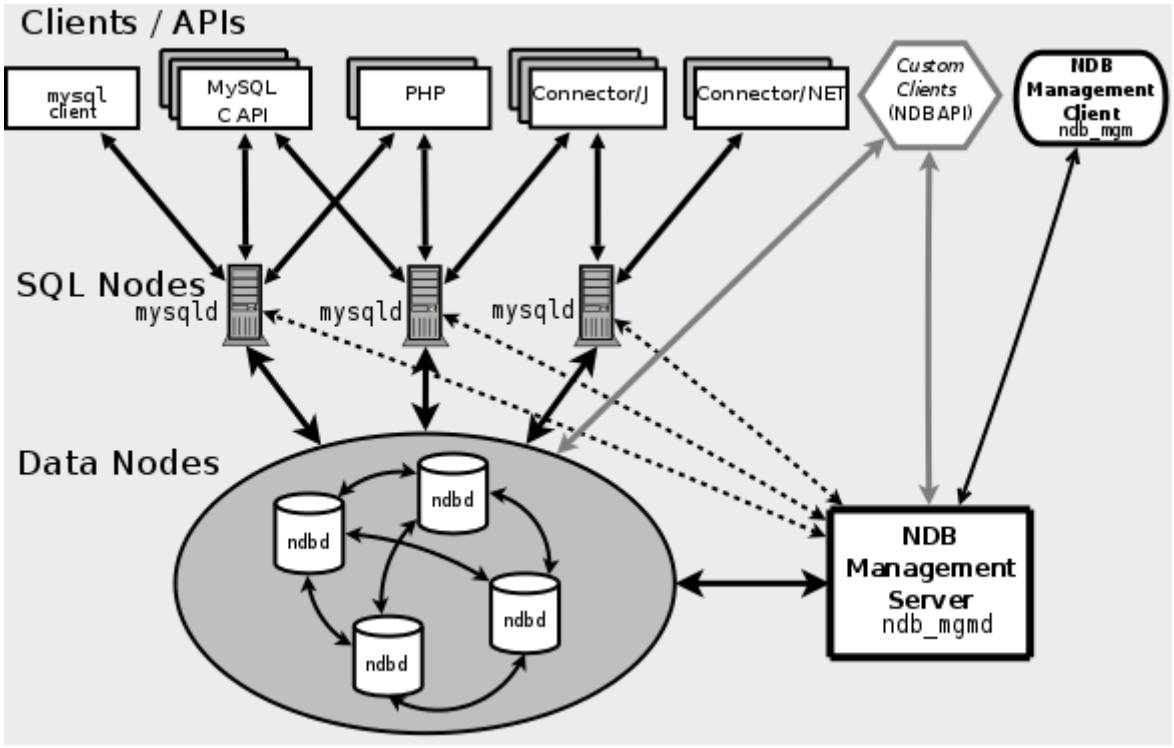
1.4.2 MySQL Cluster

MySQL cluster 和 Oracle RAC 完全不同，它采用 Shared-nothing 架构。MySQL Cluster 由一组计算机构成，每台计算机上均运行着多种进程，包括 MySQL(API) 服务器，NDB 的数据节点，MGM 管理服务器，以及数据访问程序。所有的这些节点构成一个完成的 MySQL 集群体系。数据保存在“NDB 存储服务器”的存储引擎中，表（结构）则保存在“MySQL(API) 服务器”中。应用程序通过“MySQL(API) 服务器”访问这些数据表，集群管理服务器通过管理工具 (ndb_mgmd) 来管理“NDB 存储服务器”。

一个 Mysql Cluster 的环境主要由以下三部分组成:

1. 负责管理各个节点的 Manage 节点主机 ：管理节点负责整个 Cluster 集群中各个节点的管理工作，包括集群的配置，启动关闭各节点，以及实施数据的备份恢复等。管理节点会获取整个 Cluster 环境中各节点的状态和错误信息，并且将各 Cluster 集群中各个节点的信息反馈给整个集群中其他的所有节点。由于管理节点上保存在整个 Cluster 环境的配置，同时担任了集群中各节点的基本沟通工作，所以他必须是最先被启动的节点。
2. SQL 层的 SQL 服务器节点（后面简称为 SQL 节点） ，也就是我们常说的 Mysql

Figure 1.3: MySQL Cluster 架构图



Server：主要负责实现一个数据库在存储层之上的所有事情，比如连接管理，query 优化和响应，cache 管理等等，只有存储层的工作交给了 NDB 数据节点去处理了。也就是说，在纯粹的 Mysql Cluster 环境中的 SQL 节点，可以被认为是一个不需要提供任何存储引擎的 Mysql 服务器，因为他的存储引擎有 Cluster 环境中的 NDB 节点来担任。它们与标准的 MySQL 没有区别。能够从用 PHP、Perl、C、C++、Java、Python、Ruby 等编写的现有 MySQL 应用程序直接访问。

3. Storage 层的 NDB 数据节点，也就是上面说的 NDB Cluster ：NDB 是一个内存式存储引擎也就是说，它会将所有的数据和索引数据都 load 到内存中，但也会将数据持久化到存储设备上。不过，最新版本已经支持用户自己选择数据可以不全部 Load 到内存中了，当然，从磁盘上读取数据的速度要比从内存慢的多。数据节点的数目与副本的数目相关，是片段的倍数。例如，对于两个副本，每个副本有两个片段，那么就有 4 个数据节点。

2

系统的性能测试

软件系统的测试方法有很多种，常见的白盒测试、黑盒测试、单元测试。对基于 Web 的应用我们常使用的是压力测试。

2.1 压力测试

系统的压力测试其实包括四种:

1. 压力测试 (load test) ：模拟系统的实际使用情况，如用户并发数、峰值时段。测试的目的是了解系统对设计的使用场景会如何应对，如用户响应时间、系统使用效率。
2. 强度测试 (stress test) ：这是压力测试的极端情况，通过不断升高用户并发数（即系统内使用多线程模拟多用户并发）对系统加压直至其性能出现拐点，这个拐点即代表了设计的系统的最大负荷。测试的目的除了找到拐点外，还可以测试出系统的瓶颈，如

系统是 CPU bound（即运算能力是瓶颈）还是 I/O bound（即数据读取是瓶颈）。针对不同类型的瓶颈也就可以设计不同的优化策略了。

3. 浸泡测试 (soak test) ：这也是压力测试的一个变种，即对系统长时间加接近峰值的恒定压力。测试的目的在于发现系统中不易被察觉的性能隐患，如 web 服务器的内存泄漏会在长时间的测试中最终导致系统内存用量不断升高直至系统无法响应。

4. 尖峰测试 (spike test) ：也是压力测试的一种，即突然增加或减少系统的使用压力（压力一般都超过最大峰值）。测试的目的在于发现系统对突发情况下的反应，例如模拟多个负载均衡的数据库节点突然下线，于是其他的节点的压力将随之突然增加。

压力测试常用工具有惠普的 LoadRunner 和开源的 Apache jMeter。它们的工作原理类似，通过记录然后回放对一个系统的实际操作，然后加入多线程模拟系统压力和真实的系统使用情况。在测试过程中不断收集系统数据，如 CPU 使用情况等，然后给出性能报告。工具本身还支持直接调用编写的程序或脚本进行测试，所以十分灵活。

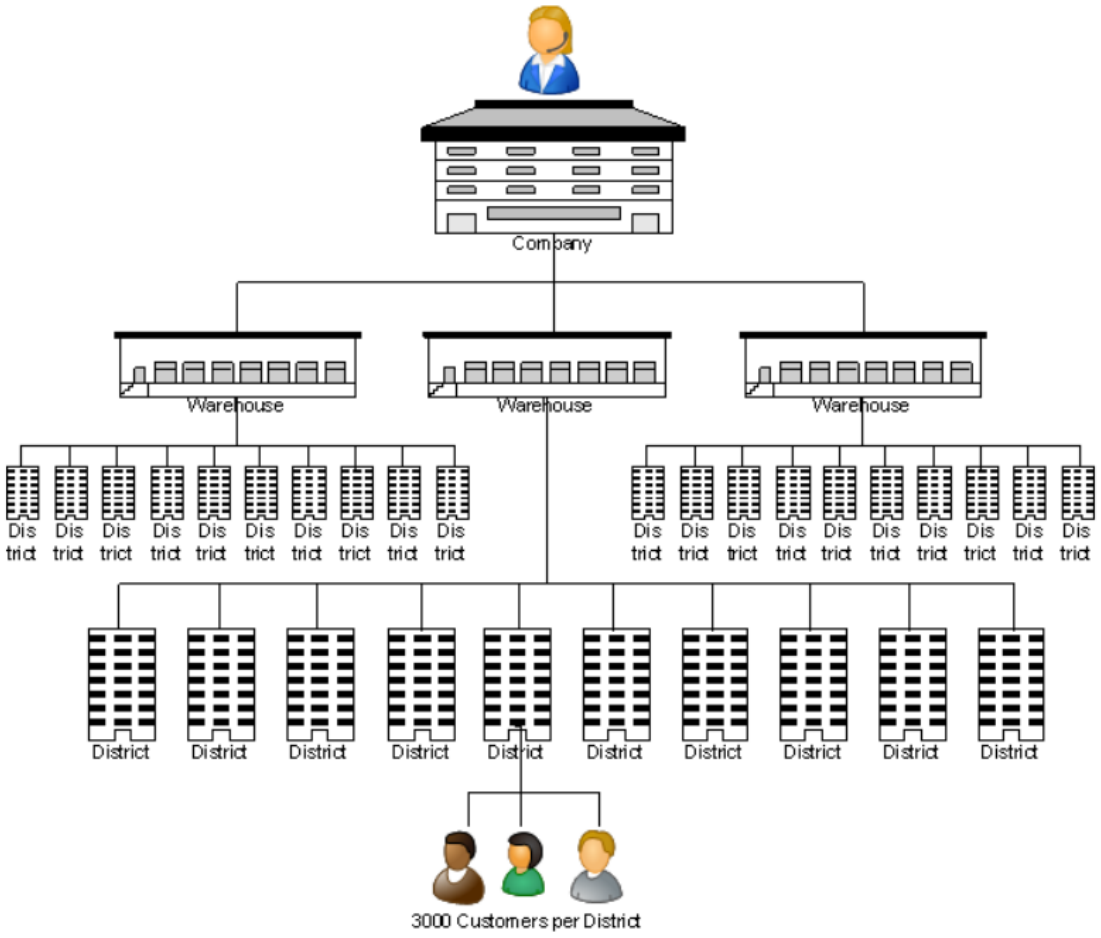
2.2 TPC-C Model

TPC-C 是专门针对联机交易处理系统（OLTP 系统）的规范，一般情况下我们也把这类系统称为业务处理系统。TPC-C 测试用到的模型是一个大型的商品批发销售公司，它拥有若干个分布在不同区域的商品仓库。当业务扩展的时候，公司将添加新的仓库。每个仓库负责为 10 个销售点供货，其中每个销售点为 3000 个客户提供服务，每个客户提交的订单中，平均每个订单有 10 项产品，所有订单中约 10% 的产品在其直接所属的仓库中没有存货，必须由其他区域的仓库来供货。同时，每个仓库都要维护公司销售的 10 万种商品的库存记录 (见图2.1)。

对每一个地区的销售人员来说，他可以在任何时间选择执行以下的五个业务场景。为了模拟真实的业务，TPC-C 还对每一种操作分配了侧重比例：

1. 新订单（New-Order）：客户输入一笔新的订货交易，45%；
2. 支付操作（Payment）：更新客户帐户余额以反映其支付状况，43%；
3. 发货（Delivery）：发货（模拟批处理交易），4%；

Figure 2.1: TPC-C 公司组织结构



4. 订单状态查询（Order-Status）：查询客户最近交易的状态，4%；

5. 库存状态查询（Stock-Level）：查询仓库库存状况，以便能够及时补货，4%。

TPC-C 的指标是业务流量 (Tranaction per Minute, tpm-C)，即每分钟实现的新订单数。如果把其他的事务都想象成系统/业务内必不可少的噪音的话，那新订单数无疑代表了一个公司有意义的贸易量。这个指标越高，则代表系统的处理能力越强。

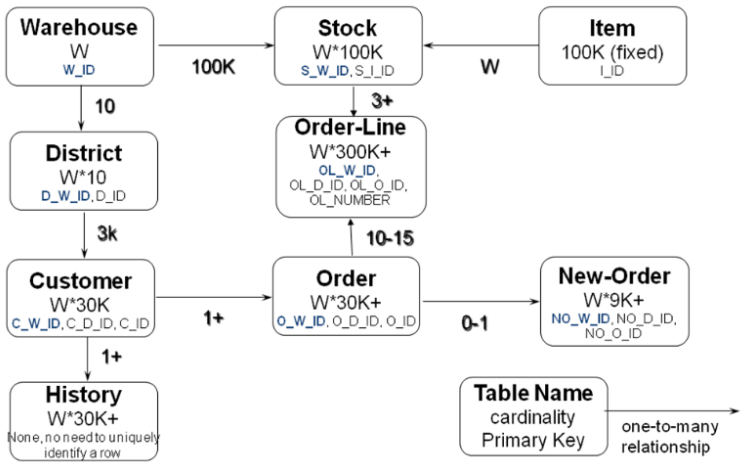
Figure 2.2: TPC-C 公布的单机性能排名前 10 位的服务器

Rank	Company	System	Performance (tpmC)	Price/tpmC	Watts/KtpmC	System Availability	Database	Operating System	TP Monitor	Date Submitted	Cluster
1	ORACLE	SPARC T5-8 Server	8,552,523	.55 USD	NR	09/25/13	Oracle 11g Release 2 Enterprise Edition with Oracle Partitioning	Oracle Solaris 11.1	Oracle Tuxedo CFSR	03/26/13	N
2	ORACLE	Sun Server X2-8	5,055,888	.89 USD	NR	07/10/12	Oracle Database 11g R2 Enterprise Edition w/Partitioning	Oracle Linux w/Unbreakable Enterprise Kernel R2	Tuxedo CFS-R	03/27/12	N
3		IBM System x3850 X5	3,014,684	.59 USD	NR	09/22/11	IBM DB2 ESE 9.7	SUSE Linux Enterprise Server 11 SP1 for X86_64	Microsoft COM+	07/11/11	N
4	CISCO	Cisco UCS C240 M3 Rack Server	1,609,186	.47 USD	NR	09/27/12	Oracle Database 11g Standard Edition One	Oracle Linux w/Unbreakable Enterprise Kernel R2	Microsoft COM+	09/27/12	N
5		IBM Flex System x240	1,503,544	.53 USD	NR	08/16/12	IBM DB2 ESE 9.7	Red Hat Enterprise Linux 6.2	Microsoft COM+	04/11/12	N
6		IBM System x3650 M4	1,320,082	.51 USD	NR	02/25/13	IBM DB2 ESE 9.7	Red Hat Enterprise Linux 6.4 with KVM	Microsoft COM+	02/22/13	N
7	HP	HP ProLiant Blade BL685c G7	1,263,599	.51 USD	NR	05/23/11	Microsoft SQL Server 2005 Enterprise Edition x64 SP3	Microsoft Windows Server 2008 R2 Enterprise Edition x64	Microsoft COM+	05/23/11	N
8	CISCO	Cisco UCS C250 M2 Extended-Memory Server	1,053,100	.58 USD	NR	12/07/11	Oracle Database 11g Release 2 Standard Ed One	Oracle Linux w/Unbreakable Enterprise Kernel R2	Microsoft COM+	12/07/11	N
9	HP	HP ProLiant DL380 G7	1,024,380	.65 USD	NR	06/20/11	Microsoft SQL Server 2005 Enterprise Edition x64 SP3	Microsoft Windows Server 2008 R2 Enterprise Edition	Microsoft COM+	05/04/11	N

2.2.1 HammerDB

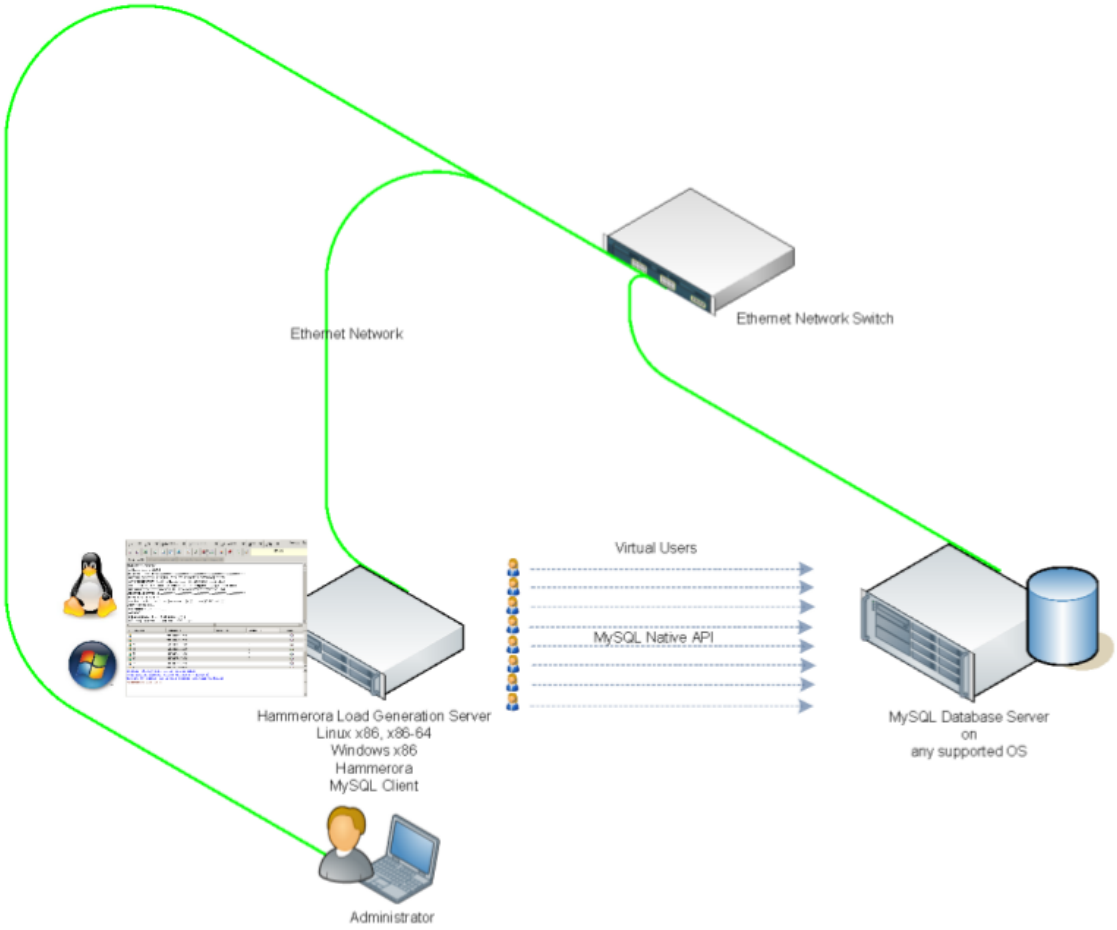
TPC-C 的测试工具有很多，这里我介绍一个开源软件 HammerDB。HammerDB 首先实现了 TPC-C 的数据模型（见图2.3）：

Figure 2.3: HammerDB TPC-C 的数据库模型



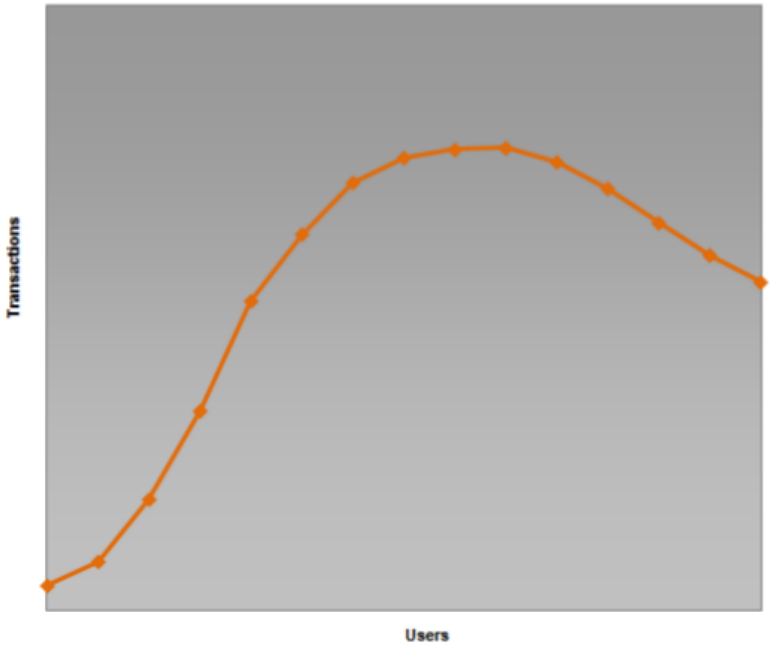
在此基础上，HammerDB 需要至少 3 个服务器节点：一个是测试对象的服务器，一个是模拟用户（即压力）的服务器，还有一个管理服务器收集测试对象的情况。把三个分开是为了减少互相的影响，尤其是对测试对象，以增加测试结果的准确性。

Figure 2.4: HammerDB 的网络拓扑图



在 MySQL 环境下，设置 25 个商品仓库大约等同于 2.5G 的数据量，即一个商品仓库约 100MB。HammerDB 支持设置最大 5000 个仓库，也就是说将模拟最大 500G 的数据库。用户数没有限制，主要依据底层的硬件能支持多少线程，应用本身需要 10MB 内存，每一个模拟的用户需要 2MB 内存。测试可以逐步增加用户数，每一次 HammerDB 都会计算出 NOPM，即每分钟的新订单数。当压力最终引发瓶颈，系统的性能将随着压力的增加而下降。细节这里略过，最终工具给出一个系统的压力曲线（见2.5)。

Figure 2.5: HammerDB 系统压力曲线



HammerDB 还可以自动完成以上的曲线压力, 用户只需制定一个用户数序列, 如”5,11,31,41”, HammerDB 就会按照设定的用户数执行，每次记录 TPC-C 值。

3 总结

云计算和云平台是多种计算机技术的大规模集成。现代 IT 系统最核心的是数据，而数据的存储依赖于数据库和底层的存储技术。云平台的建设和选型，尤其是在 IaaS 层主要考虑的将是这两层的技术方案。本文希望通过对一些基础知识的总结帮助集团和相关人员了解现代的数据库技术和目前正在流行的趋势，在评审技术架构和供应商的过程中有更多的依据。

无论平台的建设以何种方式进行，最终对集团都存在一个如何评估系统的问题。本文介绍了最常用的压力测试方法和一个开源的测试工具，旨在帮助读者熟悉压力测试的几种类型，和初步了解压力测试可以起到的作用。这里用 TPC-C 作为实例，行业还有其他的测试标准，如 Specweb 等，也同样可以使用。无论采用哪种衡量标准，关键在于理解测试背后的设计理念和指标的意义，选择一个适合集团应用场景的工具。