分 布式**key-value** 键值数据库与关系数据库**-NoSql**

[2010-11-21 19:03:00 | By: 梦翔儿]

关系数据库的死期到 了?

最近,大量新的非关系式数据库如雨后春笋般出现在云里云外。这其中所释放出的一个关键信息是:“如果想获得丰富而 随需应变的可伸缩性,你需要一个非关系数据库。”如果这是真的,那么这是不是一个迹象,表明曾经强大的关系式数据库终于在它的盔甲上出现了裂缝?关系数据库的日子是不是到头了?该隐退了?在本文中,我们 将检视当前这种在特定情况下摆脱关系数据库的趋势,并分析这对于关系数据库的未来意味着什么。

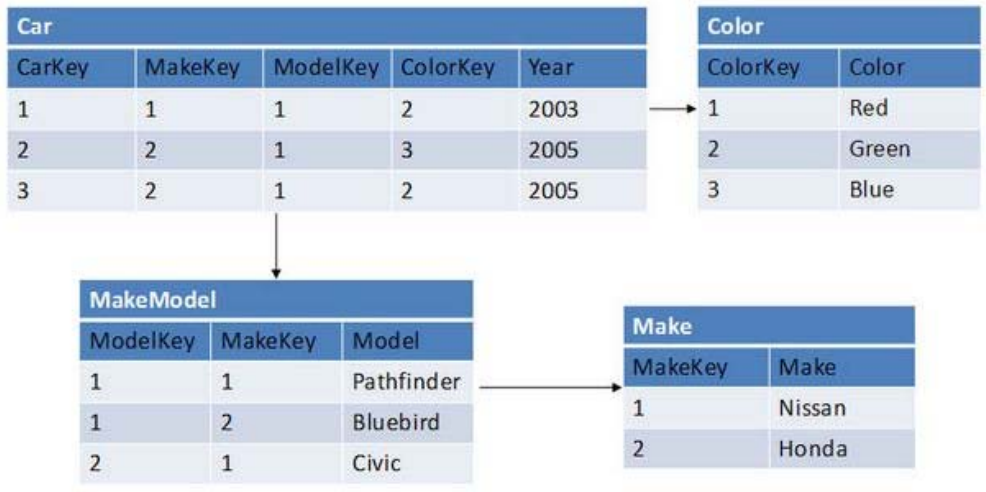
最近,大量新的非关系式数据库如雨后春笋般出现在云里云外。这其中所释放出的一个关键信息是:“如果想获得丰富而 随需应变的可伸缩性,你需要一个非关系数据库。”

如果这是真的,那么这是不是一个迹象,表明曾经强大的关系式数据库终于在它的盔甲上出现了裂缝?关系数据库的日子是不是到头了?该隐退了?在本文 中,我们将检视当前这种在特定情况下摆脱关系数据库的趋势,并分析这对于关系数据库的未来意味着什么。

关系数据库已过而立之年。在此期间,短暂爆发过一些所谓终结关系数据库的革命。当然,最终都失败了,丝毫没有动摇到关系数据库的主导地位。

先了解一些背景

一个关系数据库基本上就是一个表(实体)集合。表由列和行(变量集)构成。这些表存在约束,相互之间定义了关系。关系数据库使用SQL进行查询,结 果集通过访问一个或多个表的查询生成。单个查询里被访问到的多个表,一般是利用在表关系列里定义的范式被“连接”到一起的。 规范化 是关系数据库使用到的一种数据结构模型,能保证数据一致性并消除数据冗余。



Example of a Typical Relational Data Model

关系数据库在 关系数据库管理系统 (RDBMS)的帮助下得以促进。我们今天所使用的绝大部分数据库系统都是RDBMS,包括 Oracle, SQL Server, MySQL, Sybase, DB2, TeraData等等。

关系数据库占据统治地位的原因并非微不足道的。它们持续提供了简单性、健壮性、灵活性以及性能的最佳组合,并带来了通用数据管理的兼容性。

不过,为了实现这一切,关系数据库的内部不得不复杂得难以置信。举个例子说,一个相对简单的SELECT语句可能有着上百条执行路径,优化程序在运 行时必须进行评估。这一切都被隐藏起来,对用户都是不可见的,RDBMS通过使用类似基于代价的算法来决定最佳响应请求的“执行计划”。

关系数据库的问题

尽管RDBMS为数据库用户提供了简单性、健壮性、灵活性、性能、可伸缩性以及兼容性的最佳组合，但它在其中每个领域里的性能，不一定就优于其他追 求某一项好处的独立替代方案。迄今为止这不是太大的问题，因为RDBMS的普遍优势已经压制了开疆拓土的需求。不过，如果你的确存在着通用关系数据库无法 满足的要求，替代的方案则总可以填补这些壁龛。

今天，形势略有不同。对于数量不断增长的应用程序而言，上述的其中一项好处变得越来越重要；虽仍被视为特殊，但正迅速成为主流，以至于对于不断增长 的数据库用户来说，其在重要性方面已经开始侵蚀掉其它的好处了。这项好处就是可伸缩性。随着越来越多如Web服务之类承受大规模工作负荷的应用的发行，其 对可伸缩性的需求，首先有可能会改变得非常迅速，其次会变得无比庞大。第一种场景下，如果你只有一个龟缩在一所房子内的服务器中的关系数据库，情况将难以 管理。举个例子，如果你的负载一夜之间增加了2倍，你能用多快的速度去升级硬件？第二种情况下，一般的关系数据库是也很难管理的。

关系数据库的确能伸缩自如，但通常只能单台服务器节点上进行。一旦单节点的能力抵达上限，你就得通过多服务器节点来往外扩展来分发负载。这时候关系 数据库的复杂性就开始影响其潜在的扩展规模了。试图扩展到成百上千个节点，而不是几个，将导致不堪复杂性之重负，这一特点使得RDBMS在大型分布式系统 平台市场里的生存能力被大幅削减。

为了让云服务变得可行，供应商不得不突破这种限制，因为一个缺乏伸缩性的数据仓储的云平台根本就不能算一个平台。因此，为了能向客户提供的一个伸缩 自如的空间去存放应用数据，供应商实际上只有一种真正的选择。他们不得不实现一种新型的关注于可扩性的数据库系统，而牺牲掉关系数据库所带来的其他好处。

这些努力，再加上那些已有的特殊供应商，已经带来了一种新型的数据库管理系统。

新品种

这种新型的数据库管理系统通常被称为键/值存储。实际上，尚无正式的名字，因此你可能会看到它被称为面向文档的、面向互联网的、面向属性的、 分布式数据库 (尽管这也可以是关系式的)、共享排序数组、 分布式哈希表以及键/值数据库。虽然每个名字都指出了这种新方案的某种特征，但都是基于一个主题 的派生，这个主题就是我们将要命名的键/值数据库。

不关你怎么称呼它，这个“新型”的数据库其实已经在某些普通关系数据库不合适的特殊应用里使用了很长时间了。不过如果没有Web和云应用所带来的伸 缩性的话，它很可能还得继续呆在深闺大院里。现在的挑战是我们得弄清楚，究竟是它还是关系数据库更适合于特定应用。

关系数据库和键/值数据库从根本上来说是完全不同的，分别被设计用于满足不同的需求。迄今为止，一项一对一的比较能有助于你理解这种差异性，不过在 开始之前，先让我们来看看下面：

Database Definition	
Relational Database	**Key/Value Database**
• Database contains tables, tables contain columns and rows, and rows are made up of column values. Rows within a table all have the same schema. • The data model is well defined in advance. A schema is strongly typed and it has constraints and relationships that enforce data integrity. • The data model is based on a “natural” representation of the data it contains, not on an application's functionality. • The data model is normalized to remove data duplication. Normalization establishes table relationships. Relationships associate data between tables.	• Domains can initially be thought of like a table, but unlike a table you don't define any schema for a domain. A domain is basically a bucket that you put items into. Items within a single domain can have differing schemas. • Items are identified by keys, and a given item can have a dynamic set of attributes attached to it. • In some implementations, attributes are all of a string type. In other implementations, attributes have simple types that reflect code types, such as ints, string arrays, and lists. • No relationships are explicitly defined between domains or within a given domain.

没有实体连接

键/值数据库是面向项目的，这意味着所有与项目有关的数据都被存储进该项目中。一个域（你可以把它视为表）可以包含大量不同的项目。比如说，一个域 里可以同时包含客户项目和定单项目。这意味着在一个域内，不同项目间的数据通常是重复的。这在实践上是可行的，因为磁盘空间相对廉价。但这个模型允许一个 单一的项目包含完所有相关数据，就可以通过消除对多表的数据连接的需求来改善可扩性。而在关系数据库中，那样的数据需要被连接到一起，以便能重组为相关属 性。

Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto

Example of a Typical Key/Value Domain

不过虽然在关系数据库中的需求在键/值数据库已大为减少，有些东西还是不可避免的。那些关系一般存在于核心实体之间。比如说，定单系统会有这样一些 项目，其中包含有客户、产品及定单的数据。 这些是否在相同或不同的域都是无关紧要的；但是当客户下单时，你大概不会想把客户和产品的属性都放进同一张定单里吧。

相反，定单需要包含相关键值指向客户和产品。尽管这在键/值数据库是完全可行的，这些关系却不会在数据模型本身内进行定义，因此数据库管理系统是无 法强制要求这些关系的数据一致性的。这意味着你可以删除客户及其已订购产品。确保数据完整性的责任完全落在了应用的身上。

Data Access	
Relational Database	**Key/Value Database**
• Data is created, updated, deleted, and retrieved using SQL. • SQL queries can access data from a single table or multiple tables through table joins. • SQL queries include functions for aggregation and complex filtering. • Usually contain means of embedding logic close to data storage, such as triggers, stored procedures, and functions.	• Data is created, updated, deleted, and retrieved using API method calls. • Some implementations provide basic SQL-like syntax for defining filter criteria. • Basic filter predicates (such as =, !=, <, >, <=, and >=) can often only be applied. • All application and data integrity logic is contained in the application code.

Application Interface	
Relational Database	**Key/Value Database**
• Tend to have their own specific API, or make use of a generic API such as OLE-DB or ODBC. • Data is stored in a format that represents its natural structure, so must be mapped between application code structure and relational structure.	• Tend to provide SOAP and/or REST APIs over which data access calls can be made. • Data can be more effectively stored in application code that is compatible with its structure, requiring only relational "plumbing" code for the object.

键/值存储:优点

有两点键/值数据库是明显优于关系数据库的。

云的最佳搭档

第一个好处是它们简单，并因此比关系数据库伸缩起来要自如得多。如果你正在打算把一个内部系统聚拢起来，试图把预期中规模庞大的伸缩需求交给数据仓 储背后那数十上百台服务器去处理，那么就请考虑一下键/值存储。

由于键/值数据库简单，且动态可扩，它们也是提供多用户、web 服务的平台数据存储的供应商的选择。这种数据库提供了相对廉价的设计存储平台，并拥有庞大的扩充潜力。用户通常只需用多少就给多 少，而其需求增长时配额能随之而增。与此同时，供应商能基于总用量动态扩充平台，整个平台的大小几乎不受限制。（译者：最典型的就是各种所谓支持多少G的 邮箱应用了）

编码得心应手

关系数据库模型和应用代码对象模型通常是以不同方式建立起来的，这导致了不兼容性。开发人员通过将代码映射到关系模型去克服这种不兼容性。这个过程 一般被称为 对象-关系映射，基本上等于是"管道"代码，没有直接明确的价值，却耗费掉了应用开发的大量时 间和精力。另一方面，许多键/值数据库在结构中保留的数据，与底层代码中的对象类的映射关系却要直接得多，从而显著减少了开发时间。

其它一些支持这种数据存储的理由，比如"关系数据库相比之下更为笨拙（不管这意味这什么）"等，则不太令人信服。不过在跳上这趟键/值数据库的列车 之前，请先考虑一下它的缺点。

键/值存储: 缺点

关系数据库固有的约束保证数据在最低层次拥有完整性。违反完整性约束的数据在物理上进不了数据库中。这些约束在键/值数据库中是不存在的，因此确保 设计完整性的责任全部落到了应用程序的肩上。但是程序会经常出现Bug。在一个设计得当的关系数据库里，Bug通常不会导致数据完整性问题；但是在键/值数据库里的bug就很容易引起数据完整性问题。

关系数据库另一项关键的好处就是它强迫你经过一个数据建模的过程。如果做得好，这个建模过程所创建出的数据库的逻辑结构，就应该是映射它所要包含的 数据，而非映射应用程序的结构。然后数据从某种程度上就变得是独立于应用的，意味着其他应用同样也能使用统一数据集，从而应用逻辑的改变不会影响到底层的 数据模型。而键/值数据库要实现这一过程的话，就要以类的建模实践替代关系数据建模，以便在数据的自然结构基础上创建出通用的类。

还有别忘了兼容性。面向云的数据库不像关系数据库，并没有多少办法去共享标准。尽管它们都有着类似的概念，却各有着自己的API、特定的查询接口及 特性。因此，你真的要信任你的供应商，因为就算你对服务不满意也覆水难收了。还有，由于目前所有的键/值数据库均处于测试阶段，这种信任的风险可比对老派的关系数据库要高的多。

分析上的限制

在云内，键/值数据库一般是多租户的，这意味这许多用户和应用将使用同一系统。为了防止任一进程导致

共享环境过载，大部分 云数据存储严格限制了单一查询可能导致的总影响效应。举个例子，在SimpleDB里，你不能执行过程超过5秒钟的查询。而在Google的 AppEngine Datastore里，任何查询结果都不允许超过1000条。

这些限制对于你的面包-奶油式的应用逻辑（添加、更新、删除、获取少量内容）而言不成问题。但是当你的应用取得成功的时候会发生什么呢？你已经攫取 了许多用户，获得了大量数据，现在你想为客户创造新价值，或者也许还想利用这些数据产生新的收入。你就会发现自己被严重限制了，甚至连直接的分析型查询都 很困难。在此类平台上，类似追踪（用户的）使用模式、基于用户历史提供建议等事情，即便不是不可能也是非常困难的。

这种情况下，你将不得不实现一个从键/值数据库分离出来的，独立的分析型数据库，以便执行那样的分析。再想想你该在哪里才能做这样的事情？又该如何 去做？是不是应在云上维护它？还是投资于一个现场（译者：on-site对应于off-site，一般在IT外包中应用）的设施？你和云服务供应商之间的 延迟会不会成为问题？你现在的基于云的键/值数据库支持它吗？如果你的键/值数据库有10亿个条目，可是每秒钟却只能提供1000条结果，这样的查询该执 行多久才能完事呢？

归根结底，虽然规模是一项考虑因素，不要把它排在你将数据转换成为资产的能力的前头。如果你的用户，因为竞争对手拥有更酷更人性化的特性而跑掉，这 世上的一切伸缩性都将一无是处。

云服务竞争者

一些网络服务供应商现在提供了基于即用即付的多租户键/值数据库。大部分都符合我们这里讨论到的标准，但是每一个都有其独特之处，与迄今描述的一般 标准有所不同。现在让我们看看这些特定的数据库，分别是SimpleDB， Google AppEngine Datastore和SQL Data Services。

亚马逊**: SimpleDB**

SimpleDB 是一个亚马逊网络服务平台的一个面向属性的键/值数据库。SimpleDB 仍处于公众测试阶段；当前，用户能在线注册其“免费”版 --免费的意思是说直到超出使用限制为止。

SimpleDB有几方面的限制。首先，一次查询最多只能执行5秒钟。其次，除了字符串类型，别无其它数据类型。一切都以字符串形式被存储、获取和 比较，因此除非你把所有日期都转为ISO8601，否则日期比较将不起作用。第三，任何字符串长度都不能超过1024字节，这限制了你在一个属性中能存储 的文本的大小（比如说产品描述等）。不过，由于该模式动态灵活，你可以通过追加“产品描述1”、“产品描述2”等来绕过这类限制。一个项目最多可以有 256个属性。由于处在测试阶段，SimpleDB的域不能大于10GB，整个库容量则不能超过1TB。

SimpleDB的一项关键特性是它使用一种最终一致性模型。这个一致性模型对并发性很有好处，但意味着在你改变了项目属性之后，那些改变 有可能不能立即反映到随后的读操作上。尽管这种情况实际发生的几率很低，你也得有所考虑。比如说，在你的演出订票系统里，你不会想把最后一张音乐会门票卖 给5个人，因为在售出时你的数据是不一致的。

Google AppEngine Data Store


Google AppEngine Datastore 是在BigTable之上建造出来的， 是Google的内部存储系统，用于处理结构化数据。AppEngine Datastore其自身及其内部都不是直接访问BigTable的实现机制，可被视为BigTable之上的一个简单接口。

AppEngine Datastore所支持的项目的数据类型要比SimpleDB丰富得多，也包括了包含在一个项目内的数据集合的列表型。

如果你打算在Google AppEngine之内建造应用的话，几乎可以肯定要用到这个数据存储。然而，不

像SimpleDB，使用谷歌网络服务平台之外的应用，你并不能并发地与 AppEngine Datastore进行接口 (或通过BigTable)。

微软**: SQL**数据服务

SQL 数据服务 是微软 Azure 网 络服务平台的一部分。该SDS服务也是处于测试阶段，因此也是免费的，但对数据库大小有限制。 SQL数据服务其自身实际上是一项处在许多SQL服务器之上的应用，这些SQL服务器组成了SDS平台底层的数据存储。你不需要访问到它们，虽然底层的数 据库可能是关系式的；SDS是一个键/值型仓储，正如我们迄今所讨论过的其它平台一样。

微软看起来不同于前三个供应商，因为虽然键/值存储对于可扩性而言非常棒，相对于RDBMS，在数据管理上却很困难。微软的方案似乎是入木三分，在 实现可扩性和分布机制的同时，随着时间的推移，不断增加特性，在键/值存储和关系数据库平台的鸿沟之间搭起一座桥梁。

非云服务竞争者

在云之外，也有一些可以独立安装的键/值数据库软件产品。大部分都还很年轻，不是alpha版就是beta版，但大都是开源的；通过看看它的代码， 比起在非开源供应商那里，你也许更能意识到潜在的问题和限制。

CouchDB

CouchDB 是一个免费、开源、面向文档的数据库。它来自于键/值存储，使用JSON（译者：JavaScript Object Notation，一种轻量级的数据交换格式）来定义项目的模式。CouchDB允许通过JavaScript动态创建“视图”，意在跨越面向文档型数据 库与关系数据库之间的鸿沟。这些视图将文档数据映射在类似表的结构上，可被索引和查询。

现在， CouchDB 还不是真正的分布式数据库。它的复制功能允许数据在服务器间同步，但 这并非建设高可扩性的环境所需的那种分布类型。毫无疑问，CouchDB团队将朝此目标继续努力。

Project Voldemort

Project Voldemort 是分布式的键/值数据库，旨在横向扩展于大量的服务器中。它产生自LinkedIn所完成的工作，据报告在 那儿为几个有着极高可扩性要求的系统所使用。Project Voldemort也使用了Amazon的最终一致性模型。

Project Voldemort还很新；它的网站前几周才刚开张。

Mongo

Mongo是由Geir Magnusson和Dwight Merriman (提到DoubleClick你可能就想到他)在10gen开发出来的数据库系统。跟CouchDB一样，Mongo是一个面向文档的JSON数据库，除 了它是被设计为一个真正的对象数据库，而不是一个纯粹的键/值存储这一点之外。起初，10gen关注于整合出一个完整的网络服务栈;然而最近，它已经把重 点转移到Mongo数据库上了。其beta测试版计划在二月中发布。

Drizzle

Drizzle可被认为是键/值存储要解决的问题的反向方案。Drizzle诞生于 MySQL（6.0）关系数据库的拆分。在过去几个月里，它的开发者已经移走了大量非核心的功能（包括视图、触发器、已编译语句、存储过程、查询缓冲、 ACL以及一些数据类型），其目标是要建立一个更精简、更快的数据库系统。Drizzle仍能存放关系数据；正如MySQL/Sun的Brian Aker所说那样：“没理由泼洗澡水时连孩子也倒

掉”。它的目标就是，针对运行于16核（或以上）系统上的以网络和云为基础的应用，建立一个半关系型数据 库平台。

决策

最终，有四条理由支持你为应用选择非关系式的键/值数据库平台：

1. 你的数据很大程度上是面向文档的，使得它比关系数据库更自然地适合于键/值数据模型。
2. 你的开发环境严重地面向对象时，键/值数据库能尽量减少对“管道”代码的需求。
3. 数据存储很廉价，易于集成进供应商的网络服务平台。
4. 你优先考虑的是随需应变的高端可扩性 -- 也就是说，那种通过简单的扩充所无法获得的大规模、分布式的可扩性。

但是在作决定的时候，要记得这种数据库的限制，以及从关系式大路出走时所面临的风险。

对于其他需求而言，你也许在RDBMS那里能得到最好的满足。因此，关系数据库的死期是不是到了？显然不是。嗯，至少还没有。

原文转自：http://article.yeeyan.org/view/%E5%8D%9A%E8%B4%A4/29756?from_com

- 标签：key－value 数 据库

- 上一篇：沁苑工 作室老成员，你们还好吗？沁苑网络多媒体工作室毕业生调查活动。
- 下一篇：CSDN 积分兑换的网址