Google F1：异步模式变更算法

czt1999

日期：June 30, 2022

1 F1 特性与基础设计

F1 是 Google 研发的分布式关系型数据库管理系统 [3]，在保证一致性、高可用性的基础上，提供一种新型的关系型数据模式演化（schema evolution）协议。模式演化是指在不丢失数据的情况下改变数据库定义的能力，已经被业界研究了近三十年。Google 提出 F1 的动机是为了让其广告系统具备更好的可扩展性，该系统的存储层最初是基于分片的 MySQL 构建的，难以应付日益增长的业务数据和相对频繁的模式变更需求。

1.1 特性及约束

F1 的模式演化协议以如下所述的系统特性为基础 [2]：

1. *大规模的分布式*。一个 F1 实例由数百个独立的服务器组成，运行在分布在全球许多数据中心的共享机器集群上。
2. *关系型模式*。每个 F1 服务器都有一个描述表、列、索引和约束的关系型模式的副本，对模式的任何修改都需要分布式机制来更新所有服务器。
3. *共享数据存储*。所有数据中心的 F1 服务器都可以访问下层的键值存储系统（这里是 Spanner）中的所有数据，服务器之间不存在数据分区。
4. *无状态服务器*。F1 服务器必须能够容忍机器故障、抢占、网络资源的不可用，因此被设计成无状态的——客户端可以连接到任何 F1 服务器，同一事务的不同语句也可以由不同的 F1 服务器执行。
5. *无全局成员*。由于 F1 服务器的无状态特性，F1 不需要实现全局成员关系协议。没有可靠的机制来确定当前运行的是哪个服务器，无法（同时也不需要）实现显式的全局同步。

此外，F1 的模式演化协议还受到如下的业务需求约束：

1. *完全的数据可用性*。F1 是 Google 业务基础设施的关键部分，数据的不可用会直接影响经济效益，因此，对模式的修改（例如，锁定一个列以建立新索引）不能使数据库的任一部分脱机。
2. *最小化对性能的影响*。上面提到，F1 的主要任务是支持 Google 的广告系统。由于该系统在多个团队和项目之间共享，模式更改非常频繁，如果每次更改都需要以明显的性能降级为代价，势必会削弱整个系统的可用性。
3. *异步的模式变更*。F1 中不存在全局成员关系，因此无法对所有服务器进行同步的模式更

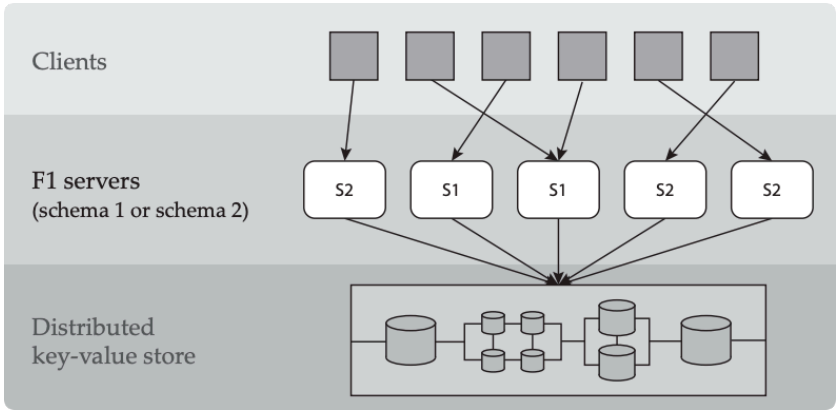

图 1: F1 模式变更基础设计

Example

first_name*	last_name*	age	phone_number
John	Doe	24	555-123-4567
Jane	Doe	35	555-456-7890

(a) Relational representation.

key	value
Example.John.Doe.*exists*	
Example.John.Doe.*age*	24
Example.John.Doe.*phone_number*	555-123-4567
Example.Jane.Doe.*exists*	
Example.Jane.Doe.*age*	35
Example.Jane.Doe.*phone_number*	555-456-7890

(b) Key–value representation.

图 2: F1 行表示方式

改，不同的 F1 服务器会在不同的时间过渡到新模式。换言之，同一时间的不同 F1 服务器可能应用着不同的模式，如图 1 所示。

1.2 存储引擎与行表示

前面提到，F1 服务器需要的状态都放置在下层的键值存储系统中，Google 实际上选择 Spanner 作为这个存储系统的实现，这是一个 Google 在更早的时期提出的可扩展、多版本、全局同步复制的数据库，能提供外部一致的读写操作 [1]。Spanner 的外部一致性保证主要是源于 TrueTime API，直接暴露分布式系统中时钟的不确定性。得益于良好的抽象设计，在 F1 的视角中，Spanner 就是一个提供外部一致性的键值存储引擎，无需考虑额外的实现细节。F1 对关系型数据到键值数据的映射方式如图 2 所示，这一过程在实现中又被称为编码。

1.3 并发控制

F1 使用一种基于时间戳的乐观并发控制机制，类似于本文第 **??** 节介绍的 Percolator（同样构建在分布式键值存储之上）。在 F1 的模式中，每个表中的每一列都与一个乐观锁关联，这个乐观锁控制着不同事务对该行数据的并发访问。

当客户端在一次事务中读取列值时，会从覆盖这些列的锁中获取最后修改的时间戳；在后续提交时，会将这些时间戳交给服务器进行验证，以确保它们没有更改。如果验证成功，与这些列相关联的所有锁的时间戳将被更新为当前时间戳。这一策略保证了事务的原子性。

由于用户可以向表添加新锁并将其与该表中的任意列关联，因此 F1 必须考虑锁机制对模式变更的影响。

2 异步模式变更的难点与对策

2.1 异步模式变更的过程

如 1.1 所述，F1 实例中的所有服务器共享一组位于键值存储中的键值对。为了将这些键值对解码为行，每个 F1 服务器需要在其内存中维护一个特定版本的模式实例，并使用该模式将关系操作符转换为键值存储所支持的操作。换言之，模式决定了 F1 服务器如何将下层的键值数据解释为业务端需要的关系型数据。

模式本身也会被编码为特殊的键值对，对所有的 F1 服务器均可见。当使用新版本的模式替换旧版本的模式时，模式变更就启动了，F1 开始将新模式传播到实例中的所有服务器。由于其大规模的分布式特性，在这些服务器之间进行同步是无法实现的，不同的服务器可能在不同的时间转换到新模式。只有当实例中的所有 F1 服务器都加载了新模式时，模式更改才算完成。

2.2 异步模式变更的难点

由于所有 F1 服务器共享底层存储，因此，不正确地执行异步模式更改可能会破坏数据库。例如，如果新版本的模式中为表 T_i 添加了一个新的索引 $I_{i,k}$，仍然在旧模式上运行的服务器在对 T_i 中的数据进行写操作时，无法同时对 $I_{i,k}$ 进行维护。最终，任何执行索引 $I_{i,k}$ 读取的查询都将返回错误的结果。

这种错误的根本原因是对模式的变更过于突然。旧模式上的服务器不知道索引，而新模式上的服务器在所有操作中使用它，两种行为的差异破坏了数据的一致性。尽管这里以添加索引为例，但是系统中所有基本的模式更改操作都会出现类似的问题，这正是异步模式变更的难点所在。

2.3 异步模式变更的安全执行方案

为了解决上述问题，Google 的工程师设计了一个通过中间状态来执行模式更改的协议。在该协议中，模式的元素（表、列、索引等）可以被标记为若干种中间状态，从而限制对它们的操作。这种方式将单个危险的模式更改分解为一系列安全的模式更改，最终确保数据的一致性。

具体而言，F1 为模式的元素规定了两种非中间状态：1）公有（*public*），此时表、列或索引存在，且可以接受任意的读写操作；2）不存在（*absent*），此时表、列或索引已被移除，不接受任何操作。同时，为了在模式变更期间限制对元素的某一特定操作，F1 又为模式的元素规定了两种中间状态：1）只删（*delete-only*），此时表、列或索引关联的键值对只能被删除，不允许插入新的键值对，也不能被读取；2）只写（*write-only*），此时表、列或索引关联的键值对可以接受任何写操作，但也不能被读取。

这里我们以添加一个必需（required）元素为例，介绍这四种状态在模式变更过程中的应用。元素的初状态为 *absent*，末状态为 *public*，通过 2.2 的分析可知，直接进行这两种状态的转换会破坏数据的一致性，需要借助中间状态来安全过渡。四种状态的转换次序如下：

$$absent \rightarrow delete\ only \rightarrow write\ only \rightarrow public$$

文献 [2] 对上述任意两个相邻状态的转换安全性进行了形式化的推导，证明了：规定 F1 实例中的服务器最多只能有两个不同状态，这一转换次序能保证模式变更前后数据的一致性。

此外，针对宕机、网络连接丢失的情况，F1 设计了模式租约（schema lease）机制。每个 F1 服务器需要在规定的租期（例如几分钟）结束后从底层存储中重新读取模式来更新租期，这一行为称为续签。续签失败的服务器将自动终止，稍后在其他正常运行的节点上重新启动。

参考文献

[1] James C. Corbett et al. “Spanner: Google’s Globally Distributed Database”. In: *ACM Trans. Comput. Syst.* (Aug. 2013). DOI: 10.1145/2491245. URL: https://doi.org/10.1145/2491245.

[2] Ian Rae et al. “Online, Asynchronous Schema Change in F1”. In: *Proc. VLDB Endow.* 6.11 (Aug. 2013), pp. 1045–1056. ISSN: 2150-8097. DOI: 10.14778/2536222.2536230. URL: https://doi.org/10.14778/2536222.2536230.

[3] Jeff Shute et al. “F1: A distributed SQL database that scales”. In: *Proceedings of the VLDB Endowment* 6 (Aug. 2013), pp. 1068–1079. DOI: 10.14778/2536222.2536232.