超扩展 A MySQL-Compatible urora 以应对突然的流量增长

AWS 规范性指导

AWS 规范性指导: 超扩展 A MySQL-Compatible urora 以应对突然的流量增长

Table of Contents

超大规模 Aurora MySQL 兼容版应对流量激增问题

Oliver Francis、Shyam Sunder Rakhecha 和 Vikram singh Rai，Amazon Web Services（AWS）

2022 年 10 月

您的互联网业务可能会面临前所未有的高速增长，而您现有的应用程序基础设施无法应对这种情况。这可能是由各种因素造成的，例如您的产品广告引起了超出预期的兴趣，或者客户购买模式突然从面对面转为在线购买。

为了应对这些意外负载，您必须对基础设施进行超大规模扩展。扩展应用程序层需要添加更多服务器或容器组。但实现超大规模扩展最困难的部分是数据库。您可以将数据库实例扩展到最大可用实例大小，但这可能无法解决您的问题。

如果您在 Amazon Aurora MySQL 兼容版上运行数据库工作负载，本指南提供了一些建议，可以实现数据库的超大规模扩展以满足突然的高速增长。并非所有这些建议都是长期最佳实践。如果您预计会出现超高速增长，并有时间计划解决这一问题，请参阅 资源 部分所列的博客文章。这些文章可以帮助您实施长期最佳实践。另一方面，如果您面临意外的高速增长，本指南将帮助您管理负载并实现合理的稳定性，同时为您的业务规划和实施长期解决方案。

请注意，本指南中列出的建议需要您的开发团队提供实施帮助。

目标业务成果

本指南中介绍的方法有助于您做到以下几点：

- 在出现意外的高速增长时稳定您的业务。实现足够的稳定性，以实施针对高速增长的长期最佳实践。
- 防止经济损失。超大规模环境中的突然中断可能会导致客户在应用程序上执行的业务交易减少。在某些情况下，这可能会导致重大的经济损失。稳定的超大规模环境是防止导致业务损失的长时间中断的关键。

管理连接

随着对应用程序需求的增长，前端流量也随之增加。在典型场景中，您可以在应用程序层设置自动扩展，以处理激增的传入流量。因此，应用程序层开始自动扩缩，并添加更多的应用程序服务器（实例）以满足流量的增长。由于所有应用程序服务器都预先配置了数据库连接池设置，因此数据库的传入连接数与新部署的实例成比例增长。

例如，20 台应用程序服务器配置 200 个数据库连接，将总共打开 4000 个数据库连接。如果应用程序池扩展到 200 个实例（例如，在高峰时段），连接总数将达到 40000 个。在典型的工作负载下，大多数连接可能处于空闲状态。但连接的激增可能会限制您的 Amazon Aurora MySQL 兼容版数据库的扩展能力。这是因为即使是空闲连接也会消耗内存和其他服务器资源，比如文件描述符。Aurora MySQL 兼容版通常使用比 MySQL 社区版更少的内存来维持相同数量的连接。但是，空闲连接的内存使用量仍然不是零。

配置变量

您可以使用两个主要服务器配置变量来控制数据库允许的传入连接数：`max_connections` 和 `max_connect_errors`。

配置变量 max_connections

配置变量 `max_connections` 来限制每个 MySQL 实例的数据库连接数。最佳做法是将其设置为略高于每个数据库实例预计打开的最大连接数。

如果您还启用了 `performance_schema`，请特别注意 `max_connections` 设置。Performance Schema 内存结构根据服务器配置变量来自动调整大小，包括 `max_connections`。变量设置得越高，Performance Schema 使用的内存就越多。在极端情况下，这可能会导致较小的实例类型 out-of-memory出现问题。请注意，启用 Performance Insights 将自动启用 Performance Schema。

配置变量 max_connect_error

配置变量 `max_connect_errors` 来确定允许来自给定客户端主机的连续中断连接请求次数。如果客户端主机超过指定的连续失败连接尝试次数，服务器将会阻止主机。来自该客户端的进一步连接尝试会产生错误。

```
Host 'host_name' is blocked because of many connection errors. Unblock with
'mysqladmin flush-hosts'
```

如果您遇到 “主机被屏蔽” 错误，请避免增加`max_connect_errors`变量的值。而是调查 `aborted_connects` 状态变量和 `host_cache` 表中的服务器诊断计数器。使用收集的信息来识别和

修复遇到连接问题的客户端。另请注意，如果 `skip_name_resolve` 设置为 1（默认），则此参数无效。

有关以下内容的详细信息，请参阅 MySQL 参考手册：

- max_connect_errors 变量
- “主机被屏蔽” 错误
- aborted_connects 状态变量
- host_cache 表

实施连接池

扩展事件可能会添加更多的应用程序服务器，这反过来又可能导致数据库服务器超过满载的活动连接数。在应用程序服务器和数据库之间添加连接池或代理层就像漏斗一样，可以减少数据库上的连接总数。代理的主要目的是通过多路复用来重用数据库连接。

一方面，代理通过数量受控的连接来连接到数据库。另一方面，代理又接受应用程序连接。它还提供了其他功能，比如查询缓存、连接缓冲、查询重写和路由以及负载均衡。需要对连接池层进行配置，使数据库的最大连接数保持在满载数量以下。Amazon RDS 代理是一个完全托管的代理，您可以实施它来达到此目的。对于大多数应用程序，Amazon RDS 代理不需要更改代码，而且您无需管理任何额外的基础设施来实施该解决方案。

您还可以探索以下适用于 Aurora MySQL 兼容版的第三方代理：

- ProxySQL
- MariaDB MaxScale
- ScaleARC
- Heimdall Data

避免连接风暴

考虑在数据库过载或副本落后主节点太多的情况下连接池的行为。配置代理服务器或连接池时，确保不要因为数据库响应缓慢（由底层硬件或存储问题或数据库资源限制所致）而重置整个连接池。

突然启动数百个连接会产生连接风暴，因为同时发起大量对数据库的新连接请求。这种风暴需要大量资源。在 MySQL 中创建新的数据库连接是一项昂贵的操作，因为后端会交换多个网络数据包，以进行初

始握手、生成新进程、分配内存、处理身份验证等。如果短时间内收到大量请求，数据库可能会出现无响应的情况。

MySQL 有一种机制可以防止这种连接请求激增。将 back_log 变量设置为在 MySQL 暂时停止响应新请求之前的短时间内可以堆叠的请求数。该值由连接处理线程强制执行，而线程本身可能会被连接风暴淹没。有关更多信息，请参阅 MySQL 参考手册。

如果您的连接配置为在数据库运行缓慢时重置，那么您将一次又一次地启动循环。同样，如果您预计在一天中的某些时间（例如，股市开盘时）数据库流量会突然增加，请预热您的连接池，这样就不会在高流量负载开始的同时尝试打开许多连接。

查询工作负载调优

经过良好调优的工作负载将对您在实现高速增长的稳定解决方案方面大有帮助。如果您的工作负载没有得到很好的调优，无论您使用的 Amazon Aurora 集群有多强大，都将遇到瓶颈，从而降低性能并影响应用程序的用户体验。最佳做法是从一开始就有一个流程，帮助您识别和调优系统中有问题的查询。

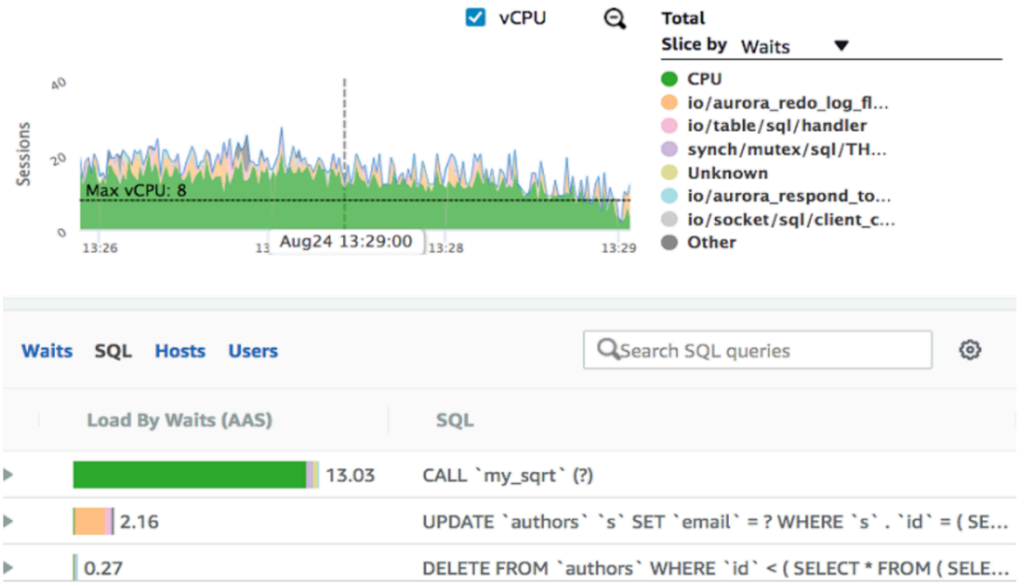
跟踪和调优工作负载中有问题的查询

当遇到高速增长时，拥有经过良好调优的工作负载是成功的一半。要了解实时工作负载的性质和 Aurora 集群面临的性能问题，确保您的团队从写入器和读取器实例中收集有问题的查询。需要调优这些有问题的查询，以使您的工作负载以最佳状态运行。亚马逊 Aurora E MySQL-Compatible dition 为您提供了两种方法来实现这一目标：

- 性能详情
- 慢速查询日志

使用 Performance Insights

Performance Insights 根据平均活动会话（AAS）来跟踪 Aurora 写入器或读取器实例上的负载。AAS 值是使用采样和等待 CPU 拾取查询工作负载并进行处理的活动会话数来计算的。Performance Insights 提供了一个图形界面，您可以在其中检查因等待活动会话而导致最高负载的 SQL 语句。



在上一个屏幕截图中，对存储过程 `my_sqrt` 的调用导致平均有 13.03 个会话等待处理其负载。合乎逻辑的下一步是调优该过程。您应该找出读取器和写入器中对各自实例造成负载的 SQL 语句，并对其进行调优以提高性能，直到 AAS 值下降并保持在 Performance Insights 的最大 vCPU 虚线以下。如果您的调优工作已达到极限，但仍看到 AAS 超过最大 vCPU 线，则可以选择更大的实例类来处理您的工作负载。如果不先尝试调优查询工作负载，就不要选择更大的实例，因为不断增长的流量将开始暴露工作负载中由错误查询造成的故障线。

使用慢速查询日志并将其发布到 CloudWatch

慢速查询日志是 MySQL 的原生功能，是对 Performance Insights 的补充。最佳做法是同时使用这两种方法，提前发现可能对您的实例造成严重破坏的有问题的查询。慢速查询日志会记录任何比动态变量 `long_query_time` 慢的查询。无需重启集群实例即可设置此变量。

为了在调优练习中提供灵活性和隔离性，请为写入器和读取器实例使用单独的参数组。如果您使用读写分离，这一点尤其重要。根据需要为集群实例中的 `long_query_time` 设置一个适当的限值。在调优负载时，您可以在 `long_query_time` 变量中设置激进的值，因为您可以将阈值设置为毫秒级。凭借高并发性和良好调优的工作负载，几乎所有查询都应该以毫秒为单位运行。

当您将 Amazon Aurora MySQL-Compatible Edition 设置为 MySQL-Compatible 将慢速查询记录到文件时，Aurora 会将慢速查询日志写入 Aurora MySQL-Compatible 文件系统并保留 24 小时。要将慢速查询日志保留更长的时间以供分析，请将其发布到 Amazon CloudWatch。您还可以构建 CloudWatch 控制面板来监控您的慢速查询。有关更多信息，请参阅博客文章创建亚马逊 CloudWatch 控制面板来监控亚马逊 RDS 和 Amazon Aurora MySQL。除了分析您在亚马逊上的慢查询外 CloudWatch，您还可以使用 Percona Toolkit 中的工具 pt-query-digest 来分析慢速查询日志。

您也可以选择自动执行下载和分析查询的过程，以提高团队的效率。您的团队应检查频繁运行和间隔较长的查询，并优先对其进行调优。目标是在慢速查询日志中记录很少的查询，随着您对工作负载的了解和调优，可以减少 `long_query_time`，使其更加激进。

查询调优指南

在发现工作负载中有问题的查询后，应对每个查询进行调优。使用以下调优指南使您的工作负载更高效地运行。

尽量减少扫描行数

尽管看起来很简单，但这是您在调优查询时可以使用的一条很好的建议。使用 EXPLAIN 语句，查看行列，了解优化程序在每个联接处扫描的行数。尝试通过创建最佳索引来减少扫描的行数，然后重新解释查询以确认您的工作。有关更多信息，请参阅 MySQL 文档。

如果您使用分区表，请务必使用启用分区修剪的 WHERE 子句来查询它们，这样优化程序就不必扫描每个分区。如果 WHERE 子句包含分区列的常量，优化程序就会知道要查找哪个分区，从而提高查询效率。

该建议的另一个方面是数据库的设计。查询中的表越少，查询速度就越快。如果可以对数据库设计进行去规范化，就可以让优化程序扫描更少的行，从而提高查询性能。

尽量减少临时表的使用和磁盘上的临时表

如果 Aurora MySQL-Compatible 优化器无法直接从索引中获得所需的查询结果，它会在 RAM 和磁盘上创建临时表。因此，调优的一个重要方面就是为您的工作负载提供正确的索引。但是，工作负载中可能存在不能仅依赖索引的查询，因此某些操作可能会在临时文件中执行。只要将这些保持在最低限度，并确保在磁盘上创建的表很少，就没问题。当临时表的大小太大而无法存储在内存中时，MySQL 就会创建磁盘表。MySQL 用来检查内部临时表大小的逻辑是两个变量值 tmp_table_size 和 max-heap-table-size 中较小的一个。您可以根据工作负载将这些变量调整为最佳值，这样在无法阻止临时表的情况下，仅在极少数情况下将它们推送到磁盘。

避免文件排序

如果您的工作负载有大量的 `ORDER BY` 查询，处理这些查询的最佳方法是在表上使用正确的索引。确保多列索引设计良好，避免在文件中排序。如果未使用常量扫描前面的列，则无法对列进行排序（`in`、`>`、`<`、`!=` 和 `BETWEEN` 不允许对右侧的下一列进行排序）。在 MySQL 中进行排序的最佳方式是放置一个多列索引，该索引将包含查询中提供的常量值的列放置在连续结构中排序列的左侧。在迫不得已的情况下，如果不进行文件排序，您的查询就无法返回结果，请将排序移至应用程序。

避免在高并发下运行聚合查询

您的工作负载可能有少量聚合查询，以符合应用程序中的某些功能。这个用例需要非常小心。InnoDB 引擎专为适当的联机事务处理（OLTP）负载而设计，但即使是几个高并发的分组查询也会对 CPU 造成很大的负担，并会迅速降低集群的性能。要解决需要聚合结果集的用例，请将数据预聚合到随时可读的表中，这样就可以完全避免 group by 查询。

测试查询的并发性

调整单个查询时，请记住，这些查询在 Aurora 中的多个 vCPU 上同时运行。 MySQL-Compatible在您的测试环境中，查询可能在几毫秒内运行一次。但这不是全部。务必在生产集群上使用预期的并发级别测试您的查询，并对其性能进行基准测试。只有当查询符合您的并发目标时，才将其发布到生产环境。确保在测试脚本中使用优化程序 `hint sql_no_cache`，以避免从缓存中获取结果。您可以使用 mysqlslap 等工具以并发方式执行测试，并对结果进行基准测试。

调优写入工作负载

实现负载均衡并释放写入器实例有助于您的写入工作负载在高峰期更好地执行。要在高并发下获得更好的写入性能，请执行以下额外的步骤。

将引用完整性移至应用程序层

虽然引用完整性检查很重要，但是对于超大规模，它们可能会对您的负载造成不利影响。对于每次写入，必须在写入本身运行之前执行额外的扫描，这会导致性能下降。如果您的应用程序需要严格的完整性检查，请将其构建到应用程序层，以防止检查限制写入操作。

避免使用繁重的主键

保持主键轻便。InnoDB 存储引擎会将主键附加到您在表中创建的所有其他索引。当主键较大时，会影响索引的大小。如果主键很大，数据页的存储和检索速度会变慢。一个常见的例子是使用通用唯一标识符作为主键。如果要在超大规模环境中提高性能，这种方法并不可取。

使用分区交换将数据加载到分区表中

如果要向分区表中写入大型数据集，LOAD DATA FROM S3 和分区交换的组合可以提高性能，因为插入操作不会访问主表。分区交换涉及数据定义语言（DDL），它在表上放置元数据锁。确保在表上运行的查询最少或没有查询时执行此操作，以便分区交换 DDL 无需等待即可获取元数据锁。交换只需几毫秒即可完成。

删除未使用的索引

InnoDB 会根据数据的增长优化其查询计划，最好检查数据库中是否有未使用的索引并将其删除。当应用程序向表中写入数据时，未使用的索引会消耗 IO。检查未使用的索引列表，并确认它们不是在极少数情况下使用的索引，比如季度报告。另外，请注意，某些索引用于强制执行唯一性或排序，也必须予以考虑。

确保有效清除旧行版本

在 InnoDB 的多版本并发控制（MVCC）实现中，当修改记录时，被修改数据的当前（旧）版本首先在撤消日志中记录为撤消记录。历史列表长度（HLL）值增长表示 InnoDB 垃圾回收线程（清除线程）无法跟上写入工作负载，或者清除被长时间运行的查询或事务阻止。当垃圾回收被阻止或延迟时，数据库可能会产生相当大的清除滞后，这可能会对查询性能产生负面影响。您可以使用以下建议来优化清除过程。

- 减少事务。

- 对于读取查询，请使用 READ COMMITTED 隔离级别。
- 增加清除线程数（innodb_purge_threads）和（innodb_purge_batch_size）。请注意，调优这些参数需要重新启动。
- 定期监视 HLL，并解决任何阻碍垃圾回收进行的工作负载问题。

确保日志记录不会引起额外争用

一般查询日志记录客户端连接和断开连接，服务器按接收顺序接收所有语句。激活后，日志记录同步进行，这可能会对繁忙的系统造成严重的性能损失。除非需要，否则我们建议停用一般日志。

慢速查询日志记录运行时间超过 long_query_time 秒数的语句，默认设置为 10 秒。当设置为 0 时，将同步记录所有语句，这可能会对繁忙的数据库造成性能损失。

降低负载

缩短响应时间并增加关键工作流程的可用资源可能需要清除无关的负载。本节介绍的许多解决方案都是权衡决策。它们对应用程序有影响，必须仔细考虑。在以下情况下，考虑使用这些解决方案：

- 您已在使用最大大小的实例，尤其是对于主可写数据库实例。
- 作为最后的手段，在短期内提供足够的空间来实施其他更改。

即时更改包括：

- 将非关键读取流量从主数据库实例中移出。
- 存档或清除旧数据。
- 移除引用完整性。
- 禁用二进制日志（如果正在使用）。
- 推迟非关键提取、转换、加载（ETL）流程。
- 暂停或降级非必要的应用程序功能。

在采取这些行动之前，请根据长期业务目标和风险对其进行评估。

分离读写工作负载

运行由 MySQL 提供支持的应用程序时，一种常见的技术是将读取操作从写入器（主）数据库实例卸载到一个或多个只读数据库副本上。通过卸载读取，您可以减少主数据库实例的总体负载，并为写入操作

留出空间。确保仅针对不依赖于副本的立即写后读一致性的读取。更有效的方法是将所有读取流量移动到副本，并计划在发生复制延迟时重新尝试写后读。可以卸载一些独立的读取工作负载，如报告服务。其他读取操作需要在应用程序级别进行更改，在此级别发布读取的上下文是众所周知的。

另一种方法是实现数据库代理解决方案，作为应用程序和数据库之间的中介，它可以提供读写分离和查询路由的功能，而无需应用程序感知。ProxySQL、MariaDB MaxScale、Scale ar c 和 Heimdal l Dat a 是一些可用的兼容 MySQL 的代理解决方案。这些产品提供多种附加功能，如缓存或连接多路复用。当您遇到流量突然激增并在应用程序堆栈中实施新技术时，我们建议您先从拆分 read/write 查询并测试行为和性能的基本功能入手，然后再使用可能产生意想不到的副作用的其他功能。

存档或清除旧数据

另一种提高数据库性能的方法是将历史数据卸载到另一个表、数据库或 Amazon Simple Storage Service（Amazon S3）。许多数据库保留了应用程序整个历史记录中的所有内联数据。在正常情况下，在典型的面向用户的应用程序中，这将使用户能够查看其所有历史订单。当需求激增时，应用程序上的许多活跃用户可能是新用户，或专注于下新订单。如果历史数据联机驻留在包含数十亿行的单个表中，情况会更加复杂。该表可能还有较大的索引。表数据和索引都存储在树结构中。在表中拥有更多条目需要树上的更多级别，这需要更多的 I/O 操作才能访问这些行。这会增加查找单个记录的访问时间。更重要的是，它会导致索引中大量不需要的部分驻留在数据库页面缓存（InnoDB 缓冲池）中。

将旧数据存档到单独的表、单独的数据库或 Amazon S3 可以减少最终用户的访问时间，释放宝贵的缓存，提高整体应用程序性能。在事件发生之前存档旧数据（例如，仅保留 30 或 90 天）会限制关键表上的表和索引的大小。这一更改要求对应用程序进行修改，以便仅针对明确要求查看历史数据的一小部分用户，从二级位置查询旧数据。

推迟非关键 ETL 过程

在超大规模条件下，从主数据库系统中提取 ETL 过程可能会给高事务性和并发工作负载带来稳定性风险。ETL 过程具有以下特点：

- Long-running 交易
- 宽锁
- CPU、内存和 I/O 消耗
- 与服务于关键最终用户路径的事务性工作负载争用。

可变或不可预测的 ETL 过程，例如，`INSERT INTO ... SELECT FROM ...;`）之类的查询会增加负载和争用的总体可变性，从而增加稳定性风险。

在可能的情况下，推迟或减少 ETL 过程的运行频率，尤其是在它们不提供关键功能的情况下。对于关键 ETL 过程，对其进行修改，使其在有界限的工作单元中运行，例如，处理固定行数的批处理（例如，使用 LIMIT 子句），使用单独的事务，或在较长时间内拉取少量数据以减少数据库实例的峰值资源需求。

关闭不太重要的应用程序功能

在处理激增的请求时，适度降低用户体验或删除非核心功能，可以为关键功能节省数据库资源。这可能需要稍微改变客户体验，但不同的流程总比站点崩溃要好。也许可以暂时禁用总是进行数据库调用的站点首页上的个性化设置。或者，您可以在选择产品时停止向老客户提供定制优惠。暂时关闭功能可以使页面无需访问数据库即可被缓存或传送。

其他示例包括具有轮询机制的前端，用于检查需要数据库调用的数据更改。减少轮询频率将立即减少数据库调用。如果用户界面需要分页，或为用户提供检索排序结果集的选项，而这些结果集又离最前面的结果较远，那么数据库调用的开销就会逐渐增加。限制结果集中的页数，以保护数据库免受开销最大的数据库调用。在应用程序层中使用功能标志，使运营团队能够随着应用程序或数据库负载的增加而关闭或降级功能。

参数调优

除了与连接相关的参数外，您还可以调整一些参数，以提高兼容 Amazon Aurora MySQL 的版本集群在面临高速增长时的性能。更改任何数据库参数时，请务必使用以下最佳实践：

- 一次更改一个参数，以便衡量更改的影响。
- 某些参数在更改后可能需要一段预热期才能显现效果。在观察和衡量与 Aurora MySQL 兼容版集群的性能时，请考虑这一点。
- 避免使用错误的参数值，因为这可能会阻止数据库实例启动。

这些参数包括：

- `sync_binlog`
- `table_open_cache`
- `innodb_sync_array_size`
- `innodb_flush_log_at_trx_commit`
- `autocommit`
- `tmp_table_size`
- `max_heap_table_size`

有关配置这些参数的指南，请参阅 AWS 文档中的 Aurora MySQL 配置参数、A urora MySQL 中的 MySQL 功能推荐和 Aurora MySQL 中的临时表行为。

资源

- 采用云原生架构之旅系列：#1 - 迎接应用程序的高速增长（博客文章）
- 采用云原生架构之旅系列：#2 - 最大限度提高系统吞吐量（博客文章）
- 使用 Amazon RDS 代理
- 缓存策略
- 打开和关闭 Performance Insights
- InnoDB 存储引擎
- Aurora 副本
- MariaDB 连接器/J
- MariaDB MaxScale
- ProxySQL
- Heimdall Data

文档历史记录

下表介绍了本指南的一些重要更改。如果您希望收到有关未来更新的通知，可以订阅 RSS 源。

变更	说明	日期
初次发布	—	2022 年 10 月 5 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。