MySQL Cluster 5.1 新特性技术白皮书

2006 年 9 月

北京万里开源软件有限公司

北京市朝阳区建国门外大街甲 12 号新华保险大厦五层　　电话：(010) 65054020　　网站：www .greatlinux.com

引 言

MySQL Cluster 是一个高性能、可扩展、集群化数据库产品，其研发设计的初衷就是要满足电信业中许多的业界最严酷应用要求。这些电信应用中经常要求数据库运行的可靠性要达到 99.999%。自从 2004 年开始 MySQL Cluster 发布以来，其新特性的变化就不断的被更新到技术白皮书中。这增加了 MySQL Cluster 在新的应用领域、市场、行业中的需求量。MySQL Cluster 目前已经不仅仅应用于传统的电信业务中，如 HLR(Home Locator Registry)或 SLR(Subscirber Locator Registry)，它还被广泛的应用在 VOIP、网络计费，session 管理，电子商务网站，搜索引擎，甚至是传统的内勤应用中。

在这篇文章中，我们将介绍 MySQL CLuster 5.1 的新特性。

新特性简介

在 MySQL 5.1 中，很多新特性和新功能已被引入到倍受客户青睐的 MySQL Cluster 里，这些特性和功能包括：

1）支持基于磁盘的数据存取

2）行级复制

3）快速添加/删除索引

4）高效率的变尺寸 record

MySQL Cluster 架构简介

在开始详细阐述 MySQL Cluster 新特性之前，有必要快速回顾一下 MySQL Cluster 产品的架构及其工作原理。

MySQL Cluster 是一个以独特的 shared nothing 架构和标准 SQL 接口构建的高可用数据库产品。整个集群由多个节点构成，通过节点间的切换以确保了系统在出现节点或网路故障时的持续高可用性。MySQL Cluster 使用专有的存储引擎来存取数据，这套引擎有一组数据节点构成，可以通过 MySQL Server 用标准 SQL 来访问。

MySQL Cluster 容许几个数据节点同时出现故障，并且在重新配置集群的设置之后可以屏蔽掉这些故障。这种自我修复的特性、集群数据存储分布和按应用类别分区存储的透明性形成了一个简洁的编程模型，这个模型使数据库开发人员在不需复杂的底层代码编写情况下，很容易在他们的应用程序中获得系统的高可用性。

MySQL Cluster 由三类节点组成：

1、数据节点

数据节点用以存储所有属于 MySQL Cluster 的数据。这些数据在数据节点之间被复制以保证在一个或多个节点出现故障时集群仍然持续可用。而且数据节点也管理数据库的事务处理。随着数据复制份数的增加整个系统的数据冗余性相应提高。

2、管理节点

管理节点用以控制系统启动时的初始配置，在集群设置发生改变时又被重新利用。通常只需配置一个管理节点，然而为了排除单点故障需要，有可能的话，尽量增加管理节点的数量。

管理节点只在集群启动和发生配置变化的时候起作用，集群启动以后，不论管理节点处于什么状态， 整个集群都将保持其在线和可用状态。

3、MySQL Server 节点

服务节点用于存取已集群数据节点上的数据，给软件开发者提供了一个标准的 SQL 语言编程接口。服务节点负责向数据节点传送访问请求，这使 mysql 使用者无需知道具体的集群过程，也无需进行数据库操作的底层编程。特别是增加服务节点数量，可以提高集群系统性能。这种设计自然就给 MySQL 增加可扩展性、数据规模和系统性能提供更广的方法和措施。

下图 1 是一个基本的 MySQL 集群配置，包括：

一个 MySQL 服务节点

一个管理节点

四个数据节点（组成两个数据节点组）

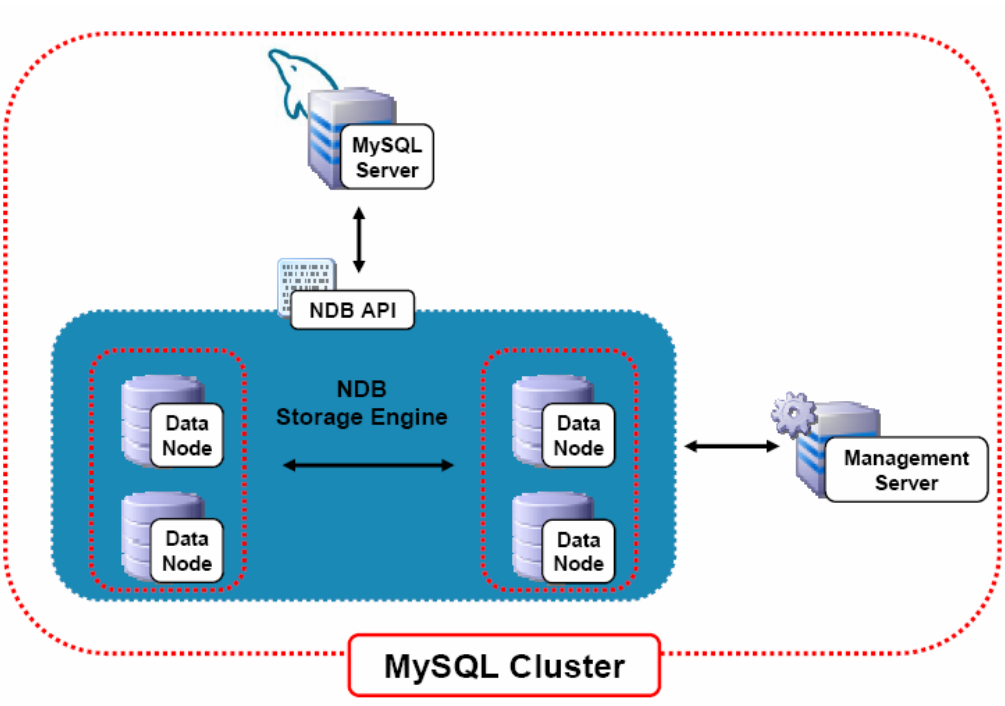

图 1

图 2 中描述了一个为了提供性能和规模而扩展了的 MySQL Cluster 配置。为了实现系统全冗余，增加了两个 Server 节点和一个管理节点，使整个集群的配置变成：

三个 Server 节点

两个管理节点

四个数据节点（组成两个数据节点组）

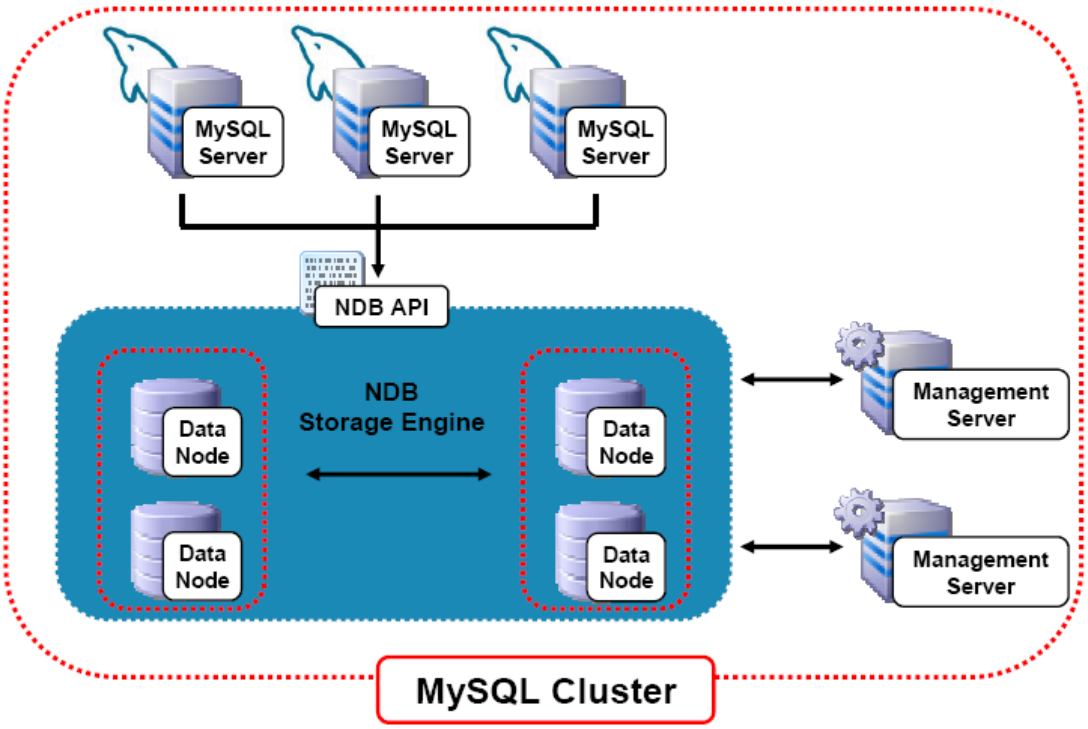

图 2

如图 2 所示，每一个 MySQL Server 节点都被连接到所有的数据节点，而且在同一个 MySQL 集群中可以有多个 Server 节点。在 Server 节点上的所有事务处理过程的执行都由数据节点组来最终完成。这意味着在一个 Server 节点上执行的事务只要一完成，通过已连接到集群的所有 Server 节点都可以马上查到。

这种基于节点的 MySQL 集群架构是为实现 MySQL 的高可用性设计的。

- 如果数据节点出现故障，Server 节点就可以用其他的数据节点来处理该事务请求。
- 在一个数据节点上的数据可以在多个节点上进行复制，如果其中一个数据节点出故障，至少有一个其他的数据节点上存储着与该故障节点同样的信

息。

- 管理节点可以被关掉和重起而不影响数据节点上的正在运行的操作。如前所述，管理节点只有在集群启动和重新配置集群的时候才起作用。

MySQL Cluster 的这种节点式设计由于把单点故障的可能性降到最低，实现了集群系统的可靠性和高可用性。任何一个节点可以被删除而不至于影响集群系统的整体功能。即使数据节点宕机，应用也能照样运行。另外，还有别的一些技术也被运用来提高 MySQL 集群系统的可靠性和抗药性。这些技术有：

- 数据在所有数据节点之间同步复制，这使在有节点宕机时系统只会出现极少的宕机时间。
- 集群的各类节点运行在不同的主机上，使集群即便在节点硬件发生故障时照样可用。
- 各类节点基于 shared-nothing 架构设计，每个数据节点有自己的磁盘和内存作为数据存储介质。
- 单点故障的可能已经被降到最小，任何节点能被停止，而不会造成数据丢失、正在操作数据库的应用中止。

使用 NySQL Server 节点连接到 MySQL 集群的应用程序系统有以下优点：

- 数据独立性，意味着即使不具备数据的物理存储知识，也能开发数据库应用程序。数据通过数据引擎存储，存储引擎复杂存储的底层细节操作，比如数据复制和自动故障切换等。
- 集群的网络结构和分布状态清晰，这意味着应用程序开发既不需考虑集群网络的操作细节，也不需考虑数据节点上的数据分布问题。
- 数据复制和分区的透明性，这意味着应用程序可以统一开发方式，而不用考虑数据被复制与否，也不用关心如何被分区。
- 标准的 SQL 接口对应用开发者和 DBA 们来说都很容易使用，并使得用户不需要任何底层的编程也能获得系统的高可用性。

这些特性使 MySQLCluster 在系统故障时可能动态的被调整，而不需要在应用程序中去修改代码。

要知道更多更细的 MySQL Cluster 架构和工作原理介绍，请查阅 MySQL Cluster 架构介绍白皮书。可在以下地址获得：

http://www.mysql.com/why-mysql/white-papers/cluster-technical.php

磁盘存取数据

MySQL Cluster 5.1 的一个最受期待的特性是集群已支持基于磁盘的数据存取。对于那些已经熟悉 MySQL 集群的人，众所共知 NDB 存储引擎大大增强了 MySQL Cluster 的性能，该引擎之前是一个 100%的内存数据库引擎。这种引擎对可以运行在内存里的数据库来说是极好的选择，现在基于磁盘的存取数据支持
使 MySQL 5.1 集群拓展了数据库规模，使 MySQL 集群用户有能力创建更大的数据库而且能对其有效的管理。

对于必须具备很高可用性，但不苛求像基于内存数据存储那样的高性能特性的数据，利用磁盘存取的方式将是最好的选择。另外，那些由于操作系统或硬件条件所致的硬性局限，使他们达到了存取性能极限的人，你现在可以考虑移植基于内存存取的库到磁盘存取的库当中，或者在支持磁盘存取数据的 MySQL Cluster 上开发新的应用。

在这一节，我们将关注一下 MySQL Cluster 中创建、移植和维护基于磁盘存取的数据表。请切记对基于磁盘存取数据的支持已经在 5.1.6bata 版中正式推出。

创建一个基于内存存取的表

通常创建 NDB 表的时候，只要发出如下的语句，表和索引都会在主存里建立起来，并且已处于就绪状态。

```
CREATE TABLE table1 (col1 int, col2 int, col3 int, col4 int,
                     PRIMARY KEY(col1), index(col1,col2)) ENGINE=ndb;
```

上面语句所建表的属性如下：

- col1 和 col2 列驻留内存，因为它们是索引的一部分，（这一点即使在使用基于磁盘的存储的 MySQL 5.1 里面也有同样的限制，原因是 NDB 索引仍然必须全部驻留于内存里，
- col2 列和 col4 列也驻留与内存，不过我们在 5.1 里面可以改成磁盘存储，

通过在 MySQL Cluster 定义一个表空间，然后把这些列指向表空间实现。

创建一个日志文件组和表空间

如前所述，利用 MySQL Cluster 的磁盘数据存取新特性时，我们首先进行一些初始步骤准备。这些步骤包括创建日志文件组和表空间。

创建日志文件组时会在每个数据节点上为存储 UNDO 日志而新建一个文件.。（请注意在 5.1 版中，只能支持一个 logfile 组，然而，从理论上讲，以后一个无限制 undofiles 数量的特性将被支持的。Undo 缓冲区的缺省值是 8MB。）

```
CREATE LOGFILE GROUP logfg1 ADD UNDOFILE 'undofile1.dat'
                    INITIAL_SIZE 16M UNDO_BUFFER_SIZE = 1M
                    ENGINE=ndb;
```

接下来我们将在每个数据节点上创建一个文件来存储基于磁盘存取表的分区。我们在表空间里面建的表都与一个 logfile 组相关联。

```
CREATE TABLESPACE tsp1 ADD DATAFILE 'datafile1.dat' USE
        LOGFILE GROUP logfg1 INITIAL_SIZE 12M    ENGINE=ndb;
```

创建基于磁盘存取的表

由于我们已经创建了 logfile 组和表空间，现在就可以创建第一个基于磁盘存取的表了。

```
CREATE TABLE table3 (pk_col1 int, fname_col2 VARCHAR(24), lname_col3
                    VARCHAR(255), ssno_col4 int , PRIMARY KEY (pk_col1),
                    INDEX (pk_col1, ssno_col4)) TABLESPACE tsp1 STORAGE
                    DISK ENGINE=ndb;
```

上面所建表的属性如下：

- pk_col 列和 ssnp_col4 列驻被保存到内存，因为它们是索引的一部分，这一点前面已有所述，这是源于整个 5.1 版本 MySQL 其索引必须存于内存中。
- fname_col2 列和 fname_col3 列不属于索引，将被保存到磁盘上

磁盘上的数据管理与维护

如果以后某一时候需要给表空间增加数据文件，我们可以用下面的命令完成：

```
ALTER TABLESPACE tsp1 ADD DATAFILE 'datafile2.dat' INITIAL_SIZE
                    16M ENGINE=ndb;
```

添加另一个 undofile 到 logfile 组可以通过下面的方法来做：

```
ALTER LOGFILE GROUP logfg1 ADD UNDOFILE 'undofile2.dat'
                    INITIAL_SIZE 16M ENGINE=ndb;
```

你也可以给在磁盘上建好的表添加一个索引，具体过程如下：

```
CREATE TABLE table3 (col1 int, col2 int, col3 int, col4 int, PRIMARY KEY
(col1))
                    TABLESPACE tsp1 STORAGE DISK ENGINE=ndb;
ALTER TABLE table3 ADD INDEX col1and2_index (col1, col2),
ENGINE=ndb;
```

这两步操作将创建适当的索引，并将相应的索引列移到内存，而把其他的；列存于磁盘上。

关于表空间和 logfile 组方面需要注意的是，在表空间和 logfile 组被删除之前，必须先删除表空间和 logfile 组中的对象。否则将产生与下面相似的结果：

```
DROP TABLESPACE tsp1 ENGINE=ndb;
ERROR 1507 (HY000): Failed to drop TABLESPACE
DROP LOGFILE GROUP logfg1 ENGINE=ndb;
ERROR 1507 (HY000): Failed to drop LOGFILE GROUP
```

对基于磁盘的集群结构的管理可以简要概括如下：

- 如果相依赖的表空间存在则 logfile 组不能被删除
- 如果相依赖的数据文件存在则表空间不能被删除
- 如果表空间中相依赖的表存在，则相应的数据文件就不能从表空间中删除。

因此，在开始上述维护操作之前，我们必须先删除数据表。

```
DROP TABLE table2;
DROP TABLE table3;
```

然后删除和表空间相关联的数据文件。

ALTER TABLESPACE tsp1 DROP DATAFILE 'datafile1.dat' ENGINE=ndb;

ALTER TABLESPACE tsp1 DROP DATAFILE 'datafile2.dat' ENGINE=ndb;

最后删除表空间和 logfile 组

DROP TABLESPACE tsp1 ENGINE=ndb; DROP LOGFILE GROUP logfg1 ENGINE=ndb;

移植内存表到磁盘表

现在我们来看怎么移植一个内存表到一个磁盘表，首先，我们先创建一个简单的内存表：

CREATE TABLE table4 (col1 int(5), col2 int(5), col3 int(5), col4 int(5), PRIMARY KEY(col1), INDEX(col1,col2)) ENGINE=ndb;

所建表的属性如下：

- col1 和 col2 列由于是索引，被保存到内存。
- col3 和 col4 列因为没有定义与之对应的表空间，因此也被保存于内存中。

如前所述，存于磁盘的数据需要有一个表空间和 logfile 组，因而我们先来创建这两项：

CREATE LOGFILE GROUP logfg1 ADD UNDOFILE 'undofile1.dat' INITIAL_SIZE 16M UNDO_BUFFER_SIZE = 1M ENGINE=ndb;

CREATE TABLESPACE tsp1 ADD DATAFILE 'datafile1.dat' USE LOGFILE GROUP logfg1 INITIAL_SIZE 12M ENGINE ndb;

现在，我们执行下面的 ALTER 命令来移植内存表到利用上面定义的表空间的磁盘表。

ALTER TABLE table4 TABLESPACE tsp1 STORAGE DISK ENGINE=ndb;

移植后的表属性如下：

- col1 和 col2 列被保存于内存中，因为他们仍是索引的一部分。
- col3 和 col4 列将被保存到磁盘，因为他们不属于索引。

参数

以下是当使用基于磁盘的数据时，你应该添加到集群配置文件 config.ini 的 NDBD 部分的一些参数。

DiskPageBufferMemory= X

这个参数定义磁盘数据页缓存大小，每一页分配 32K bytes。

SharedPoolMemory= X

这个参数设定能被各种资源所使用的空间大小，目前使用共享池的对象包括：

- 表空间
- logfile 组
- 数据文件
- undo 文件
- 分区信息
- log buffer waiter
- log sync waiter
- 页请求
- undo 缓冲

磁盘数据存取的注意点：

- 现在还不支持磁盘索引
- 可变大小的表属性会消耗数据文件和 page buffer 缓存的已分配空间。
- 当分区中的所有存储页被释放出来以后，分区就不再让与一个表空间。例如当你做随机插入和删除的时候，分配给表的磁盘空间（从表空间中分配得）将被在已使用的最大空间处。值得注意的是这种局限性在内存数据里面也存在。
- 关于磁盘数据崩溃的可能性方面，目前 5.1 版本集群里面已经不支持 OPTIMIZE TABLE，而这在 MyISAM、InnoDB 和 BDB 存储引擎里面各自都还存在。
- 只能定义一个 logfile 组，这使在不同的数据表之间分配负载的功能受到了一定的限制。

- 5.1 版本集群自动分区目前还不支持
- 在每个 CREATE LOGFILE GROUP、 ALTER LOGFILE GROUP、CREATE TABLESPACE or ALTER TABLESPACE 语句里面都必须加上一个 ENGINE 字段，ENGINE 的值可以为 NDB 和 NDBCLUSTER。

复制

MySQL Server 5.1 引入了行复制的功能。而之前的 MySQL 集群都不能利用 4.1 和 5.0 版 MySQL 里面缺省的基于语句复制的功能。现在随着 MySQL Cluster 里面对行复制的支持，使从集群到集群或其他 MySQL server 的复制变成了现实。同时也使 MySQL 应用规划中进行如下的主/从配置成为可能：

- 从 MySQL 集群到另一个 MySQL 集群的复制
- 从 MySQL server （MyISAM，InnoDB 等）到 MySQL 集群的复制
- 从 MySQL 集群到 MySQL server 的复制

很显然，如果要做到最好的高可用性，从 MySQL 集群复制到另一集群将是最好的选择。

在 MySQL 的复制功能经常应用的所有场合，MySQL 集群现在也都可以被采用。一些众所周知的需要应用复制功能的理由是：

- 能扩展 MySQL 数据库规模，提高性能
- 被复制出的数据库可用于实现故障切换功能
- 增加数据库读写的可扩展性
- 被复制出的数据库在升级，测试，备份等维护操作中都可以使用
- 复制功能增强不论是数据中心还是广域网中的数据库的高可用性。

首先，不管你是否用过 MySQL 集群，让我们回顾一下关于 MySQL 复制功能的一些基本知识，复制的实现过程中要有一个一个主服务器和一个从服务器，其中主服务器是被操作源，提供要被复制的数据，从服务器是主服务器数据的接受者，在下图 3 中描绘了复制过程的这种配置架构。

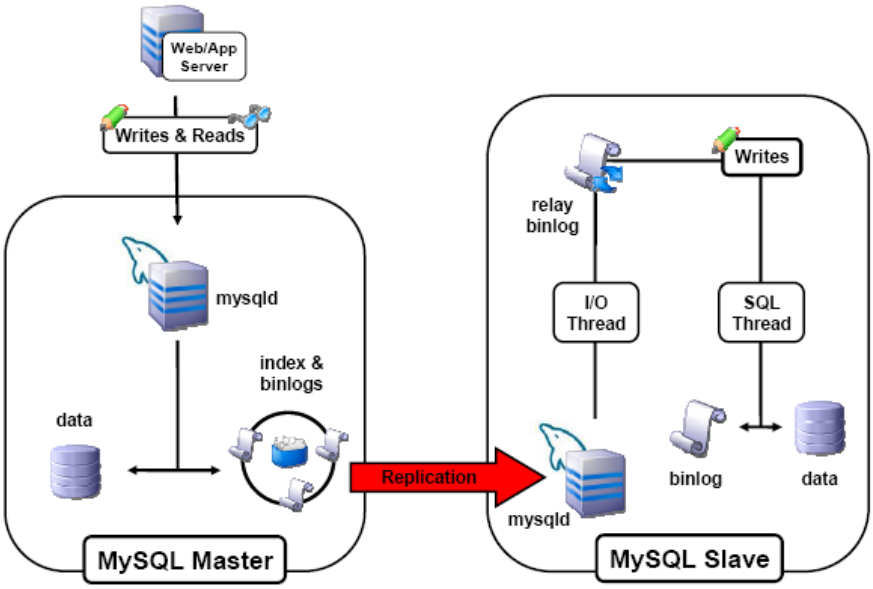

图 3

虽然 MySQL Cluster 里复制功能的应用从结构上看并不庞大，但还是有些不同之处需要说明一下。比如 MySQL 集群的内部架构相当一部分依赖于 NBD 存储引擎，而 NDB 则很显然是不支持标准 SQL 的，从一个 MySQL 集群到另一个集群的复制过程如图 4 所示：

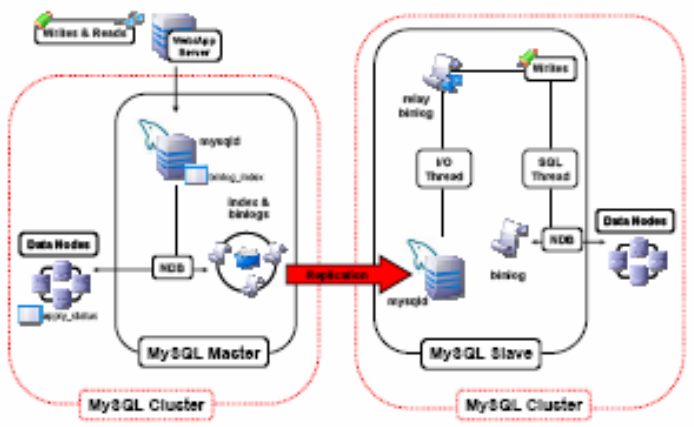

图 4

由上图可见，复制功能是主集群连续的把各个状态写到从集群日志并把数据存到从集群的过程。这个过程是由一个叫 NBD 二进制日志注入的线程完成的，这个线程运行在每个 MySQL server 上，其作用是产生二进制的日志文件（也称 binlog）。它保证了产生二进制日志文件的集群中发生的所有改变（并不只是那些由 MySQL

server 操作所致的改变）都会以正确的顺序被插入相应的二进制日志文件中。这一点很重要，因为 NDB 存储引擎支持 NDB API，NDB API 让用户可以绕过 MySQL server 和一般的 SQL 语句，开发与 NDB kernel 直接接口的应用程序。

图 4 的说明也涉及到了主从集群之间通讯数据流或数据队列，这些数据的流向常被当作‘复制通道’。在图 4 之中已用红色箭头标出。

复制通道

一个完整的复制通道要求有两个 MySQL server 来作为复制服务器（一个为主的，一个是从的）

每个集群中用来做复制的 MySQL server 在整个复制环境里所有参与复制的 MySQL server 中都必须是唯一的（就是说不能让主从集群中复制服务器共用一个 ID）。这一点可以通过使用—server-id=id 启动项去启动每个 MySQL server 来做到，此时 id 是一个不能重复的整数。

参与到复制过程的 MySQL server 在所支持的复制协议版本和 SQL 标准集上必须是互相兼容的。保证这种兼容性的一个最简单也最容易的办法就是所用又来做复制的 MySQL server 都使用同一版本 MySQL。

为方便阐述下面的章节，我们假设复制过程中的从 MySQL server 或从 Cluster 完全只是主 MySQL Server 或主 Cluster 的复制，而且没有任何其它与此过程相反的事务操作被提交。

schema 和表的复制

经常应用复制功能的集群必须用到一个名为 cluster 的数据库中大量专有表，该 Cluster 数据库位于复制通道中的 Master 和 Slave 服务器的 MySQL server 节点上（不管 Slave 服务器是单机还是 Cluster）。Cluster 数据库在 MySQL 安装时创建，该数据库中有一个存储二进制日志索引数据的表 binlog_index。Binlog_index 表位于每一个 MySQL Server 节点上，而且没有被集群化到 Cluster 中，它使用的时 MyISAM 引擎。这就是说，binlog_index 必须在属于 master 集群的所有 MySQL Server 节点上被创建。Binlog_index 的表结构如下：

```
CREATE TABLE `binlog_index` (
```

```
`Position` BIGINT(20) UNSIGNED NOT NULL,
`File` VARCHAR(255) NOT NULL,
`epoch` BIGINT(20) UNSIGNED NOT NULL,
`inserts` BIGINT(20) UNSIGNED NOT NULL,
`updates` BIGINT(20) UNSIGNED NOT NULL,
`deletes` BIGINT(20) UNSIGNED NOT NULL,
`schemaops` BIGNINT(20) UNSIGNED NOT NULL,
  PRIMARY KEY (`epoch`))  ENGINE=MYISAM DEFAULT CHARSET=latin1;
```

请注意一点，MySQL 5.1.8 以前，cluster 数据库被命名为 cluster_replication.

另一个专有表叫 apply_status，被用来保存已经从主服务器复制到从服务器的操作记录。在 apply_status 表中的数据并不指到（从）集群的任何一个 SQL 节点，因而 apply_status 能使用 NDB Cluster 存储引擎。如下所示：

```
CREATE TABLE `apply_status` (
  `server_id` INT(10) UNSIGNED NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  PRIMARY KEY USING HASH (`server_id`))  ENGINE=NDBCLUSTER
DEFAULT CHARSET=latin1;
```

Binlog_index 和 apply_status 表在独立的数据库中被创建是因为这两个表不应该被复制。创建和维护两个表中的任何一个正常情况下都不需要用户干预。这两个表都由 NDB 注入线程维护。这使主 MySQL server 节点保持因 NDB 存储引擎执行所致的变化而更新。NDB 二进制日志注入线程直接从 NDB 存储引擎
接收运行事件。NDB 注入线程负责收集集群中所有的数据操作事件，并确保所有的数据变化，数据插入，数据删除被记录在 binlog_index 表中。从服务器上的 I/O 线程将把主服务器上的 binlog 转化成从服务器上的 relay log。

Replication 建立

为复制而在 MySQL Cluster 上做的准备包括以下几步：

1、在主集群上创建具有适当权限的 slave 用户

```
GRANT REPLICATION SLAVE
```

ON *.* TO 'slave_user'@'slave_host'

IDENTIFIED BY 'slave_password';

这里 slave_user 是 slave 用户的用户名，slave_host 是复制服务器的主机名或 IP 地址，slave_passward 是给复制用户的密码。

出于安全方面的原因，使用一个专用的复制用户是更可取的做法，

2、配置从服务器和主服务器连接，使用从服务器上的 MySQL 客户端，发出 CHANGE MASTER TO 语句：

CHANGE MASTER TO

MASTER_HOST='master_host',

MASTER_PORT=’master_port,

MASTER_USER='slave_user',

MASTER_PASSWORD='slave_password';

这里 master_host 是参与复制的主服务器的主机名或 IP 地址，master_port 是提供给从服务器以连接主服务器，slave_user 是在主服务器上为从服务器连接而建的用户名，slave_password 是上一步建立的 slave 用户的口令。

你也能通过修改从服务器上 my.cnf 文件中的启动选项来配置 slave 和 master 之间的连接。为了和上面用 CHANGE MASTER TO 语句的例子效果保持一致，下面的信息应该被写到从服务器的 my.cnf 中。

[mysqld]

master-host=master_host

master-port=master_port

master-user=slave_user

master-password=slave_password

3、为了提供复制后的备份兼容性，在开始复制过程之前，你也需要增加 ndb-connectstring 选项到从服务器的 my.cnf 文件。

ndb-connectstring=management_host[:port]

4、如果主集群已经在用了，你能创建一个主集群的备份，然后加载这个备份到从服务器上以节约同步从服务器和主服务器的时间

5、如果在复制方案的从服务器上你不使用 MySQL cluster，你能在 shell 下通过在主服务器上执行下面的命令做一个备份。

mysqldump --master-data=1

6、接下来拷贝导出的数据到从服务器。完成之后你就可以使用 mysql 客户端从导出文件把数据导入从数据库：

mysql -u root -p db_name < dump_file

此处 dump_file 是使用在主服务器上用 mysqldump 导出的文件，db_name 是被复制的数据库名。

如果你是用这种方式拷贝数据到从服务器，你必须确保从服务器启动的时候在命令行用了—skip-slave-start 选项，或者在从服务器的 my.cnf 文件中已包含 skip-slave-start，以便使从服务器不去连接主服务器而在数据还没完全加载到从服务器时开始复制。一旦数据加载完成，就可以开始做下一节说明的另外几个步骤的了。

确保每一个 MySQL server 作为一个复制方案中的主服务器方法的是给 MySQL server 配置一个唯一的 ID，然后打开 binlog 功能、行级复制功能。这些配置项也可以在 my.cnf 配置文件里面设置。或者在启动主 MySQL server 的过程中在命令行指定。

开始复制

这一节说明使用单复制通道来复制 MySQL Cluster 的过程。

1、通过 shell 命令行发出以下命令启动主服务器 MySQL Server

mysqld --nbdcluster --server-id=id \ --log-bin --binlog-format=row &

这里 id 是这台主服务器的 ID，上述命令启动 mysqld 进程，打开 binlog 功能，使用特定的行级日志格式。

2、启动复制方案中的从服务器

mysqld --ndbcluster --server-id=id &

这里 id 是从服务器的唯一 ID，在从服务器上不必打开 binlog 功能。

要注意的是，除非你想让复制过程在系统启动之后马上就开始，否则你要在命令行里添加—skip-slave-start 或者在 my.cnf 文件里面包含 skip-slave-start 项。

使用了这个选项之后，正如下面第 4 步所述，复制要到 START SLAVE 命令被发出之后才正式开始。

3、有必要使从服务器和主服务器的 binlog 同步，如果 binlog 在主服务器上还没运行，在从服务器上执行下面的语句。

```
CHANGE MASTER TO
MASTER_LOG_FILE='',
MASTER_LOG_POS=4;
```

这个操作使从服务器开始从复制的开始点读主服务器的 binlog。

4、最后，你必须让通过在从服务器 MySQL 客户端去执行下面的命令来启动复制过程。

```
START SLAVE;
```

这一步执行之后就开始了从主服务器到从服务器的复制数据传送。

MySQL 集群复制另外要注意的地方

下面是使用 MySQL 5.1 的 Cluster 做复制时现在的一些局限：

- 数据定义语言（DDL）例如 CREATE TABLE, DROP TABLE, and ALTER TABLE 是被记录在发出这些执行语句的 MySQL server 节点的二进制日志中。
- 参与复制的 MySQl server 节点在使用 ndb_restore 来发现和建立集群表的复制之后必须启动或被重启。另一种方法是，你可以在集群的所有数据库上发出 SHOW TABLES 命令来达到同样的目的。
- 同样，当使用 CREATE SCHEMA 语句时，新的数据库并不是自动可被 MySQL Server 发现的。因此，这条语句必须在参与集群的 MySQL server 上都执行一遍。
- 用—initial 选项重启集群将导致 GCI 和元数据重新回到 0（这是普遍情况，并不只限于集群复制中）。参与复制的 MySQL Server 在这种情况下应该重复复制过程。之后，你要使用 RESET MASTER 和 RESET SLAVE 语句来清除不正确的 binlog_index 和 apply_status 表。

注：关于两双通道复制、双通道中使用 fail over 和应用了复制的 MySQL 集群怎么配置备份方面，请参考 MySQL 5.1 Manual。

http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-replication.html

快速添加和删除索引

MySQL Cluster 5.1 引入了新的功能，添加索引的时候不中断活动事务，或不像以前版本那样在添加索引的时候引起系统资源瓶颈。让我们回顾一下在 MySQL4.1 和 5.0 上是怎么添加索引的。

1、我们创建一个有两个相关索引(index 1 和 index 2)的表（table_1）

2、第二步、一个临时表（temp table）被创建。

3、表 table_1、索引 index1、index2 和新的索引 index3 上的数据被创建。

4、一旦上述过程完成，原始表 table_1 被删除，临时表被重新命名为 table_1.

图 5 是 MySQL Cluster 4.1 和 5.0 版的索引创建过程描述。

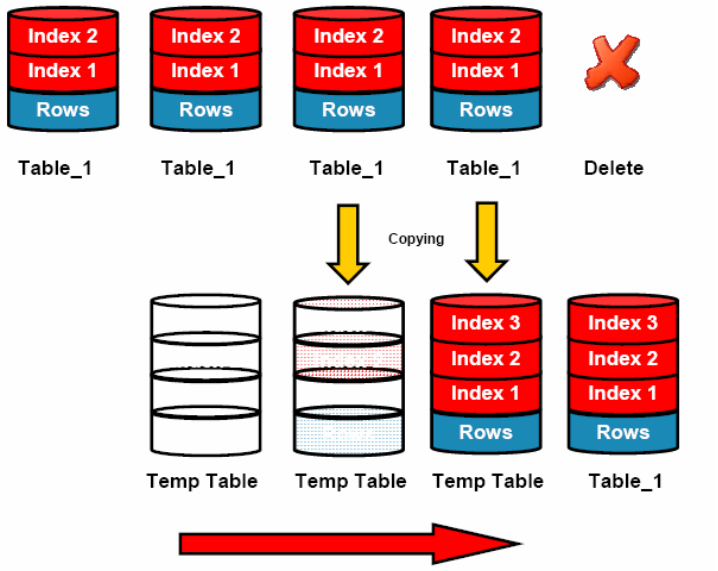

Figure 5

图 5

MySQL Cluster 5.1 中添加索引的过程如图 6 所示，添加的过程非常快。这意味着不需要再创建临时表，也不需要重新创建和删除数据。使用快速 ADD INDEX 的好处是表的维护操作加快了，内存和磁盘方面的要求降低了，因此你在维护表的

时候不再必须有足够的存储空间来维护表的双份数据拷贝，而且集群也更能适应它所支持的应用的变化需求。

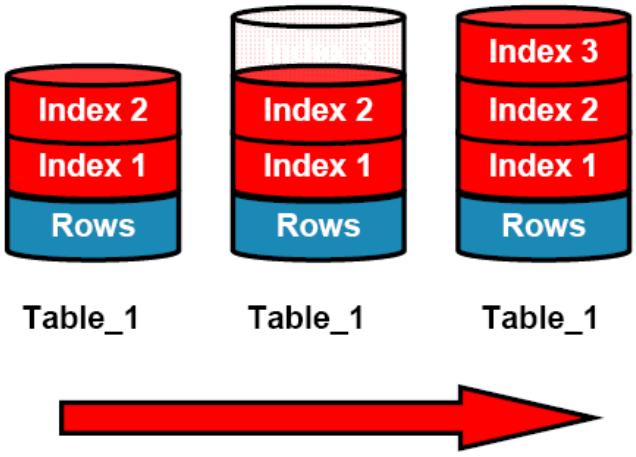

Figure 6

图 6

下面是一个在 5.0 和 5.1 上创建和删除索引性能差别的例子，正如你所见，5.1 要快将近 4 倍。

Version 5.0

```
mysql> CREATE INDEX b ON t1(b);
Query OK, 1356 rows affected (2.20 sec)
Records: 1356 Duplicates: 0 Warnings: 0
mysql> DROP INDEX b ON t1;
Query OK, 1356 rows affected (2.03 sec)
Records: 1356 Duplicates: 0 Warnings:
```

Version 5.1

```
mysql> CREATE INDEX b ON t1(b);
Query OK, 1356 rows affected (0.58 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> DROP INDEX b ON t1;
Query OK, 1356 rows affected (0.46 sec)
Records: 1356 Duplicates: 0 Warnings: 0
```

删除索引时结果也相似。

高效的可变大小的行记录

在以前的 4.1 和 5.0 版集群中，变大小的行占用比它所需要大的空间。而现在的 5.1 版中，变大小的行以更高效的方式处理。着意味着只有实际记录在行里面的数据才占用内存。这带来的好处是没有内存空间浪费，没兆字节空间能存储更多的行，内存数据中能管理更大的数据集。这方面在新旧版集群之间的差别如图 7 所示。请注意一点，正如前面提到的，磁盘数据不支持内存表的这种高效的变大小行的处理方式，

int	int	varchar(255)	
1	2	the quick brown fox	wasted space
3	4	the quick brown fox	wasted space
5	6	the quick brown fox	wasted space
7	8	the quick brown fox	wasted space
9	9	the quick brown fox	wasted space

Version 5.1

int	int	varchar(255)
1	2	the quick brown fox
3	4	the quick brown fox
5	6	the quick brown fox
7	8	the quick brown fox
9	9	the quick brown fox

图 7

结论

本文详细探讨了 MySQL Cluster 5.1 里面引入的一些新特性，

- 支持磁盘表
- 行级复制
- 高性能的添加/删除索引
- 更高效的变尺寸行记录

随着对磁盘数据支持的引入，MySQL Cluster 现在已能支持比以前受操作系统和内存限制的规模更大的数据量。

行级复制的支持使 MySQL Cluster 现在能实现更高水平的可用性和可扩展性。这个功能可用于提高扩展性和性能：为高可用切换而复制数据库或者将复制数据库用于升级或备份等操作。MySQL 集群的复制也能被用于通过广域网实现数据中心之间的冗余部署。

索引操作性能的提高以及对变大小行记录的支持使 MySQL Cluster 成为一个更快更高效的数据库服务器。

MySQL Cluster 接下来的发展方向集中在给客户一个集群数据库方面引人注目的低 TCO，同时发展利用普通硬件和开源软件实现大规模应用的一套方法。