

# MySQL Cluster 7.0架构与新特性

上海爱可生-中国最领先的MySQL服务提供商

# 目录

1 概述 .....	3
2 新特性简介 : .....	3
3 MySQL 集群架构概览 .....	3
4 增强的可扩展性和性能 .....	9
4.1 多线程数据节点 .....	9
4.2 在线添加节点 .....	9
4.3 磁盘数据的多线程访问 .....	12
4.4 改进的大记录处理能力 .....	13
4.4.1 长信号事务 .....	13
4.4.2 压缩读 .....	14
5 Windows 平台 .....	15
6 简化的 Cluster 监控和管理 .....	15
6.1 MySQL 集群的快照备份选项 .....	15
6.2 配置数据缓存 .....	17
6.3 模式变动相关的事务 .....	17

## 1 概述

MySQL Cluster 是一个具有高性能、可扩展性、集群化的数据库产品，其研发设计的初衷就是要满足电信业中许多业界最严酷应用的要求。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(Subscriber Locator Registry)，它还被广泛的应用在 VOIP、网络计费、会议管理、电子商务网站、搜索引擎，甚至是传统的后台应用中。

在本文档中，我们将介绍 MySQL Cluster 7.0 的新特性。

## 2 新特性简介：

- 1) 可在线添加节点
- 2) 支持磁盘数据多线程访问
- 3) 改进大数据记录处理能力
- 4) 支持 Windows 平台
- 5) 支持后端电信级目录
- 6) 可配置数据缓存
- 7) 事物支持改变结构

## 3 MySQL 集群架构概览

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 是一个以独特的无共享体系架构和标准 SQL 接口构建的高可用数据库产品。系统由一系列的通信进程，或是分布于各机器上的节点构成，哪怕在服务器出现故障或是网络故障时，都可以提供一个持续可用的系统。MySQL Cluster 使用专有的存储引擎来存取数据，这套引擎由一组数据节点构成，可以通过 MySQL Server 用标准 SQL 来访问或是通过 NDB API 进行实时的访问。

NDB API 是 MySQL Cluster 使用的面向对象的应用程序接口，它实现了索引，扫描，事务，事件处理。NDB 事物是遵循 ACID 准则的。集群正是通过这种方式提供了将多个操作组成一组，要么全部执行成功（提交），要么作为一个整体失败（rollback）。

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无需进行复杂的底层代码编写的情况下，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 (Data Nodes)** 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。应用程序使用 NDB API 直接访问数据节点，而不是通过 MySQL 服务器。

2、**管理节点 (Management Server Nodes)** 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，请增加管理节点的数量。以提供管理操作的高可用性。

管理节点只在集群启动和系统重配置后起作用，集群启动以后，无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 节点 (MySQL Server Nodes)** 用于存取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。MySQL 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。特别是通过增加 MySQL 服务节点数量，就可以提高集群系统性能。这种设计给 MySQL 增加可扩展性、数据规模和系统性能提供了更广的方法和措施。

应用程序如果对实时性能要求较高，请直接使用 NDB API，而不是通过 MySQL 节点。现在可以使用 C++、Java、HTTP 版本的 NDB API，这给开发人员提供了极大的方便，并且使得 MySQL 集群可以应用于广泛的 web 和企业级应用。

数据库在内部被分割为一个个分区，这些分区分布于数据节点之间。为了避免单点故障，至少会有两个以上的节点会用于存储特定分区的数据，这些节点组成一个节点组。

图 1 是一个基本的 MySQL 集群配置，包括：

- 一个 MySQL 节点
- 一个管理节点
- 四个数据节点（组成两个数据节点组）以提供更好的性能，容量，及可用性

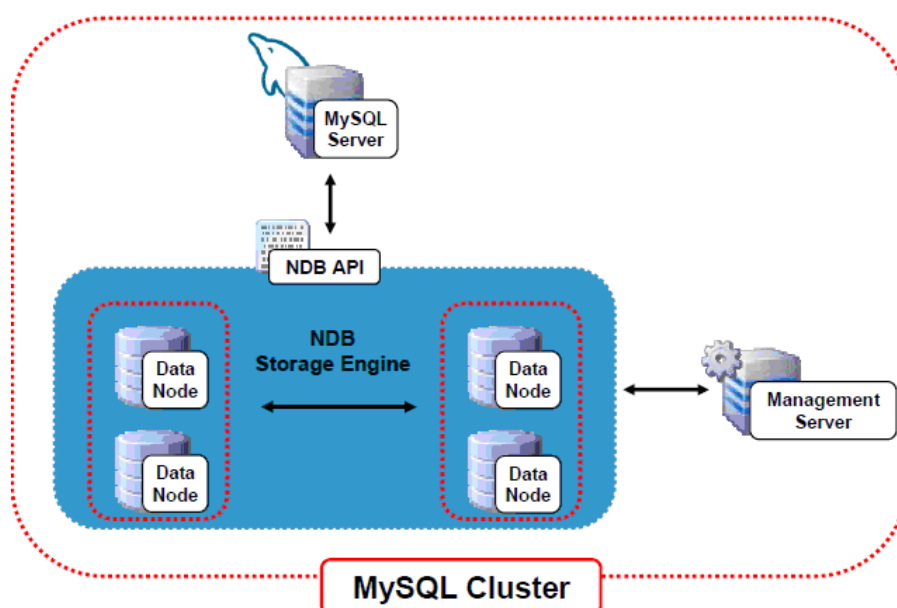


图 1

图 2 描述一个为提高性能而扩展了的 MySQL Cluster 配置。做法是：增加两个服务节点和一个管理节点，整个集群的配置构成如下

- 三个服务节点
- 两个管理节点
- 四个数据节点（组成两个数据节点组）以提供更好的性能，容量，及可用性

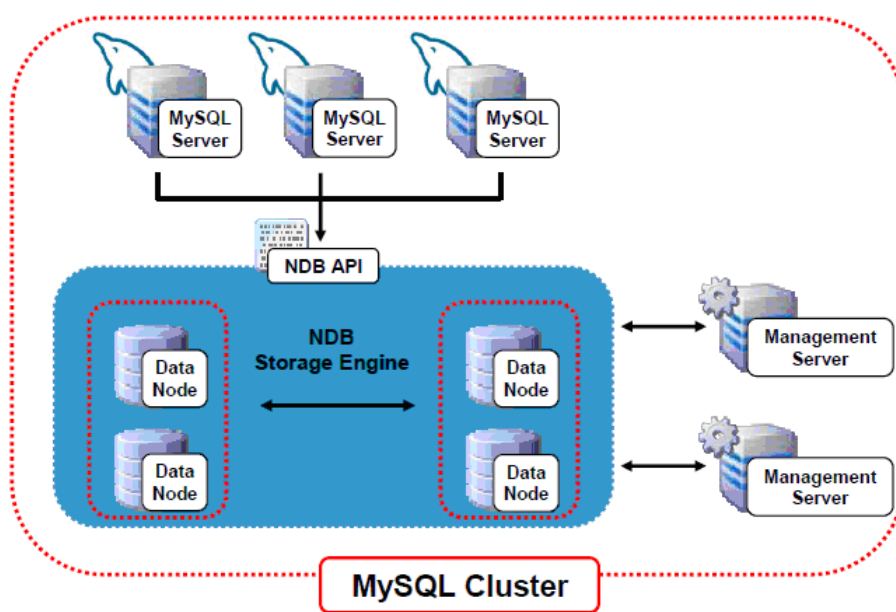


图 2

图 2 显示，每一个 MySQL 服务节点都被连接到所有的数据节点，而且在同一个 MySQL 集群中可以有多多个服务节点。在服务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。这意味着在一个服务节点上执行的事务只要一完成，通过已连接到集群的所有服务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 的高可用性而设计的。

- 如果数据节点出现故障，服务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节点组内的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启而不影响数据节点上正在运行的操作。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了最低，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都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，一些其它的技术也被运用来提高 MySQL 集群系统的可靠性和可用性。这些技术有：

- 数据在所有数据节点之间同步复制，这将大大减少系统的宕机时间。

- 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。
- 各类节点基于无共享架构设计，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。

除了提供 site 级的高可用（通过集群的冗余架构），地理冗余可以通过 2 个或是多个集群的异步复制来实现。

使用行复制，你可以将集群复制到另一个集群或其他非群集的 MySQL 数据库。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（MyISAM，InnoDB 等）的复制

如果目标是最高级别的可用性，那么从 MySQL 集群到另一个 MySQL 集群的复制使用双通道，效果将会是最理想的。

一些众所周知的需要使用复制功能的理由如下所示：

- 复制功能可以增强不论是数据中心还是广域网中的数据库的高可用性。
- 被复制出的数据库可用于实现故障切换功能
- 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
- 在不同的地理位置上提供极少的数据访问延迟
- 被复制出的数据库可用于复杂数据数据分析而不至于影响生产库。

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主服务器和一个从服务器，其中主服务器是操作源，提供要被复制的数据，从服务器是主服务器数据的接受者。在图 3 中描绘了复制过程的这种配置架构。一个单一的服务器在不同的时间可以是主或是从，在一些拓扑结构中，主和从是同一个服务器。

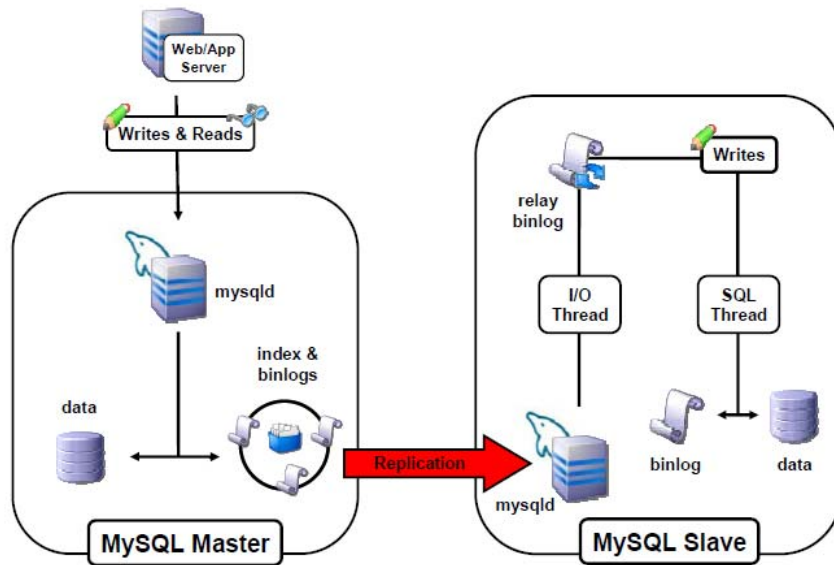


图 3

虽然在 MySQL Cluster 内部复制的结构与 MySQL 复制类似，但是还是有少许的不同之处需要解释。MySQL Cluster 和 MySQL Cluster 间的复制配置请见下图：

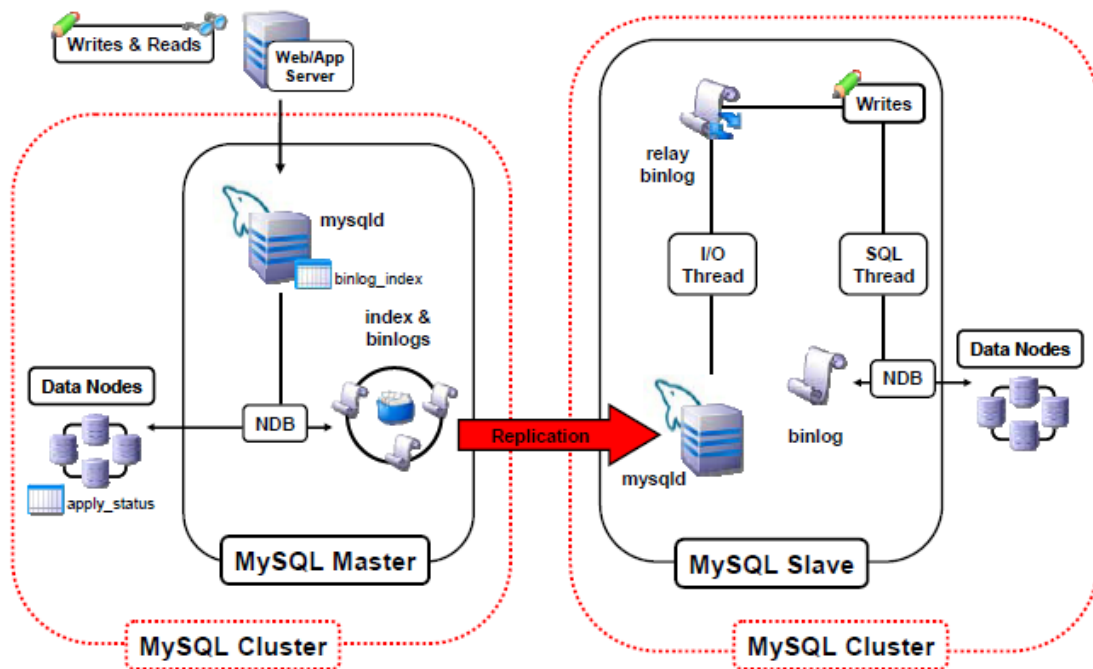


图 4

由上图可见，复制功能是主集群持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 NDB 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 内核直接接口的应用程序。

上图描述的是单 MySQL 服务器 (mysqld) 作为一个 SQL 服务器同时也作为主从复制的主；为了获得最大的复制的性能，你应该指定一台机器专用于复制。

MySQL 服务器支持基于语句级的复制 (SBR) 和基于行级的复制 (RBR)；MySQL 的复制使用的是异步的复制。请参考“MySQL Cluster Reference Guide”以得到关于 Cluster 复制的更多的信息---包括不同的主/从复制结构，冲突检测与解决，多复制通道，在线集群备份时二进制日志的同步。

在 MySQL 集群中，不是所有的数据都需要占据内存以获得相同的读/写性能的。MySQL 集群可以将一部分的数据存于磁盘上而非内存。这种特性使用你可以创建更大的数据库集群，也可以更加有效的进行管理。

那些要求高可用的，但是不要求像基于内存的数据那样需要高性能的数据可以考虑存于磁盘上。另外，因为操作系统或是硬件的缘故，内存达到使用极限的时候，可以将基于内存的数据放于磁盘上。

另一方面，那些不要求磁盘提供持久性，而需要高速的读写性能的数据可以存于临时表上。这样就提供了最大的事物处理能力同时也提供了容错能力。这种方法是管理 Session 数据的首选方法。

使用 MySQL 服务器和 NDB API 来连接 MySQL 集群有以下好处：

- 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透明，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。

另外，使用 NDB API 还有以下的好处：

- 提升性能
- 降低延迟
- 一些附加的功能不能通过 SQL 接口使用，比如表修改的通知。使用 API 就不会有这个问题。

但是，通过 MySQL 服务器连接却有以下的好处：

开发人员或是 DBA 可以使用一个标准的 SQL 接口来操作，而无需要做任何的底层编程以达到高可用的目的。

以上这些特性使得 MySQL 在出现故障的情况下，动态的配置其本身，而无需在应用程序里做编码。



## 4 增强的可扩展性和性能

### 4.1 多线程数据节点

在 MySQL Cluster 7.0 当中增加了一项新的特性，允许数据节点更好的利用单一系统中的多线程，多核或是多个 CPU。在这个发行版本里，每一个 NDB 数据节点都可以有效的利用 8 核 CPU/线程。数据由 NDB 进程的一系列线程所分割。这种设计使得各自线程可以处理尽可能多的工作量，并且大大降低了线程间的通信需求。

#### 配置 NDB 数据节点使用多线程

ndbmtbd 是一个多线程版本的 ndbd，该进程用于在 NDB 数据节点里处理表中的数据。ndbmtbd 用于多核 CPU/线程的计算机。

多数情况下，ndbmtbd 的功能与 ndbd 一样；应用程序无需知道 ndbd 已为 ndbmtbd 所取代。用于 ndbd 的命令行参数和配置参数同样可以用于 ndbmtbd。任何运行 ndbd 的操作系统都可以运行 ndbmtbd。换句话说，可以先停止在数据节点上运行的 ndbd，接着用 ndbmtbd 取代之，然后重启。这个过程，不会造成任何数据的丢失。

ndbmtbd 与 ndbd 的区别主要在两个方面：

- ◆ 必须在 config.ini 文件中为 MaxNoOfExecutionThreads 设置合适的值。如果没有这么做，ndbmtbd 会运行在单线程模式---即，效果与 ndbd 一样。
- ◆ 由 ndbmtbd 进程生成的监控文件（trace files）用于记录严重的错误信息。

#### 执行线程的个数

MaxNoOfExecutionThreads 用于决定运行 ndbmtbd 生成的执行线程的个数。

该值在 config.ini 中的[ndbd]或是 [ndbd default]中配置，且只对 ndbmtbd 起作用，

对 ndbd 不起作用。此值可以设置为 2 至 8 之间的任何整数。一般来说，你应该根据数据

- ◆ 事务协调者（不超过 1 个线程）
- ◆ 连接方式（不超过 1 个线程）
- ◆ 复制（不超过 1 个线程）
- ◆ 本地查询句柄（MaxNoOfExecutionThreads 为 2 时，该值为 1；MaxNoOfExecutionThreads 为 4 时，该值为 2；MaxNoOfExecutionThreads 为 8 时，该值为 4）

### 4.2 在线添加节点

MySQL 集群一个最吸引人的特性是它可以从一个规模小的，成本低的集群开始建设，随着数据容量的增加或是对性能的更高要求，可以通过添加刀片机或是服务器来进行集群扩展。

在之前的 MySQL 集群版本，不支持在一个处于服务状态的集群中添加一个节点。

从 MySQL Cluster 7.0 开始，允许在线添加节点组（也就是在线添加新的数据节点），不再需要关闭集群然后重新加载数据。MySQL Cluster 7.0 在数据的重新分区方面作了加强（即，将一个部分的数据从一个现有的组移动至一个新的组）。

这部分将介绍一个例子，在一个已经有一个节点组的集群中再添加一个节点组，并且使数据在两个节点组间重新分布。图 6 显示的是最初始的系统。

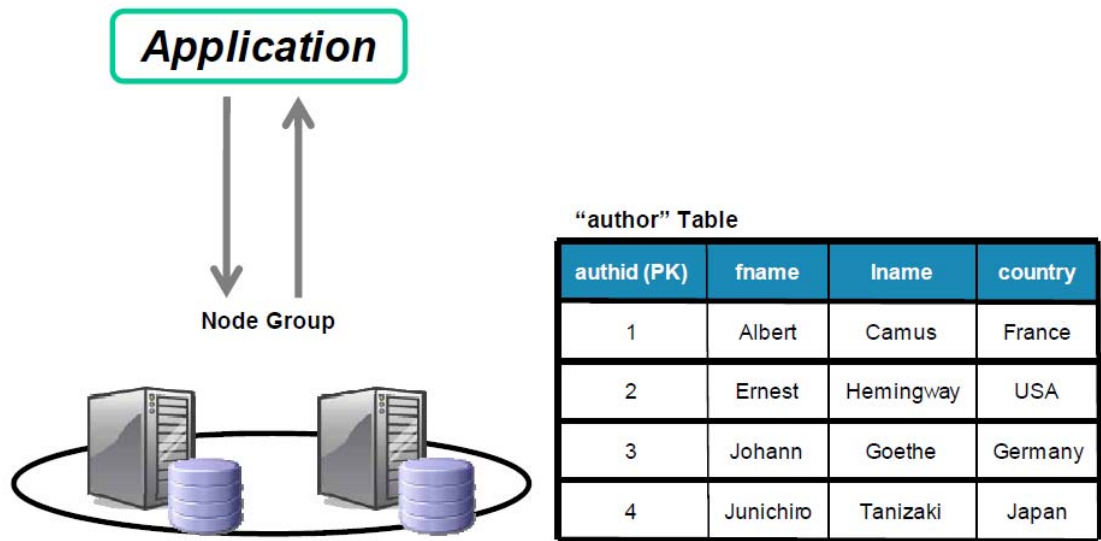


图 6

将新的节点组添加入集群的过程如下所示：

第一步：在所有的管理节点上编辑 config.ini，如图 7 所示：

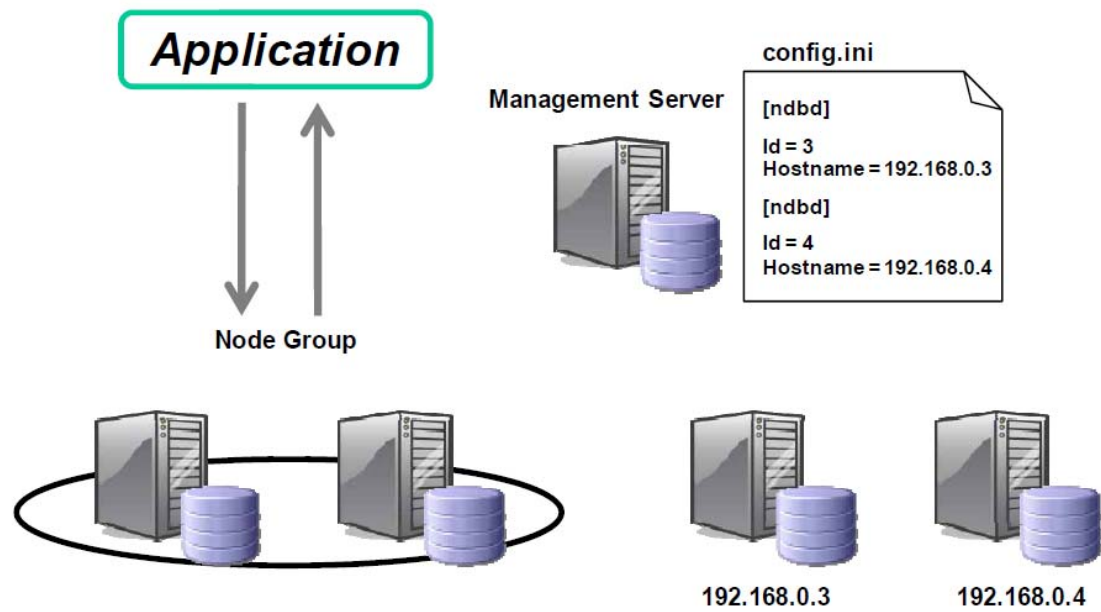


图 7

第二步：重启管理服务，原有的数据节点和 MySQL 服务器。

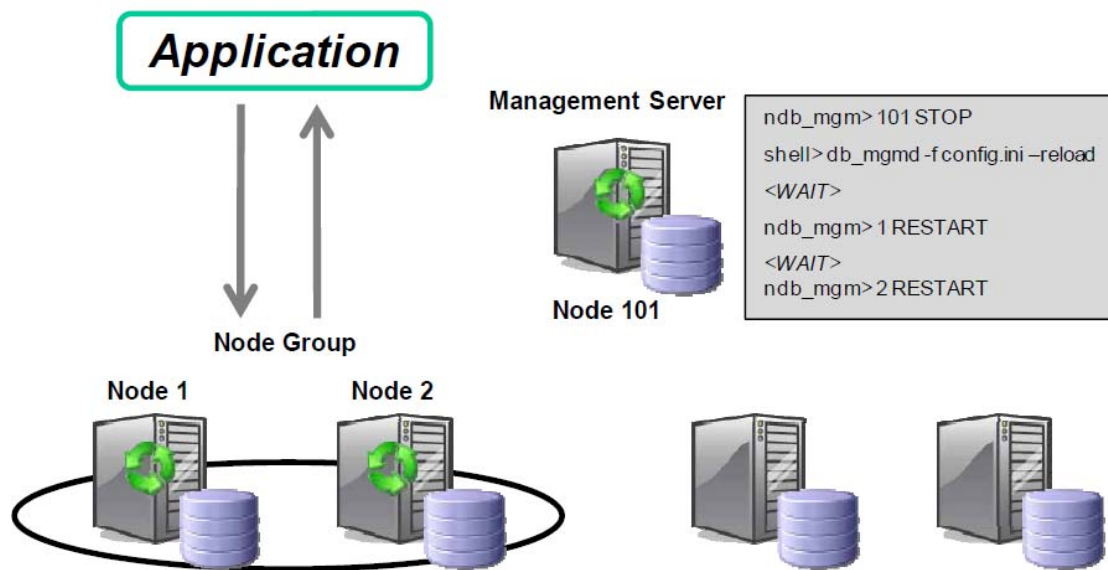


图 8

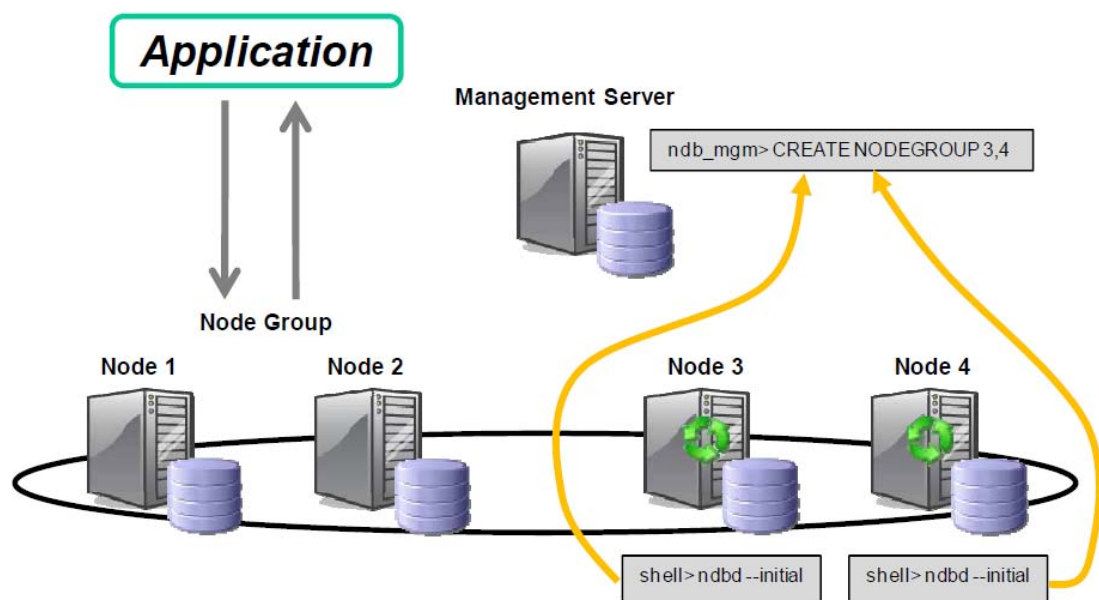
另外，所有连至集群的 MySQL 服务器都应该重启。运行如下的命令：

```
shell> mysqladmin -uroot -ppassword shutdown
```

```
shell> mysqld_safe
```

第三步：创建一个新的结点组

准备启动两个新的数据节点，并且将它们组织为一个结点组，如下图 9 所示



第 4 步：重新分配数据

现在，新的数据节点已经是集群的一部分，但是所有的数据还都是在原先的节点组当中（在节点 1 和节点 2）。注意到一旦新的节点作为新的节点组的一部分被添加，新的表将会自动的在所有的节点之间重新分配数据。

图 10 展示的是，当 MySQL 服务器上运行了相应命令以后，表数据（磁盘或是内存）是怎么重新分配的。

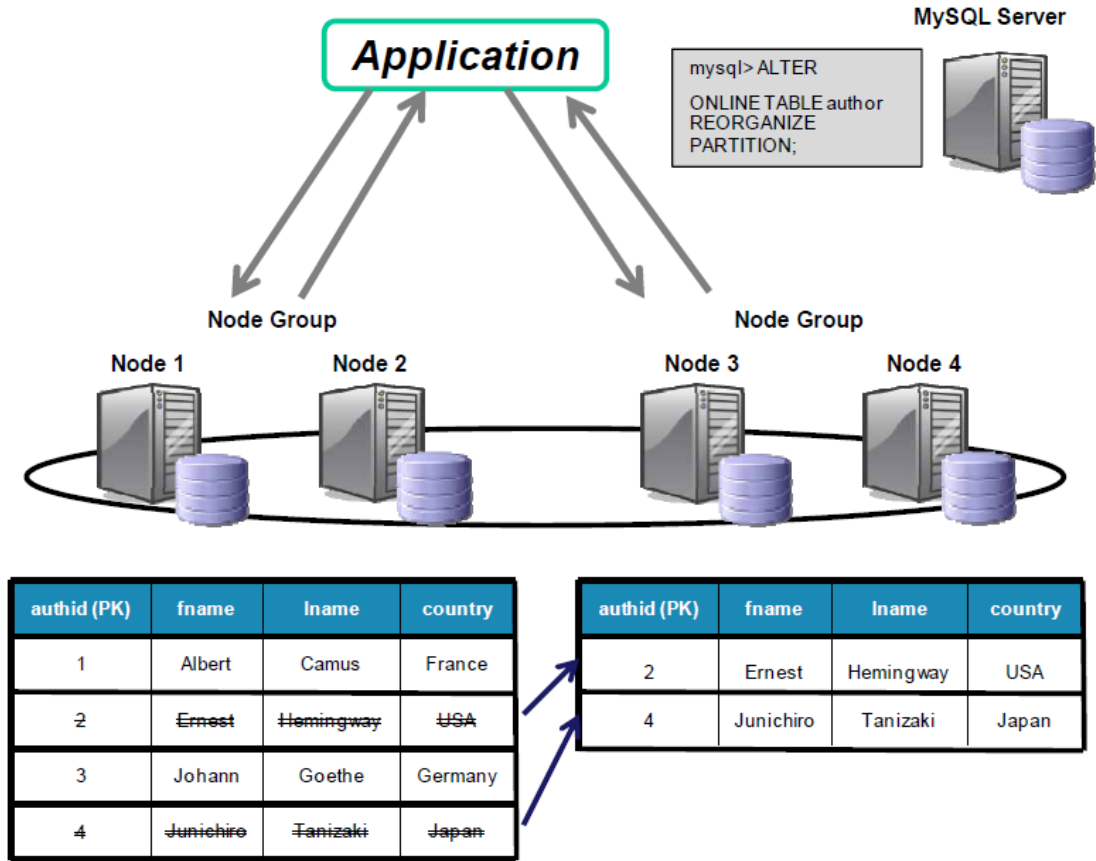


图 10

需要引起注意的是 ONLINE REORGANIZE 和 CREATENODEGROUP 操作是事物安全的，所以在运行这两个命令时，如果一个节点或是集群的故障不会导致数据库的损坏。

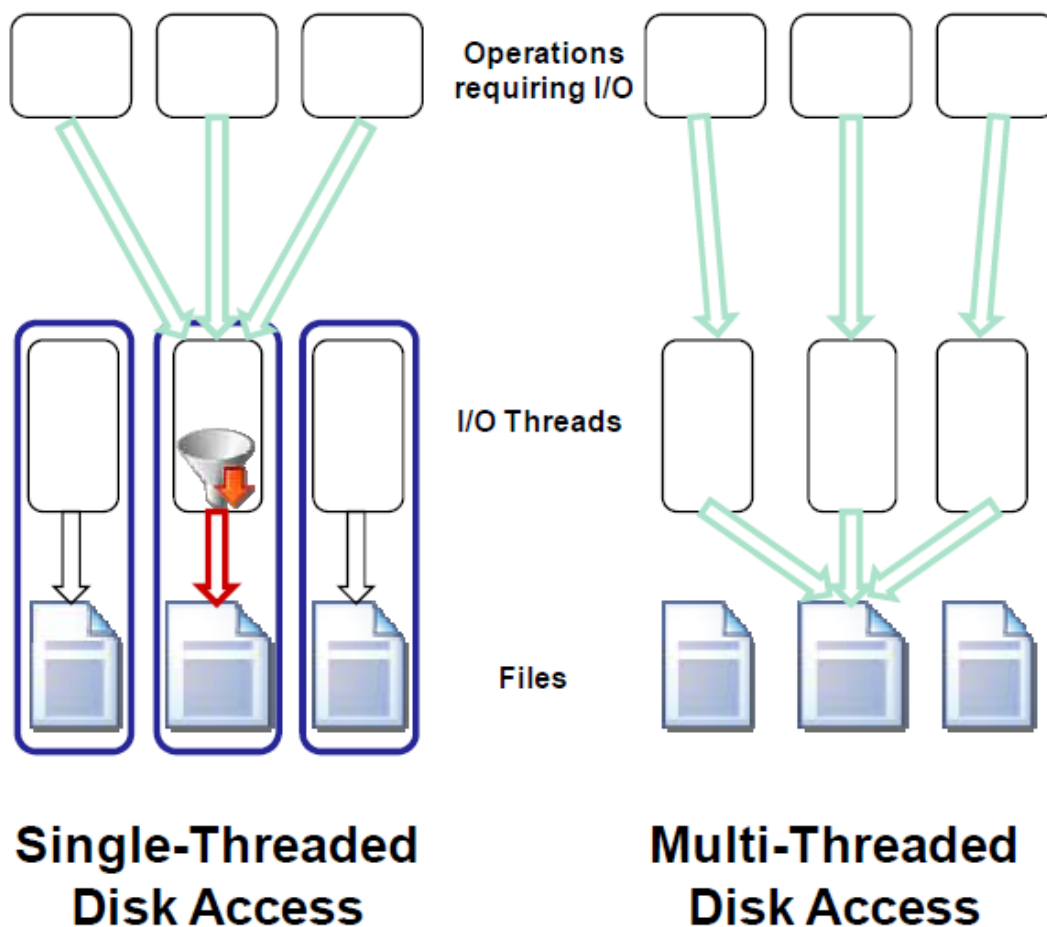
### 4.3 磁盘数据的多线程访问

使用基于磁盘的集群可以允许创建大容量的数据库。MySQL Cluster 7.0 对文件的访问架构进行了修改，提高了对数据的存取性能。

MySQL Cluster 中对文件的访问是通过 I/O 线程来实现的，在之前的版本中，打开的文件和 I/O 线程是一一对应的关系。如果一个文件被频繁访问，那么单一的 I/O 线程就会成为瓶颈并导致系统吞吐量下降。一个可行的方法是 将这些文件分割成一个个小的文件，但是如此一来，就又要需要很多的管理工作。

MySQL Cluster 7.0 中引入了线程池这个概念，其中每个 I/O 线程并不是和一个打开的文件一一对应的。一个文件被频繁访问的文件在某一个时间点可能和许多的 I/O 线程有关联的，如此就增加了系统吞吐量。如图 11 所示：

这种特性最主要的好处是 因频繁使用的文件而需要进行 I/O 操作的性能得到了大大的增强，这提高了基于磁盘表数据存储的吞吐量和反应时间。



因为 I/O 线程不再专用于所有打开的文件( 不管它们是处于访问或是不在访问的时候 ), 所以减少在系统中的 I/O 线程的总的个数可以大大减轻内存的压力。需要引起注意的是, 一些文件还是使用单线程磁盘访问 ( 特别是 redo 日志 )。

#### 4.4 改进的大记录处理能力

在客户端和 NDB 数据节点之间及各个数据节点之间的消息的传送是可能导致集群性能问题的一个可能瓶颈。这些消息会消耗两边的 CPU 的资源, 继而减少了 CPU 可以为其它工作花费的时间。

这一节的内容, 介绍目前集群是如何通过控制消息的大小和数量来优化对网络使用的。

通过这些控制, 可以提高集群 ( 乃至应用程序 ) 的性能, 如此一来, 在有些场合, 就没有必要迁移网络架构 ( 通常情况下, 迁移需要耗费更多资金 )

##### 4.4.1 长信号事务

Long Signal Transactions 的使用, 大大降低了传向 NDB 数据节点的复杂请求的消息 / 信号传输的数量。

在 MySQL Cluster 7.0 之前, 对于简单的请求, 请求信号都是 100 字节以内的, 这么多容量足矣, 但是, 当请求变的复杂时, 请求信号应该在多个消息间分离开来。图 12 给出的是在 MySQL Cluster 7.0 之前, 主键查找的例子 ( 信号称作 TCKEYREQ )。在这个例子里,

在 100 字节内没有足够的空间存放主键值详情( KEYINFO )和非主键属性值( ATTRINFO )。在这种情况下,KEYINFO 和 ATTRINFO 被置于各自的消息当中。如此一来,不仅浪费带宽,也消耗运行 NDB API 的机器和 NDB 数据节点的 CPU 资源---因为需要将消息分割然后再在后端组装。

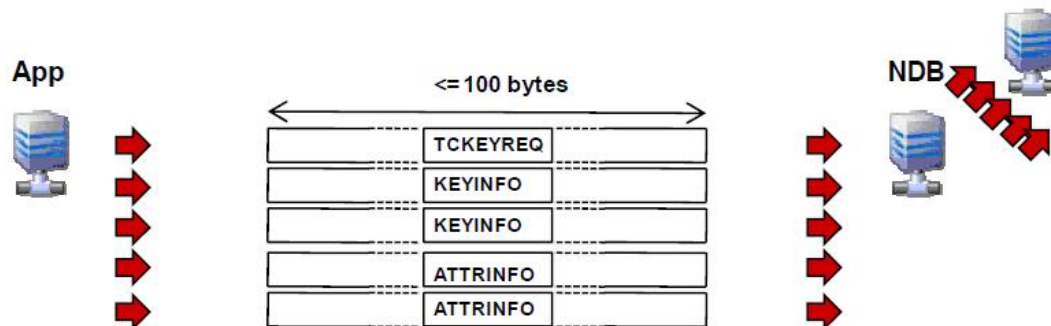


图 12

图 13 显示在 MySQL Cluster 7.0 中是如何将这些信息包含在一个单一的,可变大小的消息里面的。在这里请注意,此处的单一信息消耗的空间比之前的多个消息使用的空间总和小---因为每个消息都包含了相同的协议信息,比如:头信息和实际的有效负载数据。消息现最大可为 32kb,这足以处理大多数的查询需求。

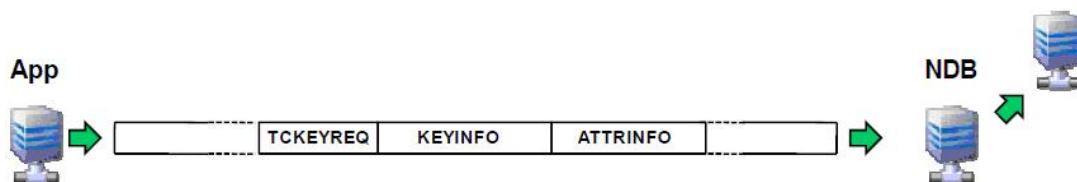


图 13

除却降低对网络产生的压力, NDB 数据节点也无需承受重组请求消息而带来的资源消耗。

这个新的功能特性对应用是透明的---不管是直接使用 NDB API 或是通过 MySQL 服务器节点访问集群。

#### 4.4.2 压缩读

该特性在 MySQL Cluster 6.3 中就已经被引入。在 MySQL Cluster 6.3 之前,读请求/信号包含了需要检索列的列表;在消息里每一个列的标示使用 4 字节。随着被取的列的个数增加,消息也会变的更大。

当从数据节点返回响应时,会使用一种结构,将每个列的结果打包至 4 个字节,使用这种方式,在这些列的内容很小时,是极其缺乏效率的。这种行为可见图 14。在这个图中,



红色的阴影显示的是因这种打包规则而浪费掉的空间。

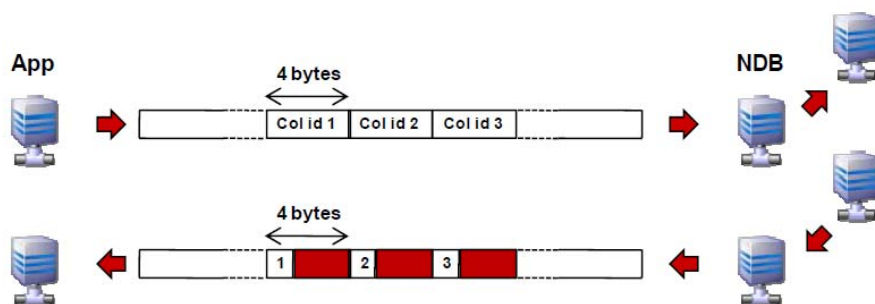


图 14

在 MySQL Cluster 7.0 当中，用于读操作的消息在两个方向上都做了优化。

在读查询中使用 bitmap 来指定要取哪些列----在需要请求大量列的情形下，这可以大大缩小消息。从数据节点的响应来说，4 个字节打包规则不再使用，所以在消息里只需要消耗少量的空间。图 15 显示的是在两个方向上，减少的消息的大小。

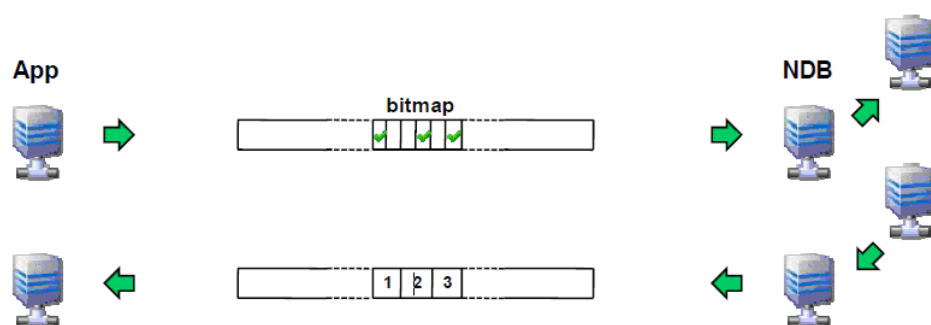


图 15

这种优化是隐藏于 NDB API 内部的，对于应用程序来说是透明的，不管应用程序是直接使用 API 或是间接的访问 MySQL Server 节点。但是，为了得到最大的效率，应用程序的设计应该将它们的表结构设计成利于打包的结构---比如中，将 bit 列放的紧凑。

## 5 Windows 平台

MySQL Cluster 的客户端，向来是支持基于 Windows 平台的，这些客户端是通过 MySQL Server Connectors 与 MySQL Cluster 取得联系的。

MySQL Cluster 7.0，其本身可以运行在 Windows 操作系统。这使得 DBA 和系统管理员除了可以选择 Linux 和 UNIX 环境，更多了一个可以考虑的平台选项。

## 6 简化的 Cluster 监控和管理

### 6.1 MySQL 集群的快照备份选项

MySQL Cluster 支持在线数据备份，这确保了服务不被中断。之前版本的 MySQL Cluster 的备份方法会在备份操作结束的时候抓取数据库的状态。在 MySQL Cluster 7.0 中，支持基于时间点的备份特性，数据库的状态在一开始备份的时候就会被抓取到。这确保了用

户可以在任意的时刻获取数据库的一致性。

图 16 展示在采用原先的 MySQL Cluster 备份策略的情况下，备份是如何进行的。如果数据库在拷贝至磁盘的时候，正在进行升级，那么升级的地方在磁盘上将是一个中间状态，所以在备份的时候，会建立一个 redo log，并和备份文件一起保存。

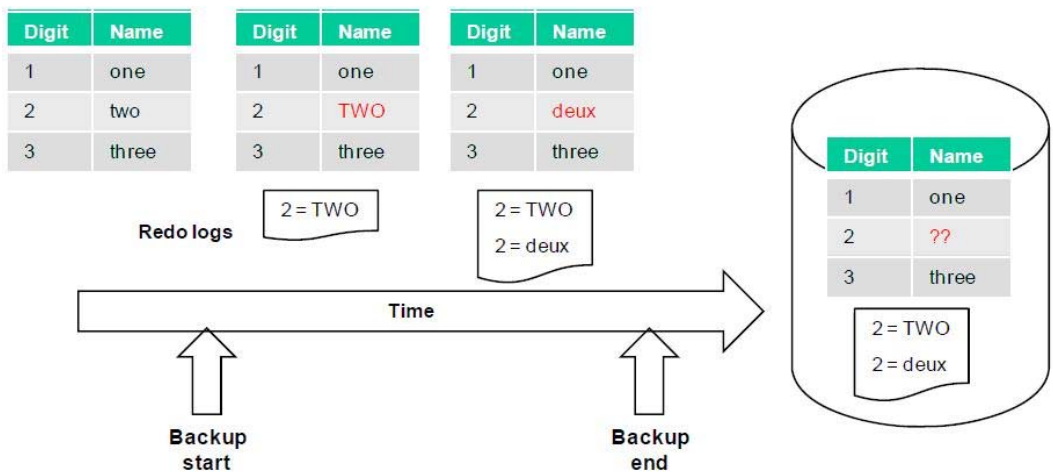


图 16

图 17 展示的是恢复的过程。备份文件从磁盘中读入内存，然后从头至尾的执行在 redo 日志中的更新。结果是，在恢复的结束时刻，数据库看起来就是在备份结束的那一刻状态一致。

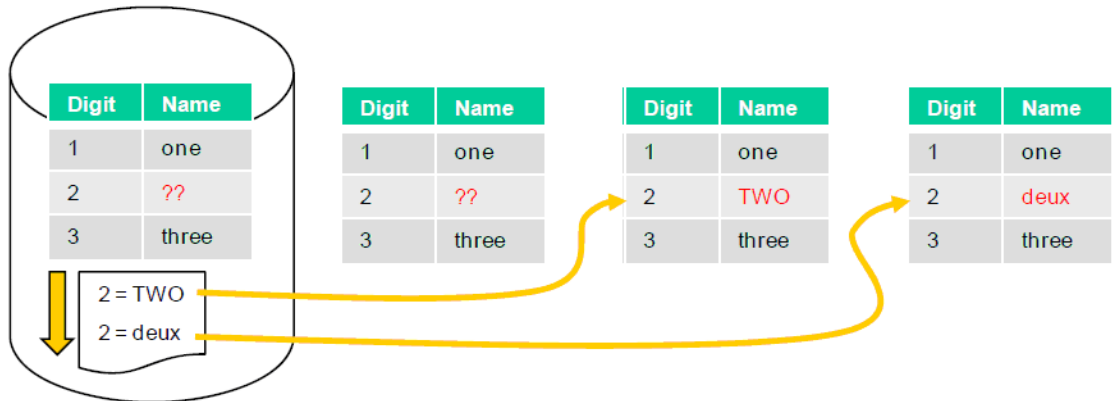


图 17

图 18 展示的是备份过程是在备份开始的时候就抓取了状态的，而不是在最后才抓取状态的。与将变更写入 redo log 不同，变更相反的操作被写入 undo 日志。



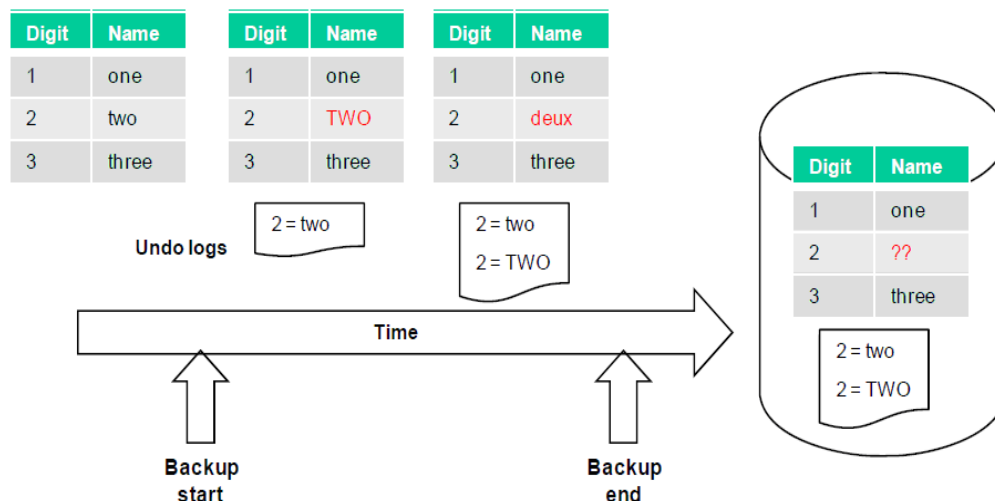


图 18

图 19 展示的是恢复的过程，undo 日志是怎样从文件的结尾处反向执行的。根据这种方式的恢复，数据库看起来就像是从一开始就备份的样子。

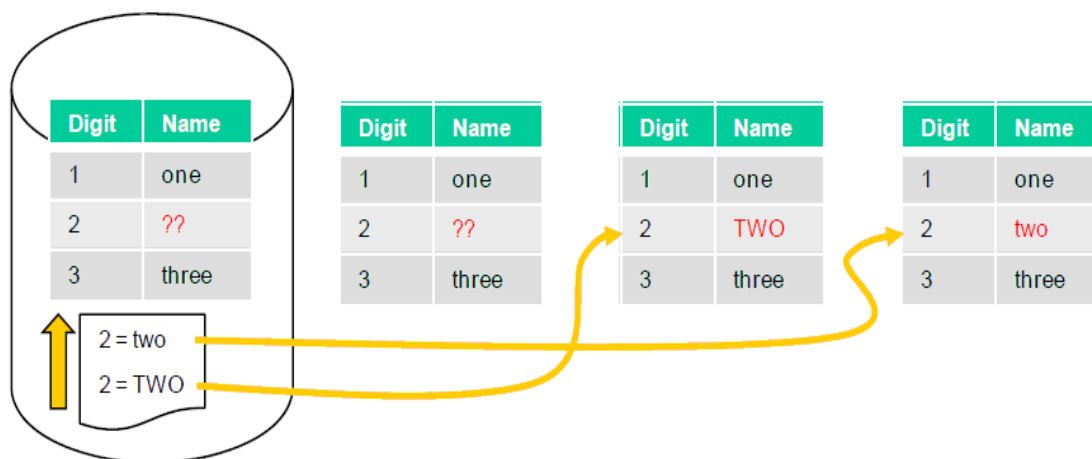


图 19

MySQL Cluster 7.0 中恢复时，使用 `ndb_restore` 可以指定灵活更多参数，你可以只是恢复特定的表或是数据库，或是在恢复时排除特别的表或是数据库。使用的是以下的参数：  
`--include-tables`, `--include-databases`, `--exclude-tables`, and `--excludedatabases`

## 6.2 配置数据缓存

之前，MySQL Cluster 的配置是无状态的---即，配置信息会在每次 `ndb_mgmd` 启动的时候重新从集群的全局配置文件（通常是 `config.ini`）中加载。从 MySQL Cluster NDB 7.0 开始，集群的配置信息会在内部进行缓存，全局的配置文件在管理节点重启时，不再自动的重新读入。

这个行为可以通过管理节点三个选项 `--configdir`, `--initial`, `--reload` 进行控制。

## 6.3 模式变动相关的事务

在 MySQL Cluster 7.0 之前，DDL（数据定义语言）操作（比如 `CREATE TABLE` 或是

ALTER TABLE ) 在数据节点故障时，是不受保护的。而在 MySQL Cluster 7，在数据节点故障时，则确保了这些操作可以优雅的回滚。之前，如果在进行 DDL 操作时，数据节点发生故障，那么 MySQL Cluster 的数据节点就会被锁定，这时，除非重启集群，否则你没有办法进行更多的 DDL 操作。现在，这种功能的增强，使得在线修改变得更加安全。