文章
Qiao Peng · 一月 8, 2021 阅读大约需 9 分钟

InterSystems IRIS 和 Caché 中的多模型数据并行处理

众所周知，InterSystems IRIS
提供了很多的工具来提升应用系统可伸缩性。尤其在提升数据并行处理能力方面，InterSystems
做了很多努力，例如在 SQL 查询中使用并行处理，以及在
IRIS中引入最具吸引力的特征：分片（sharding）。然而，许多成熟的开发成果最初是在
Caché中完成的，而且已经迁移到 IRIS 中。这些成熟的开发成果大都使用
DBMS（数据库管理系统）的多模型功能，实现在单独的数据库中共存不同的数据模型。例如， HIS qMS
数据库同时包含语义关系（电子病历）、传统关系（与 PACS
的交互）和层次数据模型（实验室数据以及与其他系统的集成）。这些数据模型大多是通过 SP.ARM 的 qWORD
工具（一种直接访问 Global的小型数据库管理系统）实现的。遗憾的是，由于查询未使用 IRIS
SQL，无法利用并行查询处理的新功能进行扩展。

而且，随着数据库规模的不断增长，大型关系型数据库所固有的大多数问题开始出现在非关系型数据库中。这就是我们关注可用于扩展的并行数据处理技术的一个主要原因。

在本文中，我将围绕多年来在解决任务时用到的并行数据处理展开多方位讨论，而这些是我在大数据问题探讨中很少提到的。我将重点讨论数据库的技术改造，或者更确切地说，是数据库转换技术。

众所周知，在系统开发的早期阶段（通常远远早于项目完成的时间）就已经选择好了数据模型、存储架构以及软件和硬件平台。但是在系统完成部署几年后，通常会因为这样或那样的原因需要对数据进行迁移。下面是一些常见的任务（所有示例均为真实案例）：
1. 一家公司正计划走向国际市场，必须将其 8 位编码的数据库转换为 Unicode 格式。
2. 将过时的服务器替换为新的服务器时，但是由于许可限制，无法在服务器之间进行日志的无缝传输（使用 IRIS
的系统功能 Mirroring 或 Shadowing），或者当在您尝试处理任务时，缺少满足如 1 提出的这种需求。
3. 你需要更改数据库之间的 Global 分配，例如将一个包含图像的较大的 Global 迁移到单独的数据库。

你可能会问，这些方案有那么难吗？只需要停止旧系统，导出数据，然后导入到新系统中就可以了。但是，如果你正在处理的是几百 GB（甚至几 TB）的数据库，而且系统在 7x24 运行，那么就无法使用标准的 IRIS
工具解决上述这些任务了。

实现任务并行处理的基本方法

"垂直" 并行

假设可以将一个任务分解成几个组件。幸运的话，你会发现可以将其中一些并行解决。例如：

- 准备报告数据（计算、数据聚合…）
- 应用样式规则
- 打印报告

可以让多个报告同时执行所有操作：一个报告正处在准备阶段，同时另一个报告已经开始打印，等等。这种方法并不新鲜。早在 60 年前批量数据处理技术出现时，它就开始发挥作用了。虽然这不是一个新的概念，但是非常有用。然而，只有当所有子任务的执行时间具有可比性时，才会实现显著的加速效果，实际情况却并非总是如此。

"水平" 并行

当任务的操作顺序可以按任意顺序执行并迭代时，它们可以同时执行。例如：

- 全局范围内的上下文搜索:
  - 把 Global 分解成多个部分（$order 作为首个索引）.
  - 分别对其进行搜索.
  - 合成搜索结果.
- 通过套接字或 ECP 将 Global 传输到另一台服务器:
  - 把 Global 分解成几个部分.
  - 分别进行传输.

这些任务具有以下共同特点:

- 相同的子任务处理过程（包括共享相同的参数），
- 最终结果的正确性不受子任务的执行顺序影响，
- 生成结果报告的过程中不需要任何资源密集型操作，因此仅在此过程中子任务和 “ 父 ” 任务之间存在弱连接。

这些简单的示例表明，水平并行在数据转换任务中是常用到的，事实也的确如此。在接下来的内容中，我们将重点关注此类并行处理。

"水平" 并行

方法之一：MapReduce

MapReduce
是由谷歌引入的一种分布式计算模型。它在处理大量信息的同时也会执行此类操作。目前流行的开源项目构建在 Apache Hadoop和[Mahout](https:// ru.bmstu.wiki/apachemahout) 的组合之上。
使用该模型的基本步骤：映射（在处理程序之间分配任务）、实际处理和归约（合并处理结果）。

对此感兴趣的读者如果想了解更多信息，我推荐 Timur Safin 的系列文章，他在《Caché MapReduce——大数据和 MapReduce 概念简介》（第一部分）中介绍了在 IRIS/Caché中创建MapReduce 工具的方法。

请注意，因为 IRIS 具有将数据快速写入数据库的"先天能力"，所以归约这一步通常是微不足道的，类似 WordCount 分布式版本
中的情况。在处理数据库转换任务时，可能完全没有必要进行这一步。例如，使用并行处理程序将一个大型的 Global 移动到单独的数据库，则不需要任何其他操作。

需要多少台服务器?

并行计算模型的创建者（例如 MapReduce）通常将其扩展到多个服务器，即所谓的数据处理节点，但是在数据库转换任务中，通常一个这样的节点就足够了。事实上，连接多个处理节点（例如，通过企业缓存协议（ECP））是没有意义的，因为数据转换所需的 CPU 负载相对较小——这里不用考虑处理过程中的数据量。在这种情况下，初始数据仅使用一次，也就意味着无法通过分布式缓存带来任何性能提升。

经验表明，比较方便的做法是使用两个角色不对称的服务器。下面稍微进行一下简化:

- 将源数据库安装在一台服务器（*源数据库*）上。
- 将转换后的数据库安装在第二台服务器（*目标数据库*）上。
- 只在其中一台服务器上配置水平并行数据处理；此服务器上的操作进程作为*主进程*执行。
- 相应的，在第二台服务器上运行的进程作为*从属进程*；在使用 ECP 时，这些是 DBMS 系统进程（ECPSrvR、ECPSrvW 和 ECPWork），而当使用面向套接字的数据传输机制时，这些是TCP 连接的子进程。

可以说，这种分配任务的方法结合了水平并行（用于在主服务器内分配负载）和垂直并行（用于在主服务器和从属服务器之间分配 “ 职责 ” ）。

任务和工具

让我们思考一下最普通的数据库转换任务：将全部或部分数据从源数据库传输到目标数据库，同时可能对 Global 执行某种重新编码（可以是编码更改、排序规则的更改等等）。在这种情况下，新旧数据库是保存在本地的不同数据库服务器上的。下面列出的是架构师和开发人员需要解决的子任务：

1. 给服务器分配角色。
2. 选择数据传输机制。
3. 选择 Global 转移策略。
4. 选择在多个进程之间分配任务的工具。

接下来我们逐个进行分析。

给服务器分配角色

众所周知，即使 IRIS 以 Unicode 方式安装，它也可以挂载 8 位数据库（本地和远程）。但反之则不行：8 位版本的 IRIS 无法支持 Unicode 数据库，否则会出现错误。在决定将哪个服务器（源服务器或目标服务器）作为主服务器时，必须考虑到数据转换过程中字符编码是否发生了更改。然而，想要决定最终的解决方案，还需要考虑下一个任务，那就是：

选择数据传输机制

你有以下几个选项:

1. 如果两台服务器上的许可证和 DBMS 版本均支持 ECP，则应选择 ECP 作为传输机制。
2. 如果不支持，最简单的解决方案就是在目标服务器上本地处理两个数据库（源数据库和目标数据库）。这时必须利用任何可用的文件传输工具将源数据库文件复制到适当的服务器。当然，这个过程将占用额外的时间（通过网络复制数据库文件）和空间（存储数据库文件的副本）。
3. 为了（至少）避免把时间浪费在复制文件上，你可以通过 TCP 套接字机制实现服务器进程之间的数据交换。这种方法支持以下情况:
    - 由于某些原因无法使用 ECP，例如，为源数据库和目标数据库提供服务的 DBMS 版本不兼容（比如源 DBMS 安装的是非常旧的版本），
    - 或者：用户无法停止在源系统上工作，则源数据库在传输过程中发生的数据修改必须反映在目标数据库中。

在选择方法时，我的优先级非常明确：如果 ECP 可用，并且在传输期间源数据库保持静态，则选择方法 1；如果 ECP 不可用，但数据库保持静态，则选择方法 2；如果修改了源数据库，则选择方法 3。结合这些情况及主服务器的选择条件，我们可以生成如下可能性矩阵：

传输期间源数据库是静态的吗?	**ECP 协议是否可用?**	**源数据库的位置**	**主系统**
是	是	目标系统远程	目标
是	否	目标系统本地（副本）	目标
否	不重要，因为我们将使用TCP 套接字机制传输数据.	源系统本地（原始）	源

选择 Global 转移策略

乍一看，似乎可以简单地通过读取全局目录逐个传递 Global。但是，同一个数据库中的Global 的大小可能相差很大：我最近遇到的一个生产数据库中的 Global 的大小范围在 1MB -600GB 之间。假设我们有多工作进程（nWorkers）可供使用，并且至少有一个 Global（Globals）^Big 符合下面条件：

^Big 的大小 >（所有^ Globals 的大小）/ nWorkers

然后，无论分布式 Global 转移过程中的其他任务完成的多么顺利，最终分配给^Big 的处理任务将一直保持忙碌，并

且可能在其他进程完成之后很久才能完成其任务。如果想要改善这种情况，你可以预先按照大小对 Global 进行排序，并从最大的 Global 开始处理。但是如果^Big的大小明显偏离所有 Global 的平均值（这是 MIS qMS 数据库的典型情况）：

^Big 的大小 >>（所有^ Globals 的大小）/ nWorkers

这个策略不会提供太大的帮助，因为它必定会产生好几个小时的延迟。因此，需要将大型Global 拆分成多个部分，通过多个并行进程对其进行处理。我想强调的是，这个任务（列表中的第 3 项任务）是本文讨论的最难的任务，而且花费了我（而不是 CPU！）很多时间。

选择在多个进程之间分配任务的工具

与并行处理机制进行交互的方式如下:

- 创建一个后台工作进程池。
- 为这个后台工作进程池创建一个队列。
- 发起程序（我们称其为本地管理器）将工作单元放入队列（这是在步骤 3 中预先准备好的计划）。工作单元通常包括某类方法的名称和实际参数。
- 工作进程从队列中检索工作单元并执行处理，这里可以理解成调用了一个类方法，将实际参数传递给工作单元。
- 本地管理器在接收到队列中工作单元已处理完毕的确认信息后，将释放并按需结束这些工作进程。

幸运的是，IRIS 提供了一个非常适合该方案的、出色的并行处理引擎（可以通过%SYSTEM.WorkMgr 类实现）。我们将在运行的实例中应用改引擎，并在计划发表的系列文章中进行深入探讨。

在下一篇文章中，我将更加详细地阐述任务 3 的解决方案。

在第三篇文章中（如果大家对我的文章感兴趣的话），我将讨论任务 4 中的细微差别，包括%SYSTEM.WorkMgr 的局限性及其解决方案。

#大数据 #开发运维 #Caché #InterSystems IRIS

源URL:https://cn.community.intersystems.com/post/intersystems-iris-%E5%92%8C-cach%C3%A9-%E4%B8%AD%E7%9A%84%E5%A4%9A%E6%A8%A1%E5%9E%8B%E6%95%B0%E6%8D%AE%E5%B9%B6%E8%A1%8C%E5%A4%84%E7%90%86