[1] Zhang X, Yu H, Zheng W. BirdStore: Block-grained snapshot management system for instant disaster recovery[C]// IEEE International Conference on Software Engineering and Service Science. IEEE, 2014:800-803.

**摘要：**块粒度快照管理是新兴的即时灾难恢复系统的要求，这些系统通常使用按需数据获取策略。 本文介绍了BirdStore，一种高效的块粒度快照管理系统。由于查询每个块的快照版本需要多次磁盘IO，因此难以组织块粒度快照版本。 BirdStore通过将共享相同备份历史记录的块分组到一个段中来减少磁盘IO。 因此，每个段发出一次磁盘IO，而不是每个块发出一次。 评估表明BirdStore的算法比现有算法快33.5倍，并且存储效率非常高。

**1、Introduction**

随着信息化的爆炸性发展，灾难恢复（DR）技术对于确保IT基础设施灾难时数据服务的可靠性和可用性变得越来越重要。 DR方法的恢复效率通常由恢复时间目标（RTO）来评估，其表示灾难之后的服务停机时间。主要有两种最先进的DR方法：故障转移和故障恢复。故障转移方法维护生产站点的热备用。如果发生灾难，他们可以立即切换到备份站点，从而实现接近零的RTO。但是，远程热备用站点的维护可能非常昂贵。故障方法会定期将生产站点的持久存储数据备份到远程备份站点。要恢复数据服务，需要首先恢复持久存储数据。返回方法要便宜得多，但RTO通常非常大，因为在将所有持久存储数据提取到本地磁盘并重新启动生产操作系统之前，无法重新启动数据服务来自新恢复的存储数据。

在最近的研究中，出现了新的DR技术，以低成本提供短RTO。作为代表，BIRDS利用按需数据获取实现非常短的RTO，为故障恢复DR方法提供即时服务恢复。然而，这些新技术为构建远程备份中心带来了新的挑战，这是本文的重点。

对于传统的故障恢复DR方法，持久性存储数据会定期复制到远程备份中心并存储为快照映像文件。为了减少存储空间和网络带宽消耗，大多数备份系统采用增量快照机制。为了恢复指定时间点的状态，首先获取并恢复最新的先前完整快照。然后，必须获取并应用完整快照和指定时间点之间的所有增量快照，以便将系统还原到指定状态。从远程备份中心的角度来看，所有恢复请求都以各个快照映像的粒度发布。

但是对于像BIRDS这样的按需数据获取方案，恢复请求是以各个块的粒度发出的。在BIRDS中，数据服务在恢复过程中立即重新启动，在恢复持久存储数据之前。然后BIRDS捕获磁盘IO未命中和取出这些数据块是按需方式从远程备份中心阻止的。特定时间点的持久存储的完整视图通常是一个完整快照和许多后续增量快照的组合。远程备份中心负责选择包含特定时间点的数据块的正确副本的正确快照映像。为此，远程备份中心必须维护每个块的备份版本的历史记录，并有效地处理快照版本查询。

本文介绍了BirdStore，这是BIRDS的块粒度快照管理系统。 BirdStore的主要要求是为生产站点存储增量备份快照映像，并在即时服务恢复期间提供按需数据块检索。 **BirdStore采用了一种新颖的基于段的快照版本管理算法，以实现块粒度快照版本的高效管理。** 评估表明，BirdStore中基于段的算法比现有算法快33.5倍，并且在将快照版本信息存储在磁盘上时非常节省空间。

**2、设计思路**

**2.1 Problem Definition**

针对Bird按需即时服务恢复系统的远程备份中心的功能要求可以概括为两个方面：1）多个快照图像的存储; 2）块粒度快照版本的管理。增量快照机制的存储格式设计是一个相对充分研究的问题。因此本文主要关注第二个问题。

典型的备份序列包括一个完整备份，后跟一系列增量备份。完整备份会生成包含所有数据块的快照。每个增量备份都包含自上次备份以来修改的部分数据块集。

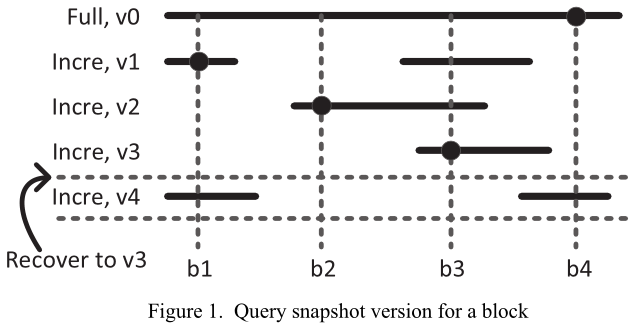


图1是简化的例子。 由版本v0标识的完整快照之后是四个增量快照v1，v2，v3和v4。 块b1由快照v0，v1和v4覆盖，表示在备份v1期间修改并复制b1，然后在备份v4期间再次修改和复制b1。 假设我们要恢复到快照v3的状态，则应从快照v1中检索块b1，其中包含快照v3时块b1的最新数据副本。同样，应检索块b2，b3和b4 当我们恢复到快照v3时，分别从快照v2，v3和v0。

值得注意的是，在实际系统中，完整快照之后的增量快照数量可能非常大。例如，在BIRDS中，每周进行一次完整备份，每5分钟进行一次增量备份。在这种情况下，两个相邻的完整快照之间有超过2000个增量快照。

**2.2 Requirements**

要解决块粒度快照版本管理的问题，我们需要注意以下两个要求：

1) Maintain Extra Snapshot Information

要处理特定块的快照版本查询，我们必须维护有关每个块的备份历史记录的必要信息。 基本上，我们需要保留每个快照包含的块列表的记录。 同样，我们可以保留每个块的相关快照列表。 如何有效地保存这些信息是我们需要解决的第一个问题。

2) Keep Runtime Overhead Low

在利用按需数据块检索的恢复过程中，必须从远程备份中心获取并传输恢复的数据服务所需的每个丢失块。 无法避免磁盘读取IO和网络传输的开销。为了确保恢复数据服务的质量，我们不应该为每个块的快照版本查询引入明显的额外运行时开销。

**2.3 An Existing Solution**

存在Sapuntzakis等人采用的算法。管理虚拟机备份的块粒度快照版本。此算法为每个快照维护一个位图文件。位图文件使用一位表示每个块，指示该块是否存在于相应的快照中。

要在恢复期间检索数据块，将执行搜索以按相反顺序遍历快照位图文件。位图文件包含数据块的第一个快照是要读取的快照。由于完整快照包含所有数据块，因此保证搜索成功。

但是，对于具有大量增量快照（如BIRDS）的系统，此解决方案不能很好地工作。假设单个生产操作系统的持久存储为500GB，我们使用一位表示每个4KB数据块，然后是每个快照的位图文件 约15.6MB。 在此配置下，2000个增量快照将占用31.2GB空间，这太大而无法放入主内存。

由于位图文件驻留在持久存储上，因此在遍历位图文件期间对每个块的反向搜索将涉及多个磁盘IO，这将考虑到磁盘的高延迟而带来显着的运行时开销。 此外，我们必须对每个块执行搜索，即使可能存在一系列共享相同备份历史的块。

**3、实现细节**

在本节中，我们将介绍基于段的算法，其灵感来自以下两个观察结果，这些观察结果揭示了减少磁盘IO的次数和开销。

**3.1 Observations and Improvements**

1) Reduce IOs per Block: Reverse the Bitmap

第一个观察是我们可以通过按块对快照版本信息进行分组，将IO降低到每个块一个IO。 每个块的快照版本信息存储在一起，因此可以从磁盘整体加载，无需多个IO。

这是通过反转位图文件来完成的。我们不是为每个快照保留位图文件，而是为每个块保留一个快照列表文件，其中包含所涉及快照的列表。处理每个块的查询所需的所有信息都包含在相应的快照列表文件中。

2) One IO for Multiple Blocks: Combine the Duplicates

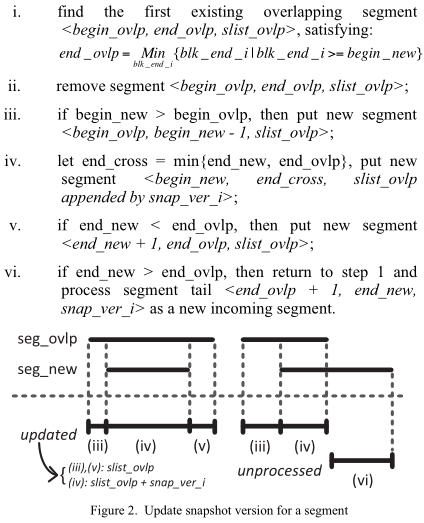
第二个观察是，当我们为每个块保留一个快照列表文件时，有许多相邻的块具有相同的快照列表文件。 这是由于磁盘IO访问的空间局部性特征。如图1所示，典型的备份由多个数据段组成，而不是随机分散的数据块。同一段内的块具有相同的快照列表文件，因为它们共享相同的备份历史记录。空间局部性还表示在恢复期间，如果块被恢复的数据服务访问和获取，则同一段内的相邻块非常可能在不久的将来被访问和获取。

这激发了我们将基于块的快照列表文件转换为基于段的快照列表文件，最终导致我们基于段的快照版本管理算法。采用基于段的快照列表文件会使备份期间的更新过程变得复杂，但允许一个磁盘IO提供多个块查询，从而在很大程度上减少了整个磁盘IO。

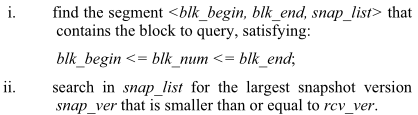
**3.2 Segment-based Algorithm**

基于段的算法可以通过三个子问题来描述：备份期间的更新过程，恢复期间的查询过程，以及用于保存基于段的快照列表数据的内存数据结构。

1) Update Process



2) Query Process



3) In-memory Data Structures

为了实现多个相邻块的一个IO，我们将新获取的段保留为内存缓存，这些缓存被组织为红黑树以便快速查找。 在快照版本查询过程中，我们首先检查红黑树缓存。 如果在缓存中找不到该段，我们执行正常的查询过程并将新加载的段放入红黑树缓存中。