- 目录是文件移动请求需要的元数据之一。被迁移目录可能是文件移动请求的参与者。

以上请求的共同点是它们在拥有目录宿主权限的前提下，与 BS 通信，查询或要求其他元数据的宿主权限。因为目录的迁移请求通过 BS 转发，所以，请求之间形成环路的可能性较大，容易导致系统死锁。

在常规文件并发控制基础上，为提高目录迁移判断的精度，需要比较目录关联的正常元数据请求与目录迁移原因之间的优先级，决定是否迁移目录的元数据。表 6.6 给出了目录迁移需要的同步结构。

表 6.6 目录并发控制的数据结构

```
struct mdt_entry {
        u32 me_state;
        u64 me_timestamp;
        atomic_t me_modicnt;
        atomic_t me_lookupcnt;
        atomic_t me_unlinkcnt;
        atomic_t me_linkcnt;
        atomic_t me_renamecnt;
};
```

在 me_state，me_timestamp 和 me_modicnt 的基础上增加了 me_lookupcnt、me_unlinkcnt、me_linkcnt 和 me_renamecnt，用来记录目录关联的文件查找、删除、创建、以及移动请求的数目。目录迁移算法将通过它们完成目录元数据请求的并发控制。

6.4.1 目录并发算法

表 6.7 目录迁移的优先级判定表

关联 \ 迁移	将被删除的目录	文件移动的目标父目录或者旧目录	文件移动的新目录
创建文件	文件创建成功将导致该目录非空，迁移请求可以提前预测“目录非空”，删除目录请求失败	可以迁移，但需要等待创建请求完成	文件创建成功将导致该目录非空。由于文件移动要求已存在目录必须为空，所以可以提前预判文件移动请求失败
删除文件	可以迁移，但需要等待文件删除请求完成。如果目录有多个孩子，可以提前预测目录非空，删除目录请求失败	可以迁移，也可以抢断文件删除请求。文件删除请求需要重新检查父目录的宿主，向新的宿主请求删除文件	可以迁移，也可以抢断文件删除请求。如果目录有多个孩子，可以提前预判文件移动请求失败
查询文件	可以迁移，可以抢断。查询文件请求不需要向新的宿主重新发起文件查询请求		
文件移动的源父目录	可以迁移，但需要等待移动请求完成。如果目录由多个孩子，可以提前预判删除目录请求失败	由于文件移动请求的串行化，这种情况的并发不会出现	
文件移动的目标父目录	文件移动成功将导致该目录非空，可以提前预判删除目录失败		
文件移动的旧目录	不可能出现这种并发，因为目录只可能有一个父目录		
文件移动的新目录	可以迁移，但需要等待文件移动请求完成		

目录是否迁移的判断策略需要考虑两个问题：1）目录迁移对请求的处理有利，尽量减少无效的迁移。比如，如果被删除的目录非空的话，删除目录的请求不能完成，其迁移将成无效迁移。2）防止目录在服务器间反复迁移，形成元数据迁移的“乒乓”现象。基于以上考虑，目录关联的元数据请求和目录迁移原因的优先级对比如表 6.7 所示。

结合表 6.6 的数据结构，根据表 6.7 的优先级判断，目录迁移的并发控制算法为：

1. MS 处理 AS 的元数据请求的并发算法为：

表 6.8 目录正常元数据请求部分的同步算法

```
check：
  if（(me_state & ME_FLUSHING)为 0，并且没有元数据宿主权限）{
        返回给 AS 新的 MS 信息；
  }
  if （me_timestamp>0 并且还没有超时，或者（me_state & ME_FLUSHING)为 1) {
        等待；
        跳到 check；
  }
  if （me_timestamp>0 但已经超时）{
        me_timestamp = 0；
        唤醒等待 me_timestamp 的用户元数据请求；
  }
  增加 me_modicnt；
  if (需要访问 BS) {
        按照请求类别增加 me_lookupcnt/me_linkcnt/me_unlinkcnt/renamecnt;
        减少 me_modicnt；
        与 BS 通信；
        按照请求类别减少 me_lookupcnt/me_linkcnt/me_unlinkcnt/renamecnt;
        跳到 check;
  }
  完成元数据请求处理；
  减少 me_modicnt；
  /*唤醒元数据迁移请求*/
  if (me_modicnt 为 0)
        唤醒等待 me_modicnt 的进程；
}
```

2. 目录元数据迁移算法为：

表 6.9 目录元数据迁移部分的同步算法

```
if (me_modicnt > 0){
    me_timestamp =  超时时间；
    返回 BUSY；
}
/*比较请求原因和关联操作的优先级。*/
if (出于移动文件的目的迁移目录){
    if (me_linkcnt > 0) {
        if (被迁移目录是移动文件请求的将被覆盖的新目录，并且目录不为空) {
            返回”目录非空”;
```

```
            } else {
                me_timestamp = timeout_second;
                返回"BUSY";
        }
        if (me_unlinkcnt > 0, 并且被迁移目录是移动文件请求的将被覆盖的新目录、且目录的孩子数
目大于 1) {
            返回 "目录非空";
        }
}
if (出于删除目录的目的迁移){
      if (me_linkcnt > 0) {
          返回 "目录非空"
      }
       if (me_renamecnt > 0 || me_unlinkcnt > 0) {
          me_timestamp = timeout_second;
          返回"BUSY";
       }
}
me_state |= ME_FLUSHING;
me_timestamp = 0;
唤醒等待 me_timestamp 的用户元数据请求;
将元数据写回到存储设备，更改元数据分布信息;
me_state &= ~ME_FLUSHING;
唤醒等待"me_state & ME_FLUSHING =1"的进程;
```

6.4.2 算法活跃性验证

由于目录并发判断涉及到的请求种类很多，本部分仅验证目录迁移请求与删除目录下文件请求的并发控制，其 Petri Net 表示如图 6.5 所示，起始状态时位置 Pldt 有 1 个标记。

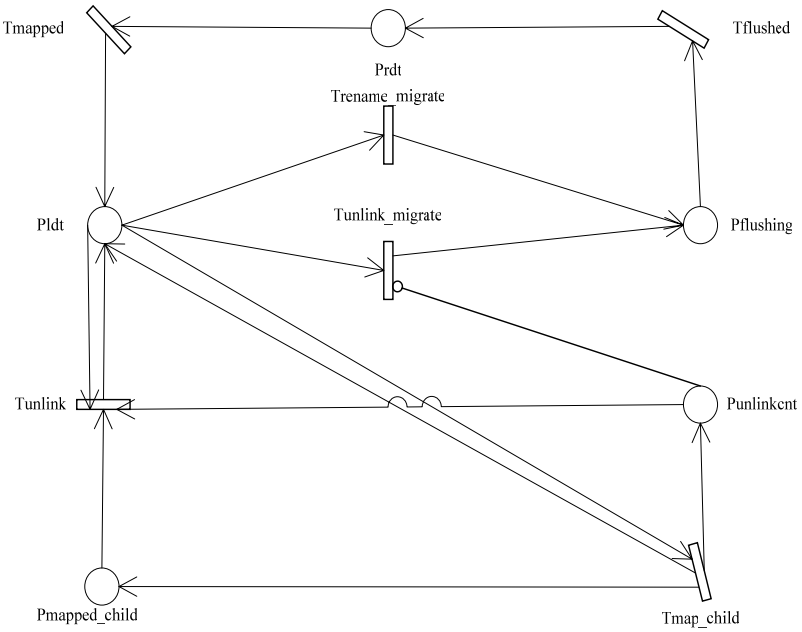

图 6.5 目录迁移与目录子文件删除的并发 Petri Net 表示

图 6.5 中位置的含义为：

表 6.10 目录并发控制图的位置含义

位置名	含义
Pldt	具有宿主权限
Prdt	没有宿主权限
Punlinkcnt	为删除文件目的，请求被删除文件的宿主权限。父目录的 unlinkcnt 记录关联的删除文件请求数目
Pmapped_child	获得被删除文件的宿主权限
Pflushing	元数据正在迁移

图 6.5 中变迁的含义为：

表 6.11 目录并发控制图的变迁含义

变迁名	含义
Tmap_child	请求被删除子文件的宿主权限
Tunlink	删除文件
Tunlink_migrate	处于删除目的请求迁移目录元数据
Trename_migrate	处于文件移动目的请求迁移目录元数据
Tflushed	文件迁移完成
Tmapped	获得元数据宿主权限

图 6.5 的下半部是删除目录下子文件的状态图。在删除文件请求 BS，获得被删除文件的宿主权限前，将目录的 me_unlink 加 1。请求完成后，me_unlinkcnt 减 1。位置 Punlinkcnt 到变迁 Tunlink_migrate 的禁止弧表明，一旦存在本目录下的文件删除操作，期望删除本目录的请求必须在目录下的删除请求完成之后才能执行，以提高迁移的精准度。

变迁 Trename_migrate 和变迁 Tunlink_migrate 的发生导致位置 Ponme 没有标记，变迁 Tunlink 不能进行。这表明如果在为删除请求迁移被删除文件过程中，删除操作的父目录被迁移走，则删除操作不能再进行，需要向新的宿主重新发起请求。

根据图 6.5，得到其可达图如图 6.6 所示，图中节点序列为（Pldt, Punlinkcnt, Pmapped_child, Pflushing, Prdt）。

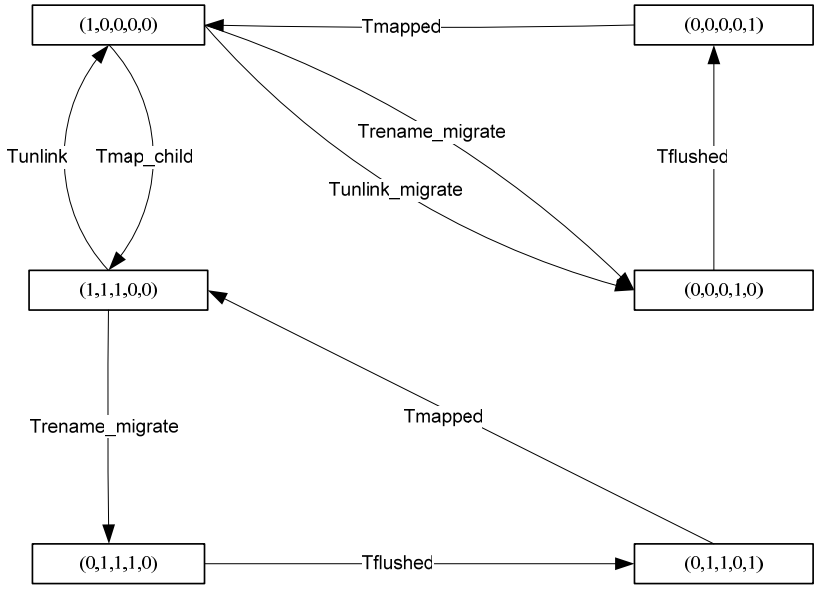

图 6.6 目录迁移与目录子文件删除并发可达图

通过类似的方法，其他的目录迁移并发控制算法的活跃性也能得到证明。综合所有的情况，目录迁移的并发控制算法是活跃有效的。

6.5 本章小结

并发进程对共享资源的竞争是导致系统死锁的根源，分布式系统的非集中结构使得系统的死锁检测和消除尤为困难。已有研究从死锁预防、死锁避免和死锁的检测与消除等角度对系统的死锁问题进行研究。请求的因果关系发现是解决系统死锁的另一种思路，通过分析并发请求之间可能存在的因果关系，为系统死锁问题解决提供支持。投机执行是增加请求并行度、提高系统处理效率的有效方法。

在这些研究成果基础上，BWMMS 在各个 MS 上以元数据请求的元数据对象为核心，结合文件系统元数据请求的语义，分析可能的请求并发。MS 需要同步的请求可以分为 AS 的正常元数据请求之间的同步、其他 MS 的元数据迁移请求与 AS 的正常元数据请求的同步两类。

AS 间的元数据请求的同步对象是元数据，它可以利用文件系统的同步机制完成请求的并发控制。而迁移请求与正常请求的同步对象是元数据的分布信息，需要在元数据分布信息层进行控制。

文件的正常元数据请求由单个元数据服务器完成，并且文件迁移目的简单。所以，通过元数据分布信息协助，采用简单的并发算法，即可完成文件迁移与正常元数据请求的并发控制。在本章中，常规文件元数据的并发控制算法通过 Petri Net 描述并验证。

目录关联的正常元数据请求种类繁多，且迁移的目的较多。并且，目录的迁移对系统的性能有明显的影响，需要提高其迁移决策的准确性。为提高迁移决策的精准度，迁