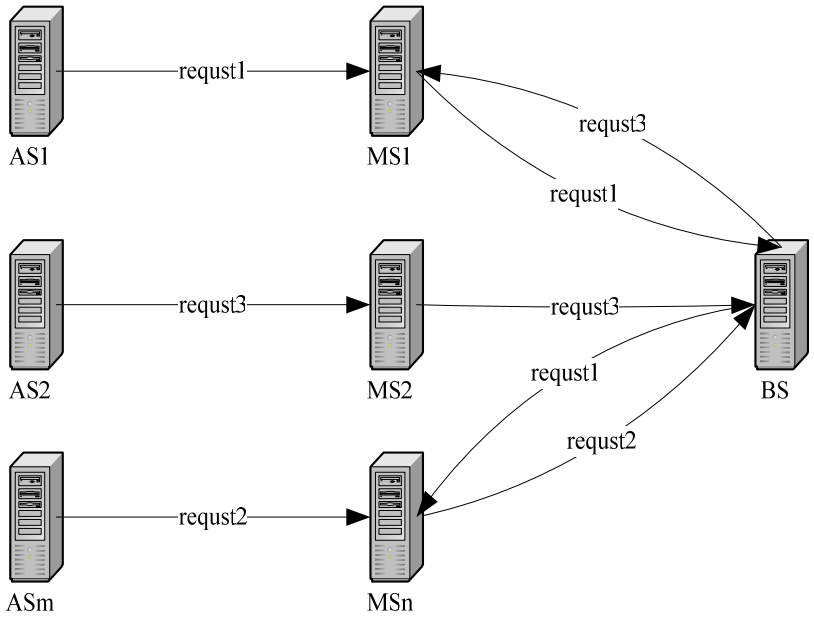

图 6.1 BWMMS 可能的请求并发

如图 6.1 所示，系统可能在两个层次出现死锁：

- 服务进程层的死锁。系统由于各个服务器没有空闲的服务进程处理新的请求而出现死锁。在图 6.1 中，BS、MS1 都可能出现这种情况。假定 MS1 在发出 request1 之后、响应 request3 之前，没有服务进程可以处理新的请求，BS 转发来的 request3 将不能得到处理。而 BS 向 MS1 转发 request3 后，它也没有服务进程处理 request1。那么，MS1 和 BS 间相互等待，导致死锁的出现。
- 文件系统层的死锁。多个元数据请求访问的元数据可能存在关联性。如果请求在拥有一个元数据的宿主权限后，还需要获得其他元数据的宿主权限，元数据请求之间就可能出现循环等待元数据宿主权限的情况。

通过通信协议的超时重发机制，服务进程层的死锁能够得到解决。本章主要分析并解决文件系统层次的死锁问题。BWMMS 特定的服务器交互模式加深了系统出现死锁的可能。在传统分布式系统的请求并发控制基础上，还需要结合文件系统元数据请求的特定语义，进一步分析可能的请求并发，检测和消除系统可能的死锁。

6.2 BWMMS 元数据请求并发控制

在 BWMMS 环境中， MS 需要处理 AS 的正常文件系统元数据请求和 BS 转发的元数据迁移请求。对 MS 而言，BS 转发的元数据迁移请求类似于计算机系统的中断。它要求元数据迁移请求能够中断 AS 的正常元数据请求流。请求中断过程如图 6.2 所示。

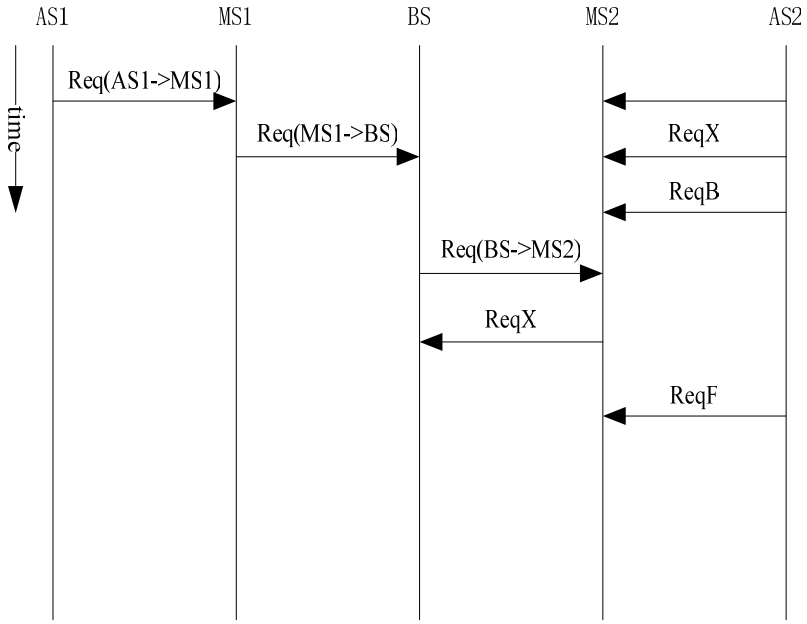

图 6.2 元数据迁移中断正常元数据请求流

假定 MS2 是 ObjX 当前的宿主。在 MS2 上，它需要处理 AS2 请求 ObjX 的正常的元数据请求流（…, ReqX, ReqB, ReqF, …）。在收到 BS 转发的 Req(BS->MS2)时，该迁移 ObjX 的请求需要中断正常的元数据请求流，迁移 ObjX 的 Req(BS->MS2)插入到 ReqB 和 ReqF 之间。迁移成功后，ObjX 的宿主为 MS1。所以，ReqB 及其前的正常元数据请求可以处理，而 ReqF 则需要等待迁移请求完成后，再返回给 AS2。AS2 重新向新的宿主 MS1 发起请求。

6.3 常规文件元数据的迁移

6.3.1 常规文件并发算法

AS 对常规文件的正常元数据请求包括文件索引节点的内容访问、文件访问权限验证、文件数据的设备块号映射、文件数据的并发访问控制信息获取等。这些请求针对单个索引节点进行，不会在持有该文件宿主权限的情况下，再向 BS 请求其他元数据的宿主权限。MS 迁移常规文件元数据的目的包括删除文件、创建到文件的硬连接、或者文件是文件移动请求的旧文件或者新文件等。所以，常规文件的请求并发仅需要控制元数据是否能被迁移，处理元数据迁移过程中收到的元数据请求等情况。依据表 4.1，常规文件元数据迁移同步控制的相关信息如表 6.1 所示。

表 6.1 常规文件并发控制的数据结构

```
struct mdt_entry {
        u32 me_state;
        u64 me_timestamp;
        atomic_t me_modicnt;
};
```

Me_modicnt 用来控制本元数据是否可以被迁移，其值表示不希望元数据被迁移的请求数目，初始值为 0。仅当 me_modicnt 为 0 时，元数据才可能被迁移。

Me_timestamp 是元数据迁移不能进行时，记录的预留时间范围，初始值为 0。在其值表示的时间内，新的正常元数据请求不能使用该元数据，需要等待迁移请求的重新到来或者超时。当超过时间范围，元数据仍然没有被迁移时，me_timestamp 被清除，正常元数据请求可以继续使用该元数据。

Me_state 记录元数据迁移请求的进行过程。在元数据迁移请求能够进行但还没有开始时，它将 me_state 赋值 ME_FLUSHING。新的元数据请求检查到 ME_FLUSHING 时，需要等待元数据迁移请求的完成。

综合上面所述，常规文件迁移同步的控制包括两个部分：

1. MS 对 AS 正常的元数据请求处理过程的同步部分的算法为：

表 6.2 常规文件正常元数据请求部分的同步算法

```
check：
  if（(me_state & ME_FLUSHING)为 0，并且本 MS 没有元数据宿主权限）{
      返回新的宿主 MS 信息给 AS；
  }
  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；
      唤醒因”me_timestamp > 0”而等待的元数据请求；
  }
  增加 me_modicnt；
  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 > 0”的进程；
  }
```

2. MS 处理常规文件迁移请求部分的算法为：

表 6.3 常规文件元数据迁移部分的同步算法

```
if (me_modicnt > 0){
      me_timestamp = 超时时间值，并返回“繁忙，再试”；
}
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，并更改元数据分布信息；
me_state &= ~ME_FLUSHING；
唤醒等待”me_state & ME_FLUSHING 为 1”的进程；
```

6.3.2 算法活跃性验证

通过扩展图 4.4，常规文件迁移并发控制算法的 Petri Net 表示如图 6.3 所示，起始状态只有位置 Pldt 拥有 1 个标记。

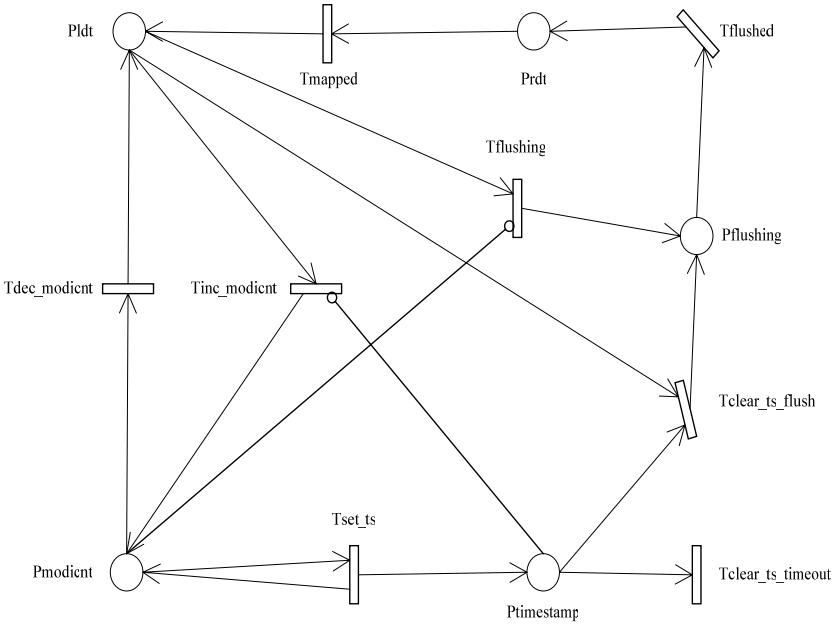

图 6.3 常规文件并发控制算法 Petri Net 表示

其中的位置含义如表所示。

表 6.4 常规文件并发控制图的位置含义

位置名	含义
Pldt	具有文件元数据的宿主权限
Prdt	不具有文件元数据的宿主权限
Pmodicnt	元数据的 me_modicnt 的数目，0 表示没有元数据请求限定“该元数据不能被迁移”
Ptimestamp	Me_timestamp 是否被设置
Pflushing	元数据迁移请求是否在进行

其中变迁的含义为：

表 6.5 常规文件并发控制图的变迁含义

变迁名	含义
Tdec_modicnt	减少 me_modicnt，允许元数据被迁移
Tinc_modicnt	增加 me_modicnt，禁止元数据被迁移
Tflushing	正在进行元数据迁移，将元数据写回到存储设备
Tclear_ts_flush	元数据迁移请求的处理清除 me_timestamp
Tclear_ts_timeout	超时，没有元数据迁移请求清除 me_timestamp，由正常的元数据请求清除。
Tflushed	完成元数据迁移
Tmapped	重新从 BS 获得宿主权限

位置 Pmodicnt 到变迁 Tflushing 间的禁止弧表示如果已经有正常使用元数据的请求

存在，元数据迁移请求不能进行。位置 Ptimestamp 到变迁 Tinc_modicnt 间的禁止弧表示元数据迁移请求阻塞后续的正常元数据请求。变迁 Tset_ts 仅在有正常请求存在的情况下进行，表明元数据迁移请求需要设置 me_timestamp，并返回。

根据表 4.2 的 Petri Net 可达树求解算法，合并可达树中的相同节点得到如图 6.4 所示的可达图，节点序列为（Pldt, Pmodicnt, Ptimestamp, Pflusing, Prdt）。图 6.4 中每个节点都可以通过一定的变迁序列到达其他节点，常规文件迁移的并发控制算法是活跃的。

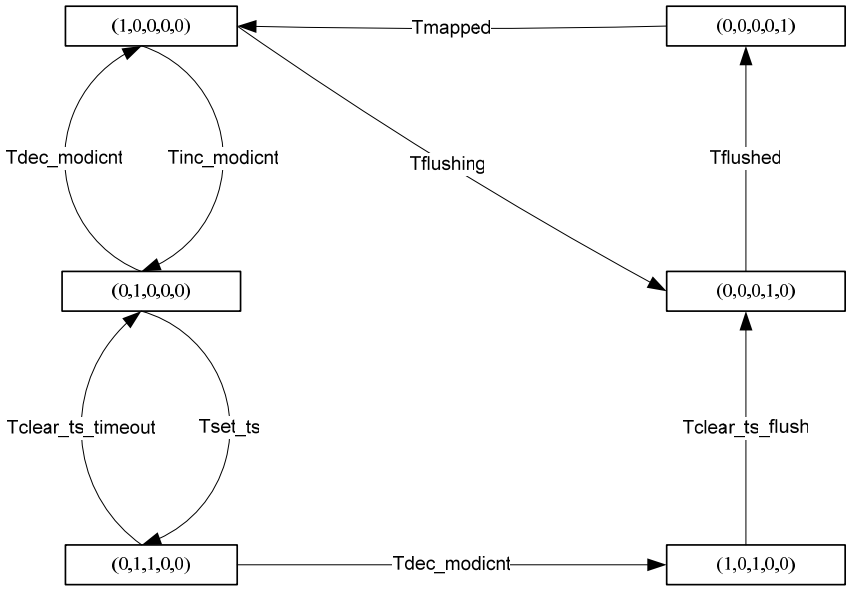

图 6.4 常规文件并发控制可达图

6.4 目录元数据的迁移

由于目录关联的元数据请求种类繁多，目录的元数据迁移带来的并发非常复杂。并且，由于元数据迁移可能导致可以集中完成的请求转换成分布式请求，将影响系统的请求处理效率。所以，目录的迁移决策需要尽可能准确地判定目录是否可以迁移。

被迁移目录可能参与的元数据请求包括：

- 创建文件操作。被迁移目录是创建文件请求的父目录。在为新创建的文件决定元数据宿主过程中，如果父目录被迁走，新创建文件可能还没有写到存储设备，其他 MS 不能使用新创建文件的最新内容。
- 查找文件操作。查找操作可能需要为被查找的文件请求元数据宿主映射。由于查找操作不会更改任何元数据，所以其父目录可以被迁走，不需要进行任何的更新设备内容的操作。
- 删除文件操作。文件的删除需要更改目录和被删除的文件。在文件删除进行前，可能需要向 BS 申请被删除文件的宿主权限。所以，父目录的迁走将导致请求的失败，AS 需要向新的父目录宿主重新请求删除文件。