假定 MAP 协议返回该元数据的宿主为 MS1。MS1 将使用获得的元数据分布信息处理后续的元数据请求。当该元数据不再活跃，需要解除该元数据的请求分布管理时，MS1 主动通过 UNMAP 协议，请求 BS 解除对该元数据的请求分布管理。UNMAP 协议的格式为（索引节点号）。在获得“成功解除”的结果时，MS1 释放缓存中的信息。

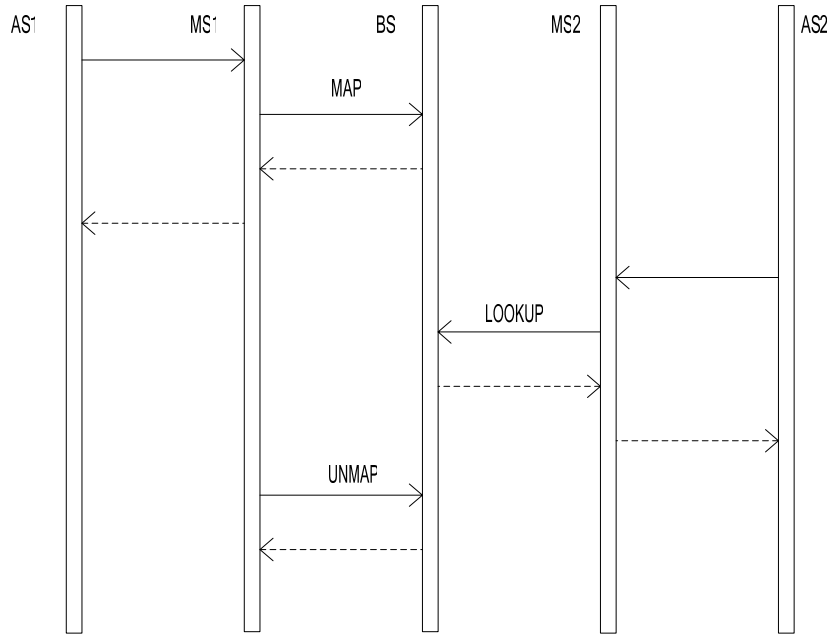

图 3.3 元数据请求分布管理协议

其他需要明确该元数据请求分布状况的元数据服务器，需要跟 BS 通信，获得分布结果。对应到图 3.3 右半部分，MS2 受 AS2 驱动，需要明确该元数据的请求分布信息。所以，它通过 LOOKUP 协议与 BS 通信，获得结果。LOOKUP 协议格式为（索引节点号），BS 返回“该元数据宿主为 MS1”的结果。

从协议的语义看，元数据请求分布的有效性在很大程度上取决于 BS 对 MAP 请求的响应，其算法的有效性将决定系统的元数据服务能力。

3.4.4 请求分布算法

元数据请求分布算法的终极目标是为活跃元数据选择最合适的元数据服务器作为其宿主，其发生在 MAP 协议的处理。BS 对 MAP 协议的处理算法将决定元数据请求处理的效率和元数据服务器的负载，需要综合考虑多种因素。

算法首先需要考虑用户在元数据服务器和文件系统两个方面是否存在特定的需求。出于性能或安全性等因素的考虑，用户可能要求其元数据请求只能由能够满足特定要求的元数据服务器处理。在这种情况下，请求分布算法应该在满足要求的元数据服务器范围选择。另一方面，用户的文件使用模式、用户间的数据共享模式，决定某些文件的元数据请求采用某种特定的分布策略更为合适。比如，每个用户在私有目录中存放文件的使用模式。由于用户间没有文件共享，并且各个用户的访问负载相差不大，目录子树分

区法能够支持有效的元数据服务。再比如，如果用户很少按照目录树遍历查找文件，哈希法能够支持元数据服务器的快速定位，提升系统的元数据服务效率。总之，元数据请求分布算法应该能够支持用户特定的需求。

算法还需要考虑元数据间存在的访问相关性。根据元数据访问协议，文件跟其父目录同时被访问的概率非常高。但它与其父目录的父目录间不一定存在访问相关性，元数据间的访问相关性仅存在有限范围中。在一般情况下，考虑与其父目录的相关性即可。

最后，算法需要考虑元数据服务器的负载情况。有效的分布决策算法需要在保证具有较高统计频率的元数据请求的有效处理的前提下，尽可能平衡服务器间的负载。如果某些服务器负载过高，可能限制元数据服务整体的扩展能力。

根据 MAP 协议，请求分布算法的输入参数为（索引节点号，父目录索引节点，元数据类型，映射模式)。其中索引节点号是被映射的元数据的标识。元数据类型表明被映射的是目录、或者普通文件。映射模式是 MS 给出的参考映射方式，包括“强制映射在本 MS”、“尽可能地映射在本 MS”、“尽可能地映射到其他 MS”和“强制映射到其他 MS”等模式。算法的正确输出是元数据宿主的标识。错误的时候输出相应的错误，由 MS 根据错误情况处理。算法的具体过程如图 3.4 所示。

算法首先明确候选元数据宿主服务器的范围。如果没有满足条件的元数据服务器，返回错误结果给 MS。在明确服务器范围后，它还需要明确该元数据是否关联特定的分布策略。如果有关联的特定分布策略，按照特定的分布策略选择候选宿主服务器。如果没有特定的分布策略绑定，则按照 BWMMS 的动态请求分布决策法（Dynamic）完成请求的分布。算法最后需要检测候选宿主的负载情况，如果不能分布新的元数据请求，则选择预测负载最小的服务器作为宿主服务器返回给 MS。

Dynamic 结合元数据的类型、映射模式和服务器负载，按照如下策略进行：

- 对于文件，首先选择父目录当前的宿主作为候选宿主。然后检查 MS 给出的参考映射模式。如果要强制映射到其他服务器，BS 选择预测负载最小的服务器作为宿主；
- 对于目录，首先以概率 α（比如 98%）在不包括父目录宿主服务器的元数据服务器集合中，选择预测负载最小的服务器作为其候选宿主。然后检查 MS 给出的参考映射模式。如果要强制映射到父目录宿主服务器，BS 选择父目录当前的宿主作为宿主。否则，选择候选宿主作为目录的宿主。

原型系统目前的实现采用元数据服务器当前分布的活跃元数据数目作为服务器的负载参数。还需要在后续工作中丰富负载参数构成，深化负载预测算法的研究。

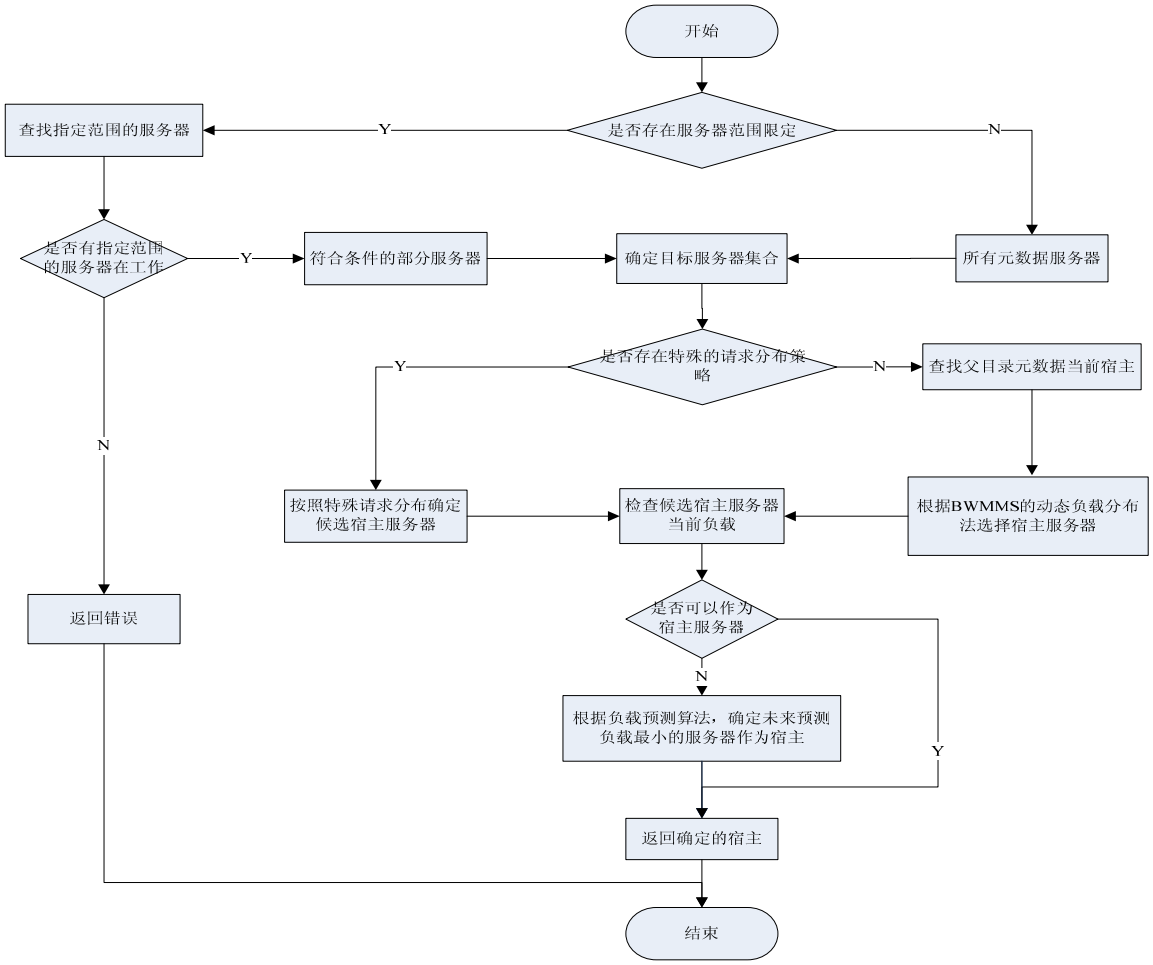

图 3.4 元数据请求分布决策算法

3.5 动态请求分布法的有效性评估

本节通过对比不同请求分布策略对系统的影响，说明 BWMMS 动态请求分布法的有效性。由于哈希函数的设计难度很大，测试仅仅对比动态请求分布法(Dynamic)和目录子树分区法(subdir)在各个客户端的请求负载不同时的表现。

评测环境包括 6 个元数据服务器和 7 个客户端，1 个采用 120GB 的 SATA 硬盘提供存储空间的存储设备，1 个 BS 和 1 个虚拟存储管理器。每个节点是 3.4GHz 的 Intel® Xeron®, 3GB 内存，RedHat® Linux® 8.0 操作系统。每个客户端在自己私有目录中运行 postmark[postmark2002]。Postmark 测试包括测试文件集生成、文件的事务[2]访问两个阶段，其主要模拟邮件服务器、文件服务器等企业应用的 I/O 访问模式。测试由不同的客户端负载构成，客户端 AS1-AS3 在单一目录下进行 10000 个事务，客户端 AS4-AS7 在 10 个子目录下总共运行 10000 个事务。

表 3.2 中的事务吞吐率是 7 个客户端聚合的每秒能够完成的事务数目，服务器工作

[2] Postmark 的事务由两个阶段构成，第一阶段是随机选择 1 个文件，读该文件全部数据，或者以追加写方式写入一定数量的数据。第二阶段是随机选择创建 1 个新文件，并写入一定数目的数据，或者删除测试集中已经存在的文件。

时间是 6 个元数据服务器的服务进程的工作时间总和。

表 3.2 两种分布策略的整体对比

	目录子树分区法	动态请求分布法	提高比例(%)
聚合事务吞吐率(ops)	622	802	28.94
聚合 MS 服务时间(s)	62.82	43.78	30.31

表 3.3 是各个客户端的事务吞吐率明细，单位为事务数目/秒。从表 3.3 可以看出，由于动态请求分布法结合服务器负载情况，动态地分布元数据请求，各个用户的元数据请求可以更合理地分布到服务器，用户的事务吞吐率相对能够得到保证。而目录子树分区法固定单个客户端的请求由既定元数据服务器处理，无法很好地利用服务器的处理能力，客户端的事务吞吐率将明显受客户端负载的影响。

表 3.3 两种分布策略的事务吞吐率明细

	AS1	AS2	AS3	AS4	AS5	AS6	AS7	总和
目录子树分区法	102	97	110	70	80	76	87	622
动态请求分布法	113	103	108	124	118	117	119	802

表 3.4 是各个服务器的具体工作时间，单位为秒。从表 3.4 可以看出，目录子树分区法的服务器负载的平衡性较差，存在服务器很少甚至没有工作的情况，服务器处理能力不能得到很好的利用。而动态请求分布法能够获得相对较好的服务器负载平衡。

表 3.4 两种分布策略的服务时间明细

	MS1	MS2	MS3	MS4	MS5	MS6	总和
目录子树分区法	14.76	16.17	15.43	16.45	0	0.00564	62.82
动态请求分布法	6.72	2.89	10.17	10.82	11.09	2.08	43.78

3.6 本章小结

元数据请求服务是文件系统元数据服务有效性的保证。元数据请求分布是元数据请求服务的核心问题，它将决定服务器的负载，影响元数据服务的扩展能力。

用户的文件访问表现出动态性和局部性的统计特征，仅仅依据静态信息进行元数据请求分布决策的目录子树分区法和哈希法等元数据请求分布策略，不能有效地支持元数据服务的动态扩展。

根据用户文件访问特征和系统扩展需要，BWMMS 提出后端集中决策机制，层次化管理元数据请求分布信息。综合考虑用户的特定需求、元数据间的访问相关性、元数据服务器的负载变化等因素，动态地完成元数据请求的分布决策，为元数据服务器规模和用户访问负载的动态变化提供支持。

请求分布策略有效性的对比评估试验结果表明，相对于目录子树分布法而言，BWMMS 的动态请求分布法能够获得近 30%的聚合事务吞吐率的提高，并能够更好地平衡元数据服务器的负载。