摘 要

对于存储系统，数据的定位、搜索路由的效率以及安全性等方面，分布式存储系统较集中式系统表现出色。本文对全球首个分布式存储系统 OceanStore 的路由协议进行讨论，并利用 PeerSim 平台对其进行仿真验证。对于 OceanStore 所采用的两个不同协议，本文利用相应的算法思路进行仿真程序设计及实现，并控制相应条件下两者的转换。通过仿真，本文得出了 OceanStore 存储协议的一些重要优良特性，并对其进行了分析。

关键字：P2P 仿真，OceanStore，PeerSim

Abstract

With respect to location of data, efficiency of routing, security and so on, distributed storage system performs better than traditional central system. In this paper, OceanStore is the first distributed storage system in the world. Its mechanism of routing is discussed and simulated via PeerSim simulation platform. This paper uses corresponding algorithms respectively for the two different routing protocols, and controls the switching of them according to certain situation. With the help of the results of simulation, this paper presents some significant characteristics of OceanStore and analyzes them.

Key Words: Peer-to-Peer simulation，OceanStore，PeerSim

目 录

第1章 引言

1.1 课题背景

近年来由于宽带的发展、企业对存储系统需求的增加和传统企业网络化趋势的增强，网络存储正在迅速兴起。但由于网络存储是新兴技术，还不十分成熟，存在着价格昂贵、扩展性差、容量小、容错能力低等缺点。而未来网络存储系统必然是分布式的、异构的、虚拟化的存储系统，能够进行自我管理、自我优化和 自我恢复。例如最近十分流行的云存储就是一个良好范例。因此，研究分布式存储对未来存储趋势是把握是非常重要的。

1.1.1 分布式存储的当前研究热点

1.1.1.1 基于 P2P 的分布式存储

基于 P2P 的分布式存储系统是一种基于对等网络技术的数据存储系统，它可以提供高效率、可扩展、鲁棒和负载平衡的数据存取功能。对于存储系统，用户关心数据的吞吐量以及定位、搜索和路由的效率。传统的集中方式无法满足大规模数据存取的要求，就需要采用新的体系来管理系统中的数据。基于 P2P 的分布式存储系统可以利用大量节点的计算和带宽资源用于数据存取，具有弱结构化、没有单一故障点、可靠性好、易于扩展、数据吞吐率高等优点。

1.1.1.2 集群存储

服务器集群技术已经非常成熟，应用也非常广泛，效果也非常显著。应用集群技术，不仅可以有效提升数据中心服务器系统的稳定性、可用性及可管理性，同时，允许用户使用价格相对低廉的配置(如刀片)捆绑来替代昂贵的单块集成电路的高端服务器，在不影响性能的情况下节约了存储成本。在传统的集群系统中，每一个节点服务器都有自己的本地存储，这些存储资源并没有被统一利用，在节点之间也没有一致的视图。如果能够将集群中除了计算资源外的存储资源也利用起来，既可以提高存储资源利用率，又可以互为容错与备份，这是集群存储的内在要求。目前市面上出售的存储集群产品主要分为两大类：一类是集群文件系统，一类是建立在集群的架构之上的独立硬件设备。不过，集群存储效率有待提高。

1.1.2 P2P 技术在分布式存储中的应用

P2P 分布式存储系统具有类似于分布式文件共享的功能和构造，但侧重于分布

式系统中文件系统管理。此类系统主要包括两个类型：

非结构化 P2P 系统：例如 Farsite[1]就属于此类系统。Farsite 通过使用密钥加密文件的内容，并把密文的备份发布到可信任的节点上。每个节点根据获得的文件内容，组织成编目的文件系统。

结构化 P2P 系统：此类分布式文件系统基于 DHT 的思想，将文件发布到 DHT 上，并组织成树状的文件系统。每个目录都组织成一个描述块的形式，每个描述块都对应一个块的 Hash 值，每个块中包含有所有子目录描述块的 hash 值，叶子节点是文件的描述块，所有这些描述块分布在 DHT 中以供检索。此类系统包括基于 Chord[2]的 CFS[3]、基于 Tapestry[4]的 OceanStore[5]等。

1.2 课题研究目标和意义

从体系结构的角度看，分布式存储将是存储领域影响最大的一个发展趋势。传统的网络存储系统采用集中的存储服务器存放所有数据，存储服务器成为系统性能的瓶颈，也是可靠性和安全性的焦点，不能满足大规模存储应用的需要。分布式网络存储系统采用可扩展的系统结构，利用多台存储服务器分担存储负荷，利用位置服务器定位存储信息，它不但提高了系统的可靠性、可用性和存取效率，还易于扩展。因此，分布式安全存储是未来存储界不可避免的重要方向，对其研究将具有重要意义。

OceanStore 分布式存储系统作为全球首个分布式安全存储系统，其目标在于构建全球性的共享网络，通过其提出的各种机制来提高文件的获取速度、文件的健壮性、文件的私密性、文件的安全性等。在此系统中，各种性能表现突出。本文主要研究 OceanStore 的存储路由机制。通过对路由机制的研究，理解文件在分布式网络中的基本存储方式，并且通过仿真平台对其协议进行模拟，为将来的学习和项目工作打下良好的基础。

1.3 课题内容及文章编排

本课题内容主要涉及对全球首个分布式存储系统 OceanStore 路由协议的详细分析，以及对其进行的仿真。

按照时间先后顺序，本文作者主要工作如下：

1.工作准备及调研部分

P2P 技术的迅速发展使得出现了许多与分布式存储相关的论文。在这些论文中，首先对当前所存在的分布式存储系统进行调研，确立研究目标。最后，确立了以 OceanStore 作为本文的主要研究对象。此外，为了选择合适的仿真平台进行工作，对当前各种仿真平台进行调研后确立 PeerSim 作为仿真实现的主要工具。

2.理论知识学习部分

对 OceanStore 系统的协议部分进行深入的学习，查阅部分 OceanStore 引用或采用的方法原型（如 Bloom 过滤器、Plaxton 路由机制等）。此外，查阅 Sourceforge 上 PeerSim 官方教程和一些其他网络资源，进而熟练运用 PeerSim 进行系统仿真。

3.仿真代码实践部分

利用 Java 语言在 PeerSim 仿真平台上实现仿真程序。验证 OceanStore 系统中的某些特性，帮助更加透彻地理解相关原理。

此外，本文后续章节安排如下：

第二章主要介绍仿真平台。其中包括当前一些主流的 P2P 仿真平台，以及对本文实验中所采用的 PeerSim 仿真平台进行描述。第三章主要介绍 OceanStore 系统的路由协议。其中主要包括局部快速协议和全局慢速协议以及他们所涉及的相关知识详尽阐述。第四章主要介绍本文作者如何通过 PeerSim 仿真平台对 OceanStore 的路由协议进行具体的仿真。其中主要描述了在 PeerSim 上对 OceanStore 路由的两个核心部分（局部快速协议以及全局慢速协议）所进行仿真的具体算法以及核心代码实现。

第2章 PeerSim 仿真系统

2.1 P2P 仿真综述

P2P 仿真的主要目的是通过对 P2P 分布式网络结构进行模拟，进而获取所需数据的一种方法。

一般来说，新的 P2P 算法和协议在使用前需要对其正确性和性能进行评价。目前，对 P2P 算法和协议进行性能评价主要有三种方法：分析法、实验法和仿真法。分析法使用数学方法对 P2P 算法和协议进行建模和推演，由于建模和推演过程中需要对假设条件进行大量简化，因此分析结果一般无法直接应用到实际系统中；实验法是在实际环境中对算法和协议进行验证，此方法需要大量的软硬件资源；仿真法使用 P2P 模拟器和仿真工具对算法和协议进行模拟验证，可以模拟大量节点，无须引入过多资源，并且仿真代码与实际代码相似。因此目前对 P2P 系统进行性能评价时，一般使用仿真法。仿真需要使用 P2P 模拟器，P2P 模拟器不仅可以验证 P2P 协议的性能，还能将实际网络中的一些因素加到仿真中去，使得仿真结果更加真实可信。

在 P2P 仿真中，用户可以通过各种计算机硬件资源，借助相应的软件工具，利用特定的一种或多种方法来对一些特定的 P2P 网络场景进行模拟，从而构建一套模拟真实 P2P 网络的仿真环境，并在此基础上模拟某种 P2P 协议的各种功能（如 Gnutella、BitTorrent 和 Pastry 等），通过分析 P2P 仿真运行过程和系统行为特征，获取相应的性能数据，进而有助于测试人员对其有更加直观和现实的了解。通过 P2P 仿真，不仅能够分析现有 P2P 协议和 P2P 网络的固有特性，为 P2P 的行为特征分析、流量检测、流量控制、内容识别等各方面提供直接依据，而且能够对一些新的 P2P 技术的成功性、成熟性进行验证，为进一步改进提供参考依据。此外，P2P 仿真还广泛用于分析 P2P 系统对互联网的影响、P2P 安全防御各种攻击等众多方面。

2.2 当前主流仿真平台简介

从层次化的角度来看，网络模拟器可分为底层网络模拟器(underlying Network Simulator)和覆盖层网络模拟器(Overlay Network Simulator)。底层网络模拟器在数据包层进行仿真，主要模拟底层网络的情况，如网络拓扑、数据包延迟、丢包率等。常见的底层网络模拟器有 NS2[6]，Omnet++[7]和 SSFNet[8]等。其中，离散事

件模拟器 NS2 用于对因特网协议、路由协议以及广播协议进行仿真；基于组件的仿真环境 Omnet++主要用于通信网络的模拟；SSFNet 由基于 Java 和 C 抖的组件构成，主要用于网络连通性的仿真。覆盖层网络是一种构建在物理网络之上的逻辑网络，用于描述节点之间的逻辑关系。覆盖层网络模拟器即 P2P 模拟器，主要用于 P2P 算法和协议的正确性验证并对协议的性能进行分析，P2P 模拟器一般忽略底层网络结构，因此可以进行超大规模的 P2P 系统仿真。

2.2.1 Narses

Narses[9]在 Java 环境下开发，支持分布式仿真。Narses 为进行仿真的应用提供一个用于发送和接收数据的传输层接口，该接口与 UNIX socket 接口类似，使得用户可以方便地将进行仿真的应用移植到真实的操作系统中去。Narses 提供了一系列不同详细程度的网络模型。对底层网络模拟越详细，仿真的效率就越低。用户可以交替地使用不同的模型，在效率与精确性之间权衡。例如，有一种“naive”的简化模型，它不考虑流量的影响，用户可先用“naive”模型建立协议的原型来验证协议的正确性，为了使仿真的结果更接近真实情况，可以再使用更为详细的网络模型进一步仿真。Narses 中的简化模型通过对链路、网络与传输层进行近似地估计来降低仿真的复杂眭。在实现协议方面，Narses 没有实现任何覆盖层网络协议。

2.2.2 NeuroGrid

NeuroGrid[10]是 P2P 查询协议项目，该项目包含一个单线程离散事件模拟器，NeuroGrid 实现了 NeuroGrid，Freenet[11]以及 Gnutella 三种协议，并对这三种协议进行了对比。NeuroGfid 通过定义若干抽象类再派生新的子类来实现可扩展的功能。它包括六个基本的抽象类：Keyword，Document，Message，Node，Network 和 MessageHandler。NeuroGrid 工作在覆盖层，支持结构化和非结构化网络的仿真。运行时的模拟参数是可调的，如网络模拟的节点数目、每个节点的初始连接个数及查询次数等，仿真结果会保存到文件中。NeuroGrid 的不足之处主要有：①没有提供集成其他网络覆盖或拓扑生成器的接口。②没有提供一个请求集合产生器，即缺乏请求分布构件，每次循环随机选择请求结点和请求文件，因此不能模拟实际环境的请求分布特征。③没有提供基于 Gnutella 的经验模型。Gnutella 是目前应用最广的对等系统之一，因此基于 Gnutella 的经验模型是模拟实际对等网络环境的重要模型。④只提供了一个非常简单的动态网络模型，在模拟过程中结点只能

增加而不能减少。

2.2.3 P2PSim

P2PSim[12]是一种离散事件的模拟器，由 C++编写而成，仅支持结构化网络仿真，不支持分布式仿真。P2PSim 上实现的协议有 Chord，Kelips[13]，Tapestry，Kademlia[14]。可以通过设置 P2PS．衄的事件脚本对节点的波动行为进行仿真，但是波动行为和节点失效行为的统计数据都不详细。在查询方面，P2PSim 支持迭代和递归查询。目前，由于缺乏文档，P2PSim 使用起来并不方便。

2.2.4 PlanetSim

PlanetSim[15]是一种离散事件模拟器，在 Java 环境下开发，支持结构化与非结构化的覆盖网络仿真，实现的协议有 Chord 与 Symphony。

在性能方面，有实验表明，对于 Chord 而言，稳定一个 100 个节点的网络需要 8 秒，1000 个节点需要 16 秒，1000~个节点需要 46 秒。

PlanetSim 使用通用的 API，设计简洁且易于理解。PlanetSim 模拟器分为三层：网络层、节点层和应用层。网络层负责仿真循环、底层的拓扑以及信息在仿真网络中的路由；覆盖层实现了一些 P2P 算法，如 Chord；在覆盖层之上的是应用层，实现一些 P2P 服务。这种分层结构便于用户在相同的网络层上实现不同的覆盖层，或者在不同的覆盖层上实现相同的应用服务，可以对各种覆盖网络协议和 P2P 应用服务的性能进行比较。

PlanetSim 在底层可以模拟简单的随机或环形网络，这些简单的网络不考虑流量和带宽的影响。当然，也可以使用专用的网络拓扑生成器来生成更加真实的底层网络拓扑。PlanetSim 包含一个可调节的底层，使得上层服务和覆盖层路由算法可以在各种级别的底层网络拓扑上进行测试。

为了获得良好的性能，PlanetSim 进行了一系列的优化，比如，使用消息池(MessagePoo1)机制，不必在每次需要消息时都实例化一个消息对象，而是采用对象重用机制，回收以前使用过的消息对象以便下次使用，避免实例化过多的对象降低模拟器的性能。

虽然 PlanetSim 没有良好的统计数据收集机制，但是该模拟器支持对节点波动与失败行为的仿真，并且可保存仿真结果以便以后使用。在文档方面，PlanetSim 有详细的指导文档和源代码注释，便于学习和使用。

2.2.5 OverlayWeave

OverlayWeaver[16]是对 P2P 协议进行研究和测试的工具，在 Java 环境下开发而成，仅支持结构化覆盖网络层的仿真，支持分布式仿真，并且支持迭代和递归形式的路由仿真，但不支持底层网络模拟，OverlayWeaver 上实现的协议有 Chord，Kademlia，Pastry，Tapestry 和 Koorde。

用户通过脚本文件定义仿真环境，如创建节点。并且，脚本生成器提供一个用于定义波动行为的函数，对波动行为的仿真需要创建一些节点，这些节点在一个特定的时间段内加入或离开网络。

OverlayWeaver 包含一个图形化的实时可视化工具用于观察协议的运行状况。同时，该模拟器还包含一个简易的统计数据收集工具，通过修改源代码可以获得更为详细的统计数据。

由于 OverlayWeaver 主要用于 P2P 协议辅助设计，所以它的仿真用途是次要的。因此，在仿真方面，OverlayWeaver 存在很多缺点，如不易对统计数据进行采集、文档匮乏等。

2.3 PeerSim 仿真平台介绍

PeerSim[17]在 Java 环境下开发，支持结构化及非结构化网络的仿真，该模拟器的结构是基于组件的，可以快速搭建协议原型。PeerSim 支持两种仿真模式，循环模式和事件模式。循环模式比较简单，具有良好的扩展性，但循环模式忽略了实际网络中的一些细节，比如并发与传输层仿真等，节点之间直接进行通讯并且周期性地对节点进行控制；事件模式则考虑了网络中的实际情况，但该模式的仿真效率较低，扩展性不好。PeerSim 中有一个网络拓扑生成器，产生一个简单的网络拓扑模型——FKP 模型，并可打印出该模型的度分布情况。PeerSim 仿真的大体过程是：首先要确定网络的大小(节点的数量)；然后选择要进行仿真的协议并进行初始化；接着选择所关心的属性，对其进行监控，并可通过配置文件配置一些参数(如，网络的大小，协议的内部状态等)，最后调用类 Simulator 运行仿真。循环模式仿真的过程如下：首先，以命令行的方式读取配置文件，配置文件是普通的 ASCII 码文本文件，由一系列键值对组成；然后，模拟器初始化网络中的节点和节点中的协议；初始化结束后，循环驱动机制在每次循环时调用所有的组件(协议和控制器)一次，直到达到特定的循环次数或仿真结束为止。在 PeerSim 中，所有在仿真过程中创建的对象都是实现一个或多个接口的类的对象，其中主要的接口有：

Node，CDProtocol，Linkable 以及 Control。P2P 网络是由节点组成的，节点是协议的容器，通过 Node 接口可以访问该节点所包含的协议；CDProtocol 是一种特定的协议，该协议运行在循环模式下，对每次循环时要执行的操作进行简单的定义；Linkable 接口为其他协议提供访问邻居节点集合的服务；实现 Control 接口的类可在仿真过程中某个特定的时间执行，这些类一般用于对仿真过程进行观察或修改。PeerSim 具有良好的可扩展性和网络动态进化能力但它不支持分布式仿真。在文档方面，PeerSim 为循环模式提供了详细的文档，而事件模式的文档则相对匮乏。

第3章 OceanStore 路由协议

3.1 OceanStore 系统整体介绍

OceanStore 是一个广域的 P2P 网络文件存储系统，P2P 存储系统的基本目标是帮助用户把数据分布到广域网的多个结点上，并且保证数据的完整性、一致性、可靠性和可用性。OceanStore 采用的是基于 Tapestry 的分布式数据存取系统，其目标是提供全球范围的广域、持久性数据存取服务，如图 3-1 所示。在这个系统中，任何一台计算机都可以加入，贡献自己的存储空间，同时获得他人存储的内容。OceanStore 对数据提供传统的复制、缓存功能，以提高存取速度和可用性。由于建立在一个广域、动态、不可靠的网络基础上，因此对所有数据、元数据都提供了加密或者认证的功能并且采用“拜占庭式容错提交协议”保持副本间的强一致性。

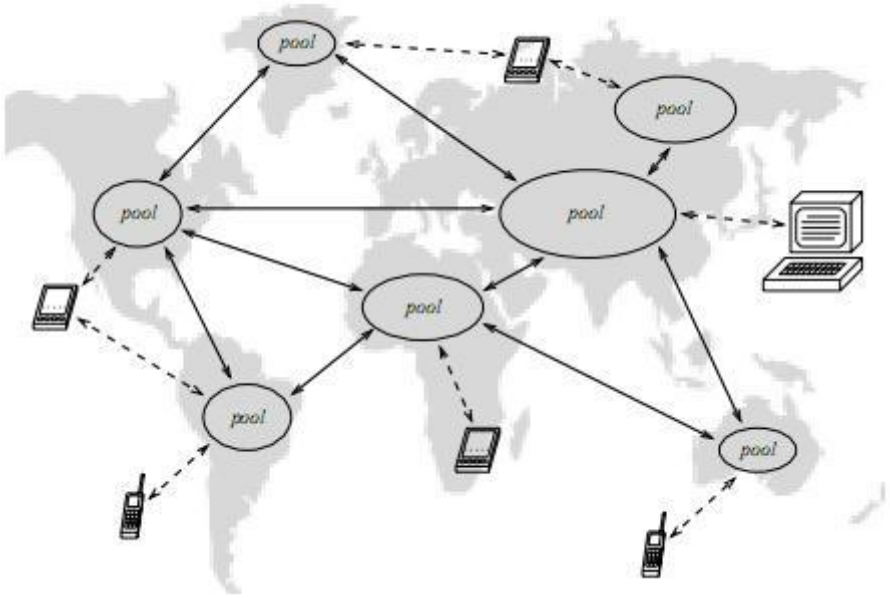

图3-1　OceanStore 系统概观

与传统的基于集群或局域网的分布式存储系统相比，P2P 存储系统具有以下优势：

存储容量更大。P2P 存储系统中的一个存储结点既可以是一般个人用户的 PC 机，也可以是基于集群的大型存储设备。因此，P2P 存储系统实际上是把全球各地具有存储能力的结点整合成一个更大的统一整体，提供上百 TB 甚至上千 TB 的存

储空间。

可靠性、可用性更高。P2P 存储系统在整个因特网中搭建，数据在全球范围内分布。因此，任何一个地理局部的灾难都不会给整个系统以毁灭性打击。也就是说，P2P 存储系统可以通过在广域网范围内进行数据冗余管理增强数据的可靠性和可用性。

分布式访问效率更高。数据在全球范围分布的另一个好处是当数据有多个备份时可以就近访问，对于一些经常处于移动中的用户这一点尤为重要。

3.2 OceanStore 协议部分介绍

在 OceanStore 中，网络中的各个实体是通过一个或多个 GUID 而加以识别的。功能等价的实体，例如，相同对象的不同复制，使用相同 GUID 加以标识。客户端与这些实体使用一系列协议消息交互，如后面的小节所描述的。为了支持位置无关的寻址，OceanStore 消息以一个目的地 GUID、一个随机数和一个小的谓词作上标签。OceanStore 路由层的角色是将消息直接路由到匹配谓词并具有期望 GUID 的最近节点。而在这些消息中目的地 IP 地址不出现。在路由过程中，OceanStore 网络层咨询一个分布式的、容错的数据结构，该结构显式地跟踪所有对象的位置。相应地，路由分为两个阶段过程。消息眼分布式数据结构从节点到节点开始路由，直到发现一个目的地。在那个点，消息直接路由到目的地。重要的是指出，OceanStore 路由层不取代 IP 路由，而是在 IP 之上提供额外的功能。

路由机制是一种两层方法，其特点是一个快速的、概率性的算法由一个较低速的、可靠的等级化方法所备份。这种两等级层次结构合理的理由是，被频繁访问的实体极可能驻留于它们正在被使用的附近；确保这种局部性的机制在 3.2.2 节描述。因此，如果所需要的文件在局部临近范围的话，概率性的算法快速地路由到实体。如果这种方式没有成功，则以一种巨大规模等级层次数据结构就用来定位不能局部找到的实体。因此，可以说 OceanStore 的路由协议主要是由两部分构成的。下面将分别对其进行介绍。

3.2.1 OceanStore 中的局部快速路由协议

在 OceanStore 中，当一个全新的查询产生时，OceanStore 将首先采用一种快速的、局部的协议进行路由。在这种协议中，采用的是一种先进的、可以对资源位置进行预测的数据结构来获知资源位置，而非盲目地洪泛查询。通过这种方式，

OceanStore 中的资源如果存在于发起查询的节点的周围，则资源能够被很快地发现，并且路由到此处而不会产生泛洪，不会造成大量的带宽消耗。需要指出的是，尽管这种方法可以很大概率上保证结果的正确性，但是仍然不排除会进行误判，即：某资源并不存在某节点中，但是却被误认为存在于该节点。在这里，OceanStore 利用一个被称为 Bloom 过滤器[18]的特殊过滤器来实现这一功能。因此，下文主要就 Bloom 过滤器以及其在 OceanStore 中的运用进行介绍。不过当出现此类情况的时候，OceanStore 采用全局的、相对较慢的、可靠的路由机制来重新对资源进行搜索，这将在 3.2.2 节理进行介绍。

3.2.1.1Bloom 过滤器——一种重要的数据结构

Bloom 过滤器对数据集合采用一个位串表示并能有效支持集合元素的哈希查找操作。由于它仅仅利用位数组很简洁地表示一个集合，并能判断一个元素是否属于这个集合，因此 Bloom Filter 是一种空间效率很高的随机数据结构。但是，Bloom Filter 的这种高效是有一定代价的。由于其表示算法的随机特性，存在某元素不属于数据集合而被指称属于该数据集合的可能性，其大小记为误称率。只要这种可能性足够地小以致应用能容忍这种误差，由于其哈希查找的常数时间和少量的存储空间开销，使得它具有非常的实用价值。因此，Bloom Filter 不适合那些“零错误”的应用场合。而在能容忍低错误率的应用场合下，Bloom Filter 通过极少的错误换取了存储空间的极大节省。Bloom 过滤器自 1970 年提出以来，被广泛应用到各种计算机系统之中表达庞大数据集及提高查找效率。下面将对 Bloom 过滤器的原理进行阐述。

1.直观的生活场景描述

我们假设在网络中有节点 A，它拥有资源 X，Y 和 Z。存在三个哈希函数，分别为 Hash1，Hash2 和 $Hash_3$。此外，还存在有一个 7 位的空向量（即向量各位均为 0）。然后，我们利用这三个哈希分别最这三个资源进行哈希。资源被其中一个哈希函数得出的值将被作为偏移量定位到向量中，而定位的到的位将被设置为 1。如图 3-2 所示，首先通过 Hash1 函数将资源 X 哈希后得到 0，于是将向量的第 0 位置 1。然后利用 Hash2 函数将资源 X 哈希得到后得到 1，再将向量的第 1 位位置 1。最后，利用 $Hash_3$ 函数将 X 哈希后得到 4，将第 4 位置 1。类似的，我们对资源 Y 和资源 Z 进行同样的处理。最后，将三个向量进行或运算，最后就可以得出节点 A 的 Bloom 过滤器值（即：该向量）。

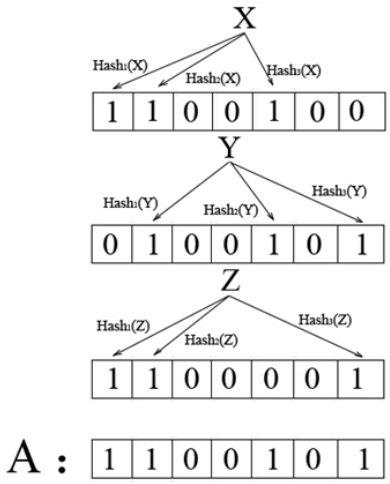

图3-2　一个节点的 Bloom 过滤器

2.Bloom 过滤器的准确理论描述

初始状态时，Bloom Filter 是一个包含 m 位的位数组，每一位都置为 0，如下图 3-3 所示。

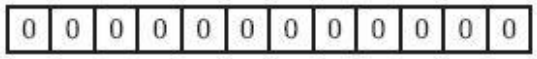

图3-3　初始向量

为了表达 S={X_1，　X_2，…，X_n}这样一个 n 个元素的集合，Bloom Filter 使用 k 个相互独立的哈希函数（Hash Function），它们分别将集合中的每个元素映射到{1，…，m}的范围中。对任意一个元素 x，第 i 个哈希函数映射的位置 $h_i(x)$就会被置为 1（1≤i≤k）。注意，如果一个位置多次被置为 1，那么只有第一次会起作用，后面几次将没有任何效果。在图 3-4 中，k=3，且有两个哈希函数选中同一个位置（从左边数第五位）。

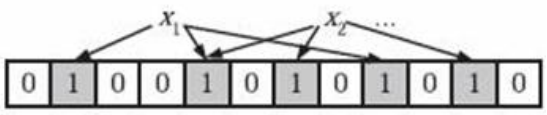

图3-4　产生集合的 Bloom Filter

在判断 y 是否属于这个集合时，我们对 y 应用 k 次哈希函数，如果所有 hi(y) 的位置都是 1（1≤i≤k），那么我们就认为 y 是集合中的元素，否则就认为 y 不是集合中的元素。图 3-5 中 y1 就不是集合中的元素。y2 或者属于这个集合，或者刚好是一个 false positive。

图3-5　利用 Bloom Filter 判断元素是否属于该集合

3.2.1.2Bloom 过滤器在 OceanStore 中的应用

在 OceanStore 中，深度为 D 的一个弱化 Bloom 过滤器可看作为 D 个常规 Bloom 过滤器的一个数组。此时，第一个 Bloom 过滤器是当前节点本地包含对象的记录。第 i 个 Bloom 过滤器是：通过任意路径距离当前节点为 i 的所有节点的所有 Bloom 过滤的并。一个弱化的 Bloom 过滤器为网络中的每条有向边进行存储。一条查询是沿这样的边路由的，该边的过滤器表明对象的存在处于最小的距离上，如图 3-6 所示。

3.2.1.3OceanStore 的局部快速路由过程

在图 3-6 中，是利用局部协议进行查询的过程：在 n_1 处的复制正在寻找对象 X，其 GUID 散列到位 0，1 和 3。n_1 的本地 Bloom 过滤器（圆形框）表明它没有该对象，但其邻居 n_2 的过滤器（非圆形的框）表明 n2 也许是到对象路径上的一个中间节点。查询移动到 n_2，其 Bloom 过滤器表明它本地没有该文档，其邻居 n_4 也没有对象，但其邻居 n_3 也许有。查询转发到 n_3，它验证它有对象。

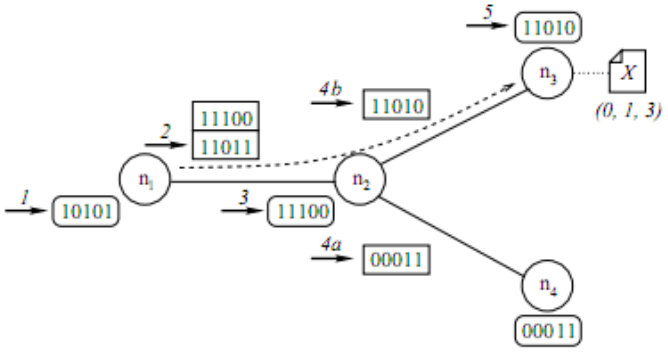

图3-6　OceanStore 利用弱化的 Bloom 过滤器进行路由

3.2.2OceanStore 中的全局慢速路由协议

我们知道，如果已经获取目标节点的 IP 地址，那么利用 IP 层路由算法，可以很快地直接连接到目的地。OceanStore 的全局算法是 Plaxton 等的随机化等级层次分布式数据结构[40]的一个变种，其中在网络中内嵌了了多棵随机树。因此，与局部算法类似，OceanStore 并不“直接”利用 IP 层路由算法，而使构建一个自己的 P2P 网络路由算法。虽然 OceanStore 使用这种数据结构的一个高度冗余的版本，但理解基本的 Plaxton 机制是有意义的。因此，下面介绍 Plaxton 机制。

3.2.2.1Plaxton 机制

在 Plaxton[19]机制中，每个节点或机器都可以扮演服务器角色（即存储资源），路由器角色（即转发消息），和客户端（请求方）。在我们的讨论中，我们用这些术语来与节点互换。同时，资源和节点在它们的定位和语义部分在名字上是独立的，以基于普通基数的随机定长比特序列的形式来描述的（如，40 个十六进制数描述 160 比特）。系统假定进入者大约是在节点数和资源命名空间都均匀分布的，它们可以使用 hash 算法的输出来完成，如 SHA-1。Plaxton 假定 Plaxton 网络是静态数据结构，没有节点或资源插入或删除。

在 Plaxton 中，资源定位按照如下工作：1）服务器 S1 通过路由一个消息到 O_1 的“根节点”来发布消息说明它拥有资源 O1。根节点在网络中是唯一节点，这个节点被放置在 O_1 的内嵌树的根部。发布的过程由向根节点发送一个消息组成，消息包含<资源标识，服务器标识>映射。这个映射将被自 S1 到 O_1 根节点的路径上的节点记录下来。2）在资源定位过程中，一个目标为 O1 的查询消息最初被路由向 O_1 的根节点。3）在每一步中，如果消息遇到一个包含 O1 位置映射的节点，它立即转向到包含 O_1 的服务器上。否则，这个消息被转发到离根节点更近的节点。如果消息到达根节点，只要 O1 的根节点没有失效它就要确保找到一个 O_1 位置的映射。Plaxton 路由协议执行时，节点 N 有一个多层的邻点图，如图，其中每一个层描述了一个 ID 数字的匹配的后缀。

0.3.2.5	X.0.2.5	X.X.0.5	X.X.X.0
1.3.2.5	X.1.2.5	X.X.1.5	X.X.X.1
2.3.2.5	X.2.2.5	X.X.2.5	X.X.X.2
3.3.2.5	X.3.2.5	X.X.3.5	X.X.X.3
4.3.2.5	X.4.2.5	X.X.4.5	X.X.X.4
5.3.2.5	X.5.2.5	X.X.5.5	X.X.X.5
6.3.2.5	X.6.2.5	X.X.6.5	X.X.X.6
7.3.2.5	X.7.2.5	X.X.7.5	X.X.X.7
8.3.2.5	X.8.2.5	X.X.8.5	X.X.X.8
9.3.2.5	X.9.2.5	X.X.9.5	X.X.X.9
A.3.2.5	X.A.2.5	X.X.A.5	X.X.X.A
B.3.2.5	X.B.2.5	X.X.B.5	X.X.X.B
C.3.2.5	X.C.2.5	X.X.C.5	X.X.X.C
D.3.2.5	X.D.2.5	X.X.D.5	X.X.X.D
E.3.2.5	X.E.2.5	X.X.E.5	X.X.X.E
F.3.2.5	X.F.2.5	X.X.F.5	X.X.X.F

图3-7　节点 0325 的邻节图

在图 3-8 为全局全互连网的一部分。假设某资源的根节点为 4598。从任意节点到任意树的根的路径可以通过一次解析根 ID 的一位数字而得以遍历。黑色箭头标明从节点 0325 到节点 4598 的一条路。数据定位使用这个结构，即：0325=>****8=>**98=>*598=>4598（其中*代表通配符）。多数对象搜索不会走完到根的所有线路。从图 3-8 得到的主键观察结果是连接形成一系列随机嵌入树，每个节点作为这些树中一棵树的根。结果，邻居连接可以用来从任意地方路由到一给定节点，方法是简单地一次一条连接地解析节点的地址——首先等级 1 连接，之后等级 2 连接等。使用这个结构用于数据定位，我们将每个对象映射到单个节点，其 node-ID 以最多的比特匹配对象的 GUID（从最低位开始）；称这个节点为这个对象的根。如果关于 GUID 的信息（例如其位置）存储在其根，则任何人都能够找到这个信息，方法是简单地跟随邻居连接，直到它们到达该 GUID 的根节点。如所描述的，这种机制具有良好的负载分布性质，因为 GUID 是随机地映射到整个基础设施中的。

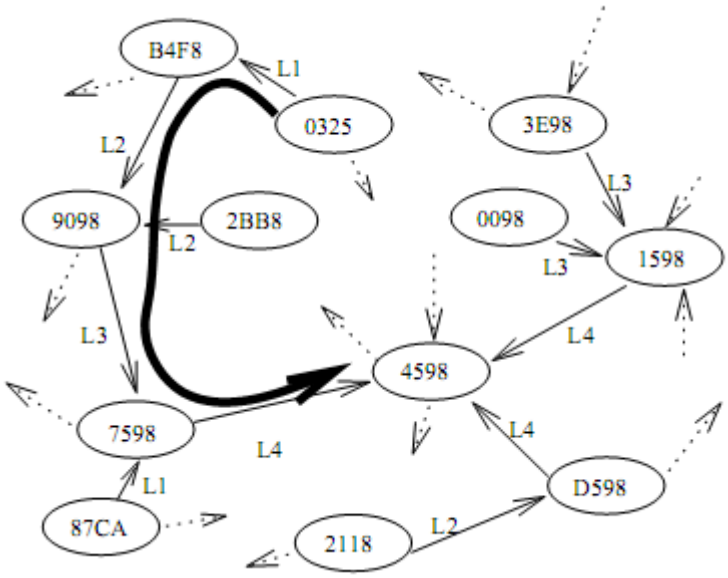

图3-8　Plaxton 机制路由执行过程

3.2.2.2OceanStore 对 Plaxton 原型的改进

问题一：单点故障

问题描述：上面描述的基本机制对许多不同的故障是敏感的。首先，每个对象具有单个根，这成为故障单点，拒绝服务攻击的潜在主体和一个可用性问题。

解决方案：OceanStore 以一简单方时处理这个缺点：它将每个 GUID 以少量的不同精选值进行散列。结果映射到几个不同根节点，因此得到冗余性，且同时使目标为单个节点以拒绝服务攻击的方式攻击 GUID 的一个范围，变得困难。

问题二：敏感性

问题描述：对连接和指针中的破坏是敏感的。

解决方案：一项重要的观察结果是，上述结构具有容人少量破坏的充足冗余性。坏连接能够立刻被检测到，且通过跳到一个随机的邻居节点，能够继续路由。为了增加这种冗余性，OceanStore 定位结构以额外的邻居连接补充上述机制的基本连接。进而，基础设施不断地监视并修复邻居连接，且服务器缓慢地重复发布过程以修复指针。

第4章 基于 PeerSim 的 OceanStore 路由协议仿真

4.1 P2P 仿真环境简介

在本仿真中，所采用的仿真环境如下：

1.运行操作系统：

Windows XP

2.程序语言及其版本：

JAVA 程序设计语言，JDK 1.6 版本

3.程序开发环境：

IBM Eclipse IDE

4.其他：

PeerSim JAR

4.2 P2P 仿真算法

由第 3 章的叙述可知，OceanStore 的路由算法主要分为两部分：快速的但是概率性的局部协议和慢速的但是稳定的全局协议。在查询报文开始的时候，首先应当对本次查询采取局部快速路由方式来优先查看周围节点集合中是否具有可能含有该资源的节点。当没有符合条件的时候，则应当利用慢速全局协议对其进行转发。此外，当局部协议产生误判的时候，也应当将该报文转化为全局协议查询。由此可见，整体来说，报文的查询过程的关键点可以分为以下三点：

1.整体查询实现过程；

2.局部快速协议实现过程；

3.全局慢速协议实现过程。

因此，以下将分别围绕这三部分进行描述。

4.2.1 整体查询实现过程

4.2.1.1 预处理阶段

首先，从接收到的报文中解析出所需要的信息，比如报文的 ID，报文当前的

TTL 和跳数等等。

4.2.1.2 主体流程

1.判断节点是否已经处理过该报文

为了避免重复发送所造成的流量呈指数上涨进而使得网络效率受到极大的影响，当一个节点收到报文的时候，首先应当判断该节点之前是否已经接收到（处理）了该报文。若该节点之前已经处理过了该报文，那么则不应当对其进行转发；如果还没有处理该报文，那么节点应当采用一些相应的转发机制来对该报文进行操作。

因此，在解析报文后，节点首先要判断自己是否已经处理过该报文。在这里，采用如下机制来判断，即：该报文是否已经存在于该节点的路由表中。因为当节点之前转发过该报文后，则报文的一些关键字必将存储于该节点的本地路由表中。借此，可以良好地判断节点是否已经接受、处理过该报文。

如果该节点的路由表中已经包含该表项，那么将该报文丢弃，不予以任何操作。如果该节点的路由表中不包含该表项，那么节点将对报文进行转发，并且与此同时按照该报文的关键信息产生一个新的路由表项，存储在路由表中，转入第 2 步。

2.判断该节点是否含有报文所查询资源

步骤 1 之后，节点就应当判断本地是否含有所需要的资源。如果该节点包含所需资源，则它应当返回资源查找成功的信息，并且产生新的响应报文。如果节点本地并没有所需资源，则跳至下一步等待之后的操作。

因此，节点应当首先其本地的资源列表进行搜索。在这里，为每个节点都含有一个节点自己的资源列表。当节点收到报文后，通过报文中查询资源的关键字在本地资源列表中查找是否含有该资源。若有，则产生一个新的相应报文添加到自己的报文队列等待之后发送。否则，执行下一步操作，转入第 3 步。

3.判断是否进行转发

在网络路由中，报文不可能无限制地路由，也就是为什么会为报文设置 TTL 来限制其在网络中能够存活的时间周期（一般为跳数）。同样地，节点若不包含所需的资源，此时则应当首先判断该报文的生命周期是否终止。如果是，则丢弃报

文，本次操作结束；否则对报文再次进行转发，转入第 4 步。

4.判断采用何种协议进行转发

由于在 OceanStore 中包含有两种协议，因此需要对报文所采用的协议进行判断。这一步之后，节点将按照 OceanStore 的某种协议（局部慢速协议或者全局快速协议）进行转发。

在这里，主要是通过在报文中设置的关键字来达到区别报文所要采用协议的目的。在报文的原有的、普遍的关键字域里，一个新的被用来标记协议类型的域被添加到报文中。这样，当节点收到该报文时，就可以解析出该域，进而采取相应的协议来转发报文。

5.执行协议

当节点已经获取报文的各种信息，并且已经明确了所要采用的协议，则节点将最该报文采用相应的协议进行转发。由于此部分为 OceanStore 路由机制的核心部分，因此将于 4.3 和 4.4 部分进行详细论述，此处不予以讨论。

4.2.1.3 结束处理

当节点已经处理了本周期应该处理的所有报文后，节点清空本地的报文队列，一面进行重复处理。然后节点不进行任何操作，等待下一个自己的周期到来再进行操作。

4.2.1.4 整体流程小结

上述步骤为节点接收到报文之后主体上所需要进行的相关处理步骤，而之后更加详细的局部快速协议和全局慢速协议将在后续部分予以讨论。在主体步骤中，主要是从进入仿真程序到仿真结束过程中所要进行的一般性操作，与其他大多数协议的报文处理没有太大差别。然而不同的是，由于 OceanStore 的路由协议主要由两部分构成，因此其处理步骤将需要比一般的处理步骤复杂，即：判断当前应当采取的协议类型。当判断协议类型结束后，再转入相应的入口执行协议函数。在图 4-1 中，清晰地展示了整体查询的实现过程。

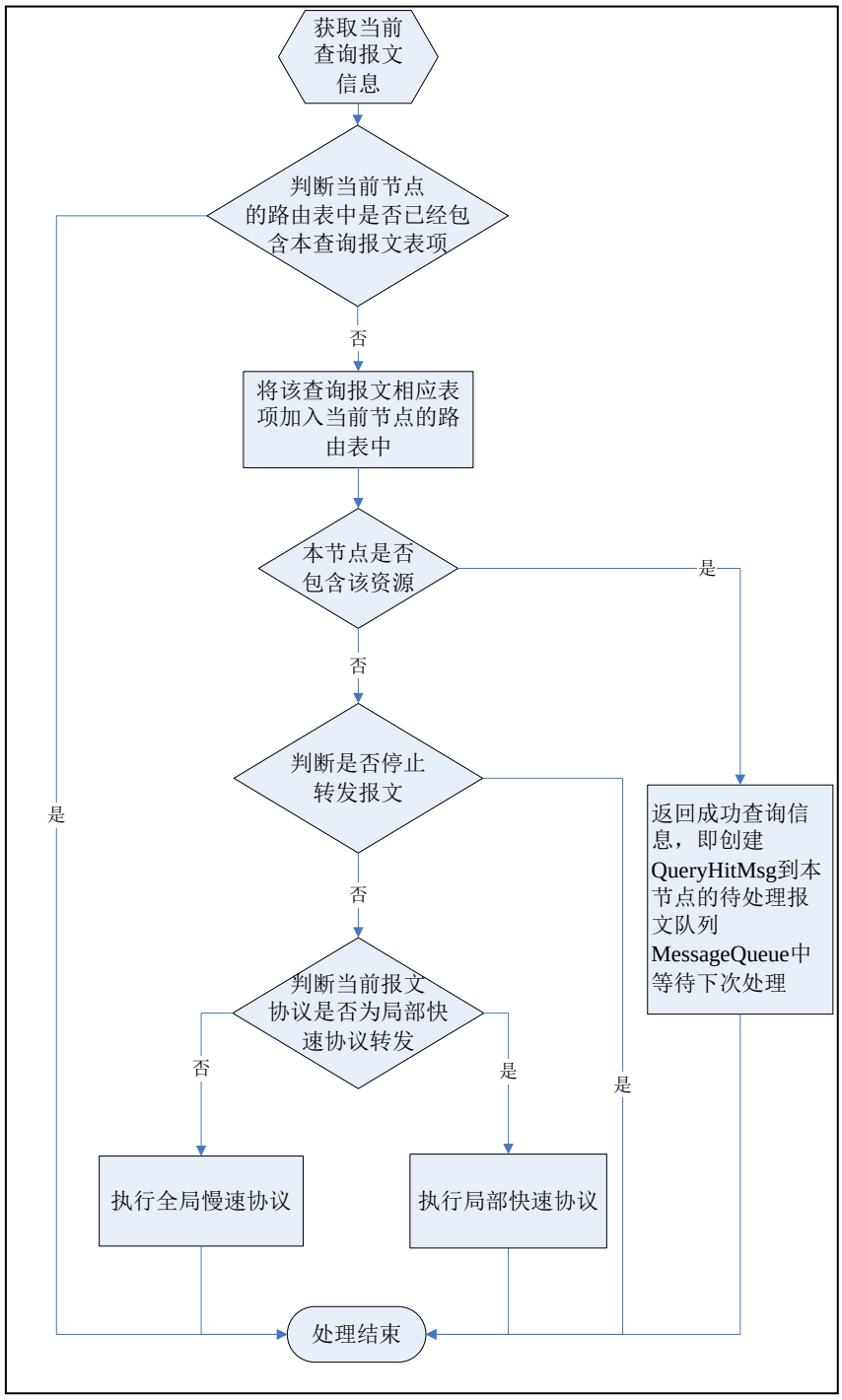

图4-1　查询过程整体流程图

4.2.2 局部快速协议实现过程

如 3.2 所述，在 OceanStore 中，新报文的最初的数跳中采用的是一个概率性地

快速路由方式（如果有符合实行该协议的邻居节点时）。局部快速协议能很快地判断一个资源是否存在于某个节点资源列表中，而仅仅是占据很小的存储空间。虽然有一定的误差率，但是误差率可以被减小得十分小。

OceanStore 中认为，用户经常查询的资源很可能就存在于节点的周围。因此，发出初始查询的节点应当首先检查在指定跳数内的周围邻居集合中是否可能有节点包含所需资源。如有，则可以极大程度地缩小查询时间，节省网络带宽等等。该路由是 OceanStore 的核心路由方式之一，下面将对局部快速协议的实现过程进行详细叙述。

4.2.2.1 预处理阶段

当执行局部快速协议的时候，节点首先需要判断局部快速协议的生命周期是否结束。因为节点不应当无限制地执行局部快速协议，当一定范围内的快速路由不成功后，将转换为可靠的全局慢速协议进行转发。因此，与报文中一般的 TTL 相似，也需要对报文设置其相应的局部快速路由生命周期。这样，可以保证当局部快速的生命周期结束的时候，能跟通知节点将转发协议更换为全局慢速协议。

由此，尽管整体流程上宏观地看应当进行局部快速路由，但是节点还是需要判断局部快速路由的生命周期是否结束。解析出相应的、表示局部快速路由生命周期的域后，若局部快速路由生命周期数值小于 0，则节点将报文切换为全局快速协议，将新报文添加到自己的报文队列等待之后以全局慢速协议进行发送；否则，节点跳入下局部快速路由的下一步。

4.2.2.2 核心处理阶段（获取一定深度邻居节点集合的 Bloom 过滤器）

在进行局部路由的时候，节点首先需要知道各个深度的邻居节点的 Bloom 过滤器列表，进而依据表的信息来判断某条路径上的邻居节点是否可能包含所查询的资源。因此，获取各个深度的邻居节点的 Bloom 过滤器就成为了重点，也是局部快速算法的核心所在。

为了获取指定深度内所有邻居节点集合的 Bloom 过滤器，这里采用的是深度遍历的迭代方式来获取。

解析报文后，节点判断判断此时的深度是否已经达到了最大值。如果已经到所指定的达最大深度，则不再向更深层次的邻居节点继续查询其 Bloom 过滤器，而仅仅是将自己的 Bloom 过滤器追加在返回报文的尾部，并且还要附带其深度；

如果没有到达最大深度，则继续将报文转发给邻居节点集合来获取他们的 Bloom 过滤器，当获得了他们的 Bloom 过滤器后节点再将自己本地的 Bloom 过滤器及其深度追加到报文尾部。过程如下图 4-2 所示。

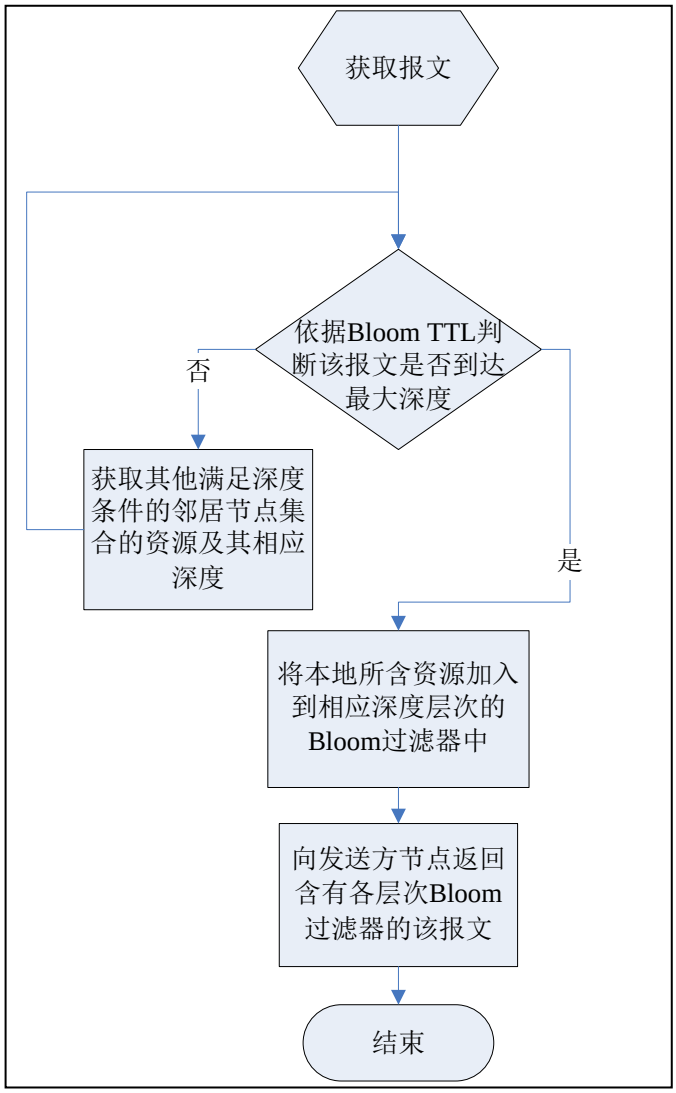

图4-2　获取 Bloom 过滤器流程图

从上图可见，某节点经过一定迭代之后，将获得所指明的深度的所有邻居节点集合的所有 Bloom 过滤器。这样，节点就可以利用各个不同深度层次的 Bloom 过滤器。

4.2.2.3 判断所要转发的邻居节点

当节点获取所有指定深度的邻居节点的 Bloom 过滤器后，节点可以通过各个

层次（深度）的 Bloom 过滤器来判断将查询报文转发向哪一个邻居。当节点在沿着某个邻居的路径上的 Bloom 过滤器中发现该 Bloom 过滤器中很可能包含该资源时，则节点记录下该邻居节点。值得一提的是，除此之外，节点还将记录下该 Bloom 过滤器的深度。在选择将要转发给的邻居节点时，有如下两种情况：

1.各个直接邻居节点的不同深度被检测出可能含有该资源时：

因为为了提高最高的查询效率、减少最小的带宽开销，发送方将只选择唯一一个邻居节点作为发送方。当前节点将选择具有可能包含该资源且具有最小深度的 Bloom 过滤器的直接邻居作为其选择的下一个转发节点。由于 Bloom 过滤器的误差可以降低到十分小的程度，因此，虽然只向一个邻居节点转发而不是洪泛的方式也能够很大程度上正确地地位到正确节点，并且选择满足条件的最小深度的 Bloom 过滤器这一判断条件保证了查询报文能跟在最小的跳数内到达目的节点。

2.不同直接邻居节点被检测出可能含有该资源，而此时有多个节点都具有最小深度的 Bloom 过滤器：

此时，由于有不只一个的直接邻居节点具有 Bloom 过滤器，且它们均为最小深度。那么节点将向所有节点转发该查询报文。这样可以依旧保证是最小的跳数到达目的节点，也可以同样地保证正确率。而且一般来说，出现这样的情况较少，这样出现深度相同的邻居节点也不会很多，因此实际上转发的报文数量不会有大幅度增长，不会对网络带宽造成影响。

上述情况为各个深度邻居节点集合可能含有资源，而可能出现在制定深度的情况下，没有任何邻居节点集合含有该资源。此时，由于局部协议已经不能找到满足局部快速协议条件的邻居节点，该节点应当利用全局慢速协议进行转发。所以，当局部快速协议查询失败时，节点就应当自动将报文的协议类型字段设置为全局慢速协议，以便之后该报文按照全局慢速协议进行转发。

4.2.2.4 结束处理

节点按照实际情况更新报文的各个字段信息，并且将新报文转发到邻居节点，完成本地的转发工作。当节点处理完报文队列中的所有待处理报文后，将节点队列清空，等待下一个周期的到来。

4.2.2.5 局部快速协议小结

从上述过程看来，最核心的部分应当是通过迭代法获取各个深度邻居节点集

合的 Bloom 过滤器进而判断转发对象。局部协议整体流程如图 4-3。

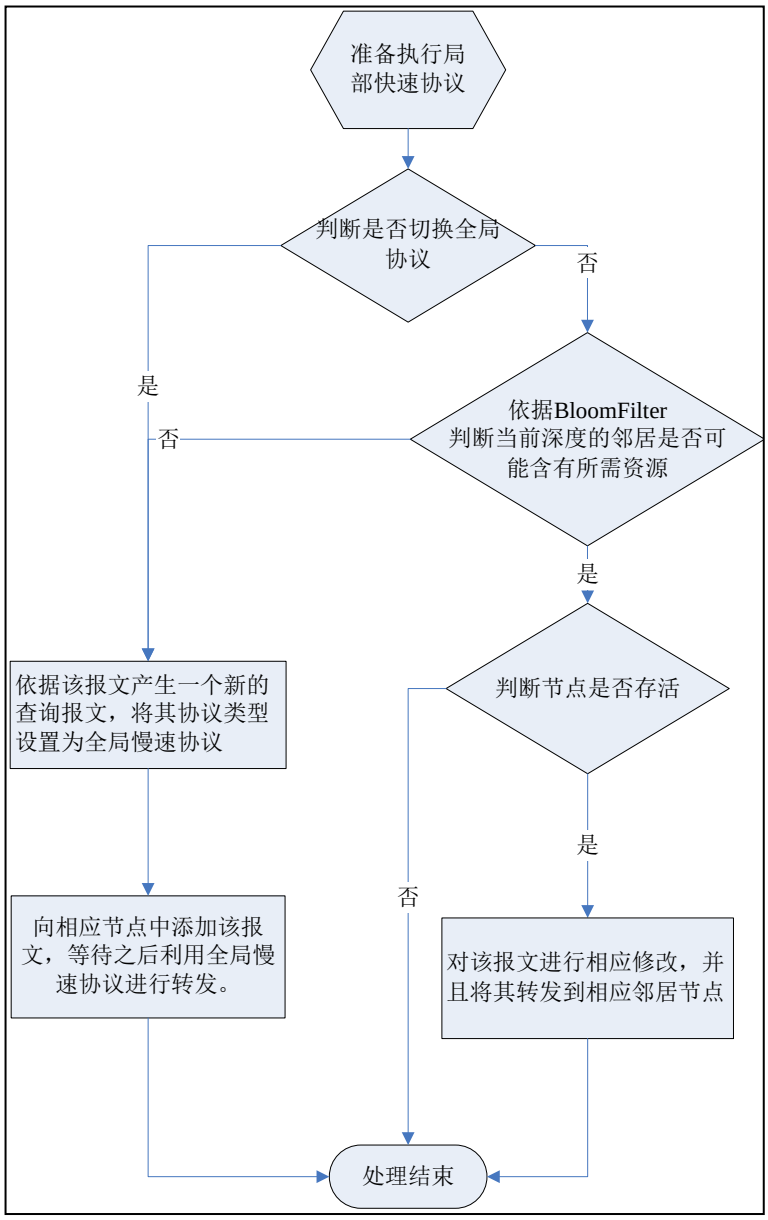

图4-3　局部慢速协议流程图

4.2.3 全局慢速协议实现过程

局部快速算法尽管能跟快速地定位到目的节点，但是并不能保证一定查找到符合条件的节点。因为有些时候节点所需要的资源很可能距离节点十分遥远，这样节点就不可能在小范围内查找到。因此就必须在全局范围内进行查询。

4.2.3.1 预处理阶段

从接收到的报文中解析出所需要的信息。

4.2.3.2 核心处理阶段（按照邻接表进行转发）

当节点解析报文后，节点依据报文中的协议类型将判断出该报文此时应当进行的全局慢速协议转发。

首先，该节点将报文中所要查询的资源的关键字转化为其 GUID。

然后，该节点将会依据 GUID 作为节点 ID 来查找本地的邻居映射表。按照邻居映射表中的各个条目，各个节点可以把消息按照目的地址一位一位地向前传递。若在邻居映射表中节点没有找到满意的邻居节点，那么节点将该报文转发拥有与理想节点的节点 ID 最接近的节点 ID 的节点上。随后该节点再按照相应的路由机制和报文情况对其进行路由转发或者停止操作等等。

4.2.3.3 结束处理

当本地节点已经决定了将要将该报文转发向哪一个节点之后，该节点向被选择的节点转发进行相应更新过的报文。随后清空自己的报文队列，等待下一周期的处理。

4.2.3.4 全局慢速协议小结

从上面的叙述可以知道，全局慢速协议的主要操作是查询本地的邻居映射表来决定转发对象的。选择对象的方式类似于 IP 分组转发过程中的最长前缀匹配方式。全局慢速协议整体过程如下图 4-4 所示。

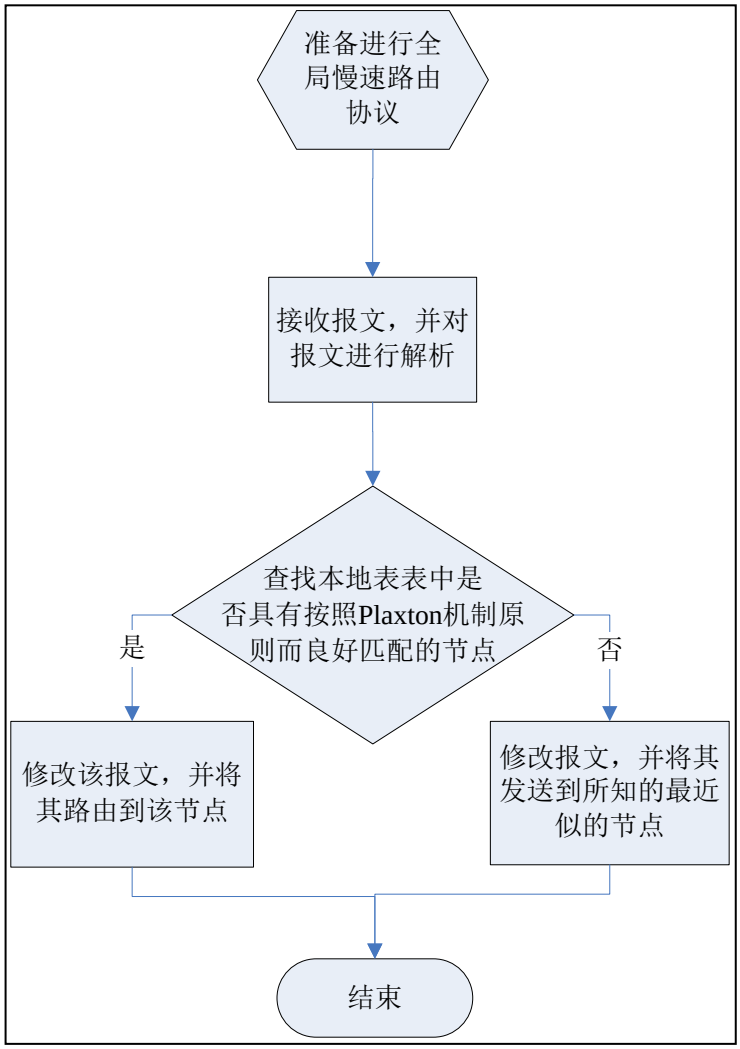

图4-4　全局慢速协议流程图

4.3 P2P 仿真实现

在 4.2 中描述了 OceanStore 路由协议的方针算法，下面结合具体的程序说明如何具体地进行仿真。

4.3.1 基本数据结构

在网络中，存在一些最基本的最基本的数据结构，比如各种报文的字段、路由表字段、节点所含有的资源格式等等。下面将介绍在 PeerSim 中对这些基本数据结构实现的类。

4.3.1.1 报文结构

在仿真中，最重要的基本数据结构应当算是报文了。因为他是整个查询过程的最基本元素，通过查询报文、相应报文等才能够发起查询操作，否则整个过程无法进行。

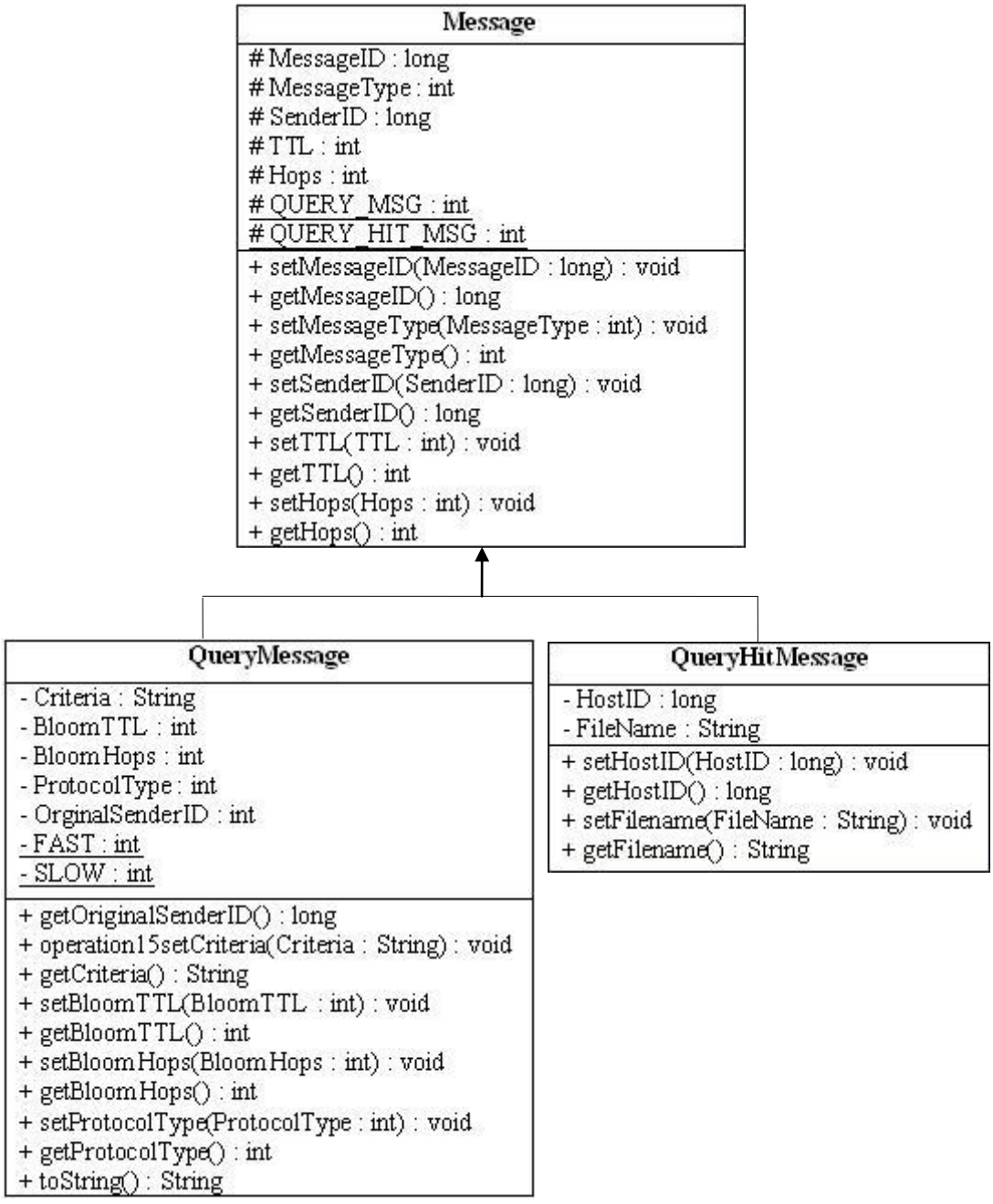

图4-5 各种报文类图

1.Message 类

Message 类是最为普通的类，它包含基本报文的基本域，如：报文 ID，发送方，TTL，跳数等等。它是 QueryMessage 和 QueryHitMessage 的父类。

2.QueryMessage 类

在这里，QueryMessage 的基本域与一般的查询报文相同。但不同的是，QueryMessage 含有一些特殊的附加的域，而这些域正是 QueryMessage 依据 OceanStore 所需数据结构而量身定做的设置。在这里，QueryMessage 利用 Bloom TTL 和 Bloom Hops 来标明 Bloom 过滤器的生命周期和当前深度，为是否进行和进行何种深度的局部快速路由提供依据。ProtocolType 用于指明当前协议类型，它的值为 FAST 或者 SLOW。相应地，静态字段 FAST 和 SLOW 则是利用整型来分别表示局部快速协议和全局慢速协议。

3.QueryHitMessage 类

QueryHitMessage 与一般的响应报文大致相同。当找到资源后，响应报文返回一个包含自身信息的相应报文。

4.3.1.2 本地资源列表结构

fileList 队列中存放的是本地共享资源的资源名。列表中的每个资源用 String 类型的数据进行表示。fileList 采用的是 LinkList 数据结构，而其元素类型就是 String。其结构如下：

fileList
(String)本地资源 1
(String)本地资源 2
……

图4-6　本地资源列表格式图

4.3.1.3 路由表结构

在本仿真中，路由表中的表项设置得非常简单，仅记录报文 ID 和上一个发送方的节点 ID。其格式如下：

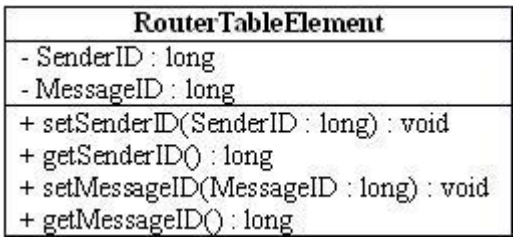

图4-7 路由表表项类图

4.3.2 通过 PeerSim 配置文件来进行网络设置

在 PeerSim 的仿真平台中，提供了一种方便的基于配置文件来对仿真进行框架式说明的设置方式。通过配置文件，底层平台可以方便地获取仿真人员所设置的各项参数，如：网络规模、执行周期、网络规模等等。

4.3.2.1 设置基本参数

simulation.cycles 15

network.size 30

network.maxsize 2000

从上面截取的部分的配置文件，可以很清楚地看到仿真周期被设置为了 15，当前网络规模（即网络中节点的数目）被设置为了 30，网络最大所能容忍的规模被设置为了 2000。

4.3.2.2 生成随机拓扑结构

protocol.0 peersim.core.IdleProtocol

protocol.0.k 3

网络拓扑结构的建立是采用 Peersim 中的 WireRegularRandom 类。该类用于随机生成一个指定节点数的正则拓扑图，并且设定网络中每个节点的最大度数，同时初始化每个节点同邻居节点的连接。

4.3.2.3 定制协议

protocol.1 OceanStoreSim.Protocol.OceanStoreProtocol

protocol.1.linkable 0

protocol.1.dcount 35

protocol.1.bloomarray 3

protocol.1.vectorbit 1000

protocol.1.hashnum 10

protocol.1.filepath D:\ Project \ P2P(NSF) \ Eclipse workspace \ OceanStore_CD \ resourcefile \ resource.txt

在设置协议时，将 OceanStoreSim.Protocol 包中的 OceanStoreProtocoll 这个类设置为协议类 1。在以后的仿真程序中，OceanStoreProtocol 的 ProtocolID 就为 1。

4.3.2.4 初始化仿真程序

在 PeerSim 中，网络结构的初始化时通过 Control 接口而完成的。在这里，该初始化包括为节点分配资源类(ResourceDistributionInitializer)以及 Query 消息初始化类(QueryMessageInitializer)。

1.生成虚拟网络

init.0 peersim.dynamics.WireKOut

init.0.protocol 0

init.0.k 3

WireKOut 是 PeerSim 中的一个用于表示链接的类，初始化这个类可以生成一个节点之间相互连接的网络结构。而该链接是方式是由 protocol 0 所指定，也就是之前被设置为 protocol 0 的 IdleProtocol。因此，节点之间的链接是随机生成的。此外，k 指定了单个节点的邻居节点个数为 3，即：该节点的出度为 3。

2.仿真程序初始化

init.1 OceanStoreSim.Initializer.ResourceDistributionInitializer

init.1.protocol 1

init.1.nodecount 10

init.1.filemax 5

init.1.filecopy 3

init.1.filepath D:\ Project \ P2P(NSF) \ Eclipse workspace \ OceanStore_CD \ resourcefile \ resource.txt

仿真程序的初始化由此开始。仿真平台首先执行 OceanStoreSim.Initializer 这个包中的 ResourceDistributionInitializer 类，该类的作用是向网络中的节点分配资源。其中，初始化所采用的协议为 protocol 1，即之前设置为 protocol 1 的 OceanStore 协议类。被分配资源的节点个数为 10。单个节点所可能含有的最多资源数为 5 个。资源在网络中存储的副本有 2 个，即某种资源一共存在 3 个。最后，filepath 指定了从外部读入资源列表的文件路径。

init.2 OceanStoreSim.Initializer.QueryMessageInitializer

init.2.protocol 1

init.2.nodecount 9

init.2.filepath D:\ Project \ P2P(NSF) \ Eclipse workspace \ OceanStore_CD \ resourcefile \ resource.txt

最后一次初始化是查询操作的初始化，调用的是 OceanStoreSim.Initializer 中的 QueryMessageInitializer 类。与 init.1 类似，对此初始化采用的协议也是 OceanStore 的协议。发起查询的节点数目为 9 个。由 filepath 指定资源字典路径。

4.3.3 主体仿真流程

当仿真程序开始执行时，PeerSim 平台按照节点 ID 的大小顺序依次执行各节点的周期。当执行完一个节点的周期后，再调用下一个节点的周期执行。为了完成这一功能，PeerSim 底层平台自动调用 nextCycle()函数来执行下一个节点的周期。当网络中所有节点的周期都被执行完一遍后，整个网络周期完成一次。

4.3.3.1 主函数: nextCycle()函数

在本 nextCycle()函数中，节点所要完成的任务是：在本周期内将报文队列中的报文（查询报文或者响应报文）处理完毕。因此，应当向节点的报文队列 MessageQueue 中循环取出待处理报文，直到报文队列为空。

而 MessageQueue 中含两种类型的报文：查询报文和响应报文。因此，节点就应当区别这些报文并执行不同的操作。在这里，采用的是 switch()函数来对报文类型加以区分，进而转向不同的处理方式（即 Query()函数或者 QueryHit()函数）。

在处理完了本节点 MessageQueue 中的所有报文后，清空 MessageQueue。为了保证 MessageQueue 中元素的完整性以及程序别写的简易性，仿真程序在获取了其中的报文后执行克隆操作创建一个副本来使用，而不是直接在 MessageQueue 上操作。因此，实际上，MessageQueue 里已经处理过的报文并没有被删除。所以在处理完了队列中的所有的报文后，需要对其进行清空。

4.3.3.2 OceanStore 路由函数: Query()函数

Query()函数就是处理查询报文的函数，所有的查询操作都是通过此函数而进入的。而在这篇文章中，主要集中在 OceanStore 的查询路由机制上。因此，这部分是整个算法思想体现的核心之处。下面将结合部分核心程序进行具体分析。

1.起始查询处理流程

if(this.routerTableList.containsMessage(queryMessageTemp.getMessageID())==-1)

代码功能描述：判断该节点之前是否已经处理过该报文，即：本地路由表中是否已经包含该报文。若已经处理，则不进行任何操作；若未处理，则在路由表中添加该报文相关表项，并且进入下一步操作。

if(this.fileList.contains(queryMessageTemp.getCriteria()))

代码功能描述：当节点决定处理该报文后，第一步操作就是首先判断报文所要查找的资源是否存在于本地资源列表上。如果在本地资源列表中查找到了所需资源，则产生一个新的响应报文，添加到报文队列末尾。如果失败，则进行下一步转发。

if(queryMessageTemp.getTTL() >= 1)

代码功能描述：当节点没有在本地查找到资源时，则需要判断该报文的生命周期是否截止。如果报文的生命周期已经截止，则节点依旧对报文不进行任何处理，本次处理结束。否则，报文将被继续转发给下一个节点，此时为 OceanStore 路由的核心部分。

2.区分报文协议类型

if(queryMessageTemp.getProtocolType()==QueryMessage.FAST)

代码功能描述：通过此 if 语句，程序判断当前报文应当采用何种协议进行发

送。仿真程序就是从这里开始对 OceanStore 的两种协议进行区分。在这个 if 语句之后，分别是局部快速协议。而相应地，else 之后则是全局慢速协议。

3.执行局部快速协议

if(!FastProtocol.fastRouting(queryMessageTemp, node, protocolID, this.numKey, this.numHashFunctions))

FastProtocol 是用来实现快速协议的类。在这个类中，只有一个用于执行的静态函数 fastRouting。在 fastRouting 函数中，各个传入参数分别为：查询报文、当前节点 ID，协议类型（此处将始终为 OceanStoreProtocol，因为本仿真涉及的协议只有 OceanStore 协议），Bloom 过滤器所使用的向量长度，Bloom 过滤器所使用的哈希函数个数。当存在满足转发条件的邻居节点时，即：成功进行局部快速路由时，函数将返回 true 来表示执行成功，否则返回 false 表示局部快速路由失败，进而指示报文进入全局慢速协议路由阶段。

由此可知，fastRouting 实际上就是充当的局部路由协议。那么究竟是如何让实现的呢？下面描述 fastRouting 内部是如何进行路由的。

a) 获取各个深度的 Bloom 过滤器

在局部快速路由中，关键的是获取指定深度内的邻居节点集合的 Bloom 过滤器。本仿真主要是通过下述函数获取 Bloom 过滤器的。

LinkedList<FileElement> fileList = getBloomFilterList(neighbor, queryMessage.getBloomTTL()-1, 0, protocolID);

在这个函数体中，主要是通过深度遍历的迭代思想来实现目的的。首先通过 if(bloomTTL>1)来判断节点是否已经处于所要求的最大深的。若已经是最大深度，那么将本地的资环添加入 Bloom 过滤器中并返回给上一级。若还未达到最大深度，则继续向周围邻居节点查询其资源列表，当获得其列表后将这些资源添加到相应深度的 Bloom 过滤器中，最后将自己的本地资源也追加到相应深度的 Bloom 过滤器中，在返回给自己的上级。这样，通过一层一层的反馈，节点最终就能通过 Bloom 过滤器预测在一定深度情况的所有邻居节点的资源含有情况。然后节点将所查询的文件一次用各个不同直接邻居节点的各个不同深度的 Bloom 过滤器进行验证。选取含有满足最小深度的 Bloom 过滤器的节点作为转发对象。

b) 向选中的邻居转发报文

peerNeighbor.messageQueue.add(new QueryMessage(queryMessage.getMessageID(),neighbor.getID(), queryMessage.getTTL()-1,queryMessage.getHops()+1, queryMessage.getOriginalSenderID(),queryMessage.getCriteria(), queryMessage.getBloomTTL()-1, queryMessage.getBloomHops()+1));

由于节点已经确定了合适的直接邻居节点，因此可以通过简单的方式向该邻居节点转发报文，即：产生一个更新（修改 Bloom TTL， Bloom Hops 等）后的报文并添加到该节点报文队列中等待该节点自己的处理周期到来后进行处理。

4.执行全局慢速协议

在全局慢速协议中，节点主要利用的是邻居映射表来实现操作。此时，节点依据邻居映射表将新的查询报文发到表中所对应的节点中去，即在此节点的报文队列中添加新的报文。

第5章 仿真实验数据分析

通过本仿真程序，可以对 OceanStore 路由机制的算法进行验证、获取所需数据等等。如无特殊说明，网络实验参数将采取如下默认参数：

1.网络规模（节点数目）：100

2.发起查询数：30

3.网络内资源数目：5

4.单个文件副本个数：3

5.Bloom 过滤器最大深度：3

5.1 网络中不同资源数目对查找结果的影响

此时，修改网络中资源的数目由原来的固定值 5 修改为从 2 至 10 的变化范围，进而可以得出网络中资源数目对查询结果的效果。

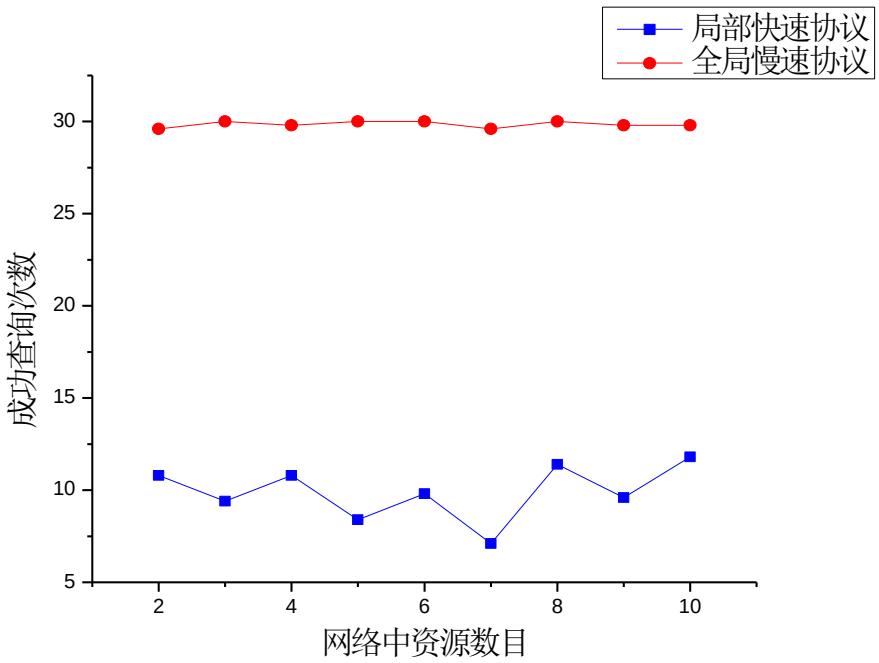

图5-1　网络中资源数目对查询效果的影响

从图 5-1 可以看出，当网络中的资源数目变化对局部协议查询效果和全局查询

效果影响均不大。此外，还可以发现通过全局慢速路由的成功次数比局部快速协议的成功次数多，并且方差小。

这表示全局慢速协议比局部快速协议准确率高。而且，从波动情况来说，全局慢速协议具有更好的稳定性，局部快速协议则相对较差。这正体现了 OceanStore 中局部快速协议是概率性的查找方式和局部快速协议只能在一定范围起作用这些特性。并且，这也验证了对于 OceanStore 这一全球广域范围这一理念的一个重大优势：无论资源在哪里，数目有多少，只要该资源存在就能在几乎相同的效率下获取。

5.2 网络中单个资源副本数目对查找结果的影响

此时，修改网络中单个资源副本数目由原来的固定值 3 修改为从 2 至 10 的变化范围，进而可以得出网络中资源数目对查询结果的效果。

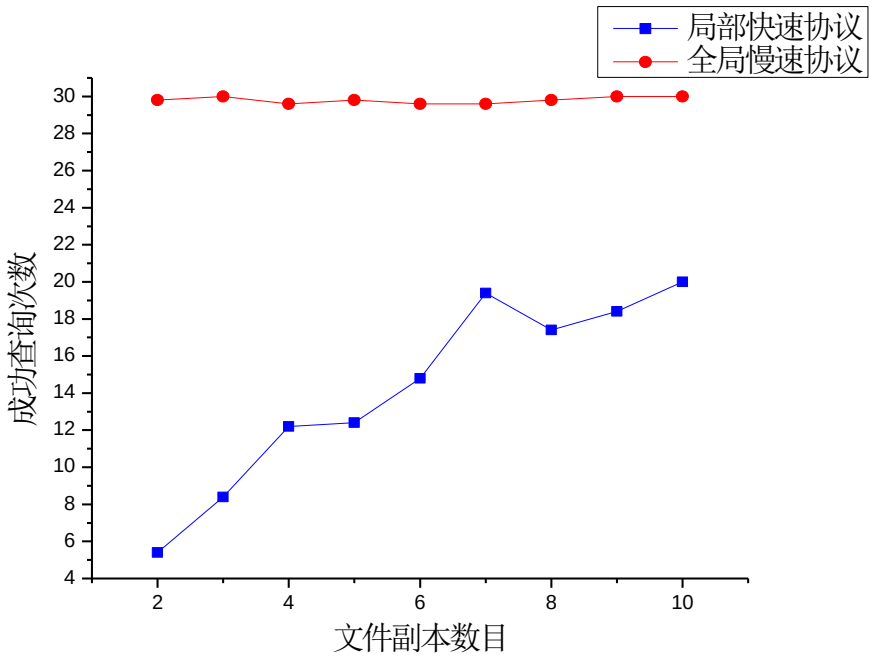

图5-2　单个文件副本数目对查询效果的影响

从图中可以看出，当文件的副本变化时，含有全局慢速协议的查询效果变动很小，局部快速协议变化较大，这与上一仿真实验得出的结论相同。此外，更重要的是，局部快速协议的查找成功次数随着单个文件的副本个数而显著上升。

由于单个文件的副本数目增多，使得网络中分布着更多的所需资源。这样，就加大了所需资源存在于查询节点周围的概率。当这一概率上升后，相应地，局部快速协议的查找成功概率也在上升，这就使得图中的查询成功次数曲线整体上呈现上升的趋势。

5.3 Bloom 过滤器最大深度对查找结果的影响

在 OceanStore 中利用了 Bloom 过滤器这一重要的数据结构来实现局部快速协议。现就修改 Bloom 过滤器由原来的固定值 3 修改为从 2 至 10 的变化范围，观察 Bloom 过滤器对于对查询结果的影响。

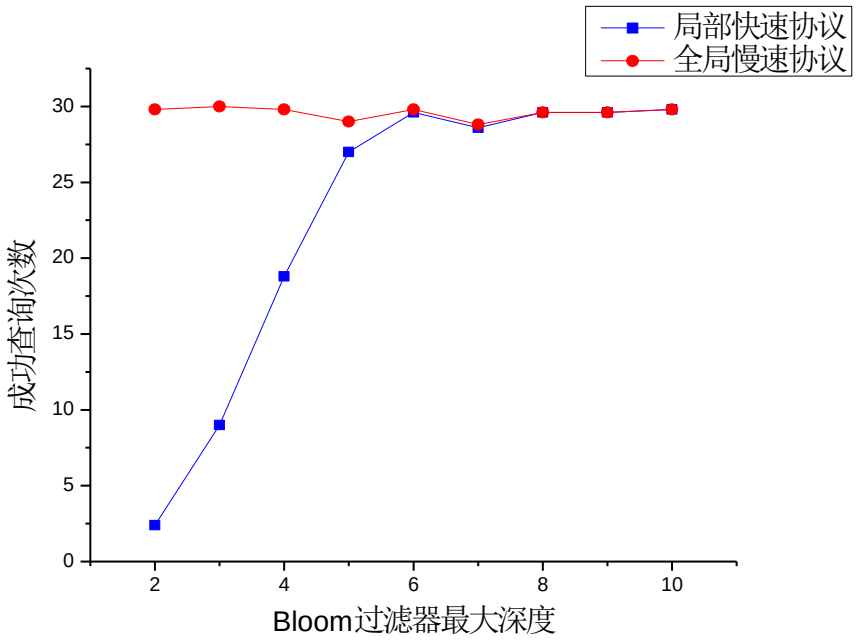

图5-3　Bloom 过滤器深度对于查询效果的影响

由图 5-3 可以很直观地发现，当 Bloom 过滤器的深度增加时，局部快速协议的成功查询次数快速上升，并且在 Bloom 过滤器深度为 8 的时候局部快速协议的查找成功次数与总成功查询（即包含了全局慢速协议的查询）次数持平。

在起始时刻，由于 Bloom 过滤器的深度过低，导致局部快速协议查询效率很低。但是，随着 Bloom 过滤器最大深度的增加，局部快速协议的查询成功次数有了很大的改观。这种增长的速度是快速的，这源于局部快速协议的基本特性。因

为每当增加一个深度（Bloom TTL 增加一跳），那么节点所能获取直接或间接邻居节点集合的信息也就更多，并且这种信息的增长是呈指数的。这种呈指数的增长与节点的度（即与之相连的邻居节点个数）有关，多得知的间接邻居的增长数目为节点的度乘以上次最深深度间接邻居节点的个数。由图中可以看出，当 Bloom 过滤器的深度达到一定程度后，其局部快速协议成功率与全局慢速协议成功率相同，这意味着局部快速协议此时仅仅能够通过 Bloom 过滤器就能得知几乎整个网络的资源归宿情况。

5.4 整体小结

通过上面的三个仿真可以总结出 OceanStore 所具有的一下几个基本特征：

1.全局慢速协议较局部快速协议稳定，并且普遍来说查询成功率较高。

2.局部快速协议受单个资源的副本影响较大，因为它是基于概率性的、仅在节点周围地区进行路由的，而文件副本的增多就使得资源分布在查询节点周围地区的概率提高，进而影响局部快速协议的查询效果。而全局慢速查询并不与资源所在位置相关，因此其查询效果与副本个数关联微弱。

3.局部快速协议在很大程度上依赖于 Bloom 过滤器的深度。当 Bloom 过滤器的深度加深时，局部快速协议能跟迅速增长其查找成功的概率。而全局协议并不依赖于 Bloom 过滤器，因此全局协议与 Bloom 过滤器的关系十分微弱。

4.尽管 Bloom 过滤器深度的增加能跟很大程度上帮助局部快速协议，但是随着 Bloom 过滤器深度的增加，节点将获得更多间接邻居节点几个的信息。而这样会增大网络带宽的消耗。Bloom 过滤器的优势在于其仅仅是利用字节向量来判断文件是否属于某一集合，而不是在原有文件上直接判断，因此节省了大量的带宽。但是这也并不意味着能够任意加深 Bloom 过滤器的深度。

结束语

分布式应用已经深入到了互联网的各个角落，成为不可或缺的重要元素。随着分布式技术的迅速发展和趋于成熟的机制，其应用被扩展到了越来越多的领域。如今分布式存储正在兴起，比如云计算、云计算等，均被 Microsoft，Amazon，IBM，Google 等公司提上日程并且已经投入商业中运营。分布式存储有着很大的开发潜力以及应用前景，因此对分布式存储进行深入的研究是非常有意义的。

本文主要就全球首个分布式存储系统 OceanStore 进行了介绍。并且采用 PeerSim 仿真平台对 OceanStore 进行系统仿真，来研究 OceanStore 这一系统的路由特性，以便对其进行更加深入透彻的理解，为进一步的科研、工作打下良好的基础。

参考文献

[1] Atul Adya, William J. Bolosky, FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, December 2002

[2] PIon Stoica, PRobert Morris, PDavid Karger, PM. Frans Kaashoek, PHari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the 2001 SIGCOMM conference, San Diego, CA, 2001

[3] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica. Wide-area cooperative storage with CFS. In Proceedings of the eighteenth ACM symposium. Alberta, Canada, 2001

[4] Ben Y. Zhao, John D. Kubiatowicz, Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report : University of California at Berkeley, Berkeley, CA, Apr.2001.

[5] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski etc.OceanStore: An Architecture for Global-Scale Persistent Storage, ACM SIGOPS Operating Systems Review, 2000, Vol34, No.5:190~201

[6] Thomas J. Giuli, Mary Baker. Narses: A Scalable Flow-Based Network Simulator. Stanford University, Technical Report, 2002

[7] OMNet++. OMNet++ Community Site. 2006. http://www.omnetpp.org/

[8] SSFNet. Scalable Simulation Framework Net. 2006. http://www.ssfnet.org/

[9] Thomas J. Giuli, Mary Baker. Narses: A Scalable Flow-Based Network Simulator. Stanford University, Technical Report, 2002

[10] Sam Joseph. NeuroGridy in Peer-to-Peer Networks. Proceedings of the International Workshop on Peer-to-Peer Computing, 2002

[11] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System, H. Federrath (Ed.): Anonymity 2000, LNCS 2009, pp. 46~66, 2001

[12] P2PSim. P2PSim: A Simulator for Peer-to-Peer (P2P) Protocols. Technical Report, 2005. http://pdos.csail.mit.edu/p2psim/

[13] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, Robbert van Renesse. Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead.

Cornell University, Ithaca, NY, USA:Springer Berlin / Heidelberg, 2003.160~169

[14] P. Druschel, F. Kaashoek, A. Rowstron. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. New York University, USA:Springer-Verlag Berlin Heidelberg, 2002.53~65

[15] Pedro Garcia, Carles Pairot, Ruben Mondejar, et al. PlanetSim: A New Overlay Network Simulation Framework. Lecture Notes in Computer Science, Software Engineering and Middleware. Springer Berlin/Heidelberg:, 2005, 3437: 123-136

[16] K. Shudo. Overlay Weaver. 2006. http://overlayweaver.sourceforge.net/

[17] Gian Paolo Jesi. The BISON Project and the Peersim P2P Simulator. Biology-Inspired Techniques for Self Organization in Dynamic Networks (BISON), Technical Report, 2005

[18] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, 1970, Vol:13, No.7

[19] C. Greg Plaxton, Rajrnohan Rajararnan, Andrea W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. Theory of Computing Systems, 1999, Vol.32, No.3:241~280

致谢

首先，感谢我的指导老师钟婷博士。在进行毕设期间，我遇到了很多困难，钟老师总是耐心细致地引导我寻找合适的方法解决这些困难。尽管钟老师事务繁忙，但每周都与我进行交流，关心我的毕业设计情况。十分感谢钟老师在这一路上对我的帮助、支持和鼓励。

此外，还要感谢孙正隆学长。孙学长对于 PeerSim 仿真平台有着深入细致的了解，当我遇见解决不了的技术问题后孙师兄总是毫无保留地将自己的经验传授给我。

最后，还要感谢刘梦娟博士、杨磊博士、杨毅同学、廖晓鹃同学。在与他们的讨论中，我获益匪浅。

外文资料原文

OceanStore: An Architecture for Global-Scale Persistent Storage

John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao

INTRODUCTION: Distributed Routing in OceanStore

Every addressable entity in the OceanStore (e.g. floating replica,archival fragment, or client) is identified by one or more GUIDs. Entities that are functionally equivalent, such as different replicasfor the same object, are identified by the same GUID. Clients interact with these entities with a series of protocol messages, as described in subsequent sections. To support location-independentaddressing, OceanStore messages are labeled with a destination GUID, a random number, and a small predicate. The destination IP address does not appear in these messages. The role of theOceanStore routing layer is to route messages directly to the closestnode that matches the predicate and has the desired GUID.In order perform this routing process, the OceanStore networking layer consults a distributed, fault-tolerant data structure that explicitly tracks the location of all objects. Routing is thus a twophase process. Messages begin by routing from node to node alongthe distributed data structure until a destination is discovered. Atthat point, they route directly to the destination. It is important tonote that the OceanStore routing layer does not supplant IP routing,but rather provides additional functionality on top of IP.There are many advantages to combining data location and routing in this way. First and foremost, the task of routing a particularmessage is handled by the aggregate resources of many differentnodes. By exploiting multiple routing paths to the destination, thisserves to limit the power of compromised nodes to deny serviceto a client. Second, messages route directly to their destination,avoiding the multiple round trips that a separate data location androuting process would incur. Finally, the underlying infrastructurehas more up-to-date information about the current location of entities than the clients. Consequently, the combination of locationand routing permits communication with “the closest” entity, ratherthan an entity that the client

might have heard of in the past. If replicas move around, only the network, not the users of the data, needsto know.

The mechanism for routing is a two-tiered approach featuring afast, probabilistic algorithm backed up by a slower, reliable hierarchical method. The justification for this two-level hierarchy isthat entities that are accessed frequently are likely to reside close towhere they are being used; mechanisms to ensure this locality aredescribed in Section 4.7. Thus, the probabilistic algorithm routesto entities rapidly if they are in the local vicinity. If this attemptfails, a largescale hierarchical data structure in the style of Plaxton et. al. locates entities that cannot be found locally.

.

翻译文稿

OceanStore:全球范围永久存储的一种架构

John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao

介绍：OceanStore 中的分布式路由

OceanStore 中的每个可寻址实体（例如，浮动的复制、归档片断或客户端）是通过一个或多个 GUID 而加以识别的。功能等价的实体，例如相同对象的不同复制，使用相同 GUID 加以标识。客户端与这些实体使用一系列协议消息交互，如后面的小节所描述的。为了支持位置无关的寻址，OceanStore 消息以一个目的地 GUID、一个随机数和一个小的谓词作上标签。在这些消息中目的地 IP 地址不出现。OceanStore 路由层的角色是将消息直接路由到匹配谓词并具有期望 GUID 的最近节点。为了执行这个路由过程，OceanStore 网络层咨询一个分布式的、容错的数据结构，该结构显式地跟踪所有对象的位置。因此路由是一个两阶段过程。消息眼分布式数据结构从节点到节点开始路由，直到发现一个目的地。在那个点，消息直接路由到目的地。重要的是指出，OceanStore 路由层不取代 IP 路由，而是在 IP 之上提供额外的功能。以这种方式将数据定位和路由相结合，存在许多优势。首先也是最重要的，一条特定消息的路由任务是通过汇聚许多不同节点的资源得以处理的。通过利用到目的地的多条路由路径，这就限制了被攻破节点对一个客户端拒绝服务的能力。第二，消息直接路由到它们的目的地，避免了一个独立的数据定位和路由过程将引发的多次往返。最后，底层基础设施具有比客户端更新的实体当前位置信息。结果就是，定位和路由的组合允许与“最近”实体的通信，而不是客户端过去听到过的一个实体通信。如果复制移动了，仅有网络，而不是数据的用户，需要知道这个信息。

路由机制是一种两层方法，其特点是一个快速的、概率性的算法由一个较低速的、可靠的等级化方法所备份。这种两等级层次结构合理的理由是，被频繁访问的实体极可能驻留于它们正在被使用的附近；确保这种局部性的机制在 4.7 节描述。因此，概率性的算法快速地路由到实体，如果它们在局部临近范围的话。如果这种企图失败，则以 Plaxton 等风格的一种巨大规模等级层次数据结构就用来定

位不能局部找到的实体。