P2P 持久存储研究综述*

田 敬[1+], 代亚非[1]

[1](北京大学 计算机科学技术系,北京 100871)

A Survey of Durable Peer-to-Peer Storage Techniques*

Jing Tian[1+], Yafei Dai[1]

[1](Department of Computer Science and Technology, Beijing University, Beijing 100871, China)

+ Corresponding author: Phn: +86-10-62751799-8010, E-mail: tianjing@net.pku.edu.cn, http://www.pku.edu.cn

Abstract: Peer-to-Peer (P2P) has been one of the most important architectures for Internet applications for its inherent scalability, fault tolerance and high performance. The research of P2P storage systems is one of the hot issues, and P2P storage system is regarded as one of the most promising P2P applications. However, to provide durable data storage is not trivial work and sets great barrier to real deployed systems. This survey paper surveys the P2P storage systems and techniques for durable storage. We first introduce the basic components of a durable P2P storage system and the advantages by using P2P architecture. After presenting the research framework, we introduce some typical P2P storage systems and the techniques they adopted. By a detailed comparing, we discuss the pros and cons of the techniques for different environments, the problems in current research and some future research issues.

Key words: survey; Peer-to-Peer; storage; durability; redundancy; placement; failure detection; maintenance

摘 要: P2P(Peer-to-Peer)的组织模式已经成为新一代互联网应用的重要形式,它为应用带来了更好的扩展性,容错性和高性能等特点.P2P 存储系统一直是研究界所关注的热点,被认为是 P2P 最具前途的应用之一.数据的持久存储是制约 P2P 存储系统发展的关键问题,也是其研究的难点.本文综述了 P2P 存储系统及数据持久存储相关技术的研究现状.首先概述了 P2P 存储系统的基本概念及其在不同应用环境中的优势,并介绍了数据冗余,数据分发,错误检测和冗余数据维护等多种持久存储的基本技术.在一个 P2P 存储系统研究框架下,介绍了目前知名的 P2P 存储系统及其使用的持久存储技术.对于各种技术进行了详细综述和对比讨论,分析各种技术的适应环境及优劣,指出了存在的问题和未来研究的方向.

关键词: 综述; 对等网络; 存储系统; 持久性; 冗余; 数据分发; 错误检测; 数据维护

中图法分类号: TP393 **文献标识码**: A

近年来，P2P (Peer-to-Peer)技术在即时通讯、文件共享及流媒体传输等应用领域均显示出了极大的优势，

* Supported by the National Grand Fundamental Research 973 Program of China under Grant No.2004CB318204 (国家重点基础研究发展规划(973))

作者简介: 田 敬(1979－),男,北京顺义,博士生,主要研究领域为 P2P 存储系统;代亚非(1958－),女,博士,教授,博士生导师,主要研究领域为 P2P 计算,分布式存储.

成为构建新型大规模互联网应用的主要结构和技术之一。由于巨大的技术挑战，P2P 存储应用并没有获得相应的商业成功，然而 P2P 存储系统一直被 P2P 科研社区认为是极富前景的一种应用，并一直是学术界研究的热点。本文将综述 P2P 存储系统发展现状，指出面向持久存储的技术挑战和目前研究的主要成果。

P2P 存储系统，也即对等存储系统，是指存储节点以一种功能对等的方式组成的一个存储网络。这种结构是与传统的客户/服务器的集中控制模式相对应的。本文中的 P2P 专指“功能对等的节点组织方式”，而非专指一般用户桌面机所组成的系统。P2P 文件共享系统一般指由用户桌面机组成的 P2P 网络，利用用户桌面机之间的带宽来解决服务器性能瓶颈问题。然而，P2P 存储系统既可以是完全由服务器节点以对等方式组成，又可以是完全由用户桌面机组成，也可以是服务器与桌面机共同以对等的方式组成的存储系统。可以说，P2P 技术既可用来组织专业的大型存储服务，也可以用来组织闲散的桌面机资源形成互助存储网络。综上所述，只要以功能对等的方式组织起来的存储系统均属于 P2P 存储系统。

由于采用对等互连的技术，P2P 存储系统相比传统的存储系统有如下优点：不依赖中央控制，系统自然具有高扩展性，且不存在单点性能瓶颈问题；各个节点功能对等，使得整个系统在缺失任意节点后仍能正常工作，也即有高容错性；高扩展性和高容错性进而使得利用廉价机群搭建大规模高性能存储服务成为可能；由于没有也没法进行中央控制，P2P 存储系统能够极大减小存储系统总的拥有开销（TCO, Total Cost of Ownership）[1]；由桌面机组成的 P2P 存储系统，每个节点将可以利用互联网的边界带宽资源存储数据，极大的提高传输速度。

虽然 P2P 系统有与生俱来的高容错潜力，但 P2P 系统中每个节点都可能随时暂时或永久的离开系统，这使得构建 P2P 存储系统极富挑战。系统中的节点均负责存储数据，一旦某节点暂时离开，存在其上的数据就将暂时不可访问，而节点的永久离开更会造成数据的丢失。因此，如何提供数据的持久存储，屏蔽这些系统错误成为近年来 P2P 存储领域的研究热点。

本文第 1 节介绍构建 P2P 存储系统及提供数据持久存储的基本技术，并将系统按动态性进行分类；第 2 节给出整个 P2P 存储系统的研究框架，并介绍系统及相关研究的发展历史；第 3 节分别介绍各重要的存储系统，讨论它们的技术特点；第 4 节介绍和讨论目前前沿的研究结果；第 5 节总结全文。

1 P2P 持久存储的基本技术及系统分类

本节将以构建一个简单的抽象 P2P 存储系统为目标，简要而直观的介绍组成系统所需要的结构化覆盖网络及保证数据持久存储的相关技术，最后本节按系统所适应的目标环境将系统分类。

这里要强调的是，本文中所指的 P2P 文件存储系统均指“基于结构化 P2P 覆盖网络”的存储系统。这也是目前学术界的默认观点。因此，本节将首先介绍一个简单的结构化覆盖网络，并在其基础之上介绍存储相关的基本技术。

1.1 结构化P2P覆盖网络基本概念

结构化 P2P 覆盖网络是一种维护节点之间在应用层上互联的组织方法。它按照一定的逻辑拓扑结构将系统中的节点互连起来，并通过路由消息使得系统中任意两个节点可以互相通信。在有节点动态加入和退出的情况下，结构化 P2P 覆盖网络要能够保证节点之间的互连性。

为了让节点互相认识，首先需要为每个节点命名。结构化 P2P 网络中节点的名字一般是一个名字空间内的一个数值，例如 Chord 路由算法[2]的名字空间是一个环形，如图 1 所示。当有节点加入系统时，节点随机选择一个环上的位置作为自己的标识符（ID），如图 1 左边一个节点加入并以 0 作为自己的 ID，而右图中则有更多的节点加入了这个系统。

每个节点获得了一个唯一标识后，就可以定义节点之间的互连关系，例如图 1 的网络中可以定义每个节点认识与其在环上左右相邻的两个节点，也即知道它们的 IP 地址。这样，图 1 中的节点 0 就认识节点 13 和节点 3，而节点 3 就认识节点 0 和节点 6，进而每个节点都可以通过自己的邻居间接的认识网络中所有其它的节点。当有新节点加入时，也需要继续保持这一规则。例如，一个节点要以 2 这个 ID 加入系统，那么它可

能先联系到了节点 13，而节点 13 就会介绍它认识节点 0 和节点 3；节点 0 和节点 3 得知有节点 2 要加入后，会更新自己的路由表，让节点 2 成为自己的新邻居，以使得这个环形拓扑继续保持。同样，覆盖网络也要处理节点离开的情况。离开处理相对复杂，这里不做介绍，可以参考 Chord[2]、Tapestry[3]、CAN[4]、Pastry[5]和 Kademlia[6]等路由算法。

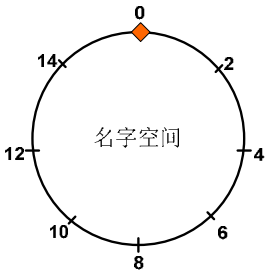
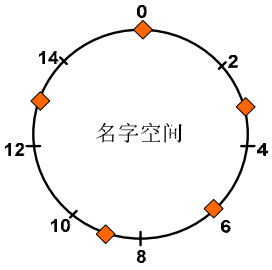


Fig.1 Namespace of structured overlay network

图 1 结构化覆盖网络的名字空间



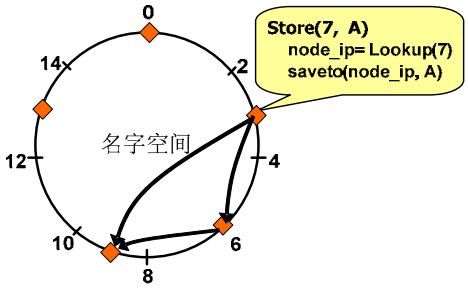


Fig.2 Store operation on a structured overlay network

图 2 结构化覆盖网络上的存储操作

要在这样的一个覆盖网路上存储数据，我们需要为数据命名，以便可以用名字取回存储的数据。数据的名字一般也是该网络名字空间里面的一个值，例如图 2 中 3 号节点将文件 *A* 命名为 7。存储时，我们希望数据存在以 7 为 ID 的节点上，以便读取时用文件名字定位。但不幸的是，7 号位置并没有物理节点。因此，我们需要定义一个区域负责制，也即谁负责那些还没有物理节点位置。我们可以简单定义为，排在这个位置之后的最近的一个节点负责这个位置。这样，图 2 中负责位置 7 的就是 9 号节点。节点 3 可以通过自己的邻居认识到节点 9，并将数据最终存储在 9 号节点上。当然，如果后来有 8 号节点加入网络，根据前面负责制的定义，*A* 文件就应该移动到 8 号节点上。这将造成不小的带宽消耗，下一小节会讨论这个问题的解决方案。

结构化 P2P 覆盖网络是一个独立的研究领域，近年来发展出了不少优秀的路由算法。它们虽然都是以分割 ID 空间和空间分区管理为核心思想，但其拓扑结构却多种多样。有代表性的包括，分割矩形空间的 CAN[4]，超立方体结构的 HyperCup[7]和基于抑或距离关系的 Kademlia[6]等。

本小节给出的只是一个简化版本的覆盖网络，其中一个节点只认识其左右两个邻居。这样势必造成节点查找任意 ID 位置的负责节点时，需要通过过多的邻接关系才能查到，效率非常低。为了解决这个问题，我们当然可以定义每个节点可以认识网络中所有其它的节点。这样，节点的任意查找都可以根据本地信息一次定位。但这种方案也增加了节点维护应用层路由表的负担。因此，一般的 P2P 覆盖网络都采用一种折衷的方案，让每个节点认识网络中 O(log*N*)个节点，这样可以使查找效率达到 O(log*N*)，其中 *N* 为网络中节点总数。优化的具体方法，请参考具体路由算法文章。

P2P 覆盖网络的另一个主要研究点是如何让网络在高抖动情况下（节点频繁加入和退出）保持正确的连通性。此研究方向，本文不展开讨论。在后续讨论中，我们简单假设覆盖网络能够保证节点之间的正确互连。

1.2 面向数据持久存储的相关技术介绍

在一个 P2P 覆盖网络中，我们一般假设系统中有海量的节点且每个节点可以自由加入和退出系统。因此，临时和永久的节点失效相比于传统系统将多很多。这样，一个 P2P 存储系统必须要用一定的策略屏蔽这些节点失效，保证数据的持久存储。本小节，我们将介绍其基本的方法。

1.2.1 数据冗余方案：副本与纠删码

对要存储的数据做一定的冗余，是在有节点失效情况下保证数据持久性的最基本和必要的手段。没有冗余的数据，节点退出后，其上的数据将必然无法恢复。

在 P2P 存储系统中，目前研究主要讨论两种冗余方法，完全副本冗余和纠删码冗余。完全副本冗余，顾名思义就是保存多个要存储的数据的完整副本。纠删码是指将要存储的数据先切分为 *m* 个部分，然后通过编

码算法变换为 n（$n>m$）个部分，其中任意 t（$t>=m$）个部分可以用来恢复原始数据。当 $t=m$ 时，我们称编码算法具有最大距离分割性质（MDS，Maximum Distance Separable）。

纠删码主要分为两大类：低密度校验码（LDPC， Low Density Parity Check）和 Reed-Solomon 码。其中低密度校验码[8]具有运算速度快的优势，但其不具有 MDS 性质；Reed-Solomon[9]码具有 MDS 性质，但是其运算速度相对较慢。

相对于完全副本方式，纠删码造成了一定的计算量，也增加了系统设计和实现的复杂度；但在一定情况下，它也能在较少冗余数据的情况下提供与副本方式相同的持久性。

1.2.2 数据分发方案：DHT 直接分发与基于目录分发

在做了数据冗余后，我们希望找到一个最合适的节点集合来存放这些冗余的数据，以达到最好的数据持久性。不适当的节点组合将可能极大的消耗系统带宽，甚至威胁系统中数据的持久性。例如，将数据的多个副本存放在一个错误相关的节点集合上，即节点集合中的节点可能由于区域断网或断电而同时离线，这样即便有多个数据副本，也容易出现数据不可用的情况。在 P2P 网络中，主要存在 DHT（Distributed Hash Table）直接数据分发和基于目录的间接分发两种形式。

DHT 直接数据分发指将数据及其副本直接存放在负责其名字对应位置的节点和其后续节点集上，如图 3 中左图所示。这种分发方式在覆盖网络中比较直观和容易实现，当负责数据的节点离开系统后，其后继节点自然成为该数据的负责节点，使得对数据的读操作仍可以成功。但其缺点也较明显，就是当有新节点加入并成为数据或其副本的新的负责节点时，就需要进行数据转移，浪费网络带宽。

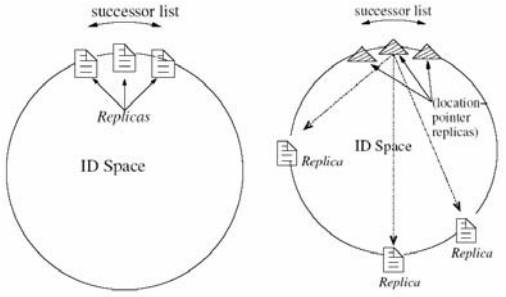


Fig.3[10] Two placement approaches

图 3[10] 不同的数据分发方式

基于目录的分发方式则将数据随意分发至网络中的任意无关节点集上，最后将这个节点集合的位置信息作为目录信息存放在负责数据名字区域的节点和后继节点集上，如图 3 中右图所示，其中三角形表示目录节点。当需要读取数据时，先读取目录信息，然后在定位到实际的数据存放节点。当新加入节点影响了区间的负责关系时，这种间接数据分发方式可以仅移动很小的目录数据，而不用移动数据本身，这就解决了 DHT 直接分发频繁移动数据的难题。另一方面，间接分发的方式增大了节点集合选择的灵活性，使得更多的基于节点性质的选择算法可以用来优化分发问题。具体的优化算法，我们将在后面介绍。

1.2.3 错误检查方案：定期心跳与失效广播

冗余的数据分发出去，会出现节点的失效，影响数据存储的持久性。因此，我们需要一套有效的机制发现系统中节点出现的错误，这就是错误检测。错误检测主要分为两种方式，定期心跳法和失效事件广播法。

定期心跳法指每个数据节点定期向其对应的目录节点报告自己存在的状态，如果对应的目录节点一段时间没有收到心跳则认为该数据节点下线，如图 4 中左图所示。另一些系统中，目录节点定期主动探测其负责的数据节点的活动状态，这种探测与被动接收心跳效果基本相同，本文不做区别。

错误事件广播则没有定期的状态报告，当出现一个节点下线后，其临近的节点将发现这个事件，并通过 P2P 覆盖网络的广播机制将这个事件广播给网络中所有其它的节点，需要检测这个节点的目录节点将会收到这个消息，如图 4 中右图所示。

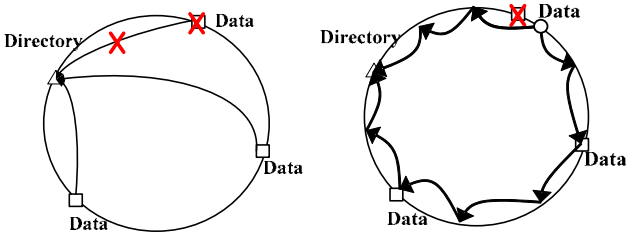


Fig.4 Failure detection: heartbeat vs. broadcast

图 4 心跳检测和错误事件广播法

1.2.4 冗余数据维护

发现系统中节点失效后，及时通过修复来补充丢失的冗余数据是维持数据长期持久的关键措施。不同于传统系统的硬件错误，P2P 系统中经常会出现节点暂时性离开，也即一段时间后节点还会回到系统中。如果每次碰到这种暂时性错误也都触发修复，将白白耗费大量带宽。因此，何时触发修复以及修复多少个的冗余数据成为一个重要的研究问题。

目前系统主要分为立即修复和延迟修复两种策略。立即修复就是发现节点失效马上进行修复，此方案多用于维护极其关键的目录信息。延迟修复则是等到失效节点数增至一定程度后才开始修复。

1.3 一个抽象的P2P存储系统

现在我们考虑利用上述基本技术构造一个抽象的可维护数据持久存储的 P2P 存储系统。假设使用上述的环形名字空间的覆盖网络，一次存储过程如下：首先，请求存储数据的节点将要存储的数据做 3 个完全副本的冗余；其次，将冗余的数据随机分发到网络中的 3 个节点上，并将这 3 个节点的网络 ID 作为目录信息保存在数据名字对应的负责节点上。

数据被存入系统后，3 个副本所在的节点定期向目录节点发送心跳数据包报告自己的在线状态；一个节点出现失效后，目录节点发现这个错误，并立即修复出一个新的数据副本，并更新自己的目录信息。

当客户要读取自己的数据时，先访问目录节点，获取当前 3 个副本所在节点的 ID 信息；然后通过覆盖网络联系到其中一个副本所在节点；最后，取回自己的数据。

1.4 系统分类

P2P 存储系统按其所适应的环境划分，可分为两大类：封闭式系统（closed system）和开放式系统（open system）[11]。

封闭式系统指，系统有较为严格的中心认证，显示审计和管理，以保证系统中节点的持续运行[11]。封闭系统中的节点可以认为相对较为稳定，虽然可能出现暂时错误，但是会及时修复并重新回到系统中，并且不会随意退出系统，节点之间非常协作。这类系统的代表是 PlanetLab[12]，它是由遍布世界的研究机构捐献的机器（目前约 700 多台）构成的一个覆盖网络，每个节点有专人负责维护，节点可靠性和可用性较高。在 PlanetLab 上搭建的存储系统包括 OceanStore[13, 14]的原型系统 Pond[15]和 Total Recall[16]等。

开放式系统指，每个节点可以随意加入或退出系统，节点不保证持续在线提供服务的系统。开放式系统中的临时错误（节点暂不在线）和永久错误（节点退出系统）相对于封闭系统更加频繁，节点之间的合作度很低，甚至有不少理性用户[17]的存在。这类系统的典型代表是桌面机所组成的网络，例如文件共享系统 Gnutella[18]、Napster[19]、Overnet[20]、KaZaA[21]和 Maze[22]等。目前由于开放式系统的动态性太强，还没有成熟的存储应用在开放式系统上成功部署。

当然 P2P 存储系统还可以按照其它的方式进行划分，如按为上层应用提供的接口语义可分为 P2P 文件系统和 P2P 存储系统。本文将以面向数据持久化存储为核心展开讨论，故系统的动态性是我们区别系统的主要指标，因此其它分类方法不展开讨论。

2 P2P 存储系统研究体系结构

前面介绍了 P2P 存储系统一些基本的概念和技术，本节将概览 P2P 存储系统的各种相关技术及其之间的关系，以及 P2P 存储系统和技术的发展关系。

2.1 P2P存储系统相关技术概览

P2P 存储系统及相关技术的研究是一个尚在起步阶段的复杂系统研究，它需要多方面技术和研究的支撑，其研究体系如图 5 所示。

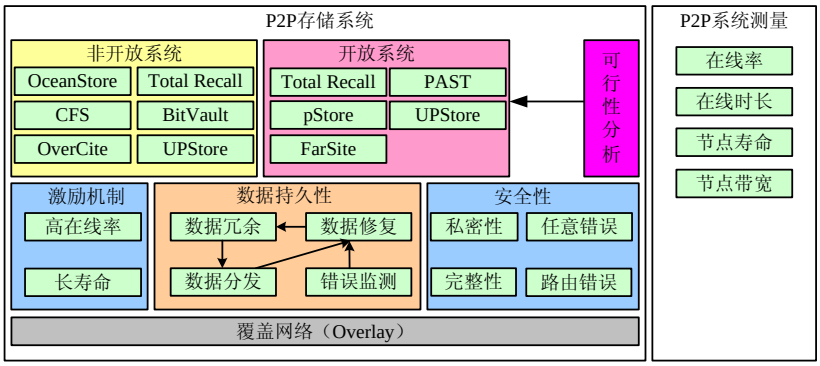


Fig. 5 Research framework of durable P2P storage system

图 5 P2P 存储系统相关技术关系

P2P 存储系统都是基于结构化 P2P 覆盖网络的，因此覆盖网络的研究是 P2P 存储研究的基础，而它本身也是一个较为独立的研究领域。

有了应用层网络互连之后，最大的问题就是将数据稳定的存储在这个不稳定的节点集上，因此面向数据持久性的技术和研究就成为核心，其中包括数据冗余方案，数据分发方案，错误检测方法和数据维护策略等。

P2P 存储是将用户的数据存储在其它的不可信节点上的，敏感数据的私密性和完整性自然就成为关键问题；另一方面，网络中的恶意节点还可能恶意破坏网络中运行的协议，破坏正常节点的路由，造成整个网络瘫痪。因此，保证数据私密性和完整性及发现和隔离恶意节点等安全措施也是 P2P 存储研究必不可少的一个部分。

P2P 网络中的普通用户一般都是理性用户，也即他们追求自己利益的最大化，他们更趋向于使用别人资源，而不贡献自己的资源。这就需要我们采用一定的激励机制来鼓励用户长时间在线，并尽可能多的贡献自己的存储资源。

不少研究机构也面向不同的应用环境建立了一些 P2P 存储系统，并通过系统的方法验证他们的相关研究。其中封闭式环境和开放式环境均有一些有代表性的系统设计，如图 5。其中特别指出，Total Recall 系统最初是考虑适应高动态的开放式环境，但由于其研究期间有 MIT 学者指出开放式环境中建立 P2P 存储是不经济的[23]，Total Recall 系统的原型就只建立在了 PlanetLab 环境上；另，UPStore[24]系统是一个开放式的存储框架，其设计目标就是利用替换框架中的个别算法模块，使得系统可以分别适应封闭式和开放式两种环境。

要进行相关技术的量化研究，我们需要实际系统中的运行数据，因此 P2P 系统测量是 P2P 存储系统研究的一个重要组成部分。

目前，在 P2P 系统测量工作的基础上，还有部分研究者在对开放式环境的 P2P 存储系统可行性和有效性进行研究。

2.2 P2P存储系统及技术发展历史

P2P 存储系统研究最初是被一些著名的 P2P 存储系统设计驱动的，也即先有了一些系统的想法，然后才

有各种相关的研究工作，因此 P2P 存储系统设计在这个研究领域非常重要，本小节将按研究机构和时间两个维度来总结和讨论各知名 P2P 存储系统和相关技术的发展。

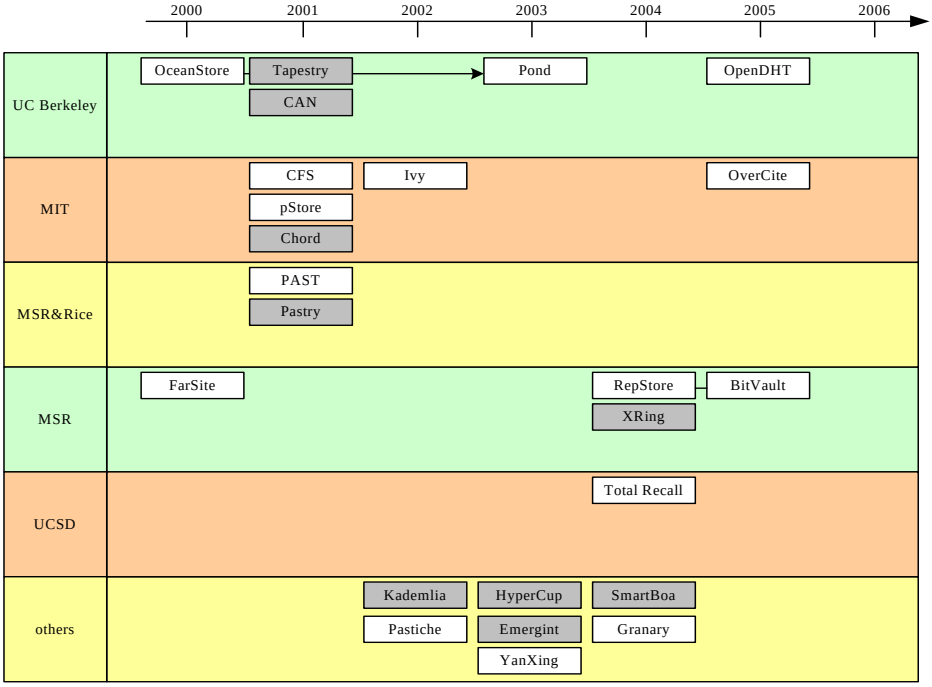


Fig. 6 History of P2P storage systems

图 6 P2P 存储系统发展历史

P2P 存储系统研究是伴随着 P2P 结构化覆盖网络产生的，它是结构化覆盖网络最典型的一种应用之一。图 6 给出了各具有代表性的研究机构提出的知名系统的时间，图中白色框是存储系统，灰色框是路由算法。从图中能清楚的看到，最初的存储系统多是伴随着相应的路由算法同时产生的，例如 Berkeley 的 OceanStore 和其相应的 Tapestry 路由算法[3]。

著名的研究机构包括 Berkeley、MIT、MSR 和 UCSD 等，主要的系统包括 OceanStore，CFS[25]，OverCite[26]，PAST[27]，FarSite[28]，BitVault[29]和 Total Recall[16]等。这些研究机构的相关研究及系统细节将在后面详细介绍。

从图 6 中可以发现，P2P 存储系统的设计集中于 2001 年和 2004 年两个时期，这是由该领域研究方法的发展历史所导致的。前面我们提到，P2P 存储系统研究是一项复杂的系统研究，任何设计都应体现在一定的系统目标下，以系统为背景。因此，研究的最初阶段是从 2001 年左右的系统设计开始的。虽然期间有不少优秀的系统设计出现，但 P2P 存储研究远未成熟，一直没有出现可实用的存储系统，自然也就没有任何实际的系统数据来支持系统中各种方法的量化研究。然而这段时间内，P2P 文件共享和 P2P 即时通讯等其它 P2P 系统迅速流行起来。这时，不少研究者通过探测这类系统来了解用户对 P2P 系统的使用行为规律，这为存储系统中各种方法的量化研究提供了有力的实际数据支持。随后，在这些测量的结果基础上，一些分析研究开始比较各种系统的策略和方案，并给出了分析上的指导方案。这些分析上的结果使研究者对 P2P 存储系统有了新的认识，从而又推动了 2004 年左右的新一轮系统设计。图 7 给出了这个系统研究中的各个子模块，以及其之间的依赖关系。

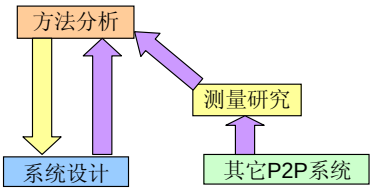


Fig. 7 Dependency of research fields

图 7 P2P 存储系统各方面研究的依赖关系

3 知名 P2P 存储系统介绍

3.1 MIT的CFS（Cooperative File System）

CFS[25]是一个只读的文件系统，它提供文件系统的语义，只允许信息发布者更新内容，不支持同步更新的语义。系统主要分为三个层次：CFS 使用 MIT 提出的 Chord[2]做底层覆盖网络；DHash 负责块级别的数据存取，并维护数据冗余；客户端的文件系统层（FS）负责提供文件系统接口与数据块之间的转换。

CFS 客户端将整个文件系统按小数据块的方式组织起来，如图 8a 所示，目录信息存成一个块，数据文件被分为多个小块存储，并用 inode 块来记录数据块之间的关系。每个小块以都其块数据内容的哈希值为 ID。CFS 的文件系统客户端负责将数据块最终转换为上层应用需要的只读文件系统接口。通过抽象出独立的文件系统层，CFS 使得下层 DHash 可以较为自由处理语义无关的数据块。同时，将文件分成大量的数据块也使得 CFS 可以方便的做负载均衡和处理热点文件的问题。

DHash 负责每个小块在系统中的存储。DHash 首先将一个数据块做 k 个完全副本，然后用 DHT 直接分发的方式将 k 个副本顺序的放置在数据块 ID 对应的负责节点及后继节点上，如图 8b 所示。当 Chord 网络检测到有一个副本丢失时，数据块的主节点负责立即再修复出一个副本。

Chord 是一种环形拓扑的覆盖网络，它负责维护 CFS 系统中节点的连通性。通过让每个节点认识大约 $O(\log N)$个其它节点（其中 N 是网络中的总节点数），Chord 中任意两个节点可以通过约 $O(\log N)$跳互相认识。

对比前面提出的抽象存储系统，CFS 底层采用 Chord 算法；利用副本方式冗余；直接 DHT 方式分发冗余数据；通过底层的覆盖网络来检测邻居节点的失效；采用积极修复的策略。CFS 认为存储空间不是稀缺资源，因此不采用纠删码技术，但忽略了纠删码也是节约系统维护带宽的重要手段；直接的 DHT 分发方式由于缺乏灵活性，在其它系统中并没有被广泛使用；系统的积极修复策略不利于系统有效的处理节点的暂时失效。

CFS 通过 IP 来认证节点，以防止恶意节点的攻击。节点的 ID 以其 IP 的哈希值为基础，节点要加入网络时，其邻居节点可以向其号称的 IP 地址发请求以确认其身份，这使得节点很难伪造身份。进而，CFS 实现了基于 IP 的分布式存储配额管理机制。随着 P2P 网络测量研究的发展，我们发现共享系统中节点的 IP 别名现象非常严重[30]，这种环境中的一个 CFS 节点将有多个网络 ID，为配额等管理带来麻烦。

系统使用信息发布者的公钥来认证信息发布者对其根目录的更新权限，以保证数据完整性。数据块的 ID 使用数据块内容的哈希做 ID，也保证了数据块级别的完整性。另外，CFS 系统支持数据块级别的缓存，而数据块的更新必然导致新块的 ID 的变化，这样被缓存的陈旧副本也就自然失效。当不同节点的空间差异悬殊时，可以通过在空间大的机器上运行多个 CFS 节点来均衡系统负载。CFS 系统不存在删除操作，其中每个数据都有有效期，用户为了保持自己数据的持久存储，需要不断的刷新自己数据的有效期。

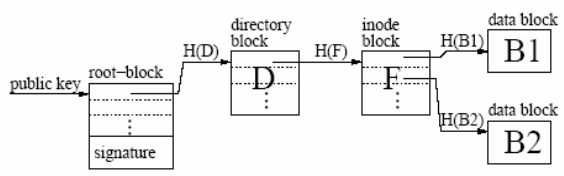


Fig. 8a CFS file system structure

图 8a CFS 文件系统结构

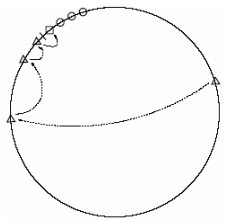


Fig. 8b CFS replication

图 8b CFS 的副本

3.2 Berkeley的OceanStore及其原型Pond

OceanStore[13, 14]系统构建在较为稳定的由服务商提供的节点集合上，节点间通过协约保证互相提供连续的服务；系统假设每个节点都可能不可信，但其系统整体上又是可信的；系统中的数据是不断演化的，因此系统能够自调整；系统中的数据是可以共享和全局可访问的，系统既保证数据私密性又保证其完整性；系统提供一定的数据一致性保证。

OceanStore 底层采用 Berkeley 提出的 Tapestry[3]路由算法，其通过对 ID 的后缀路由提供 O(log*N*)的消息路由效率。冗余策略上，OceanStore 一方面使用纠删码存储归档的数据以减小空间和带宽消耗，另一方面使用完整副本来提高数据访问的效率。OceanStore 把冗余的数据碎片存放在网络中无错误相关性的节点集合（即节点不会出现相关错误，例如区域停电造成的同时错误）上，然后将碎片的位置信息保存在文件 ID 对应的根节点处，如图 9 所示，当有客户请求数据时，先根据数据的 ID 联系到数据的根节点，进而获取每个数据碎片。数据的根节点通过混合的心跳机制检测每个碎片所在节点的状态。当出现节点错误时，数据的多个根节点将联合决定何时及由谁来修复丢失的冗余数据，OceanStore 采取延迟修复的策略。

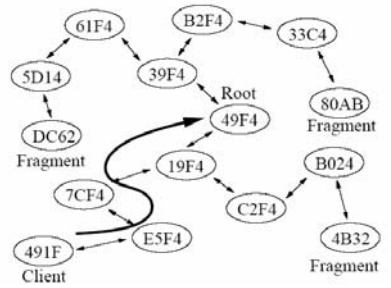


Fig. 9 Architecture of OceanStore

图 9 OceanStore 系统结构

Pond[15]是 OceanStore 的一个原型系统，被用来评估 OceanStore 可能的性能问题。系统使用双层的结构来组织节点，稳定节点负责组织，不稳定节点负责贡献存储资源。系统采用 Java 语言实现，主要利用 Berkeley 开发的 SEDA 事件驱动模型开发，系统部署在 PlanetLab 上。模拟发现，其性能比 NFS 系统高 7.3 倍。同时也发现，其性能的一些限制来源于纠删码的计算上。

OceanStore 还提供可基于事务的更新，以及容拜占庭错误的协议。系统通过对数据的多版本管理来简化一致性的设计。多版本使得每个版本中的数据块都是只读的，数据更新则创建新的数据块，这样避免了块级别的一致性问题，而将一致性转移到版本的维护上。为了抵抗拜占庭错误，数据的主副本分布在一个节点集合上，称为数据的内环（inner ring）。版本的一致性由内环节点负责维护，内环节点负责序列化一切更新操作。内环使用拜占庭容错协议，对抗系统中的拜占庭错误。内环更新后，向所有缓存的副本（缓存使用 LRU 策略）和归档数据（使用纠删码存储）发出消息通知。在一致性语义上，Pond 提供一套操作语义，以及一套一致性

谓词描述客户需要的一致性。

数据安全方面，除了容忍拜占庭错误，OceanStore 还采用了端到端的加密算法保证数据私密性，采用数据块互相自校验的方法来保证数据的完整性。

OceanStore 设计提供的目标语义非常强大，包括事务的更新操作等，这使得系统的设计和实现都过于复杂。另外，它对于系统中的错误模型假设也过强，认为每个节点都可能出现任意错误，导致设计中使用复杂的拜占庭协议。由于系统主要是部署在稳定节点集合上，节点的暂时性错误不是非常严重的问题，故该研究对于数据可用性的讨论不多，只有对数据可靠性的讨论。

3.3 微软研究院的BitVault

BitVault[29]是一个面向较少更新的参考数据（reference data）的存储系统，它是由较为稳定的机房内部存储节点构成的。系统采用 P2P 方式组织机房中节点，其目标包括：用廉价服务器群构建的超可靠和可用的服务；降低系统总的拥有开销；系统设计足够简单。

BitVault 底层使用微软亚洲研究院提出的 XRing[31]覆盖网络，此覆盖网络可以根据目标环境来自适应的调节自己的路由表大小。在非常动态的环境下，其路由表的大小保证 O(log*N*)的路由效率；在动态性小的环境中，XRing 不断扩充节点路由表，最终达到 O(1)的路由效率。

为了提供高性能的数据访问并保持简单的设计，BitVault 系统采用了完全副本的方式做数据冗余。

BitVault 提出了一种限制型分发的数据分发机制[32]。虽然机房系统中不会出现频繁的节点加入/退出造成直接 DHT 分发浪费大量带宽，但 BitVault 的研究者认为顺序的放置一个数据的多个副本将严重影响数据修复的速度，进而降低数据的可靠性；另一方面，完全无限制的随机分发也会造成系统可靠性的降低。关于此分发方法，我们将在下一节详细讨论。BitVault 的系统结构如图 10 所示，每个系统中的节点都分别具有索引和数据两个功能区域。每个数据对象的多个副本按一定规则存储在一个节点集合的数据区中，它们共享同一个数据 ID；在数据对象 ID 对应的根节点的索引区会建立软状态（即不永久存储）指针指向所有副本位置。

BitVault 采用基于广播的错误检测机制。由于 BitVault 面向机房系统，节点失效的概率非常低，因此频繁心跳的错误检测将浪费大量的带宽。BitVault 首先利用 XRing 的软状态路由表（soft-state routing table, SSRT）在节点间建立一个广播树结构，当有节点出现失效后，其邻居节点可以通过 XRing 协议发现这个错误，然后利用广播的方式通知所有节点。例如，*A* 节点的错误消息被广播，某数据对象的索引节点发现有一个数据副本在 *A* 节点上，也即现在丢失了一个副本，此索引节点就可以开始修复数据。

BitVault 在一定条件下积极的修复任何错误带来的数据丢失，并具有最后副本可修复（last copy recall）的性质。BitVault 不区分暂时错误与永久错误，对任意错误都积极修复；但当副本数超过系统设置的上限时，系统就不再修复。数据对象的索引信息是通过冗余副本主动发送自己的位置信息建立起来的软状态。当索引节点出现错误后，拥有数据副本的节点将得到这个广播的错误消息，并在新的数据根节点处建立数据对象索引。这样，只要有最后一个副本，就可以建立起索引信息，索引节点也就可以修复出足够的副本。系统保证了最后副本可修复的性质。

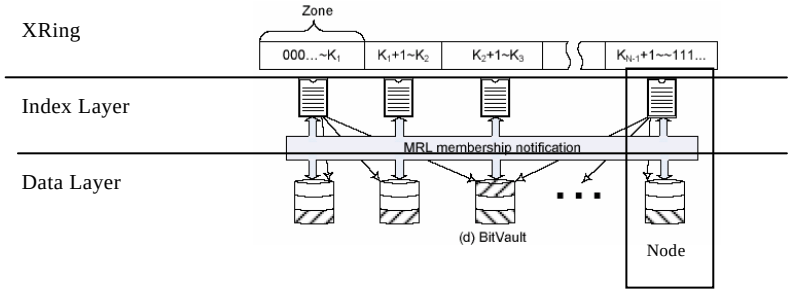


Fig.10 Architecture of BitVault

图 10 BitVault 系统结构

BitVault 的设计非常简洁，这一方面因为其使用软状态方式建立索引，使得系统不用持久存储索引和维护其一致性；另一方面是因为 BitVault 仅是一个数据对象存储池，它仅提供单个对象层面的存取操作，没有文件系统语义接口。在 BitVault 上开发的应用程序需要自己独立管理数据对象之间的关系，[29]中举例说明可以用如 SQL Server 的数据库系统来做应用的元数据管理。

3.4 UCSD的Total Recall

对等存储系统 Total Recall[16]的设计目标是自动配置系统所需要的各种参数，包括冗余方法、冗余度和修复时间等，以避免烦琐且困难的人工设置。

Total Recall 系统底层采用 Chord 路由算法。在冗余方法上，Total Recall 针对不同类型数据分别使用副本或纠删码方式或混合的方式。例如对一个不断追加的日志文件来说，可以对文件的前段采用纠删码，而后端采用副本冗余方法。冗余的数据被随机分发至网络中多个存储节点上，每个数据在其数据 ID 对应的主节点(Master node)处建立元数据文件并做多个副本，如图 11 所示。主节点负责维护数据多个副本的位置信息、检测存储节点的状态以及修复丢失的冗余数据。Total Recall 中的主节点负责周期性的探测数据节点，以检测它们的活动状态。当出现节点不在线时，Total Recall 或积极或懒惰的修复丢失的冗余数据，文章中系统对小文件采用积极修复法，对大文件采用懒惰修复法。

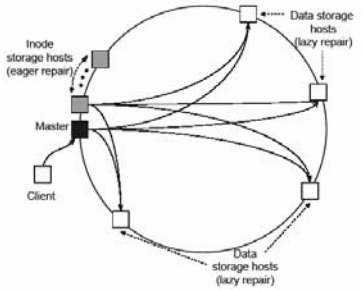


Fig.11 Architecture of Total Recall

图 11 Total Recall 系统结构

为了达到参数的自动配置，并保证数据的短期可用性和长期可用性（即可靠性），Total Recall 提出了保证可用性的三个重要技术：第一，可用性的预测，也即通过长期的观察系统中节点的可用性，预测节点短期在线概率和节点长期离开概率；第二，冗余管理，也即根据存储节点的动态性和目标的可用性要求，自动选择冗余方案和需要的冗余度；第三，数据修复，也即根据对节点长期离开率的预测，系统应能不断修复丢失的冗余数据。

在短期不考虑节点永久离开系统的情况下，Total Recall 首次给出了不同冗余方法为达到目标数据可用性所需要的冗余度计算公式。假设系统中的节点可用性为 μ，目标的数据可用性为 A，则副本方式所需要的冗余度 K_R 为，

$$k_R = \frac{\log(1-A)}{\log(1-\mu)} \tag{1}$$

若使用具有最大距离分割性质的纠删码方式，且纠删码需要 m 个碎片恢复数据，则需要的冗余度为，

$$k_E = \left(\frac{\sigma_E \sqrt{\frac{\mu(1-\mu)}{m}} + \sqrt{\frac{\sigma_E^2 \mu(1-\mu)}{m} + 4\mu}}{2\mu} \right)^2 \tag{2}$$

其中 σ_E 为对应数据目标可用性的正态分布的标准差。

通过模拟发现，Total Recall 系统可以获得的数据可用性远远高于设计的目标可用性，并发现使用纠删码能够减小修复带宽等结论。通过与一个能够准确预测节点永久离开时间的理想系统比较，发现系统在修复带宽消耗上面还有不小的优化空间。

Total Recall 虽然设计了主动探测的方法来检测每个数据节点的活动情况，但是这种设计却没有很好的扩展性，当更多的数据随机分发后，一个主节点可能需要一个周期内探测成千上万的存储节点，这将消耗极大的带宽。Total Recall 利用节点的平均在线率来评估所需要的冗余度，然而使用一组节点的平均值代入上述公式是否会造成结果很大的偏差，文章并没有给出证明。文章提出，如果进行懒惰修复，就需要有比维持短期可用性更多的冗余数据，而触发修复的条件是数据冗余度低于维持目标可用性的冗余度。很明显，这种修复的方法并没有让本用来屏蔽暂时错误的冗余数据发挥作用，这也就直接造成系统获得的可用性远高于需要的可用性，造成了系统资源的浪费。

3.5 清华大学的Granary

Granary[33]系统的设计目标是能够自适应的支持高动态系统和稳定系统，并提供面向对象的存储。

Granary 采用清华提出的 Tourist[34]路由算法，这种算法可以根据网络的动态情况自适应的改变自己的路由表大小，使得网络在稳定环境中获得 O(1)的路由效率，而在动态环境中至少保证 O(log*N*)的路由效率。Granary 采用完全副本的数据冗余方式。冗余后的数据被 Granary 均匀的分发到网络中的节点上，以防止相关性错误，冗余数据所在节点的 IP 列表以及数据对象的属性被作为元数据以 DHT 的方式存储在 P2P 网络中。Granary 利用 PeerWindow 算法向目录节点广播节点加入和离开的事件。系统为用户存放 2 倍于用户指定的数据副本数，当系统中副本数低于用户指定的副本数时，Granary 才触发修复操作。

利用路由算法的自适应性，Granary 可以自适应不同动态性的环境。目前，该项目组正利用 Granary 向互联网用户提供一个免费的存储服务。

3.6 北京大学的UPStore

UPStore[24]是一套具有开放框架的存储平台，它的设计目标是通过修改算法适应多种动态性环境。它的开放性表现为：系统为上层应用提供一系列的存储接口；系统由多个互相独立的模块组成，模块之间松耦合，每个模块允许用第三方的算法扩展或替换；最终系统要部署到互联网的众多节点上，并以独立服务的形式供其它应用使用，免去应用层部署服务的麻烦。由于策略模块可替换，UPStore 系统可以容易的利用不同算法组合适应不同的动态性环境。

UPStore 底层采用该研究小组完成的一个 Kademlia[6]网络实现 Lunar[35]。Lunar 不仅是一个覆盖网络实现，它同时是一个 P2P 开发的支持软件包，为跨平台和网络编程提供良好支持。Kademlia 覆盖网络协议是目前互联网上用户最多的一种 P2P 路由协议，其性能和容错性已经被[36]等网络应用所实际验证。UPStore 同时提供副本和纠删码两种冗余策略，其扩展的纠删码设计（SEC）[37]在冗余的同时还能提高系统中用户数据的私密性。UPStore 将网络中的节点分成多个簇（cluster），每个节点可以同时属于多个簇。在 UPStore 目前的封闭式系统实现中，冗余的数据被随机的分配到一个簇中的不同节点上。每个冗余副本（碎片）向簇的主节点（master）报告自己存储的数据，簇的主节点负责建立数据 ID 到其多个冗余副本（碎片）位置的软状态映射。簇中的数据节点周期性的通过心跳向簇的主节点报告自己的活动状态。当出现节点错误时，簇的主节点会根据错误是永久错误的概率来判断是否开始修复过程。

UPStore 系统通过分簇的机制使得心跳消息不会随网络规模增大而不可接受，解决了 Total Recall 系统完全随机分发带来的实际问题。UPStore 的懒惰修复依赖于对节点永久退出的判断准确性，而这种判别本身还是一个难题。

3.7 存储系统小结

我们总结上述典型的 P2P 存储系统的持久存储相关技术于表 1。从表中不难发现，各个系统所设计的目标工作环境是不同的。根据不同的工作环境，它们所采取的技术组合也不尽相同。其中，不同系统对副本和

纠删码的冗余方式各有偏好；分发方式上，逐渐抛弃了直接 DHT 分发的方法；错误检测根据不同的目标环境，采用定期心跳（探测）或失效事件广播法；修复方式上，系统设计逐渐趋向于用更多的副本来支持延迟修复。

知名的对等存储系统还包括 OverCite[26]，pStore[38]，Pastiche[39]和 Farsite[28]等，本文不能逐一详细介绍。另外，近期出现的 GFS[40]等文件系统设计也对 P2P 存储技术研究有重要参考价值。

Table 1 Comparison of P2P storage systems

表 1 P2P 存储系统对比

	适应环境	冗余方式	分发方式	错误检测	修复方式
CFS	稳定	副本	直接 DHT	邻居节点探测	积极
OceanStore	稳定（广域）	副本、纠删码	间接	心跳	懒惰
BitVault	稳定（机房）	副本	限制性间接	事件广播	积极
Total Recall	稳定、高动态	副本、纠删码	间接	主管主动探测	积极、懒惰
Granary	稳定、高动态	副本	间接	事件广播	—
UPStore	稳定、高动态	副本、纠删码	间接	心跳或其它	积极、懒惰

4 面向数据持久性的相关技术研究

前面介绍了面向数据持久存储的基本技术，以及每类中的不同技术方案。但在这些技术中哪种技术和哪些技术组合是最优的，以及在什么情况下是最优的等问题还需要量化研究来回答。尽管广域网大规模的 P2P 存储系统还没有成功部署的案例，但其它类型的 P2P 应用已有不少获得了成功，如即时通讯、文件共享和流媒体等。目前已有不少研究通过分析现有 P2P 应用系统的负载和动态性等特征，量化的研究 P2P 存储系统中各种技术的特点。本节将介绍一些现阶段的研究成果。

4.1 系统和数据持久性的指标定义

在进行量化研究之前，我们需要对系统和数据的持久性给出明确的指标和相应的定义。持久性主要有两个指标，面向长期持久性的系统可靠性和面向访问当时的数据可用性。

系统可靠性定义为，系统在时间 t 没有永久丢失数据的概率。系统可靠性主要受到系统中存储节点的寿命影响，而节点暂时离开造成的数据暂时不可访问则不影响系统的可靠性。传统的系统可靠性的概率分布一般被认为是指数分布，因此也经常用平均失效时间（Mean Time To Failure, MTTF）来代替可靠性的概率函数表示以衡量系统可靠性。

数据可用性定义为，数据在时间 t 可以被访问到的概率。由定义可以发现，节点暂时离开可能造成数据暂时不可被访问，会降低数据的可用性。因此，数据的可用性主要被节点的在线率（用在线时间和总观察时间的百分比表示）和节点每次会话时间长度所影响。

相比较两个概念，可靠性更加关注数据是否能长期存在于系统中，而可用性则更加关注任意时刻数据是否可以被使用。可以看出，当系统可靠时，数据可能不可用。在传统系统中，由于暂时错误比较少，因此主要衡量系统可靠性；在 P2P 系统中，暂时的节点退出比较频繁，因此数据的可用性也就成为非常重要的衡量指标。

4.2 数据冗余方案研究

4.2.1 前沿研究成果介绍和讨论

2002 年，OceanStore 项目的研究者 Weatherspoon 等人[41]就用量化的方法分析了纠删码和副本方式冗余对系统可用性的影响。此分析工作以 OceanStore 的目标运行环境为参考，考虑较为稳定的部署环境，主要考察不同冗余方式对系统可靠性的影响，采用随机过程的方法进行建模和分析。作者假设系统每隔一定时间间隔做一次全面的数据修复，系统中每个存储节点都独立且等概率的失效。在以上假设下，作者通过对比发现：

在分布式系统中使用纠删码冗余，可以在与副本冗余得到相同可靠性的条件下极大的节约系统中的存储空间和维护带宽；反之，若使用相同的存储空间和维护带宽，纠删码方式能够极大的提高系统的可靠性。因此说，纠删码有利于提高系统的可靠性。

Total Recall 的作者 Bhagwan 等人[42]在 2002 年针对系统中数据的可用性，也比较了副本和纠删码两种冗余方式。作者认为，在 P2P 系统中以整个文件的形式复制和处理大文件是耗时和麻烦的，因此要考虑将文件切分为小块。通过实验，作者对比了整个文件副本、对文件碎片做副本和纠删码三种冗余方法对数据可用性的影响。实验结果显示，纠删码在提高数据可用性方面有明显优势。

虽然前面的研究已经给出纠删码在可靠性和可用性方面的优势，2004 年香港学者 Lin 等人[43]还是重新考察了纠删码对 P2P 存储系统可用性的影响。此研究中，作者仍然使用被广泛接受的数据可用性计算公式，

$$A(m)=\sum_{i=m}^{Sm}\binom{Sm}{i}\mu^{i}(1-\mu)^{Sb-i} \tag{3}$$

其中 m 为恢复数据所需要的最少碎片数，S 为数据冗余的倍数，μ 为机器的平均可用性。不同的是，作者使用了纯分析的方法分析了可用性公式的渐进行为；而前面的研究多是代入具体环境的动态性参数，比较典型动态性下的性能。在此研究中，作者更关注高动态性的系统中纠删码方案下的数据可用性，其主要结论包括：

- ✧ 当节点可用性高时，无限多的碎片数将提高数据可用性；当节点可用性低时，碎片数多将使得数据可用性降低。因此，具有最低碎片数的‘全文件副本’在高动态环境下更有优势
- ✧ 当系统高动态时，通过增大冗余度，会使得纠删码方式显得有优势。
- ✧ 通过渐进分析，作者发现$S\times\mu>1$时纠删码方式具有优势；否则，应该使用全副本方式

Lin 等人最后提出，虽然碎片数无限大的时候能够使得数据可用性最高，但实际情况的种种约束将使得碎片数不能太大；作者还强调，当对系统中节点可用性测量不准确的时候，最好还是用副本的方式，以免纠删码方式降低数据的可用性。

2005 年，MIT 的学者 Rodrigues 等人[44]从系统可行性的角度分析了两种冗余方式对可靠性和可用性的影响，指出了纠删码方式优势的局限性。文章利用公式(1)和(2)，由数据的目标可用性反算出副本方式和纠删码方式各自需要的冗余度。作者建立了系统在稳态情况下维护带宽的模型，这个系统模型简单且有效的刻画了 P2P 存储系统的数据维护过程。根据模型，作者推导出维护系统中数据的冗余度所需要的带宽资源。作者使用 Overnet、Farsite[28]和 PlanetLab 三个系统的动态性测量结果，分析它们所需要的维护带宽，结论包括：

- ✧ 在 PlanetLab 这种稳定节点组成的系统中，副本方式已经能够在低的维护带宽情况下提供很好的可靠性和可用性
- ✧ 在 Farsite 这种中等动态的系统中，纠删码方式比副本方式并没有很明显的优势，但却会带来设计和实现上的复杂度，因此没有必要使用纠删码
- ✧ 纠删码能为 Overnet 这种高动态的系统节约存储空间和维护带宽，但是其维护带宽仍然是每个节点不可接受的，因此系统本身就不可行。

4.2.2 前沿研究成果总结和讨论

从分析方法上看，上述研究主要使用随机过程和组合概率两种分析法。这两种方法被分别用于不同的分析目标，其中随机过程法一般用于系统可靠性分析，而组合概率法一般用于数据可用性分析。

从分析结论上看，上述研究对于纠删码优劣的看法众说纷纭，甚至有些相互冲突。我们客观的总结上述研究成果，发现纠删码相比副本方式有如下特点：

- ✧ 在低动态环境中，能够提高系统可靠性和数据可用性
- ✧ 在低动态环境中，如果碎片数不大，则纠删码方式效果不明显，可以直接用副本方式
- ✧ 在高动态环境中，当冗余度不足时，可能降低数据可用性，而且碎片数越大可用性越低
- ✧ 在高动态环境中，当冗余度充足时，能够提高数据可用性，而且碎片数越大可用性越高
- ✧ 在高动态环境中，虽然纠删码方式可以节约维护带宽，但其需要的带宽仍不可接受，造成系统不

可实现

明显，纠删码的适应环境和参数问题目前还没有一致的结论，还有不少问题值得继续研究。我们提出如下的问题：

- ✧ 在纠删码方式下，要修复一个丢失的冗余碎片就需要有一个完整副本。因此，如果恢复数据需要 m 个碎片，则修复一个碎片需要读取 m 倍大小的数据，浪费系统带宽。那么我们是否可以用两层的冗余方法解决这个问题，也即先做纠删码冗余，然后对每个碎片做副本冗余。这样丢失一个碎片副本就只需要从另一个碎片副本处读取等大小数据修复。
- ✧ 目前对数据可用性的分析，研究者多采用包括公式(1)(2)(3)在内的组合概率法，然而这种方法得到的是稳态下的数据可用性[45]，不适合高动态性系统下的可用性分析。事实上，利用更精细的随机过程法分析数据可用性，我们发现纠删码有更明显的优势[45]。

4.3 数据分发研究

4.3.1 前沿研究成果介绍和讨论

前文介绍了直接 DHT 分发已经很少被使用，而基于目录的间接分发被广泛使用。间接分发方案使得可以更加灵活的选择存储节点集合，这样我们就可以利用一定的策略优化这个节点集合以得到更为持久的数据存储。本小节介绍一些前沿的分发方法研究。

Douceur 等人[46]在 Farsite 的研究中提出利用爬山算法来优化存储用的节点集合，以解决在节点可用性不同的情况下，系统中数据可用性的最优化问题。爬山算法分两个部分：首先，系统将要存储的数据制作 3 个副本，并将副本随机的存放在网络的节点上；系统定期观察每个数据的可用性状况，然后交换其中两个数据的两个副本位置，以使得系统的可用性增益最高。研究比较如下三种置换的策略，

- ✧ 遍历任意两两数据的组合，寻找最优的副本调换
- ✧ 先选可用性最低的一个数据，并遍历其它所有数据，寻找最优副本调换
- ✧ 先选可用性最低的一个数据，并与可用性最高的一个数据的副本进行调换

通过理论分析及实验的方法，作者发现“最小最大法”不如“最小和任意法”，更不如“任意和任意交换法”。

Ramanathan[47]提出系统应该为经常访问的数据提供高可用性，给不经常访问的数据提供可接受的可用性。因此，其系统总的可用性定义为，系统中每个数据的可用性与其访问频率乘积的总和。在这样的可用性定义下，作者希望找到一个启发式算法，能够不断增加系统总的可用性值。系统模型中假设节点的可用性不对等。作者首先假设 P2P 网络是分层的，下层是总的 P2P 网络，上层是根据兴趣形成的小覆盖网络，而在小覆盖网络中可以有全局信息。这样启发式算法就相对容易，即当在小的覆盖网络中缺失文件 A 时，小网络的决策者就找到所有比 A 重要性低的数据 B_i，并计算能够让总可用性增益最大的替换数据 B，然后将 B 替换为 A。通过实验模拟的方式，发现这种启发式算法相对于纯随机算法，确实能够让系统总可用性达到较好水平。

Schwarz 等人[48]也希望用一个‘基于历史的爬山算法’逐步优化副本放置的位置，最终让放置达到最优。作者首先提出节点应组成存储组（Collection），这样可以只让一个节点属于为数不多的组，从而减少大家互相检测错误的代价。而后，一个组内的数据采用纠删码方式进行冗余，碎片的分发使用 RUSH 分配算法。其优化的算法为，当某个数据在线的碎片数低于阈值的时候，就寻找组内可用性最高的节点，然后将新的冗余数据碎片修复到这个节点上。算法本质上是一种很自然的利用高在线节点的思想，故真实系统中难以实用。

2005 年，微软亚洲研究院的 Lian 等人[32]以 BitVault 系统为背景，分析了局域网内 P2P 存储结构中的分发问题。文章首先提出，副本存放主要有两种形式，顺序分发（也即 DHT 直接分发）和随机分发。这两种方式各有优劣，其中顺序的方式管理起来简单，而且如果同时出现大规模的节点失效，这种方式受到的影响可能较小；其缺点是出现节点错误后，修复节点过于集中，形成修复带宽瓶颈，影响修复速度，进而降低系统可靠性，如图 12 所示，当节点 3 失效时修复集中于其附近的几个节点。随机存储的方式更加灵活，当出现错误时，数据的修复也可以并发进行，加快修复速度；其缺点是，并发的多节点错误可能导致的数据丢失的

概率比顺序分发要大，具体讨论详见[32]。作者认为快速并行修复和避免大规模同时错误本质上是矛盾的。

文章以系统的可靠性，即 MTTDL（Mean Time To Data Loss）为量度，建立了一个在系统集合带宽（root switch）约束下的分析模型。利用该模型，作者分析了两种分发方法对系统可靠性的影响，主要结论包括：直接 DHT 分发，不能快速修复，不利于可靠性；完全随机分发，则任意 k 个节点的同时错误，系统丢失数据的概率增大。因此，分发需要一定随机，但也不能完全随机。作者提出一种叫做“限制型随机分发”的方法，也即随机的节点组合总数控制在一定程度。作者提出用条带（stripe）的方法来进行随机分发，也即多个小文件被聚集在一个条带中，以减小随机组合数。文章进而分析给出了条带法的条带容量最优值。

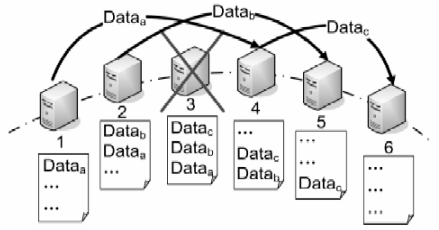


Fig.12 Data repair in sequential placement

图 12 顺序存储带来修复带宽的瓶颈问题

分析中，作者建立的分析模型是以系统中节点作为对象建立的马尔科夫模型，不同于其它以文件副本为研究对象的马尔科夫模型，它也使得分析可以建立可靠性和系统集合带宽的关系。但是此分析过于局限于机房存储系统，而且即便是机房存储系统，也可以是多级交换，因此单交换机的集合带宽约束模型过于强大。另外，此研究的方法只是让传统意义上的 MTTDL 达到了最优，但这种定义下的最优不一定适合所有应用场景，因此该结论仍值得进一步讨论。

4.3.2 前沿研究成果总结和讨论

间接的数据分发方法已经被广泛采用，它既能支持数据并发的快速修复，也能支持更加灵活的节点选择算法。在此基础上，有不少研究讨论节点集合的选择方法。然而对这个问题的研究目前仍然没有突破性进展，主要原因包括：

- ✧ 节点的在线规律还没有被很清楚的认识。这就使得根据规律寻找节点集合非常困难。因此，不少研究仅随机分发，然后利用节点在线概率的平均值计算数据可用性
- ✧ 可靠性和可用性的计算过于复杂。即便获得了节点的在线规律，如果不使用在线概率均值或假设节点寿命独立同分布，可靠性和可用性也是难以计算的。因此，不少算法使用启发式的爬山算法
- ✧ 爬山算法虽然承认节点间在线概率等方面的差异，更加接近实际情况，但它们一般依赖于对系统全局节点可用性的知识，而且它们也趋向于耗尽高在线概率节点的资源，故不实用

4.4 错误检测研究

在 DHT 直接分发方式下，错误检测一般通过 P2P 覆盖网络层对邻近节点的失效探测来实现，例如 CFS 系统。在间接分发方式下，错误检测就相对复杂，因为存有冗余数据的节点与其目录信息所在节点往往不是 P2P 网络中的邻居。

定期心跳或探测法是比较直观的错误检测方案，被 Total Recall 和 OceanStore 等系统所广泛使用。然而，当一个数据节点上存储的数据属于很多目录节点时，数据节点就要每个周期发送大量的心跳数据包，导致系统性能极度下降，甚至导致系统不可运行。Wells[13]在 OceanStore 的研究中已经发现这个问题，并采用一种混合的心跳策略来解决。

微软的 BitVault 系统则采用错误事件广播的方式代替定期心跳检测错误。通过使用 P2P 覆盖网络的广播机制，BitVault 用 $O(N\log N)$个消息让所有节点获知一个节点错误的消息，其中 N 为网络中节点总数。但很明

显，这种方法并不适用于高动态性的网络环境，因为它会引发短期内巨大数量的广播消息。

4.5 冗余数据维护研究

4.5.1 前沿研究成果介绍及讨论

维护数据的冗余度主要是为了保持数据可用性和系统可靠性。一定量的冗余数据是在有暂时错误的情况下保持数据可用的途径，根据数据的目标可用性，系统可以评估所需要的冗余度。不幸的是，P2P 系统中节点还会永久退出系统，我们必须修复那些永久离开的节点上所存储的数据，以保证系统长期可靠性。另外，P2P 系统难以分辨节点的离开是暂时的还是永久的，这使得何时修复数据以及修复多少的冗余度更加难以判断。

Total Recall 系统[16]利用数据可用性计算公式自动配置可用性所需要的冗余度参数。同时，Total Recall 为了防止暂时性错误触发修复浪费系统带宽，提出要用多余副本来屏蔽临时性错误。Total Recall 最初并没有给出确定多余副本个数的方法，而其研究者 Tati 等人[49]于 2006 年给出了多余副本个数的评估方法。例如，为了保证数据的可用性水平需要 *R1* 倍的冗余，系统就会为数据做 *R1*+*R2* 倍的冗余，只有当系统中可用的数据冗余度比 *R1* 低的时候才开始修复数据。

Total Recall 用多余副本以进行懒惰修复确实可以节约维护带宽，但其认为可用的数据冗余度降至低于 *R1* 就开始修复，则违背了用 *R1* 的冗余度屏蔽暂时错误的初衷。前文我们已经介绍了，此种方法使得系统实际获得的可用性远高于目标可用性，也即浪费了系统的存储和带宽资源。若确实要用 *R1* 的冗余度屏蔽暂时错误，就要仅当仍存在系统中的冗余度低于 *R1* 时才开始修复，而不是当时可用的冗余度低于 *R1* 时就修复。当然，这里的难题仍然是如何判断一个节点是永久离开，还是暂时离开。

OceanStore 项目的研究者 Weatherspoon 等人[50]在 2005 年对懒惰修复方法做了分析。该文章试图为广域网存储系统的各种参数的选择制定一套选择的方法框架，框架包括数据冗余方法、节点选择策略和错误检测及修复三个主要方面。其分析的目标环境是 PlanetLab 这种稳定节点集合，节点可用性水平基本相似，且节点间有一定的错误相关性。该研究认为使用更多冗余进行懒惰修复，采用间接的有选择的分发策略，以及使用纠删码都能较大的减少带宽消耗。文章还分析了节点相关错误对持久性的影响，并发现用黑名单来屏蔽可用性极低的节点能够持久性。通过分析，作者给出了系统所需冗余度和系统修复时机的评估方法。

该研究首先为整个系统建立了带宽消耗的模型，总带宽由写入操作、错误检测、永久错误和暂时错误几部分组成。其中写入带宽与数据的冗余度相关；永久错误的修复带宽是必不可少的；如果将超过阈值的离线行为都认为是永久错误，则必然有一部分暂时错误被‘误判’为永久错误，故也会消耗系统带宽。作者通过目标可用性计算公式，获得屏蔽暂时错误所需冗余度，然后根据永久错误的误判率和系统总带宽消耗公式推算出需要的多余副本数。

文章根据自己的框架，利用 PlanetLab 环境的测量数据，为 DHash、PAST、TotalRecall 和 Pond 几个系统重新计算了一下系统参数。作者发现，使用该框架计算的系统参数所消耗的带宽比原有设计都要小很多，其中 DHash 系统甚至可以节约一个数量级的带宽。

该研究将持久存储的各种相关技术统一到了总系统带宽消耗的评估中，其对带宽的评估相对其它研究更为全面。作者用时间阈值来区分永久错误和暂时错误，并指出误判率是多余副本存在的意义，从概念上明确了多余副本的意义。然而，此分析仍不甚完善，我们提出如下问题：

- ✧ 不同的时间阈值会造成错误检测器的不同漏判率，也即某些永久错误不能被识别，这将降低数据的可用性。这种漏判在高动态系统中将更为严重。但这个影响并没有体现在此分析中。
- ✧ 该研究仅以单一的时间阈值作为区分永久离开和暂时错误的方法，我们是否可以找到其它节点特征以便更为准确的区分这两种行为。更精确的区分，必能进一步减小系统资源消耗。

2006 年，UC Berkeley 的 Chun 和 Weatherspoon 同 MIT 及 Rice 大学的多个研究者又提出了一个广域网的复制算法，Carbonite[10]。文章同样以 PlanetLab 为典型应用场景，但却假设节点间没有错误相关性。Carbonite 算法以系统的可靠性为第一目标，而不注重数据的可用性指标。其主要观点包括：可靠性比可用性的维护代

价要低；维护可靠性的关键是产生新的冗余数据的速度要快于数据丢失的速度；由于系统总带宽的约束，并不是增大数据的冗余度就一定能够增加系统可靠性，但更多冗余度可以帮助避免并发错误；重用因为暂时错误而修复出来的副本，将可以有效节约系统资源消耗。

作者认为系统中的错误模型是均匀速率的离开和间歇性的并发离开两种行为的组合。通过马尔科夫模型，作者说明在带宽约束下能够维持的数据冗余度在一定系统参数下有上限。在 PlanetLab 的动态环境下，分析发现其能够维护的最高冗余度数是 6.85。在此基础上，作者讨论了如何选择冗余度。Carbonite 算法初始时，不建立多余的冗余数据。Carbonite 在系统检测到暂时错误时也创建多余副本，这些多余副本可以在以后用来屏蔽误判造成的修复。通过模拟发现，Carbonite 算法的长期运行效果接近理想效果。

不同于以往的预先创建多余副本方案，Carbonite 算法动态的创建多余副本。这使得 Carbonite 算法可以不必考虑检测器误判率的问题，也即遇到节点错误就启动修复。

Ramabhadran 等人[51]对 P2P 网络中有冗余的状态（state）的存活寿命（连续可访问的时间）进行了研究，得到了在带宽或存储约束的条件下最优的修复目标冗余度。文章假设节点的寿命和修复时间是指数分布，以状态为研究对象建立了马尔科夫分析模型，并给出了一种吸收态时间分布的解。从解的形式中可以看到，状态的寿命被副本数和修复强度两个因素所决定，并和修复强度多项式关系，和副本数指数关系。这样，在一定系统参数下，就可以评估状态的存活时间。作者进而分析在带宽约束下，状态能取得的最大寿命。系统的带宽也受到副本数和修复强度两个因素制约，因此我们需要在副本数和修复强度上折衷以取得最优的状态寿命。由于难于给出问题的解析解，作者通过对数值解的观察发现，副本数在取其能取得的最大值或最小值处时，状态的寿命能够达到最优值。最后，文章通过 PlanetLab 的动态性数据验证了节点在线时间的指数分布特性。

与[51]类似，2006 年我们也发表了以状态的连续可访问时间为量度的 P2P 系统数据持久性分析[45]。通过分析，我们发现在高动态性环境中如果做大量数据冗余，马尔科夫模型的吸收态时间分布将不再是指数分布，故不能用平均寿命时间这一指标衡量数据的持久性。因此，[51]的分析在高动态环境中可能有较大误差。另外，虽然[51]说明了 PlanetLab 的节点寿命是指数分布，但却没有证据表明开放式系统的节点寿命也是指数分布，这也阻碍了其结论在开放式系统中的应用。

同年，Sit 与 Ramabhadran 和 Weatherspoon 等人的另一个合作研究[52]提出，出现错误再触发修复可能导致系统的突发修复带宽影响其它的应用服务质量，而在没有错误时就预先修复可以解决这个问题。作者认为修复所占用的带宽应在一定预算之内，才不会让用户感到服务的突然降级。在无错误情况下可以一直后台修复数据，而有错误出现后就可以还在预算带宽下修复未完成的少部分修复工作。作者在 PlanetLab 动态性环境下的模拟表明：在消耗的总带宽可比的情况下，预先修复的方案可以达到与反应式修复一样的可靠性和可用性程度。作者另外指出，在预先修复的模式下，间接的随机分发方式并不能带来更多的好处。这是因为，随机分发的主要优势在于可以提高并发的修复速度，但在有修复带宽预算的情况下，顺序分发和预先修复则可以更好的解决问题。

Sit 等人的预先修复方案在稳定的环境可能是一种好的解决方案，因为这种方案并没有区别各个节点的可靠性，仅是随机的互相修复数据。如果节点的可靠性极度的不平衡，那么将稳定节点上的数据预先修复到不稳定节点上就非常浪费带宽。当然，在可靠性不平衡的系统中，我们可以监视和预测节点的可靠性，并倾向于修复可靠性低的节点上的数据，但这种预测本身又是相当困难的。

不同于上述以稳定环境为目标的研究，MIT 的 Blake 等人[23]在 2003 年给出了高动态环境中冗余数据维护的分析，并悲观的提出用类似 Gnutella 系统中的高动态节点组成的 P2P 网络存储服务是不经济的。作者利用数据可用性计算公式得到数据需要的冗余度，进而根据节点永久退出系统的频率推算出系统的数据维护带宽。根据此带宽评估模型和 Gnutella[18]网络的动态性测量结果，作者指出支持高动态性的节点、支持大规模的存储数据和提供高可用的数据服务三者是不能共存的，只能选择其二。作者举例说明，大约 33,000 个 Gnutella 节点所能提供的数据存储服务仅相当于 5 台专职 PC 服务器的服务水平。文章指出，一定的“准入控制”可能是解决桌面系统动态性的方法，也即只让可用性高的节点加入系统；但另一方面，准入控制又会挫

伤用户积极性，限制系统规模的迅速发展。

Blake 等人的研究用一个简单朴素的模型，有效的刻画了一个对等存储系统维护数据冗余度的带宽消耗，该模型也被后续的研究[44, 50]等所参考。作者在文中的讨论也道出了一个可用性和可靠性的折衷问题，也即节点的可靠性和可用性是同时被判断永久错误所用的时间阈值所影响的，它们是此消彼长的关系。节点可用性和可靠性的降低都会造成系统维护带宽的增大，因此我们需要精心选择判断永久错误所用的时间阈值，以让冗余数据的维护带宽达到最优值。

Blake 等人的研究虽然认为高动态环境不适合用来提供超大规模稳定存储服务，但其并不说明利用桌面机组成 P2P 存储系统是不可行的。这是因为 Gnutella 系统仅是一种共享型 P2P 桌面系统，它不能代表所有类型的桌面机 P2P 系统，例如 P2P 的 VoIP 系统 Skype 中节点的动态性就相对共享型系统低很多[53]。明显，桌面机组成的 P2P 系统中节点的动态性是和系统的应用类型相关的[54]。用户存储在存储系统中的数据是供其日后使用的，因此我们推测存储系统中用户的永久离开率会比共享系统低很多。另外，一项近期的测量工作[55]发现，桌面机 P2P 系统中新注册用户的永久离开率是非常高的，如果能够控制新注册的不稳定用户对系统的影响，桌面机 P2P 系统的动态性能够大幅降低。

由于认为广域桌面机组成的高动态 P2P 系统难以做大规模数据服务，MIT 的 Li 等人[11]在 2006 年考虑在有朋友关系的节点间建立 P2P 存储网络，即称 F2F（Friend-to-Friend）。在这样的系统中，由于各个节点所对应的用户在真实社会中有朋友关系，作者就假设节点之间是非常互相合作的，而且不用考虑节点永久退出的问题。因此，系统中只需要考虑用户误操作和磁盘错误造成的数据丢失，其中磁盘错误的频率使用 PlanetLab 上面收集的数据。通过计算，文章表明这样的组织方式大大降低了系统需要的存储空间和维护带宽。

4.5.2 前沿研究成果总结和讨论

对于一个 P2P 存储系统来说，需要维护多少的数据冗余、何时进行数据修复以及如何修复等问题，由于其相关联的系统问题多且复杂，故仍然是非常困难的研究课题。综合前面介绍的研究，不难发现目前在这个研究领域中，研究者提出的解决方案很多，例如用预先的多余副本避免修复[16, 49, 50]，或动态的增加多余副本[10]，或采用无错误时预先修复等方案[52]等。针对不同的解决方案，其分析模型又是多种多样，例如用组合概率法计算数据可用性[16, 23, 50]，或用随机过程法计算状态持久性[45, 51, 56]，或用类似半衰期的生命周期模型分析节点离开的影响[49]。

虽然在诸多的方案和分析方法中并没有统一的结论，但各研究的主要方法基本都是在带宽约束下，考察系统可以支持的修复冗余度和修复速度的最优值。多数研究中，最大的难题还在于难以判断节点的离线是暂时离开还是永久离开，这直接造成了系统不能精确的针对永久错误进行修复。因此，一个对系统节点离线行为的判别算法将可能成为今后研究的核心。

另外，我们看到目前多数系统的研究都是针对类似 PlanetLab 这种稳定系统的，而对于高动态系统只有少数对其可行性的讨论。这主要是因为目前还没有成功的高动态环境中的存储系统出现，而且在高动态环境中的其它类型 P2P 应用的测量结果也相对较少，这使得我们对高动态环境中节点行为的认识还很欠缺。因此，难以做系统的研究。

可以说，目前稳定节点环境中 P2P 存储系统的冗余数据长期维护问题正在进入被深入研究的阶段；而高动态环境的研究还在起步阶段。

5 总 结

P2P 网络应用近年来发展速度迅猛，这无疑显示了 P2P 模式的强大生命力。虽然 P2P 存储系统在数据持久存储方面仍有难题，但今后其必将成为一种重要的存储组织模式。本文较为全面的综述了近年来 P2P 存储系统及持久存储技术的研究成果，并对各种技术的适应环境及优劣做了深入的探讨和对比。我们看到稳定节点环境下的技术研究已经取得了一定的成果，极大的推动了 P2P 存储系统的发展。相比之下，高动态环境下的持久存储研究却仍处于起步阶段，其发展有赖于我们对高动态环境下用户行为的进一步认识。因此，我们需要尽快建造这样的系统以收集实际数据，或更大量的收集类似的其它 P2P 应用的动态性数据以支持此研究。

References:

1. Zhang, Z, S Lin, Q Lian, and C Jin, *RepStore: a self-managing and self-tuning storage backend with smart bricks.* In Proc. of International Conference on Autonomic Computing, 2004: p. 122-129.
2. Stoica, I, R Morris, D Karger, MF Kaashoek, and H Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications.* Proceedings of the 2001 SIGCOMM conference, 2001. 31(4): p. 149-160.
3. Zhao, BY, J Kubiatowicz, and AD Joseph, *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing.* Computer, 2001.
4. Ratnasamy, S, P Francis, M Handley, R Karp, and S Schenker, *A scalable content-addressable network*. 2001: ACM Press New York, NY, USA.
5. Rowstron, A and P Druschel, *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.* IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001. 11: p. 329-350.
6. Maymounkov, P and D Mazieres, *Kademlia: A peer-to-peer information system based on the XOR metric.* Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), 2002. 258: p. 263.
7. Schlosser, M, M Sintek, S Decker, and W Nejdl, H*yperCuP—Hypercubes, Ontologies and Efficient Search on P2P Networks. I*nternational Workshop on Agents and Peer-to-Peer Computing.
8. Mitzenmacher, M, Di*gital fountains: a survey and look forward. I*nformation Theory Workshop, 2004. IEEE, 2004: p. 271-276.
9. Plank, JS, A *tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. S*oftware Practice and Experience, 1997. 27(9): p. 995-1012.
10. Chun, BG, F Dabek, A Haeberlen, E Sit, H Weatherspoon, MF Kaashoek, J Kubiatowicz, and R Morris, E*fficient replica maintenance for distributed storage systems. P*roc. of the 3rd Symposium on Networked Systems Design and Implementation, 2006.
11. Li, J, F Dabek, UC Berkeley, and MIT MIT, F2*F: reliable storage in open networks. P*roc. of the 5th International Workshop on Peer-to-Peer Systems, 2006.
12. PlanetLab, ht*tp://www.planet-lab.org/.*
13. Wells, C, The *oceanstore archive: Goals, structures, and self-repair. U*C Berkeley Masters Report, May, 2002.
14. Kubiatowicz, J, C Wells, B Zhao, D Bindel, Y Chen, S Czerwinski, P Eaton, D Geels, R Gummadi, and S Rhea, O*ceanStore: an architecture for global-scale persistent storage. P*roceedings of the ninth international conference on Architectural support for programming languages and operating systems, 2000: p. 190-201.
15. Rhea, S, P Eaton, D Geels, H Weatherspoon, B Zhao, and J Kubiatowicz, P*ond: the OceanStore prototype. P*roc. of FAST, 2003. 2003.
16. Bhagwan, R, K Tati, YC Cheng, S Savage, and GM Voelker, To*tal Recall: System Support for Automated Availability Management. I*n Proc. of the First ACM/Usenix Symposium on Networked Systems Design and Implementation (NSDI), 2004.
17. Yang, M, Z Zhang, X Li, and Y Dai, An *Empirical Study of Free-Riding Behavior in the Maze P2P File-Sharing System. I*n 4th International Workshop on Peer-to-Peer Systems, 2005.
18. Gnutella, h*ttp://rfc-gnutella.sourceforge.net/*. 2005.
19. Napster, h*ttp://www.napster.com/*. 2001.
20. Overnet, h*ttp://www.overnet.com*.
21. KaZaA, h*ttp://www.kazaa.com*.
22. Maze, h*ttp://maze.pku.edu.cn*.
23. Blake, C and R Rodrigues, H*igh Availability, Scalable Storage, Dynamic Peer Networks: Pick Two. I*n 9th Workshop on Hot Topics in Operating Systems, 2003.
24. upstore, h*ttp://upstore.grids.cn*. 2006.
25. Dabek, Frank, MF Kaashoek, D Karger, R Morris, and I Stoica, W*ide-area cooperative storage with CFS. 1*8th ACM Symposium on Operating Systems Principles (SOSP'01). October, 2001.
26. Stribling, J, Over*Cite: A Cooperative Digital Research Library. In* 4th International Workshop on Peer-to-Peer Systems, 2005.
27. Druschel, P and A Rowstron, PAS*T: A large-scale, persistent peer-to-peer storage utility. P*roc. HotOS VIII, 2001.
28. Adya, A, RP Wattenhofer, WJ Bolosky, M Castro, G Cermak, R Chaiken, JR Douceur, J Howell, JR Lorch, and M Theimer, Fars*ite: federated, available, and reliable storage for an incompletely trusted environment. A*CM SIGOPS Operating Systems Review, 2002. 36: p. 1-14.
29. Zhang, Z, Q Lian, S Lin, W Chen, Y Chen, and C Jin, Bit*Vault: a Highly Reliable Distributed Data Retention Platform. u*nder submission, 2005.
30. Bhagwan, R, S Savage, and G Voelker, Und*erstanding availability. P*roceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003.

31. Zhang, Z, Q Lian, and Y Chen, XRi*ng: Achieving High-Performance Routing Adaptively in Structured P2P. Te*chnical Report, MSR-TR-2004-93, Microsoft Research, 2004.
32. Lian, Q, W Chen, and Z Zhang, On *the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems. D*istributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on, 2005: p. 187-196.
33. Zheng, W, J Hu, and M Li, Gra*nary: architecture of object oriented Internet storage service. E*-Commerce Technology for Dynamic E-Business, 2004. IEEE International Conference on, 2004: p. 294-297.
34. Hu, J, M Li, W Zheng, D Wang, N Ning, and H Dong, Sma*rtBoa: Constructing p2p Overlay Network in the Heterogeneous Internet Using Irregular Routing Tables. Th*e 3rd International Workshop on Peer-to-Peer Systems, San Diego, CA, USA, February, 2004.
35. Lunar, h*ttp://maze.pku.edu.cn/lunar.htm. 2*005.
36. eMule, h*ttp://emule.org/*.
37. Tian, Jing, Zhi Yang, and Yafei Dai, SEC*: A Practical Secure Erasure Coding Scheme for Peer-to-Peer Storage System. 14*th Symposium on Storage System and Technology, 2006.
38. Batten, C, K Barr, A Saraf, and S Trepetin, pSt*ore: A secure peer-to-peer backup system. Un*published report, MIT Laboratory for Computer Science, December, 2001.
39. Cox, LP, CD Murray, and BD Noble, Pas*tiche: making backup cheap and easy. AC*M SIGOPS Operating Systems Review, 2002. 36: p. 285-298.
40. Ghemawat, S, H Gobioff, and ST Leung, The *Google file system. Pr*oceedings of the nineteenth ACM symposium on Operating systems principles, 2003: p. 29-43.
41. Weatherspoon, H and J Kubiatowicz, Eras*ure coding vs. replication: A quantitative comparison. Pr*oc. of IPTPS, 2002. 2.
42. Bhagwan, R, S Savage, and G Voelker, Rep*lication strategies for highly available peer-to-peer storage systems. p*roc. of FuDiCo: Future directions in Distributed Computing, June, 2002.
43. Lin, WK, DM Chiu, and YB Lee, Eras*ure code replication revisited. In* Proc. of Fourth International Conference on Peer-to-Peer Computing, 2004: p. 90-97.
44. Rodrigues, R and B Liskov, Hig*h Availability in DHTs: Erasure Coding vs. Replication. Pr*oc. of the 4th International Workshop on Peer-to-Peer Systems, 2005.
45. Tian, Jing, Yafei Dai, Hao Wang, and Mao Yang. Und*erstanding Session Durability in Peer-to-Peer Storage System. i*n ICC*S. 2*006. UK: LNCS.
46. Douceur, JR and RP Wattenhofer, Com*petitive Hill-Climbing Strategies for Replica Placement in a Distributed File System. Pr*oceedings of 15th DISC, 2001: p. 48-62.
47. Ramanathan, MK, Inc*reasing object availability in peer-to-peer systems. In* Proc. of Parallel and Distributed Processing Symposium, 2004.
48. Schwarz, Thomas J. E., Qin Xin, and Ethen L. Miller, Ava*ilability in Global Peer-To-Peer Storage Systems. In* Proc. of 6th Workshop on Distributed Data and Structures (WDAS), 2004.
49. Tati, K and GM Voelker, On *object maintenance in peer-to-peer systems. Pr*oc. of the 5th International Workshop on Peer-to-Peer Systems, 2006.
50. Weatherspoon, H, BG Chun, CW So, and J Kubiatowicz, Lon*g-Term Data Maintenance in Wide-Area Storage Systems: A Quantitative Approach. Co*mputer, 2005.
51. Ramabhadran, S and J Pasquale, Ana*lysis of long-running replicated systems. Pr*oc. of the 25th IEEE Annual Conference on Computer Communications (INFOCOM), 2006.
52. Sit, E, A Haeberlen, F Dabek, BG Chun, H Weatherspoon, R Morris, MF Kaashoek, and J Kubiatowicz, Pro*active replication for data durability. 5*rd International Workshop on Peer-to-Peer Systems (IPTPS 2006), 2006.
53. Guha, S, N Daswani, and R Jain, An *Experimental Study of the Skype Peer-to-Peer VoIP System. Pr*oceedings of IPTPS, 2006.
54. Haeberlen, A, A Mislove, A Post, and P Druschel, Fal*lacies in evaluating decentralized systems. Pr*oc. of the 5th International Workshop on Peer-to-Peer Systems, 2006.
55. Tian, Jing and Yafei Dai, Und*erstanding the Dynamic of Peer-to-Peer Systems. Te*chnical Report, CNDS-TR-2006-10, Peking University, 2006.
56. Utard, G and A Vernois, Dat*a durability in peer to peer storage systems. Cl*uster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on, 2004: p. 90-97.