

ESStore: P2P 网络的可靠存储协议

田敬¹, 张大为¹, 代亚非¹, 韩华¹

(1. 北京大学计算机科学技术系, 北京 100871)

摘要: ESStore 是一种用于提高基于分布式哈希表的结构化 P2P 存储系统可靠性和可用性的机制。它在 P2P 网络层之上, 使用 Erasure code 对文件进行编码存储。本文描述了 ESStore 在 P2P 网络中进行文件存储、读取和删除的基本方法, 以及进行数据校验、一致性维护、修复和垃圾碎片回收的基本机制。另外, 本文给出了协议可用性的定量分析, 并通过实际系统说明了该协议的存储性能。

关键词: Erasure code; P2P; 可用性; 可靠性

ESStore: A reliable P2P storage protocol

Tian Jing¹, Zhang Da-Wei¹, Dai Ya-Fei¹ and Han Hua¹

(Department of Computer Science and Technology, Peking University, Beijing 100871)

Abstract: ESStore is a protocol used in P2P storage system to improve the reliability and availability. ESStore can be applied on all P2P networks which are based on distributed hash table, and it utilizes Erasure code to encode file content to store. In this paper, we discussed the approach to store, retrieve and delete files in detail and the mechanism of data verification, consistency, repair and garbage collection. Finally we analyze the reliability and availability, and give out the performance of our prototype system.

key words: Erasure code; P2P; reliability; availability

1. 引言

随着网络的进一步发展, 应用在人们生活中的计算设备的数目不断增多, 并产生了大量的数据。为满足日益复杂的各种信息存储需求, 人们提出网络存储, 数据统一存放在远端的服务器上, 便于数据的访问、维护、管理及共享。对海量数据的长期可靠的存储要求存储服务的空间有很好的扩展性, 目前 P2P 存储网络从理论上很好的解决了扩展性的问题。然而 P2P 网络中节点的不稳定性降低

基金资助: 国家网络与信息安全保障持续发展计划(2004 研 3-917-A-03); 国家自然科学基金资助(60303002)

作者简介: 田敬 (1979-), 男, 北京顺义, 博士生 Email: tianjing@net.pku.edu.cn

的系统的可用性。节点的不稳定性一方面影响 P2P 网络的路由,另一方面影响存储的数据的可用性。关于路由稳定性的问题已经有不少研究工作,本文将关注 P2P 系统中数据的可用性问题。提高数据可用性可以从硬件和软件两个不同层次来考虑,在硬件上可以使用 RAID 机制进行数据冗余,但其完全本地的冗余既不能满足异地容灾的需求也不能适用于分布式的 P2P 系统。为了提高 P2P 存储网络中的数据可用性,本文提出适用于 P2P 网络的 ESStore (Erasure Store) 可靠存储协议。ESStore 是构建在基于分布式哈希表 (DHT) 的结构化 P2P 网络之上的,利用 Erasure code 对数据编码分片存储的存储协议。使用 ESStore 的网络存储系统具有,

- 高可靠性: 存储在系统中的文件对象具有相当高的 MTTF (Mean Time to Failure, 平均故障时间), 这里的故障指的是文件对象的丢失。
- 高可用性: 存储在系统中的文件对象在系统运行的绝大部分时间可以被有效的访问。
- 完整性: 可以确保返回给用户的文件对象没有被篡改或损坏。

本文首先介绍 ESStore 的背景知识: 基于分布式哈希表的结构化 P2P 网络以及文件的软件冗余编码, 其次介绍 ESStore 的存储协议, 然后给出系统可靠性等参数的分析并通过原型系统得到性能参数, 最后是相关研究及结论。另, 为讨论方便, 如无特殊说明, 本文中“P2P 网络”均指基于分布式哈希表的结构化 P2P 网络。

2. 背景知识

2.1 基于分布式哈希表的 P2P 网络

P2P 网络是一个没有中心控制的对等网络, 它追求系统的扩展性, 和能够解决单点失效等问题。P2P 网络按照路由方式可以分为结构化和非结构化两种, 其中结构化的 P2P 网络目前主要是基于分布式哈希表实现的, 如 CAN[2]、Chord[3]、Tapestry[7]等。

基于分布式哈希表的 P2P 网络虽然拓扑结构的描述不同, 如 CAN 是二维平面拓扑, 而 Chord 则是环形拓扑, 但它们的核心思想是一致的, 也即系统中的所有节点分割哈希值所在区间, 使得总有且仅有一个节点负责任任给定的哈希值。根据这样的设计, 基于分布式哈希表的 P2P 网络基本都提供以下三个基本操作, 插入 $insert(key, value)$ 、读取 $get(key)$ 和删除 $delete(key)$ 。这样每个节点都可以像操作哈希表一样通过键值向系统中存储文件, 或从系统中读取文件。ESStore 就构建在这三个基本操作之上, 因此它适用于所有基于分布式哈希表的 P2P 网络。

2.2 Erasure code 理论

Erasure code 是由一个四元组 (m, n, b, r) 来确定的编码。设其编码函数为 E , 解码函数为 D , 对于消息 $M = (M_1, M_2, \dots, M_m)$, 其中 $M_i (1 \leq i \leq m)$ 为大小为 b bits 的消息包, 编码后的消息 $E(M) = (M'_1, M'_2, \dots, M'_n)$, 其中 $M'_i (1 \leq i \leq n)$ 大小仍为 b bits。设 $E(M)'$ 为 $E(M)$ 中任意 $r (r \geq m)$ 个包组成的子消息, 则 $D(E(M)') = M$, 即对 $E(M)$ 中任意 r 个消息包进行解码就可以得到原始消息。Erasure code 作为一种 FEC (Forward Error Correction) 技术主要应用在网络传输中避免包的丢失, ESStore 利用它来提高存储可靠性, 首先将要存储在系统中的文件分割成 m 块, 然后对其编码得到 n 个文件碎片并进行分布存储, 这样只需要存在 r 个可用的文件碎片, 就可以重构得到原始文件。

Erasure code 有多种实现方式, ESStore 使用其线性码 (利用线性代数性质的编码) 的实现, 并且具有 MDS 编码 (maximal distance separable, 编码后的消息中任意长度等于原始消息的子消息都可以解码得到原始消息的编码称为 MDS 编码) 性质, 即 $r = m$, 本文如不特别指出, 均为 $r = m$ 。

Erasure code 作为一种冗余机制来提高数据的可靠性可以看作是完全备份和 RAID 机制的一般情况, ($m = 1, n = 2$)的 Erasure code 相当于完全备份, RAID 5 则可以描述成($m = 4, n = 5$)的 Erasure code 系统。完全备份系统为提高可靠性只能成倍的增加冗余数据, 需要太多的额外存储容量, 而 RAID 系统又不适用于故障出现频繁的 P2P 网络。Erasure code 弥补了这些缺陷, 可以通过合理的额外存储来提供高可靠性和可用性。

3. ESStore 可靠存储协议

ESStore 是 P2P 网络的上层协议, 它直接使用 P2P 网络提供的三个基本 API, $insert(key, value)$ 、 $get(key)$ 和 $delete(key)$ 。通过包装这三个 API, ESStore 提供了三个更为可靠的存储 API: $es_store(fid, f)$ 、 $es_read(fid)$ 、 $es_delete(fid)$ 。ESStore 协议构建在 P2P 网络之上, 因此将不关心路由的容错问题, 我们假定底层 P2P 网络能够迅速检测路由错误并修复, 也即对于任意键值, 任何时刻都应能够访问到负责该键值的节点。

为了讨论方便, 我们这里给出 ESStore 的相关概念定义,

- fid_f : 系统存储的文件 f 的唯一标识 (通过文件名等信息的哈希变换得到)
- $pid_{f, n}$: 文件 f 通过 Erasure Code 形成的第 n 个文件碎片的唯一标识
- P_f : 文件 f 的属性信息, 包括 f 的当前逻辑时间戳, 编码参数和校验码等

ESStore 协议主要通过 Erasure Code 对 f 进行冗余编码, 并将碎片散布到多个节点防止单点失效来提高数据文件 f 的可用性。由于使用 Erasure Code, 我们就必须记录文件 f 的 Erasure Code 编码参数, 例如编码时填充的字节数 (Erasure Code 实现时需要要对文件填充空字符以对齐) 等。我们把相关的参数存在 f 的属性信息 P_f 中, 也即不同文件可用不同参数, 这样既能满足不同类型文件的不同可用性需求, 又可以随着系统节点增加适当调整编码参数。当然 P_f 的存在也造成了单点失效问题, ESStore 协议通过 P_f 多副本策略来解决这个问题。

3.1 简化的 ESStore 协议

本小节我们将通过 ESStore 协议的三个具体 API 来描述 ESStore 如何实现对文件的 Erasure Code 冗余存储的。为了讨论简单, 本小节是 ESStore 的一个简单版本, 暂不讨论 P_f 的多副本管理问题。

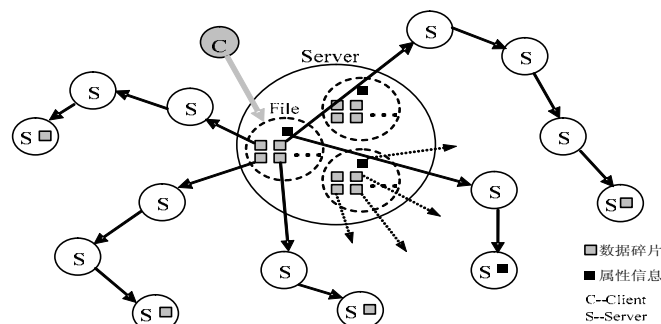


图 1 ESStore 的存储过程

Fig.1 A storing process in ESStore

3.1.1 文件存储过程

ESStore 协议的存储 API 是 $es_store(fid, f)$, 也即 P2P 存储网络的任意服务器节点 S_i 要可靠的存

储文件 f 需给出其唯一标识 fid_f , 标识可以是文件名等唯一的东西。当 S_i 调用了 $es_store(fid_f, f)$ 后, ESStore 首先在 S_i 节点上把 f 通过 Erasure Code 打散为 n 个碎片, 用 $frag_0, frag_1 \dots frag_n$ 表示, 然后调用 P2P 层 API 存储碎片, 如果存储成功的碎片数超过阈值 $K(\geq m)$ 则说明成功存储, 最后存储生成的属性信息 P_f 。图 1 是一个 ESStore 协议的存储示意, 客户将文件存入 ESStore 系统中的某个 Sever, Sever 对其编码, 将形成的碎片根据 pid 在 P2P 网络中存储, 属性信息根据 fid 在 P2P 网络中存储。ESStore 的这个过程可用图 2a 的伪代码描述,

```
//文件打散, 由 m 块变换到 n 块
ErasureEncode(f, m, n)
//存储碎片
counter = 0
for each fragi
  if insert(pidr,i, fragi)
    counter ++
//成功存储则保存属性信息, 并返回
if (counter > K)
  insert(fidr, Pf)
  return true
return false
```

图 2a 存储文件的伪代码
Fig.2a Pseudocode of storing

```
//取得 f 的属性信息 Pf
Pf = get(fidr)
for i from 0 to Pf.n
  启动新线程执行 fragi = get(pidr,i)

//等待, 直至有至少 m 个碎片或所有线程结束
wait()
if 有 m 个碎片
  f = ErasureDecode(frag, Pf)
  return f
else
  return null
```

图 2b 读取文件的伪代码
Fig.2b Pseudocode of retrieving

3.1.2 文件读取和删除

读取过程相对简单, 若 S_i 发起对文件 f 的读请求, 也即调用 $es_read(fid_f)$ 方法, 则 S_i 先得到 P_f , 然后根据编码参数确定需要取回的碎片个数, 并发的请求碎片, 最后组装出文件 f 。处理过程如图 2b 所示。可以看出存储和读取过程基本就是逆过程。

要删除一个文件, 先要得到文件的 fid , 然后向其根节点请求属性信息, 根据 fid 和时间戳获得所有文件碎片的 pid , 再向其各自的根节点请求删除碎片, 最后删除所有属性信息。另外, 未能及时删除的碎片也可以交给垃圾回收机制进行清理。

3.2 完全 ESStore 协议

前面描述的简化 ESStore 协议存在一个属性信息 P_f 单点失效的问题, 它成为系统可用性的瓶颈。ESStore 采用多副本的方法解决这个问题, 也即采用多个哈希函数, 使得每个文件 f 有多个 fid 。这样, ESStore 就在每个 fid 所在节点存一份属性信息, 其中第一个哈希函数所对应的 fid 的属性信息称为主属性信息。由于多个属性信息的存在, 造成了一致性维护的问题, 以及并发控制的问题。

针对一致性问题, ESStore 采用[6]中 Gifford 提出的加权投票方法, 通过维护属性信息的逻辑时间戳来找到当前最新的副本。这个方法要求读写操作之前, 操作者都必须得到足够多的选票, 以保证能访问到最新的副本。假定每个属性信息拥有 S 个副本, 而每个副本拥有 1 张选票。ESStore 要求写操作必须得到大于或等于 $\lfloor S/2 \rfloor + 1$ 张选票才能进行, 而读操作要得到大于或等于 $\lceil S/2 \rceil$ 张选票。这样保证了读取和更新的副本集合中总会有最新的副本, 维护了一致性。

ESStore 利用 P2P 网络的特性, 可以简单有效的解决并发控制。由于 P2P 网络路由修复机制, 使得总有一个节点负责键值 fid 所在空间, 也即文件 f 的主属性信息所在节点总是可以被访问到的。这样, 我们可以让主属性信息所在节点作为对文件 f 读写操作的协调者, 从而方便的保证序列化性质。当然, 如果主属性信息所在节点丢失了属性信息, 可以及时从副属性信息中恢复出来。

3.3 文件校验

与纠错码不同，Erasure code 必须确切的知道接收到的数据哪部分正确，哪部分存在错误，解码才能正确进行，否则将得到错误的源数据。因此，在对文件进行重构前要验证文件碎片是否正确，是否发生错误或被篡改，以保证不会返回给用户错误的文件。

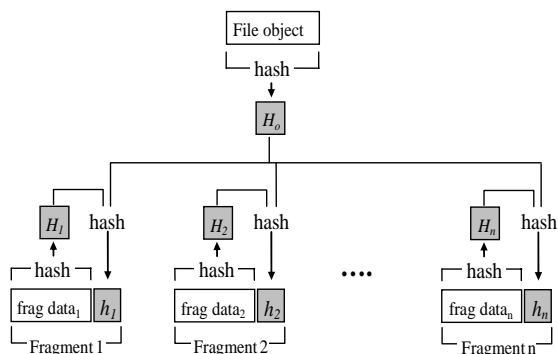


图 3 自校验文件存储

Fig.3 Self-verifying File Storage

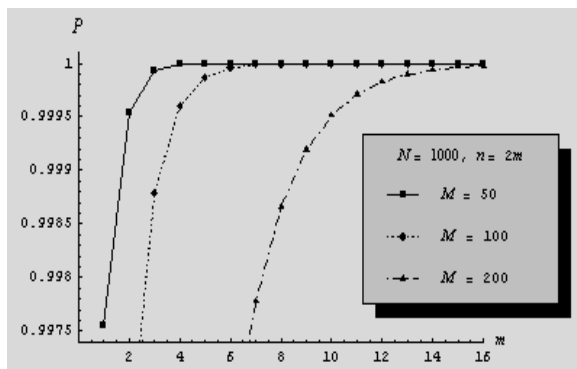


图 4 Erasure code 的可用性走势图

Fig.4 Availability Trend

ESStore 中使用如下校验方案：对每个文件对象的内容进行散列得到散列值 H_o ，存放在属性信息中。对编码后的每个碎片同样进行散列得到 H_i ，再对 H_o 和 H_i 进行散列得到散列值 h_i ， h_i 便是第 i 个碎片的校验码， h_i 与第 i 个数据碎片 $frag\ data_i$ 一起存储，如图 3。在获得碎片进行重构时，对碎片的数据部分散列求得散列值 H_i 并与 H_o 进行散列得到 h_i' ，如果与碎片的校验码 h_i 相同则校验正确。为防止恶意篡改，可以对重构后得到的文件对象的内容进行散列，判断散列值是否等于 H_o 完成进一步校验。

3.4 文件修复和垃圾回收

系统在运行一段时间后，存储在系统中的数据碎片可能会因为某些不可预知的原因而丢失或者被损坏，从而降低了存储在系统中数据对象的可靠性。因此，设计并实现一个简单而且高效的修复算法相当重要。目前 ESStore 在用户访问数据的同时，检查文件的碎片个数，如果碎片个数少于一个阈值，则启动修复进程，重构文件并重新分发碎片。这样使得经常被访问的文件有更高的可用性。

系统在运行过程中由于非正常断线等原因会造成垃圾碎片产生，目前垃圾碎片的回收还只能依靠全局状态的扫描，查看碎片是否是垃圾碎片。

4. ESStore 可用性分析

我们从系统可用性方面来对 ESStore 进行评估。我们首先假设属于某个文件的所有碎片分布存储在不同的服务器上，并且这些服务器是错误不相关的，即服务器发生的故障是互相统计独立的。

使用 Erasure code 对文件进行冗余存储，比同样冗余率的备份机制具有更高的可用性。Erasure code 带来的高可用性主要利用了大量独立部件的统计稳定性，其可用性随总碎片数目的增多而增大，随重构所需要的碎片数目减少而增大。[5]中给出了 Erasure code 所具有的可用性的计算公式，如下：

$$P = \sum_{i=0}^{n-m} \binom{M}{i} \binom{N-M}{n-i} / \binom{N}{n} \quad (1)$$

其中, P 为文件对象可用概率, n 为文件对象编码后的所有碎片的数目, m 为重构文件对象所需要的碎片的数目, N 为系统中全部机器的数目, M 为系统中当前不可用的机器数目, 即文件对象可用的概率 P 等于存放在不可用的机器上的碎片数目少于 $(n - m)$ 的存储方法数比上所有碎片存放在任意机器上的方法数。

假如系统中有 1000 台机器, 故障率为 5%, 对于使用 $(m = 16, n = 32)$ 的 Erasure code 的系统, 根据(1)算出可用性为 0.9999999999999986, 而对于完全备份系统来说, 它具有的可用性为 0.9975。显见, 同样的冗余量, 两种系统可以达到的可用性相差很大。图 4 描述了可用性随数据碎片数目变化的走势, 可以看出可用性随着数据碎片数目的增加而增大。

5. 原型系统及性能评测

北京大学网络实验室在自己设计的 Emergint[1]路由算法基础上, 使用 ESStore 协议实现了一个 P2P 存储网络——燕星 2.0 系统。燕星 2.0 系统由多台存储服务器按照结构化的 P2P 方式组织而成, 客户可以从任意一个节点请求服务。用户文件 f 的存储请求, 先会调用 P2P 层的 $insert(fid, f)$ 进行存储, 而后调用 ESStore 协议的 $es_store(fid, f)$ 创建一个文件 f 的归档保存。用户取文件时也先尝试 P2P 层的 $get(fid)$, 如果文件丢失则使用 ESStore 协议从碎片中恢复, 也即 ESStore 是燕星 2.0 文件归档提高可用性的机制。

表 1 用户与服务器之间网络传输速度受后台可靠存储程序影响的变化

Tab.1 ESStore's influence on system performance

No.	情况 1 (kbytes/s)		情况 2 (kbytes/s)	
	无 ESStore	有 ESStore	无 ESStore	有 ESStore
1	570.09	538.29	448.47	371.27
2	571.27	528.90	443.27	370.20
3	598.04	566.59	451.23	391.26
4	609.05	581.46	479.03	399.90
5	504.61	493.46	415.83	350.61
6	535.87	511.27	398.6	360.23
7	590.41	567.88	464.63	395.17
8			782.22	643.02
9			832.53	666.88
10			811.63	625.65

通过燕星 2.0 系统, 我们测试了 ESStore 对存储系统的性能影响。首先我们给出用户文件存储时间的公式, 假设文件大小为 F_{size} , 网络速度为 N_{speed} , 编码时间为 T , 则存储时间 $StoreTime$ 为,

$$StoreTime \approx \frac{F_{size}}{N_{speed}} + T + \frac{F_{size}n}{N_{speed}m} \tag{2}$$

由于燕星 2.0 中使用 ESStore 归档是使用单独线程异步方式进行的, 也即不阻塞用户操作, 因此用户的响应时间仅为公式中第一部分 $\frac{F_{size}}{N_{speed}}$ 。

我们使用 7 台分布在北大校园网内三个网段的服务器组建了一个燕星 2.0 系统, 主要为了比较系统不开启 ESStore 后台进程和开启 ESStore 后台进程时对用户传输速度的影响。情况 1 中, 我们启动了 7 台距离服务器群比较远的客户端, 同时向系统上传文件; 情况 2 中, 增加了 3 台距离服务器群较近 (与服务器之间网络带宽高) 的客户端, 共 10 台客户端同时向系统上传数据, 如表 1 所示。

从表 1 中的数据可以看出, 在后台运行可靠存储程序对于具有较低网络传输速度的用户的传输

性能影响并不大,对于具有较高传输速度的用户影响相对高一些。从情况2中8,9,10三个客户可以看出,当客户访问速度非常快的时候才会导致 ESStore 后台进程处理相对紧张,较多占用网络带宽等资源。但是目前在实际应用中,由于网络原因用户与服务器之间一般无法具有很高的传输速度,而且服务器为了减轻负载和能够为更多用户提供服务都要对用户做限速设置。因此 ESStore 可靠存储协议只牺牲小部分的网络性能却换来相当高的存储可靠性。

6. 相关研究

目前在国际上还存在着几种基于 P2P 的网络存储系统: OceanStore[7], CFS[8], PAST[9], FarSite[10] 等。这些系统都设计了自己的副本策略来提高系统可用性,但都没有能够提出普遍适用于 P2P 系统的可靠存储协议。[4]中提出了一个适合结构化 P2P 的可靠存储策略,但是其并发控制策略没有明确说明,而 ESStore 则利用 P2P 网络的特性简单有效的提供了好的并发控制,另外 ESStore 针对每个文件都可变化的编码参数给其上层应用更强的灵活性。

7. 结论

ESStore 是一种基于 P2P 网络的可靠存储协议,它使用 Erasure code 对文件进行编码存储,以相对少的冗余度获得了很高的系统可用性和可靠性,弥补了备份和 RAID 系统的缺陷。较之其它使用 Erasure Code 的冗余策略,ESStore 具有适用面广、简单有效、灵活方便等特点。另外本文的贡献还在于给出了 ESStore 协议性能的定量分析,包括可用性和可靠性两个概念,并通过原型系统给出了系统性能的实际数据,为本协议的实际工程应用提供了可靠的依据。

参考文献:

- [1] 韩华. 面向 Internet 的分布式海量文件存储系统研究[D]. 保存地点: 北京大学图书馆, 2002
- [2] Ratnasamy S, Francis P, Handley M, et al. A scalable content-addressable network[J]. In Proceedings of ACM SIGCOMM, 2001
- [3] Stoica I, Morris R, Karger D, et al. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications[J]. In Proceedings of ACM SIGCOMM, 2001
- [4] Zhang Z, Lian Q. Reperasure: Replication protocol using erasure-code in peer-to-peer storage network[J]. In Proceedings of 21st IEEE Symp. on Reliable Distributed Systems, 2002.
- [5] Weatherspoon H, Kubiatowicz J. Erasure Coding vs. Replication: A Quantitative Comparison[J]. In Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS), 2002
- [6] Gifford D. Weighted voting for replicated data[J]. In Proceedings 7th Symposium on Operating Systems Principles, 1979
- [7] Kubiatowicz J, Bindel D, Chen Y, et al. OceanStore: An Architecture for Global-Scale Persistent Storage[J]. In Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS 2000), 2000.
- [8] Dabek F, Kaashoek M, Karger D, et al. Wide-area Cooperative Storage with CFS[J]. In Proceedings of SOSP, 2001.
- [9] Druschel P, Rowstron A. PAST: A large-scale, persistent peer-to-peer storage utility[J]. In Proceedings of HOTOS, 2001.
- [10] Bolosky W, Douceur J, Ely D, et al. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs[J]. In Proceedings of ACM SIGMENTRICS'2000, 2000.