拜占庭容错纠删码分布式存储协议①

蔡鸾佳

(同济大学 电子信息工程学院，上海 201804)

摘 要：研究了从数据密集大部分拜占庭容错分布式存储协议使用复制技术，但是当存储的数据块很大时，复制技术要求大量的存储空间并占用网络带宽，效率低下。采用纠删码技术则将数据块编码为长度相同的分片，然后把各个分片分别存储在对应的存储节点上，这样可以节省存储空间和网络带宽。拜占庭容错纠删码分布式存储需要额外的开销、附加的服务器甚至要版本化存储，才能保证数据的一致性。通过对通常的情况进行优化，采用新颖的机制并引入同态指纹检验码，使得需要最少的服务器、最小的额外计算量和最小的通信回合数目，达到了低开销，并且可以保证时间戳不跳跃。

关键词：拜占庭容错；纠删码；复制技术；同态指纹；分布式存储；

Byzantine Fault-Tolerant Erasure Eoded Distributed Storage Protocol

CAI Luan-Jia

(School of Electronic and Information Engineering, Tongji University, Shanghai 201804, China)

Abstract: Most byzantine fault-tolerant protocol distributed storage protocol use replication,but if the block is very huge,replication requires network width and many spaces, so its efficiency is low. Employing erasure coding makes the data block coded into fragments of the same length, then store the according fragment on corresponding node. This can save spaces and network width. Byzantine fault-tolerant erasure coded distributed storage usually requires much overhead, extra servers and versioned storage for the consistency of the data fragments. The paper optimizes for the common cases, employs novel mechanism and introduces homomorphic fingerprints. The protocol reaches the real overhead, because it requires minimal servers, least extra computation and minimal communication rounds. Furthermore, it can make timestamp no-skipping

Key words: web information extraction; byzantine fault-tolerant; erasure code; replication; homomorphic fingerprints

随着互联网技术不断普及和进步，计算机系统在各个领域都起着越来越重要的作用，导致现代企业和个人的数据信息量正在呈爆炸式的增长，而数据作为在计算机系统内存在的形式越来越成为企业和个人最重要的财富。

分布式存储系统，就是将数据分散存储在多台独立的设备上，能满足大规模存储应用的需要。它需要有容错的能力，拜占庭容错指可以容忍组件任意类型的错误，包括能够容忍像欺骗这种类型的故障。

本文采用纠删码编码技术实现拜占庭容错分布式存储，现有的纠删码拜占庭容错协议都需要额外的服务器，多余的计算量和版本化存储，以确保数据的一致性。在故障和崩溃很少的通常情况下，采用新颖的机制进行优化，用同态指纹进行纠删码分片验证，使得编码计算量开销减少且读操作期间数据分片的验证计算量也减少，同时本文只需要 3f+1 个存储节点来容忍 f 个拜占庭故障，完成写操作只需要两个通信回合并且确保数据块分片时间戳不跳跃。

1 纠删码

一个（n,m）纠删码把 m 个源数据编码为 n（n>k）个数据，使得用这 n 个数据中任意大于等于 m 个编码数据均可重构原来 m 个源数据（如图 1 所示）。通常情况下一个（n,k）线性纠删码是可以是表示为 y=xG,

① 基金项目:安徽省教育厅自然科学基金(2005KJ004ZD)
收稿时间:2011-06-09;收到修改稿时间:2011-07-18

其中 x=(x0,x1,...,x(k-1))是源数据，y=(y0,y1,...,y(k-1)),G 为 k 行 n 列矩阵，称 G 为（n,k）线性纠删码的生成矩阵，通常 G 的任意 k 列矩阵均可逆。

1.1 纠删码编码

设客户需要存储的数据块为 B，长度大小固定为 L，对 B 进行纠删码编码生成 n 个分片，每个分片的数据长度是相同的。数据只有冗余才能实现容错，假设采取 n 取 m 纠删码编码，那么 n 中任意 m 个分片都能够解码出源数据块，并且源数据块是唯一的。

$$encode(B) \rightarrow d_1, d_2, ..., d_n \quad (|d_1| = |d_2| = ... = |d_n|)$$

$$decode(d_1, d_2, ..., d_m) \rightarrow B_1 \quad (m < n)$$

$$decode(d_2, d_3, ..., d_{m+1}) \rightarrow B_2$$

$$B_1 = B_2$$

为了容忍 f 个拜占庭错误，至少将数据块编码成 3f+1 个分片，本文模型中 3f+1 个存储节点（nodes）存储各个对应的分片，为了防止拜占庭存储节点联合欺骗使得系统发生故障，m 必须大于或者等于 f+1[3]。采用局部编码可以降低开销，只需对 2f 个分片编码，即直接把数据块分成 f+1 个长度相同的数据分片，这 f+1 个分片就是 3f+1 个分片中的前 f+1 个分片(m=f+1), 假设 n=5,m=3 且 f=1，编码方式如图 2 所示。

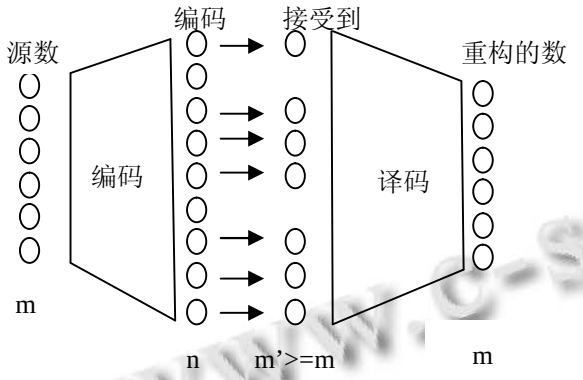


图 1 纠删码编码译码原理图

1.2 分片验证技术

每个存储节点上只存储编码后的某个分片，正确的客户要获得整个数据必须读取至少 m 个分片才能解码出源数据，但是不能确保解码出的这个数据块是否为源数据块。存在拜占庭故障的情况下，不同的 m 个分片的子集可能解码出不同的数据块，所以正确的客户在写数据分片时必须要同时写入检验数据，确保分片来自于唯一的数据块。

Krawczyk 提出了分布式指纹的概念，可以用于对数据分片的验证。实质上是一种称为交叉校验和(crossed checksum，简称 cc)的结构[5]，计算每个分片的哈希值，前提是假设哈希冲突的概率很小并且原像不能够被找到，具体数据结构如图 2 所示。

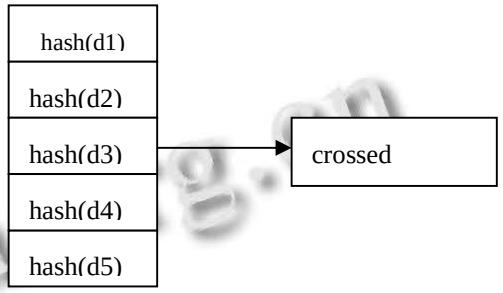


图 2 交叉校验码的计算方法

一般情况下，客户读操作时先从读取到的分片中恢复出数据块 B'，然后用纠删码将 B'编码生成 n 个分片，再分别哈希每一个分片，与交叉检验码中的对应部分相比较，如果全部都一致，那么数据验证成功，即重构的源数据是正确的，与编码前的数据是相等的，代码如下，验证过程如图 3 所示。

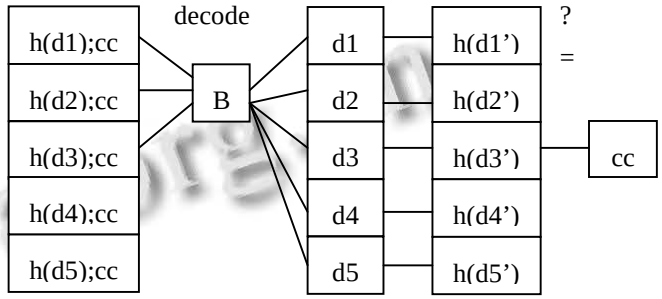


图 3 采用交叉检验码的纠删码验证

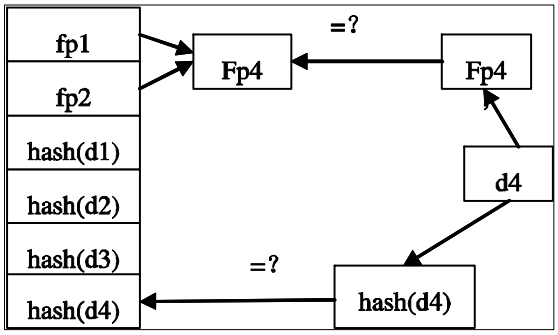


图 4 采用交叉指纹检验码进行纠删码分片验证

同态指纹在线性纠删码中的一个性质是：数据块的一个编码分片的指纹和块的指纹的编码是相等的。本文采用指纹交叉检验码（简写为 fpcc）进行数据分片验证，如果第 i 个分片的哈希与交叉检验码中的第 i 个相等且它的指纹等于同态指纹的纠删码编码的第 i 个，那么存储节点即可以判定这个分片与指纹校验码一致。更重要的是任意 m 个与指纹交叉检验码一致的分片解码出的数据块是唯一的，如图 4 所示。

2　系统模型

每一个客户和服务器之间的点对点的通信信道是可靠的并且是有序的，在实际中花费很小的开销就可以做到。通信信道是可靠的但是为异步的，即每一个发送出去的消息最终会到达但是对消息的传送延迟时间没有限制。至多 f 个存储节点和任意数目的客户是拜占庭故障，它们的行为可以是任意的方式。一个敌人可以协调所有的故障存储节点和客户，在协议中为了限制进程中缓存的纠删码分片，假设客户的数目有一个上限，每个客户的没有进入随后的执行阶段的准备请求的数目也有限制。

本协议完全不采用签名技术，而是依赖于消息验证码 MAC 和随机的 nonce 值，所有的存储节点共享成对的 MAC 密匙，客户既不创建也不验证 MACs。客户只在准备阶段收集存储节点传送的 MACs，在执行阶段再发送给存储节点，节点可以确保有足够数目的正确的存储节点也做好了准备，系统模型如图 5 所示。

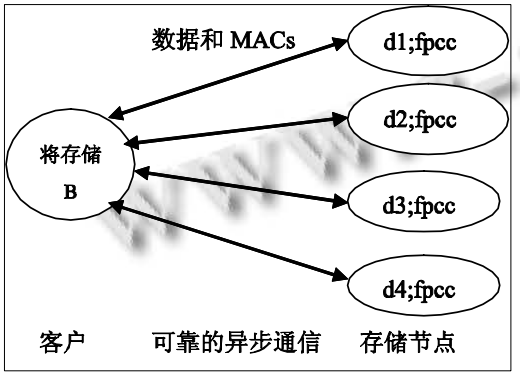


图 5　分布式存储系统模型

3　协议

本文采用 3f+1 个存储节点来容忍 f 个故障的存储节点和一定数目的故障客户（理论上是任意数目的客户）。客户进行读写操作时通过远程过程调用，完全依赖于并发的请求。双杠加上 cobegin 代表执行的并行线程的交叉点，这些进程在 end cobegin 处停止，主程序在线程交叉后继续执行，即主线程并不等待在 end cobegin 处加入交叉的线程，线程不会终止除非它们调用一个远程过程或者等待一个信号（semaphore），信号是二进制并默认为 0，一个 WAIT 操作等待一个信号，而 SIGNAL 释放所有等待的线程，return 语句停止所有的线程并返回一个值。

3.1 写操作

客户以数据块 B 作为参数调用写操作程序，先对数据块 B 进行纠删码编码生成 m+f 个分片（m=f+1），计算出分片的哈希和前 m 个分片的指纹，构成指纹交叉检验码 fpcc，接着进入准备阶段（100-103）。

(1) 准备阶段

客户向各个存储节点发送准备请求，并收集各个存储节点返回的响应，找出最大的时间戳。当收集到的响应数目大于等于 m+f 时，准备成功，但是客户并没有停止接收准备阶段存储节点返回的响应（200-208）。

正确的存储节点收到准备请求，则返回最大的时间戳和最大的准备时间戳，分别计算出其 nonce 值，修改最大的准备时间戳，计算 MACs 值（300-305）。

返回准备时间戳是为了确保时间戳不会跳跃。因为拜占庭故障的存储节点可能返回给客户一个无限大的时间戳，客户收集到的响应在执行阶段会给发送给存储节点，存储节点便可以判定此时间戳是否跳跃。

(2) 执行阶段

客户将纠删码分片、指纹交叉校验码、当前的时间戳和收集的响应集合发送给对应的每一个存储节点，如果前 m+f 个存储节点中有故障，则将整个数据块传送给存储节点。未成功写入分片的存储节点小于等于 f 个，写操作完成。客户与存储节点之间只有两个通信回合，至少 m 个分片被成功地存储（209-221）。

正确的存储节点如果收到的是分片，即为前 m+f 个分片中的第 i 个，则先与指纹交叉检验码进行验证，接着调用 s_rpc_do_commit 进行存储（400-403）；如果存储节点收到的是完整的数据块，则将其重新编码，统计与指纹交叉检验码一致的分片数目，如果有至少 m 个分片与收到的指纹交叉检验码一致，则计算出所有 n 个分片的交叉检验码 cc，生成第 i 个分片，接着调用 s_rpc_do_commit 进行存储（404-412）。

数据验证通过后，存储节点用时间戳、交叉检验码和分片以及哈希值表进行存储。如果这个时间戳比存储节点所拥有的最近的时间戳小，直接返回 SUCCESS，即分片已经被写过。存储节点先确保时间戳不会跳跃，即这个时间戳小于等于至少 f+1 个准备时间戳，因为这 f+1 个存储节点中至少有一个是正确的；还要确保至少 m+f 个节点在准备阶段成功，即其 MACs 是有效的。计算 nonce 的哈希值 nonce_hash，在这个时间戳存储分片和读操作时需要的辅助数据，删除旧的分片，更新当前时间戳（500-509）。

c_write(B):

100: $d_1,...,d_{m+f} \leftarrow encode_{1,...,m+f}(B)$

101: $for(i \in \{1,...,m+f\})$

$do\ \ fpcc.cc[i] \leftarrow hash(d_i)$

102: $for(i \in \{1,...,m\})$

$do\ \ fpcc.fp[i] \leftarrow fingerprint(hash(fpcc.cc), d_i)$

103: c_dowrite($B, NULL, fpcc$)

c_dowrite($B, ts_{commit}, fpcc$):

200: $Prepare[*] \leftarrow NULL$

201: cobegin

202: $prepare[i] \leftarrow$ S$_i$.s_rpc_prepare($ts_{commit}, fpcc$)

203: $if\ (ts_{commit} = NULL \wedge |\{j : Prepare[j] \neq NULL\}| = 2f+1)\ then$

204: $ts_{commit} \leftarrow \max\{ts' : <ts', *> \in Prepare\}$

.....

207: $if\ (|\{j : Prepare[j] \neq NULL\}| \geq m+f)\ then\ SIGNAL(prepare\_ready)$

208: endcobegin

209: $UnwriteSet \leftarrow \{1,...,n\}$

.....

219: $UnwriteSet \leftarrow UnwriteSet \setminus \{i\}$

220: $if\ (|UnwriteSet| \leq f)then$ return $SUCCESS$

221: endcobegin

S_i.s_rpc_prepare($ts_{commit}, fpcc$):

300: $if\ (ts_{commit} = NULL)\ then$

$ts_{commit} \leftarrow latest\_commit.ts + 1$

.....

305: return $<ts_{commit}, nonce, <MAC_{i,j}(<ts_{commit}, fpcc, nonce>)>_{1\leq j\leq n}, ts_{prepare}, nonce_{prepare}, <MAC_{i,j}(<ts_{prepare}, fpcc, nonce_{prepare}>)>_{1\leq j\leq n}>$

S_i.s_rpc_commit($ts_{commit}, fpcc, Prepare, d, B$)

400: $if\ (d \neq NULL)then$

.....

403: $if\ (fp = fp' \wedge h = fpcc.cc[i])then$

return S_i.s_rpc_do _commit($ts_{commit}, fpcc, Prepare, d_i, NULL$)

404: $else\ cnt \leftarrow 0$

.....

412: return S_i.s_rpc_do _commit($ts_{commit}, fpcc, Prepare, encode_i(B), cc$)

S_i.s_rpc_do_commit($ts_{commit}, fpcc, Prepare, d, cc$)

500: $if\ (<ts_{commit}, fpcc> \leq latest\_commit)$ $then$ return $SUCCESS$

.....

506: $for(<ts', fpcc'> < <ts_{commit}, fpcc>)$ $do\ \ store[<ts', fpcc'>] \leftarrow NULL$

.....

509: $else\ \ return\ \ FAILURE$

3.2 读操作

(1) 准备阶段

客户向各个存储节点发送时间戳请求（601-605），节点收到请求后返回最近的时间戳（700）。当存储节点返回的时间戳数目大于等于 2f+1 时，客户选出最大的时间戳，并尝试着在这个时间戳进行读取（606-610）。

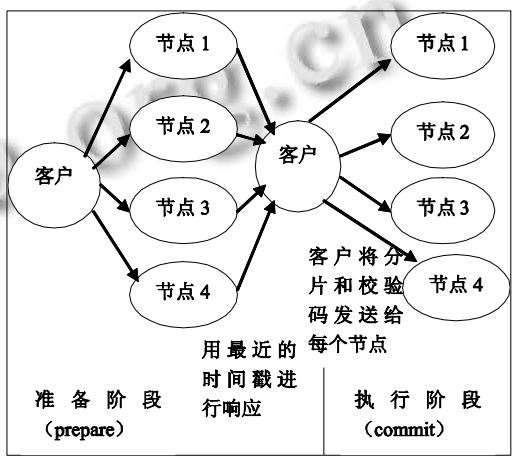


图 6 客户写操作的过程

(2) 执行阶段

客户向各个存储节点请求时间戳最大的分片，收集返回的分片，调用函数 c_find_block 解出源数据 B。如果 B 不为空，客户重新写回分片，主程序返回 B，释放旧的时间戳分片。如果有比当前时间戳更大的分

片，则重新选择时间戳（611-622）。

函数 c_find_block 根据收集的分片解码出源数据块 B（900-903），如果交叉检验码为空，则分片属于前 m+f 个分片，客户先进行指纹交叉校验码验证，如果通过，则加入分片到数组 State。通过 Nouces 值，客户统计对分片的有效性进行过确认的存储节点数目，如果小于 f+1 则无法判定返回 NULL，否则必定有一个存储节点是正确的，即这个分片是正确的，将其加入 Frags，以用于解码源数据块。如果 Frags 的元素数目大于等于 m，，即与指纹校验码一致的分片数目等于 m，就可直接解码源数据块。如果数组 Frags 的元素数目小于 m，则加入 cc 不为空的分片，即不属于前 m+f 的分片，解码出源数据。但是客户也对解码出的这个源数据块进行编码，并统计其与指纹交叉校验码一致的分片数目，如果数目大于等于 m，则返回数据块 B，否则返回 NULL（800-820）。

c_read()

600: $Timestamp[*] \leftarrow NULL; State[*] \leftarrow \varnothing$

601: cobegin

602: $\square_{i\in\{1,..,3f+1\}}$

603: $Timestamp[i] \leftarrow S_i$.s_rpc_find_timestamp()

604: SIGNAL(*found_timestamp*)

605:endcobegin

606:while(*TRUE*)do

……

610: $if\ (ts_{commit} = 0)$ then return *NULL*

611: *cobegin*

612: $\square_{i\in\{1,..,n\}}$

613: $< \overline{ts}, \overline{fpcc} > \leftarrow < ts_{commit}, fpcc >$

……

615: $B \leftarrow$ c_find_blcok($i, state, \overline{ts}, \overline{fpcc}, data$)

……

622: *end cobegin*

S_i.s_rpc_find_timestamp():

700: *return latest _commit*

c_find_block(*i,state,ts, fpcc,<d,cc,nonce_hash,Nonces>*)

800: if($d \neq$ NULL $\wedge$ cc=NULL) then

801: fp $\leftarrow$ fingerprint(hash(fpcc.cc),d)

802: $fp' \leftarrow encode_i(fpcc.fp[1],..., fpcc.fp[m])$ …… ……

817: $if\ (fp = fp' \wedge h = fpcc.cc[j])\ then\ cnt \leftarrow cnt+1$

818: $if\ (cnt \geq m)$ *then*

819: return B

820: return NULL

S_i.*s_rpc_read*

900: if(store[<ts,fpcc>]=NULL $\wedge$latest_commit><ts,fpcc>) then

……

903:return<store[<ts,fpcc>],NULL,$ts_{prepare}$,$nonce_{prepare}$, $< MAC_{i,j}(< ts_{prepare}$,fpcc,$nonce_{prepare} >) >_{1\le j\le n} >$

4 PRISM模型验证

PRISM 是一个概率模型检测器，一个对那些具有随机的或者概率的行为的系统进行形式化建模和分析的工具。它支持三种形式的概率模型离散的马尔可夫链 (DTMCs), 连续的马尔可夫链(CTMCs) 以及马尔

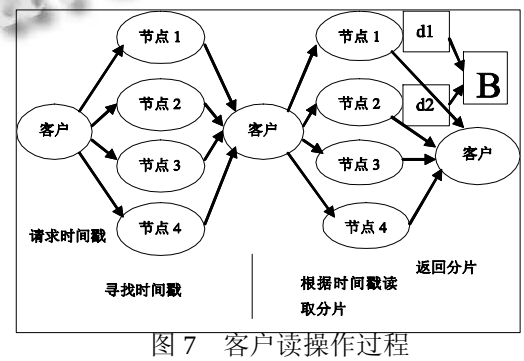
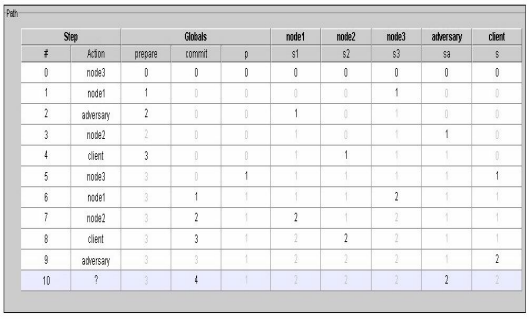


图 7　客户读操作过程

可夫表决处理(MDPs), 以及这些模型的扩展。它可以对多个领域的系统进行分析，包括通信和多媒体协议、随机的分布式算法，安全协议以及生物工程系统等。

由于系统是异步通信，只对操作成功的存储节点响应进行处理，所以用 PRISM 建模语言对系统进行建模，类型为 MDP(马儿科夫表决处理)。整个模型由三部分构成：正确的节点、拜占庭节点和客户，假设 N=4，f=1，对上述算法建模并单步执行仿真，结果如图 8 所示，客户最终会以最小的开销成功执行读写操作，与先前的拜占庭容错分布式存储相比优点如图 9 所示：


Path

Step		Globals			node1	node2	node3	adversary	client
#	Action	prepare	commit	p	s1	s2	s3	sa	s
0	node3	0	0	0	0	0	0	0	0
1	node1	1	0	0	0	0	1	0	0
2	adversary	2	0	0	1	0	1	0	0
3	node2	2	0	0	1	0	1	1	0
4	client	3	0	0	1	1	1	1	0
5	node3	3	0	1	1	1	1	1	1
6	node1	3	1	1	1	1	2	1	1
7	node2	3	2	1	2	1	2	1	1
8	client	3	3	1	2	2	2	1	1
9	adversary	3	3	1	2	2	2	1	2
10	?	3	4	1	2	2	2	2	2

图 8　系统 PRISM 单步仿真图

对比项 / 系统	存储节点数目	通信回合数目	编码分片数目	加密技术
低开销的系统	m+2f	2	m+f	MAC
懒散验证系统	m+3f	2	m+2f	数字签名
PASIS 系统	m+3f	3	m+2f	数字签名

图 9 低开销系统的优点

```
const f=1;
const M=f+1;
const N=3*f+1;
global prepare:[0..(M+f)] init 0;
.....
global p:[0..1] init 0;
module node1
s1:[0..2] init 0;
[] s1=0 & p=0 -> (prepare'=prepare+1) & (s1'=1);
[] s1=1 & p=1 -> (commit'=commit+1) & (s1'=2);
Endmodule
.....
module adversary
sa:[0..2] init 0;
[] sa=0 & p=0 -> 0.2:(prepare'=prepare+1) & (sa'=1) + 0.8:(sa'=1);
[] sa=1 & p=1 -> 0.2:(commit'=commit+1) & (sa'=2) + 0.8:(sa'=2);
endmodule
module client
s:[0..2] init 0;
[] s=0 & prepare=M+f -> (s'=1) & (commit'=0) & (p'=1);
[] s=1 & commit=M+f & p=1 -> (s'=2);
endmodule
```

5 结语

本文提出了一种时间戳不跳跃的低开销拜占庭容错纠删码分布式存储协议，使得容拜占庭故障的存储系统花费最少的硬件设备、最小的附加计算量就能做到并且通过引入同态指纹和指纹交叉校验码避免了版本化存储所带来的浪费。客户读写操作只需与存储节点进行两回合的通信即能完成，还采用局部编码等方法，真正地达到了低开销。

但是这种方法基于对通常情况进行优化，即在故障和崩溃很少的情况下，当有故障存在的情况下，系统要将整个数据发送给下一个存储节点，这样风险是很大的并且带来很大的计算量，如何提高故障出现时系统的效率，这是今后将重点研究的内容。

参考文献

1 Garth R. Efficient Byzantine-tolerant erasure-coded storage. Proceedings of the 2004 International Conference on Dependable Systems and Networks,IEEE,2004.

2 Michael MK, Jay J. Lazy Verification in Fault-Tolerant Distributed Storage Systems. 24th IEEE Symposium on Reliable Distributed Systems, 2005.

3 Hendricks J, Gregory R, Michael K. Low-Overhead Byzantine Fault-Tolerant Storage.ACM,2007.

4 田敬,代亚非.P2P 持久研究综述.软件学报,2007.

5 王宝林.纠删码分片验证技术研究.电脑知识与技术,2010-2.

6 董辉.失效检测器在拜占庭复制系统中的应用.长沙:湖南大学,2007.

7 董辉,杨金民,张大方.拜占庭容错服务的适应性失效检测研究.微电子学与计算机,2006.

8 孙周军,易峰,肖文名.基于拜占庭协议构建具有入侵容忍能力的 Web 服务研究.微电子学与计算机,2008-3.