DOI:10.11918/202406007

扩展子条带的 piggybacking 编码构造

杨红志[1],王 静[1],李 瞳[1],张洁薇[1],刘向阳[2]

(1.长安大学 信息工程学院,西安 710018;2.西北工业大学 电子信息学院,西安 710129)

摘 要: 为解决现有 piggybacking 编码存在的修复度大、校验节点修复带宽高以及无法实现多节点快速修复等问题,提出一种扩展子条带的 piggybacking 编码(substripe-added piggybacking,SAP)构造方案。SAP 构造在最大距离可分(maximum distance separable,MDS)码的基础上扩展子条带,将信息节点数据块分区均匀嵌入,校验节点数据块循环移位放置。通过理论推导,确定 SAP 信息节点和校验节点平均修复带宽率、平均修复度率。最后,将 SAP 与 RSR-I、RSR-II 和 OOP 就存储开销、修复带宽开销和修复度 3 个方面进行对比。结果表明:与 RSR-I、RSR-II 和 OOP 相比,扩展子条带的 piggybacking 编码不仅实现了修复度最优,而且在保证信息节点修复带宽开销较低的同时,明显降低了校验节点的修复带宽开销,且能快速修复多校验节点故障,明显改善了多校验节点故障修复带宽过高的不足。本文提出的 SAP 编码显著提升了 piggybacking 编码的数据恢复效率,尤其针对多校验节点故障,给出了一种快速修复算法,为 piggybacking 编码的优化提供了有效方案。

关键词: 分布式存储;最大距离可分码;piggybacking 编码;修复度;修复带宽

中图分类号: TN911 **文献标志码:** A **文章编号:** 0367-6234(2025)09-0046-10

Construction of substripe-added piggybacking codes

YANG Hongzhi[1], WANG Jing[1], LI Tong[1], ZHANG Jiewei[1], LIU Xiangyang[2]

(1. School of Information Engineering, Chang′an University, Xi′an 710018, China;
2. School of Electronics and Information, Northwestern Polytechnical University, Xi′an 710129, China)

Abstract: To address the issues of large repair degree, high repair bandwidth of parity nodes, and the inability to achieve fast repair of multiple nodes in existing piggybacking codes, a construction scheme of substripe-added piggybacking (SAP) codes is proposed in this paper. Based on maximum distance separable (MDS) codes, the proposed SAP codes extend the substripe, embed the data blocks of the information nodes by region regularly, and place the data blocks of the parity nodes using cyclic shifts. Through theoretical derivations, the average repair bandwidth rates and average repair degree rates of information nodes and parity nodes in SAP are determined. Finally, SAP is compared with RSR-I, RSR-II, and OOP in terms of three aspects: storage overhead, repair bandwidth overhead, and repair degree. The results show that, compared with RSR-I, RSR-II, and OOP, the SAP coding scheme not only achieves optimal repair degree but also significantly reduces parity node repair bandwidth while maintaining low information node repair bandwidth. Additionally, it enables rapid repair of multiple parity node failures, effectively addressing the issue of excessively high repair bandwidth in multiple parity node failures. The SAP coding proposed in this paper significantly improves the data recovery efficiency of piggybacking codes. In particular, a fast repair algorithm is provided to address multiple parity node failures, offering an effective approach for optimizing the piggybacking codes.

Keywords: distributed storage; maximum distance separable (MDS) codes; piggybacking codes; repair degree; repair bandwidth

当今数字化时代,分布式存储系统(distributed storage systems,DSSs)因其成本低、效率高、安全性好、扩展性强等优势已成为海量数据存储的首选方案[1-3]。对于 DSSs 这种超大规模存储集群,存储节点发生故障已经成为一种常态,因此需要存储冗余数据以保证系统存储的可靠性。传统的冗余机制包括三副本复制和纠删码[4-6]。三副本复制虽易于实现,但存储开销过大;纠删码在保证相同可靠性的情况下,能大大提高存储效率,但其在修复故障节点时需下载原始文件大小的数据,修复带宽开销过高。

收稿日期: 2024-06-04;**录用日期:** 2024-07-29;**网络首发日期:** 2025-01-26
网络首发地址: https://link.cnki.net/urlid/23.1235.T.20250126.1358.004
基金项目: 国家自然科学基金(62001059);陕西省重点研发计划(2024GY-YBXM-068)
作者简介: 杨红志(1998—),男,硕士研究生;王 静(1982—),女,教授,硕士生导师
通信作者: 王 静,jingwang@chd.edu.cn

针对三副本复制和纠删码的上述局限性，Rashmi 等[7]于 2013 年首次提出 piggybacking 编码。在最大距离可分(maximum distance separable，MDS)码基础上，piggybacking 编码通过线性组合部分子条带的数据块并将其嵌入至其他子条带，在保留 MDS 性质、高码率等优势的同时，克服了副本复制存储开销大及纠删码修复带宽开销过高的缺陷。目前，piggybacking 编码已被应用于 Hadoop 分布式文件系统。

Rashmi 等[7-8]提出了 3 种具体的 piggybacking 编码方案，即 RSR-I、RSR-II 和 RSR-III。其中，RSR-II 可将信息节点的修复带宽开销减少至 MDS 码的 50%，但其编解码复杂度高。Yuan 等[9]提出了保护(MDS only protected，MoP)子条带和捎带(piggybacking protected，PP)子条带组成的 piggybacking 编码，通过调节 MoP 子条带的比例，降低信息节点的修复带宽，但两者均未改善校验节点修复带宽高的问题。Li 等[10]、周悦等[11]提出了一个子条带对应一个校验节点的 piggybacking 编码(one-to-one piggybacking，OOP)构造，该构造可进一步降低信息节点的修复带宽开销，而且还能对校验节点进行快速修复，但其修复度大且子条带数目较多。张璐[12]、Sun 等[13-14]提出了双层 piggybacking 编码思想，当信息节点和校验节点的数目趋向无穷时，故障节点的平均修复带宽率趋于 0，但其构造的 piggybacking 编码修复度大且牺牲了一部分容错能力。Jiang 等[15]通过在 OOP 构造中引入 PP 子条带和 MoP 子条带，提出了一种新的 piggybacking 编码，改善了校验节点修复带宽开销高和 OOP 子条带多的问题。Shi 等[16]将 OOP 码中没有充分利用的校验节点重新进行 piggyback 函数捎带，进一步降低了修复带宽。但上述 piggybacking 编码对于多节点同时故障的情况没有相应的快速修复算法。

针对上述 piggybacking 编码存在的不足，本文提出了一种扩展子条带的 piggybacking 编码(substripe-added piggybacking，SAP)构造方案。SAP 在 MDS 码的基础上扩展子条带，将信息节点数据块分区均匀嵌入，校验节点数据块循环移位放置，旨在减少故障节点的修复带宽开销和修复度，探索一种具有较高修复效率的 piggybacking 编码，进而推动 piggybacking 编码在实际 DSSs 的应用。

1　piggybacking 框架

基于基础码，piggybacking 框架将部分子条带的数据块线性组合添加至其他子条带中，从而减少修复带宽开销。以系统 MDS 码作为基础码的 piggybacking 框架如图 1 所示。其中，k 为信息节点的数量，r 为校验节点数量，f 为子条带数。对基础码中的信息数据块 $a_{u,v}(1\leqslant u\leqslant k,1\leqslant v\leqslant f)$ 进行 MDS 码编码，生成校验数据块 $f_m(\boldsymbol{a}_v)(1\leqslant m\leqslant r,1\leqslant v\leqslant f)$，$\boldsymbol{a}_v=(a_{1,v},a_{2,v},\cdots,a_{k,v})^{\mathrm{T}}$ 表示第 v 个子条带中的 k 个信息数据块。子条带中的数据块线性组合形成 piggyback 函数 $g_{m,j}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_{j-1})(1\leqslant m\leqslant r,2\leqslant j\leqslant f)$，嵌入至其他子条带的校验数据块中组成 piggybacking 块 $f_m(\boldsymbol{a}_v)+g_{m,j}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_{j-1})$。需要注意的是，同一节点的不同数据块不能嵌入至同一个 piggyback 函数中，且校验节点的数据块不能嵌入至同一校验节点的 piggyback 函数中。

节点1	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	…	$a_{1,f}$
⋮	⋮	⋮	⋮	⋮	⋮
节点k	$a_{k,1}$	$a_{k,2}$	$a_{k,3}$	…	$a_{k,f}$
节点k+1	$f_1(\boldsymbol{a}_1)$	$f_1(\boldsymbol{a}_2)+g_{1,2}(\boldsymbol{a}_1)$	$f_1(\boldsymbol{a}_3)+g_{1,3}(\boldsymbol{a}_1,\boldsymbol{a}_2)$	…	$f_1(\boldsymbol{a}_f)+g_{1,f}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_{f-1})$
⋮	⋮	⋮	⋮	⋮	⋮
节点k+r	$f_r(\boldsymbol{a}_1)$	$f_r(\boldsymbol{a}_2)+g_{r,2}(\boldsymbol{a}_1)$	$f_r(\boldsymbol{a}_3)+g_{r,3}(\boldsymbol{a}_1,\boldsymbol{a}_2)$	…	$f_r(\boldsymbol{a}_f)+g_{r,f}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_{f-1})$

图 1　Piggybacking 框架

Fig. 1　Piggybacking framework

对于故障节点的修复：如果故障节点上的失效数据块没有作为 piggyback 函数嵌入至其他数据块，则可以利用 MDS 码译码恢复；如果失效数据块作为 piggyback 函数嵌入至其他数据块，则只需下载嵌入的 piggyback 函数中其他没有丢失的数据块和对应 piggybacking 块即可进行修复。piggybacking 编码的主要目的是通过减少 MDS 码译码的方式增加 piggybacking 块译码，从而减小故障节点的修复带宽开销。平均修复带宽率和平均修复度率可表示为：

$$\begin{cases}\gamma^{\mathrm{sys}}=\dfrac{\sum\limits_{i=1}^{k}\gamma_i}{kkf}\\ \eta^{\mathrm{sys}}=\dfrac{\sum\limits_{i=1}^{k}\eta_i}{kk}\end{cases}\tag{1}$$

$$\begin{cases}\gamma^{\mathrm{par}}=\dfrac{\sum\limits_{i=k+1}^{k+r}\gamma_i}{rkf}\\ \eta^{\mathrm{par}}=\dfrac{\sum\limits_{i=k+1}^{k+r}\eta_i}{rk}\end{cases}\tag{2}$$

$$\begin{cases}\gamma^{\text{all}}=\dfrac{k\gamma^{\text{sys}}+r\gamma^{\text{par}}}{k+r}\\ \eta^{\text{all}}=\dfrac{k\eta^{\text{sys}}+r\eta^{\text{par}}}{k+r}\end{cases}\tag{3}$$

式中：γ^{sys}、γ^{par}和 γ^{all}分别为信息节点、校验节点和节点总的平均修复带宽率；η^{sys}、η^{par}和 η^{all}分别为信息节点、校验节点和节点总的平均修复度率；γ_i($1\leqslant i\leqslant n, n=k+r$)为修复带宽，即节点 i 故障时，修复节点 i 需读取和下载的数据块数量；η_i($1\leqslant i\leqslant n, n=k+r$)为修复度，即节点 i 故障时，修复节点 i 需连接的节点数量。

2 扩展子条带的 piggybacking 编码构造

本节先介绍 SAP 框架结构和具体构造，然后阐述 $f=6$ 时 SAP 的构造实例。文中所有 SAP 构造的基础码均采用(n,k,f) MDS 码，且满足 $k\geqslant r\lfloor f/2\rfloor$。

2.1 SAP 框架结构

图 2 为 $k=r\lfloor f/2\rfloor$时 SAP 框架结构，分为 6 个区域，即 A ~ F。令 $\tau=\lceil f/2\rceil$，$\tau_1=\lfloor f/2\rfloor$。子条带 $s_i(1\leqslant i\leqslant f+1)$为第 i 个子条带，s_{f+1}为扩展子条带，主要由区域 C、F 组成。将区域 E 中的校验数据块从上至下、从左至右复制，置于扩展子条带的前 k 行，形成区域 C，确保区域 E 中的校验节点发生故障时能进行快速修复。区域 A 中的信息数据块线性组合形成 piggyback 函数 $g_{m,j}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau)$，嵌入至区域 E 中形成 piggybacking 块 $f_m(\boldsymbol{a}_v)+g_{m,j}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau)$，嵌入至扩展子条带的后 r 行形成区域 F，保证区域 A 中丢失数据块的快速修复，减少相应数据块的修复带宽开销。将第 $\tau-1$ 个子条带的校验数据块循环移位置于第 τ 个子条带，第 $\tau-3$ 个子条带的校验数据块循环移位置于第 $\tau-2$ 个子条带，以此类推，完成前 τ 个子条带的嵌入，形成区域 D。区域 B 则不进行任何操作，以保证该区域发生节点故障时，丢失数据块采用 MDS 码译码恢复。

子条带	1	…	τ-1	$\tau=\left\lceil\frac{f}{2}\right\rceil$	τ+1	…	f	f+1
节点1	$a_{1,1}$	…	$a_{1,\tau-1}$	$a_{1,\tau}$	$a_{1,\tau+1}$	…	$a_{1,f}$	$f_1(\boldsymbol{a}_{\tau+1})$
⋮	⋮	⋮	A ⋮	⋮	⋮	B ⋮	⋮	C ⋮
节点k	$a_{k,1}$	…	$a_{k,\tau-1}$	$a_{k,\tau}$	$a_{k,\tau+1}$	…	$a_{k,f}$	$f_r(\boldsymbol{a}_f)$
节点k+1	$f_1(\boldsymbol{a}_1)$	…	$f_1(\boldsymbol{a}_{\tau-1})$	$f_1(\boldsymbol{a}_\tau)+f_r(\boldsymbol{a}_{\tau-1})$	$f_1(\boldsymbol{a}_{\tau+1})+g_{1,\tau+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$	…	$f_1(\boldsymbol{a}_f)+g_{1,f}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$	$g_{1,f+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
节点k+r-1	$f_{r-1}(\boldsymbol{a}_1)$	…	D $f_{r-1}(\boldsymbol{a}_{\tau-1})$	$f_{r-1}(\boldsymbol{a}_\tau)+f_r(\boldsymbol{a}_{\tau-1})$	$f_r(\boldsymbol{a}_{\tau+1})+g_{r-1,\tau+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$	E …	$f_{r-1}(\boldsymbol{a}_f)+g_{r-1,f}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$	F $g_{r-1,f+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$
节点k+r	$f_r(\boldsymbol{a}_1)$	…	$f_r(\boldsymbol{a}_{\tau-1})$	$f_r(\boldsymbol{a}_\tau)+f_1(\boldsymbol{a}_{\tau-1})$	$f_r(\boldsymbol{a}_{\tau+1})+g_{r,\tau+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$	…	$f_r(\boldsymbol{a}_f)+g_{r,f}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$	$g_{r,f+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\ldots,\boldsymbol{a}_\tau)$

图 2 SAP 框架结构

Fig. 2 SAP framework structure

2.2 SAP 具体构造

为了便于理解和表达，将 $k\geqslant r\lfloor f/2\rfloor$时 SAP 的具体构造同样拆分为 6 个区域 A ~ F。

区域 A、B 包含所有原始信息数据块，而且没有进行任何嵌入操作。

区域 C 进行嵌入操作，具体过程如下：

将子条带 $s_i(\tau+1\leqslant i\leqslant f)$的校验数据块复制，然后置于扩展子条带 s_{f+1}的前 k 行中($k\geqslant r\tau_1$)，形成区域 C。当 $k=r\tau_1$ 时，校验数据正好全部嵌入；当 $k>r\tau_1$ 时，扩展子条带 s_{f+1}的前 k 行存在空位，以 0 填充。依据嵌入过程，区域 C 中包含的 k 个数据块从上至下依次为$(f_1(\boldsymbol{a}_{\tau+1}),\cdots,f_r(\boldsymbol{a}_{\tau+1}),\cdots,f_1(\boldsymbol{a}_f),\cdots,f_r(\boldsymbol{a}_f),0,\cdots,0)^{\mathrm{T}}$。

区域 D 的具体构造过程如下：

1)将子条带 $s_{\tau-1}$的第 $i(2\leqslant i\leqslant r)$个校验数据块嵌入至子条带 s_τ 的第 $i-1$ 个校验数据块，子条带 $s_{\tau-1}$第 1 个校验数据块嵌入至子条带 s_τ 的第 r 个校验数据块。同样的方法，完成子条带 $s_{\tau-3}$对子条带

$s_{\tau-2}$的嵌入。

2）如果 τ 为偶数，前 τ 个子条带按照 1）中的方式完成嵌入；如果 τ 为奇数，第 1 个子条带不进行操作，剩余 $\tau-1$ 个子条带按照 1）中的方式完成嵌入。区域 D 的通式表达式为

$$\mathrm{D}=\begin{Bmatrix} f_1(\boldsymbol{a}_1) & \cdots & f_1(\boldsymbol{a}_{\tau-3}) & f_1(\boldsymbol{a}_{\tau-2})+f_2(\boldsymbol{a}_{\tau-3}) & f_1(\boldsymbol{a}_{\tau-1}) & f_1(\boldsymbol{a}_{\tau})+f_2(\boldsymbol{a}_{\tau-1}) \\ f_2(\boldsymbol{a}_1) & \cdots & f_2(\boldsymbol{a}_{\tau-3}) & f_2(\boldsymbol{a}_{\tau-2})+f_3(\boldsymbol{a}_{\tau-3}) & f_2(\boldsymbol{a}_{\tau-1}) & f_2(\boldsymbol{a}_{\tau})+f_3(\boldsymbol{a}_{\tau-1}) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ f_r(\boldsymbol{a}_1) & \cdots & f_r(\boldsymbol{a}_{\tau-3}) & f_r(\boldsymbol{a}_{\tau-2})+f_1(\boldsymbol{a}_{\tau-3}) & f_r(\boldsymbol{a}_{\tau-1}) & f_r(\boldsymbol{a}_{\tau})+f_1(\boldsymbol{a}_{\tau-1}) \end{Bmatrix} \tag{4}$$

将区域 A 中的信息数据块均匀地嵌入至区域 E、F。嵌入过程需满足同一节点的信息数据块不能嵌入至同一 piggyback 函数中的嵌入规则。嵌入完成后的区域 E、F 称为信息嵌入区域。具体嵌入过程如下：

1）令 $\beta_{\tau(i-1)+j}=a_{i,j}(1\leqslant i\leqslant k,1\leqslant j\leqslant\tau)$，将$\{\beta_1,\cdots,\beta_{k\tau}\}$按照从左至右、从上至下的顺序嵌入至子条带 $s_i(\tau+1\leqslant i\leqslant f+1)$的后 r 行组成的区域中。如果该区域嵌入已满，余下的数据从第 1 个位置重新嵌入，直至区域 A 中的信息数据块全部嵌入，每个位置的所有嵌入数据块之和为一个 piggyback 函数 $g_{m,j}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau)$。

2）令 $t_{l1}=\lfloor k\tau/r(\tau_1+1)\rfloor$，$t_{h1}=\lceil k\tau/r(\tau_1+1)\rceil$，$t_1=k\tau-rt_{l1}(\tau_1+1)$。依据嵌入规则，当 $t_{h1}\neq t_{l1}$时，可以确定信息嵌入区域 piggyback 函数中包含 t_{h1}个数据块的 piggyback 函数有 t_1 个。所以，按照从左至右、从上至下的顺序，第 t_1+1 个位置开始 piggyback 函数 $g_{m,j}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau)=\sum_{i=0}^{t_{l1}-1}\beta_{ir(\tau_1+1)+t_{l1}+1}$，信息嵌入区域的通式表达式为

$$\{\mathrm{E}\mid\mathrm{F}\}=\left\{\begin{array}{cccc|c} f_1(\boldsymbol{a}_{\tau+1})+g_{1,\tau+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & f_1(\boldsymbol{a}_{\tau+2})+g_{1,\tau+2}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & \cdots & f_1(\boldsymbol{a}_f)+g_{1,f}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & g_{1,f+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) \\ f_2(\boldsymbol{a}_{\tau+1})+g_{2,\tau+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & f_2(\boldsymbol{a}_{\tau+2})+g_{2,\tau+2}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & \cdots & f_1(\boldsymbol{a}_f)+g_{2,f}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & g_{2,f+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_r(\boldsymbol{a}_{\tau+1})+g_{r,\tau+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & f_r(\boldsymbol{a}_{\tau+2})+g_{r,\tau+2}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & \cdots & f_1(\boldsymbol{a}_f)+g_{r,f}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) & g_{r,f+1}(\boldsymbol{a}_1,\boldsymbol{a}_2,\cdots,\boldsymbol{a}_\tau) \end{array}\right\}=$$

$$\left\{\begin{array}{cccc|c} f_1(\boldsymbol{a}_{\tau+1})+\sum_{i=0}^{t_{h1}-1}\beta_{ir(\tau_1+1)+1} & f_1(\boldsymbol{a}_{\tau+2})+\sum_{i=0}^{t_{h1}-1}\beta_{ir(\tau_1+1)+2} & \cdots & f_1(\boldsymbol{a}_f)+\sum_{i=0}^{t_{h1}-1}\beta_{ir(\tau_1+1)+\tau_1} & \sum_{i=0}^{t_{h1}-1}\beta_{ir(\tau_1+1)+\tau_1+1} \\ f_2(\boldsymbol{a}_{\tau+1})+\sum_{i=0}^{t_{h1}-1}\beta_{ir(\tau_1+1)+\tau_1+2} & f_2(\boldsymbol{a}_{\tau+2})+\sum_{i=0}^{t_{h1}-1}\beta_{ir(\tau_1+1)+\tau_1+3} & \cdots & f_1(\boldsymbol{a}_f)+\sum_{i=0}^{t_{h1}-1}\beta_{ir(\tau_1+1)+2\tau_1+1} & \sum_{i=0}^{t_{h1}-1}\beta_{ir(\tau_1+1)+2(\tau_1+1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_r(\boldsymbol{a}_{\tau+1})+\sum_{i=0}^{t_{l1}-1}\beta_{ir(\tau_1+1)+(r-1)(\tau_1+1)+1} & f_r(\boldsymbol{a}_{\tau+2})+\sum_{i=0}^{t_{l1}-1}\beta_{ir(\tau_1+1)+(r-1)(\tau_1+1)+2} & \cdots & f_1(\boldsymbol{a}_f)+\sum_{i=0}^{t_{l1}-1}\beta_{ir(\tau_1+1)+r\tau_1+r-1} & \sum_{i=0}^{t_{l1}-1}\beta_{ir(\tau_1+1)+r(\tau_1+1)} \end{array}\right\} \tag{5}$$

定理　本文构造的 SAP 满足(n,k)性质，即任意$n-k$个节点故障，均能从剩余的 k 个节点恢复原始数据。

证明　依据具体构造过程，子条带 s_1 没有进行嵌入操作，保持 MDS 结构，所以子条带 s_1 满足(n,k)性质。τ 为偶数时，子条带 s_2 中捎带的数据块为子条带 s_1 中的校验数据块，而子条带 s_1 的数据块均能恢复，所以进行一次异或操作，子条带 s_2 同样能恢复与 s_1 相同的结构，即子条带 s_2 满足(n,k)性质。以此类推，其他子条带也均满足(n,k)性质。而扩展子条带中的数据是校验数据的复制，在前面子条带恢复中均可得到，即对 SAP 整体(n,k)性质不影响。综上所述，SAP 满足(n,k)性质。

2.3　构造实例

本节介绍$(n,k,f)=(16,12,6)$SAP 的构造过程，其中子条带 s_7 为扩展子条带。首先，将区域 A 中的信息数据块 $a_{u,v}(1\leqslant u\leqslant 12,1\leqslant v\leqslant 3)$按照从左至右、从上至下的顺序复制，形成 piggyback 函数；其次，采用同样的方式嵌入至区域 E 的校验数据块 $f_m(\boldsymbol{a}_v)(1\leqslant m\leqslant 4,4\leqslant v\leqslant 6)$和扩展子条带 s_7 的后 r 行区域 F 中；最后，将区域 E 中的校验数据块$f_m(\boldsymbol{a}_v)$ $(1\leqslant m\leqslant 4,4\leqslant v\leqslant 6)$复制置于扩展子条带 s_7 的前 k 行区域 C 中，将区域 D 中的校验数据块进行循环移位嵌入，即将 $f_m(\boldsymbol{a}_2)(1\leqslant m\leqslant 4)$等数据块错位嵌入至 $f_m(\boldsymbol{a}_3)(1\leqslant m\leqslant 4)$等数据块中。$(n,k,f)=(16,12,6)$SAP 的编码过程完成，编码结果如图 3 所示，区域 E、F 中 piggyback 函数具体展示见式(6)。

子条带	1	2	3	4	5	6	7
节点1	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$f_1(\boldsymbol{a}_4)$
节点2	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$f_2(\boldsymbol{a}_4)$
节点3	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$f_3(\boldsymbol{a}_4)$
节点4	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{4,5}$	$a_{4,6}$	$f_4(\boldsymbol{a}_4)$
节点5	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	$a_{5,5}$	$a_{5,6}$	$f_1(\boldsymbol{a}_5)$
节点6	$a_{6,1}$	A $a_{6,2}$	$a_{6,3}$	$a_{6,4}$	B $a_{6,5}$	$a_{6,6}$	C $f_2(\boldsymbol{a}_5)$
节点7	$a_{7,1}$	$a_{7,2}$	$a_{7,3}$	$a_{7,4}$	$a_{7,5}$	$a_{7,6}$	$f_3(\boldsymbol{a}_5)$
节点8	$a_{8,1}$	$a_{8,2}$	$a_{8,3}$	$a_{8,4}$	$a_{8,5}$	$a_{8,6}$	$f_4(\boldsymbol{a}_5)$
节点9	$a_{9,1}$	$a_{9,2}$	$a_{9,3}$	$a_{9,4}$	$a_{9,5}$	$a_{9,6}$	$f_1(\boldsymbol{a}_6)$
节点10	$a_{10,1}$	$a_{10,2}$	$a_{10,3}$	$a_{10,4}$	$a_{10,5}$	$a_{10,6}$	$f_2(\boldsymbol{a}_6)$
节点11	$a_{11,1}$	$a_{11,2}$	$a_{11,3}$	$a_{11,4}$	$a_{11,5}$	$a_{11,6}$	$f_3(\boldsymbol{a}_6)$
节点12	$a_{12,1}$	$a_{12,2}$	$a_{12,3}$	$a_{12,4}$	$a_{12,5}$	$a_{12,6}$	$f_4(\boldsymbol{a}_6)$
节点13	$f_1(\boldsymbol{a}_1)$	$f_1(\boldsymbol{a}_2)$	$f_1(\boldsymbol{a}_3)+f_2(\boldsymbol{a}_2)$	$f_1(\boldsymbol{a}_4)+\sum_{i=0}^{2}\beta_{16i+1}$	$f_1(\boldsymbol{a}_5)+\sum_{i=0}^{2}\beta_{16i+2}$	$f_1(\boldsymbol{a}_6)+\sum_{i=0}^{2}\beta_{16i+3}$	$\sum_{i=0}^{2}\beta_{16i+4}$
节点14	$f_2(\boldsymbol{a}_1)$	$f_2(\boldsymbol{a}_2)$ D	$f_2(\boldsymbol{a}_3)+f_3(\boldsymbol{a}_2)$	$f_2(\boldsymbol{a}_4)+\sum_{i=0}^{1}\beta_{16i+5}$	E $f_2(\boldsymbol{a}_5)+\sum_{i=0}^{1}\beta_{16i+6}$	$f_2(\boldsymbol{a}_6)+\sum_{i=0}^{1}\beta_{16i+7}$	$\sum_{i=0}^{1}\beta_{16i+8}$ F
节点15	$f_3(\boldsymbol{a}_1)$	$f_3(\boldsymbol{a}_2)$	$f_3(\boldsymbol{a}_3)+f_4(\boldsymbol{a}_2)$	$f_3(\boldsymbol{a}_4)+\sum_{i=0}^{1}\beta_{16i+9}$	$f_3(\boldsymbol{a}_5)+\sum_{i=0}^{1}\beta_{16i+10}$	$f_3(\boldsymbol{a}_6)+\sum_{i=0}^{1}\beta_{16i+11}$	$\sum_{i=0}^{1}\beta_{16i+12}$
节点16	$f_4(\boldsymbol{a}_1)$	$f_4(\boldsymbol{a}_2)$	$f_4(\boldsymbol{a}_3)+f_1(\boldsymbol{a}_2)$	$f_4(\boldsymbol{a}_4)+\sum_{i=0}^{1}\beta_{16i+13}$	$f_4(\boldsymbol{a}_5)+\sum_{i=0}^{1}\beta_{16i+14}$	$f_4(\boldsymbol{a}_6)+\sum_{i=0}^{1}\beta_{16i+15}$	$\sum_{i=0}^{1}\beta_{16i+16}$

图 3 $(n,k,f)=(16,12,6)$ SAP 编码

Fig. 3 $(n,k,f)=(16,12,6)$ SAP coding

$$\{\mathrm{E}\mid\mathrm{F}\}=\left\{\begin{array}{ccc|c} f_1(\boldsymbol{a}_4)+\sum_{i=0}^{2}\beta_{16i+1} & f_1(\boldsymbol{a}_5)+\sum_{i=0}^{2}\beta_{16i+2} & f_1(\boldsymbol{a}_6)+\sum_{i=0}^{2}\beta_{16i+3} & \sum_{i=0}^{2}\beta_{16i+4} \\ f_2(\boldsymbol{a}_4)+\sum_{i=0}^{1}\beta_{16i+5} & f_2(\boldsymbol{a}_5)+\sum_{i=0}^{1}\beta_{16i+6} & f_2(\boldsymbol{a}_6)+\sum_{i=0}^{1}\beta_{16i+7} & \sum_{i=0}^{1}\beta_{16i+8} \\ f_3(\boldsymbol{a}_4)+\sum_{i=0}^{1}\beta_{16i+9} & f_3(\boldsymbol{a}_5)+\sum_{i=0}^{1}\beta_{16i+10} & f_3(\boldsymbol{a}_6)+\sum_{i=0}^{1}\beta_{16i+11} & \sum_{i=0}^{1}\beta_{16i+12} \\ f_4(\boldsymbol{a}_4)+\sum_{i=0}^{1}\beta_{16i+13} & f_4(\boldsymbol{a}_5)+\sum_{i=0}^{1}\beta_{16i+14} & f_4(\boldsymbol{a}_6)+\sum_{i=0}^{1}\beta_{16i+15} & \sum_{i=0}^{1}\beta_{16i+16} \end{array}\right\}=$$

$$\left\{\begin{array}{ccc|c} f_1(\boldsymbol{a}_4)+a_{1,1}+a_{6,2}+a_{11,3} & f_1(\boldsymbol{a}_5)+a_{1,2}+a_{6,3}+a_{12,1} & f_1(\boldsymbol{a}_6)+a_{1,3}+a_{7,1}+a_{12,2} & a_{2,1}+a_{7,2}+a_{12,3} \\ f_2(\boldsymbol{a}_4)+a_{2,2}+a_{7,3} & f_2(\boldsymbol{a}_5)+a_{2,3}+a_{8,1} & f_2(\boldsymbol{a}_6)+a_{3,1}+a_{8,2} & a_{3,2}+a_{8,3} \\ f_3(\boldsymbol{a}_4)+a_{3,3}+a_{9,1} & f_3(\boldsymbol{a}_5)+a_{4,1}+a_{9,2} & f_3(\boldsymbol{a}_6)+a_{4,2}+a_{9,3} & a_{4,3}+a_{10,1} \\ f_4(\boldsymbol{a}_4)+a_{5,1}+a_{10,2} & f_4(\boldsymbol{a}_5)+a_{5,2}+a_{10,3} & f_4(\boldsymbol{a}_6)+a_{5,3}+a_{11,1} & a_{6,1}+a_{11,2} \end{array}\right\} \tag{6}$$

由图 3 和式(6)可以看出,区域 A 中丢失的信息数据块利用区域 E 和 F 中的 piggybacking 块进行修复,区域 B 中丢失的信息数据块则利用 MDS 码译码修复,区域 C 中丢失的校验数据块在进行区域 B 中丢失信息数据块修复的同时重新编码修复。假设节点 1 发生故障,$a_{1,4}$ 通过下载 $a_{2,4},\cdots,a_{12,4}$ 和 $f_2(\boldsymbol{a}_4)$ 利用 MDS 码译码修复。$a_{1,5}$、$a_{1,6}$ 与 $a_{1,4}$ 的修复方式相同。$a_{1,1}$、$a_{1,2}$ 和 $a_{1,3}$ 分别被嵌入至子条带 s_4、s_5 和 s_6 的第 1 个校验数据块上,下载相应的 piggybacking 块和其中未丢失的数据块进行恢复。扩展子条带丢失的 $f_1(\boldsymbol{a}_4)$ 利用已经恢复的 $a_{1,4}$ 和恢复 $a_{1,4}$ 时下载的 $a_{2,4},\cdots,a_{12,4}$ 重新编码修复。节点 1 修复完成,共计下载 45 个数据块,连接 12 个节点。

区域 D 校验节点中丢失的嵌入数据块可利用 piggybacking 块修复,没有嵌入的丢失校验数据块则进行 MDS 码重新编码修复。区域 E 和 F 中的丢失数据块直接在区域 A 和 C 中下载,即可完成修复。假设节点 13 发生故障,利用 MDS 码编码得到数据

块 $f_1(\boldsymbol{a}_1)$、$f_1(\boldsymbol{a}_3)$ 和 $f_4(\boldsymbol{a}_3)$。然后，在节点 14 下载数据块 $f_2(\boldsymbol{a}_2)$ 并与数据块 $f_1(\boldsymbol{a}_3)$ 进行异或操作，以恢复丢失数据块 $f_1(\boldsymbol{a}_3)+f_2(\boldsymbol{a}_2)$。同理，通过在节点 16 下载数据块 $f_4(\boldsymbol{a}_3)+f_1(\boldsymbol{a}_2)$ 并与数据块 $f_4(\boldsymbol{a}_3)$ 进行异或操作，以恢复 $f_1(\boldsymbol{a}_2)$。其他丢失数据块在区域 A 和 C 中下载原始数据块进行异或操作以恢复。完成节点 13 的修复，共计下载 33 个数据块，连接 14 个节点。其他单节点故障的修复与上述描述相似。最后，依据式(1) ~ (3)得出 $\gamma^{\mathrm{sys}}=0.597$、$\gamma^{\mathrm{par}}=0.444$、$\gamma^{\mathrm{all}}=0.559$、$\eta^{\mathrm{sys}}=1.042$、$\eta^{\mathrm{par}}=1.167$、$\eta^{\mathrm{all}}=1.073$。

若图 3 中有多个节点同时发生故障，假设其中多个故障节点中包含一个或多个信息节点时，均需利用 MDS 码译码恢复故障节点信息，没有其他快速修复算法，共计需下载 72 个数据块。当多个故障节点均为校验节点时，可利用快速修复方法，只需要下载子条带 $s_i(1\leqslant i\leqslant 3)$ 中所有信息数据块和在扩展子条带 s_7 中下载失效的校验数据块即可完成修复。例如，当两个校验节点 13 和 14 同时发生故障，下载 42 个数据块即可完成修复。

3　修复分析

3.1　信息节点平均修复带宽率和平均修复度率

对于单信息节点故障，丢失的数据块存在 3 种不同情况，分别对应数据块位于区域 A、B 和 C。由上述 SAP 框架结构及构造实例中信息节点修复过程分析可知：区域 A 中丢失数据块可利用信息嵌入区域的 piggybacking 块进行修复，k 个信息节点共计需下载 $t_1t_{h1}^2+(r(\tau_1+1)-t_1)t_{l1}^2$ 个数据块；区域 B 中丢失的数据块可利用 MDS 码译码修复，需下载 $k\tau_1$ 个数据块；区域 C 中丢失的数据块，在区域 B 进行 MDS 码译码的同时可进行重新编码修复。所以，信息节点平均修复带宽率为

$$\gamma^{\mathrm{sys}}=\frac{k^2\tau_1+t_1t_{h1}^2+(r(\tau_1+1)-t_1)t_{l1}^2}{k^2f} \tag{7}$$

推论 1　对于单节点故障，当 $\tau=\tau_1$ 时，信息节点最小平均修复带宽率 $\min\gamma^{\mathrm{sys}}_{\tau=\tau_1}=1/2+1/(4r)$；当 $\tau\neq\tau_1$ 时，信息节点最小平均修复带宽率 $\min\gamma^{\mathrm{sys}}_{\tau\neq\tau_1}=(r+2)/(3r)$。

证明　信息节点平均修复带宽率 $\gamma^{\mathrm{sys}}=\dfrac{k^2\tau_1+t_1t_{h1}^2+(r(\tau_1+1)-t_1)t_{l1}^2}{k^2f}$，当 $\tau=\tau_1$，即 $f=2\tau_1$ 时，

$$\gamma^{\mathrm{sys}}_{\tau=\tau_1}=\frac{k^2\tau_1+t_1t_{h1}^2+(r(\tau_1+1)-t_1)t_{l1}^2}{k^2\times 2\tau_1}\geqslant$$

$$\frac{k^2\tau_1+(k\tau_1)^2/(r(\tau_1+1))}{k^2\times 2\tau_1}=1/2+\tau_1/(2r(\tau_1+1)) \tag{8}$$

对 $1/2+\tau_1/(2r(\tau_1+1))$ 中 τ_1 求偏导可知，$1/2+\tau_1/(2r(\tau_1+1))$ 随 τ_1 的减小而减小，所以当 $\tau_1=1$ 时，$\min\gamma^{\mathrm{sys}}_{\tau=\tau_1}=1/2+1/(4r)$。

同理，当 $\tau\neq\tau_1$ 时，$\min\gamma^{\mathrm{sys}}_{\tau\neq\tau_1}=(r+2)/(3r)$。

修复故障节点区域 B、C 中丢失的数据块共计需连接 $k-1$ 个节点。对于修复区域 A 中丢失的数据块，如果区域 A 中同一行丢失的数据块嵌入至信息嵌入区域的同一行时，需要额外连接 1 个节点；如果不在同一行时，则需要额外连接 2 个节点。所以当 $\tau\neq\tau_1$ 即 $\tau=\tau_1+1$ 时，$\eta^{\mathrm{sys}}_{\tau\neq\tau_1}=1$；当 $\tau=\tau_1$ 时，令 $t_{l3}=\left\lfloor\dfrac{k}{\tau_1+1}\right\rfloor$，$t_3=k-t_{l3}(\tau_1+1)$。当 $t_3=0$ 时，修复区域 A 中 k 个信息节点共需额外连接 $2\tau_1t_{l3}$ 个节点；当 $t_3\neq 0$ 时，修复区域 A 中 k 个信息节点共需额外连接 $2\tau_1t_{l3}+2t_3-1$ 个节点。所以，信息节点平均修复度率为

$$\eta^{\mathrm{sys}}=\begin{cases}\dfrac{k(k-1)+2\tau_1t_{l3}}{k^2}, & \tau=\tau_1\ \text{且}\ t_3=0\\ \dfrac{k(k-1)+2\tau_1t_{l3}+2t_3-1}{k^2}, & \tau=\tau_1\ \text{且}\ t_3\neq 0\\ 1, & \tau\neq\tau_1\end{cases} \tag{9}$$

3.2　校验节点平均修复带宽率和平均修复度率

对于单校验节点故障，丢失的数据块也存在 3 种不同情况，分别对应数据块位于区域 D、E 和 F。区域 E 中丢失的校验数据块，直接在区域 C 中下载即可修复，r 个校验节点共计需下载 $r\tau_1$ 个数据块；当 τ 为奇数，下载奇数子条带的所有信息数据块和嵌入的校验数据块即可修复区域 D 中丢失的数据块，r 个校验节点共计需下载 $r(k(\tau+1)/2+\tau-1)$ 个数据块。下载区域 A 中尚未下载的信息数据块即可修复区域 E 和 F 中丢失的数据块，r 个校验节点共计需下载 $k(\tau-1)/2$ 个数据块；当 τ 为偶数，修复方式类似。所以，校验节点平均修复带宽率为

$$\gamma^{\mathrm{par}}=\begin{cases}\dfrac{r\left(k\left(\dfrac{\tau+1}{2}\right)+\tau-1+\tau_1\right)+\dfrac{k(\tau-1)}{2}}{rkf}, & \tau\ \text{为奇数}\\ \dfrac{r\left(k\dfrac{\tau}{2}+\tau+\tau_1\right)+\dfrac{k\tau}{2}}{rkf}, & \tau\ \text{为偶数}\end{cases} \tag{10}$$

推论 2　单节点故障时，校验节点最小平均修复带宽率 $\min\gamma^{\mathrm{par}}=1/4+1/(4r)$。

证明 τ 为奇数时，校验节点平均修复带宽率 $\gamma^{\mathrm{par}}=\dfrac{r\left(k\left(\dfrac{\tau+1}{2}\right)+\tau-1+\tau_1\right)+\dfrac{k(\tau-1)}{2}}{rkf}$。当 $\tau=\tau_1$ 即 $f=2\tau_1$ 时，有

$$\frac{\partial(\gamma^{\mathrm{par}})}{\partial\tau_1}=\frac{kkr(1-r)}{(rk\times 2\tau_1)^2}<0 \tag{11}$$

所以，γ^{par} 随着 τ_1 的增大而减小。SAP 构造是以 $k\geqslant r\tau_1$ 为前提的，所以，当 τ_1 趋向无穷时，k 也应趋向无穷，有 $\min\ \gamma^{\mathrm{par}}=\lim\limits_{\tau_1\to\infty,k\to\infty}\gamma^{\mathrm{par}}=1/4+1/(4r)$。其他情况证明类似，不再阐述。

同时，由上述校验节点故障修复过程可推出校验节点平均修复度率 $\eta^{\mathrm{par}}=(k+2)/k$。

假设 $t(2\leqslant t\leqslant r)$ 个校验节点同时故障，子条带 $s_i(1\leqslant i\leqslant\tau)$ 中丢失的校验数据块需利用 MDS 码重新编码修复，共需下载 $k\tau$ 个数据块。而子条带 $s_i(\tau+1\leqslant i\leqslant f)$ 丢失的校验数据块，在扩展子条带 s_{f+1} 中直接下载即可，共计需下载 $t\tau_1$ 个校验数据块。其他嵌入数据块，在进行前两步修复的同时已经进行下载。所以，t 个校验节点同时故障时，校验节点的平均修复带宽率 $\gamma_t^{\mathrm{par}}=(k\tau+t\tau_1)/(kf)$。

4 性能分析

为了探究 SAP 的性能，将 SAP 与现有 piggybacking 编码 RSR-I、RSR-II 和 OOP 分别从存储开销、修复带宽开销和修复度率 3 个维度进行对比。

4.1 存储开销

在 piggybacking 编码 RSR-I、RSR-II 和 OOP 中，系统会将大小为 M 的原始文件分割成 k 份，每份长度为 f，此时节点存储开销 $\alpha_1=M/k$。本文构造的 SAP 扩展了 1 个子条带，所以每份长度为 $f+1$，则节点存储开销 $\alpha_2=(f+1)M/(kf)$。因此，与 piggybacking 编码 RSR-I、RSR-II 和 OOP 相比，本文构造的 SAP 节点存储开销增加率为 $1/f$。图 4 为节点存储开销增加率随子条带数 f 的变化曲线。由图 4 可知，节点存储开销增加率随 f 的变化呈指数递减，当 f 趋向无穷大时，节点存储开销的增加可以忽略不计。

4.2 修复带宽开销

从信息节点平均修复带宽率 γ^{sys}、校验节点平均修复带宽率 γ^{par}、节点总的平均修复带宽率 γ^{all} 以及多校验节点故障修复带宽率 γ_t^{par} 方面，将本文构造的 SAP 与 RSR-I、RSR-II 和 OOP 进行比较。

图 5 为修复带宽率对比曲线。其中，RSR-I、OOP 的子条带数 $f=6$，RSR-II 的子条带数 $f=2(2r-3)$。由于本文构造的 SAP 分 f 为奇数和偶数两种情况，同时，为了保证对比的准确性，SAP 子条带数的选择与对比编码相近，分别取 $f=6$ 和 $f=7$。由图 5(a) 可以看出，SAP 的 γ^{sys} 曲线整体位于其他 piggybacking 编码的下方。当 $f=7$ 时，无论编码参数取何值，γ^{sys} 均小于对比的 piggybacking 编码。但当 $f=6$ 时，在 $n=20$、$k=15$ 等参数下，会出现 SAP 的 γ^{sys} 大于 OOP 或 RSR-II 的情况，主要原因是当 $r=5$ 时，$\sqrt{r-1}$ 为整数，达到 OOP 信息节点平均修复带宽率的最优条件。并且，随着 r 的增大，RSR-II 子条带数增大，使其对校验节点的嵌入更加高效，从而导致上述情况出现。由图 5(b) 可以明显看出，SAP 的 γ^{par} 优于其他 piggybacking 编码，说明 SAP 达到了减少校验节点修复带宽开销的目的。由图 5(c) 看出，γ^{all} 也优于其他 piggybacking 编码，说明 SAP 具有较低的修复带宽开销。

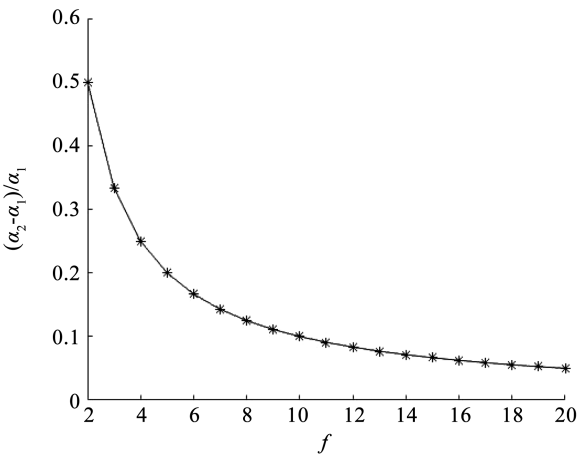
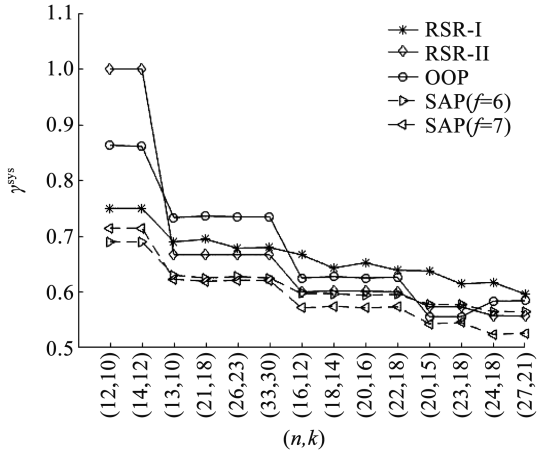


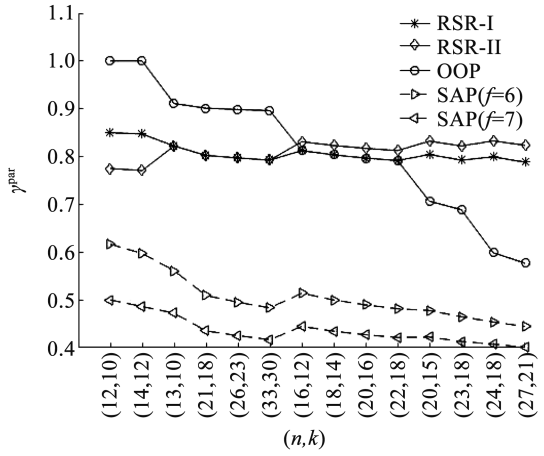
图 4 存储开销增加率随 f 的变化曲线

Fig. 4 Curve of storage overhead increase rate with f

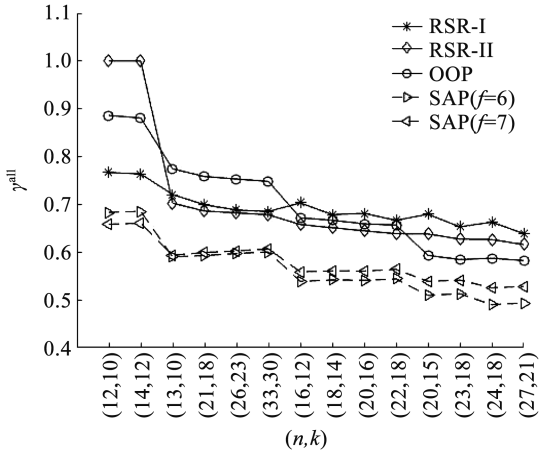
图 6 为最小平均修复带宽率随校验节点数 r 的变化曲线。可以看出，piggybacking 编码的最小平均修复带宽率均随 r 的增大而减小。这是因为校验节点 r 越大，piggyback 函数中包含的数据块越少，修复故障节点时下载的数据块就越少。由图 6(a) 可以发现：在 r 的取值范围内，当 $\tau\neq\tau_1$ 时，SAP 的 $\min\ \gamma^{\mathrm{sys}}$ 曲线一直处于其他 piggybacking 编码曲线的下方；但当 $\tau=\tau_1$ 时，从 $r=5$ 开始，SAP 的 $\min\ \gamma^{\mathrm{sys}}$ 大于 OOP 码。这是因为从 $r=5$ 开始，$\sqrt{r-1}/(r-1)\leqslant 1/2$，即 OOP 的 MDS 码译码的子条带数少于总子条带数的一半，当 $\tau=\tau_1$ 时，SAP 需要进行的 MDS 码译码的子条带数正好等于总子条带数的一半，即从 $r=5$ 开始出现 SAP 的 $\min\ \gamma^{\mathrm{sys}}$ 大于 OOP 的情况。图 6(b) 中，SAP 的 $\min\ \gamma^{\mathrm{par}}$ 一直低于其他 piggybacking 编码的 $\min\ \gamma^{\mathrm{par}}$。



(a) 信息节点



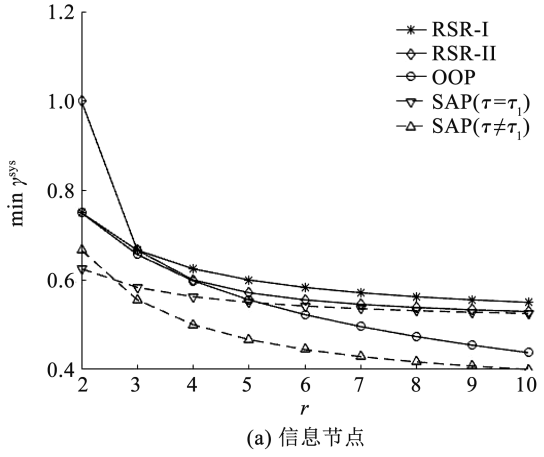
(b) 检验节点



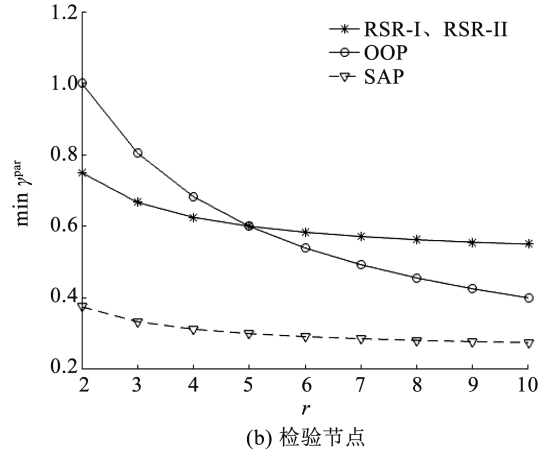
(c) 总节点

图 5　修复带宽率对比曲线

Fig. 5　Comparison curve of repair bandwidth rate



(a) 信息节点



(b) 检验节点

图 6　最小平均修复带宽率对比曲线

Fig. 6　Comparison curve of minimum average repair bandwidth rate

$t(2\leqslant t\leqslant r)$表示 SAP 中校验节点同时发生故障的个数。piggybacking 编码 RSR-I、RSR-II 和 OOP 均通过下载所有信息数据块重新编码进行恢复，修复带宽率为 1[10]，SAP 只需下载子条带 $s_i(1\leqslant i\leqslant\tau)$的所有信息数据块进行重新编码修复，其他丢失校验数据块可在扩展子条带 s_{f+1}中直接下载即可完成快速修复。图 7 为不同 piggybacking 编码在 $f=7$ 时，多校验节点故障的修复带宽率对比曲线。可以看出：SAP 中多校验节点故障的修复带宽率随着故障节点数目 t 的增大而增大，随着信息节点数目 k 的

增大而减小；相比于其他的piggybacking编码，SAP在多校验节点故障时显著地降低了修复带宽率，以$t=2$时降低的修复带宽开销最大。

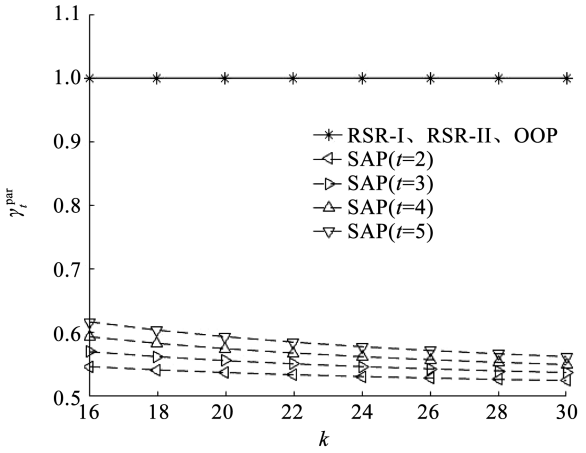


图 7　多校验节点故障修复带宽率对比曲线

Fig. 7　Comparison curve of repair bandwidth rates for multiple parity node failures

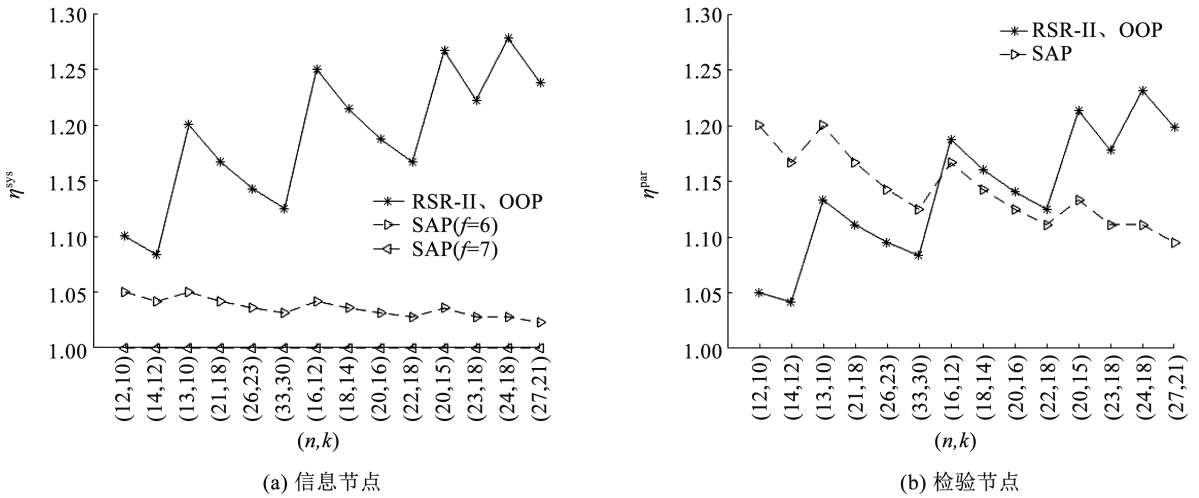
4.3　修复度率

表 1 为 RSR-II、OOP 和 SAP 关于子条带数、平均修复度率及最小平均修复带宽率的对比。由表 1 可以发现，RSR-II 和 OOP 的子条带数受校验节点数 r 的影响，而 SAP 的子条带数可自由选取，所以 SAP 的子条带数选取更加灵活，且少于 RSR-II 和 OOP 的子条带数量。图 8 为 SAP 与 RSR-II 和 OOP 平均修复度率对比曲线。由表 1 中 η^{sys} 公式和图 8(a)对比可以发现，RSR-II 和 OOP 的 η^{sys} 随 k 的增大呈递减趋势，随 r 的增大呈递增趋势；而 SAP 的 η^{sys} 不受 r 的影响且小于现有的 piggybacking 编码，所以，当 k 保持不变，随 r 的增大，SAP 的 η^{sys} 将更加低于 RSR-II 和 OOP 的 η^{sys}。由图 8(b)可以看出，在 $r=2$ 和 $r=3$ 时，RSR-II 和 OOP 的 η^{par} 小于 SAP 的 η^{par}。原因是当 RSR-II 和 OOP 的第 1 个校验节点故障时，只能利用 MDS 编码恢复，需要下载全部的信息数据块。此情况下连接的节点数较少，所以在校验节点 r 较小时，出现了平均修复度率较低的情况。但随 r 的增大，修复度率也出现上升趋势。例如，在 k 相同时，从 $r=4$ 开始，SAP 的 η^{par} 低于 RSR-II 和 OOP。

表 1　piggybacking 编码对比

Tab. 1　Comparison of piggybacking coding

piggybacking 编码	f	η^{sys}	η^{par}	min γ^{sys}	min γ^{par}
RSR-II[8]	$(2r-3)\tau$	$\frac{k+r-1}{k}$	$\frac{k+(r-1)(k+r-1)}{rk}$	$\frac{r-1}{2r-3}$	$\frac{1}{r}+\frac{r-1}{2r}$
OOP[10]	$\sqrt{r-1}+r-1$	$\frac{k+r-1}{k}$	$\frac{k+(r-1)(k+r-1)}{rk}$	$\frac{2\sqrt{r-1}+1}{2\sqrt{r-1}+r}$	$\frac{\sqrt{r-1}+1}{r}$
SAP	$f=2\tau_1\geqslant 2(\tau=\tau_1)$ $f=2\tau_1+1\geqslant 3(\tau\neq\tau_1)$	$\eta^{sys}_{\tau=\tau_1}$ 1	$\frac{k+2}{k}$	$\frac{1}{2}+\frac{1}{4r}$ $\frac{r+2}{3r}$	$\frac{1}{4}+\frac{1}{4r}$



(a) 信息节点　　(b) 检验节点

图 8　平均修复度率对比曲线

Fig. 8　Comparison curve of average repair degree rate

5　结　论

1）SAP 编码扩展了一个子条带，尽管增加了节点存储开销，但节点存储开销增加率随子条带数的增加呈指数递减趋势。因此，存储开销的增加可以忽略不计。

2）在信息节点平均修复带宽率方面，SAP 编码整体低于 RSR-I、RSR-II 和 OOP。同时，SAP 编码在校验节点平均修复带宽率和节点总的平均修复带宽率方面显著优于 RSR-I、RSR-II 和 OOP，显示出其在整体修复带宽开销方面的优势。SAP 编码显著降低了多校验节点故障修复带宽率，特别是在校验节点故障数 $t=2$ 的情况下，SAP 编码的修复带宽开销降幅最大。

3）SAP 编码信息节点修复度率始终低于 RSR-II 和 OOP。然而，在校验节点数 $r=2$ 和 $r=3$ 时，SAP 编码的校验节点修复度率高于 RSR-II 和 OOP。但随着校验节点数的增加，这一趋势发生逆转，SAP 编码的校验节点修复度率也低于 RSR-II 和 OOP。

4）SAP 编码在修复带宽开销和修复度率方面均优于 RSR-II 和 OOP，特别是在多校验节点故障情况下表现尤为显著。通过扩展子条带，SAP 编码克服了 RSR-II 和 OOP 的不足，具有重要的理论意义和应用价值。

参考文献

[1] SIDDIQA A, KARIM A, GANI A. Big data storage technologies: a survey[J]. Frontiers of Information Technology and Electronic Engineering, 2017, 18(8): 1040. DOI:10.1631/FITEE.1500441

[2] BLINOVA O, KUPRIKOV O, FARKHADOV M. Analysis of frameworks and technologies for solving big data storage and processing problems in distributed systems [C]//2023 IEEE International Conference on Information, Control, and Communication Technologies (ICCT). Astrakhan: IEEE, 2023: 1. DOI:10.1109/ICCT58878.2023.10347069

[3] YAVARI E, ESMAEILI M. Locally repairable codes: joint sequential-parallel repair for multiple node failures [J]. IEEE Transactions on Information Theory, 2020, 66(1): 222. DOI:10.1109/TIT.2019.2940975

[4] LI Jun, LI Baochun. Demand-aware erasure coding for distributed storage systems[J]. IEEE Transactions on Cloud Computing, 2021, 9(2): 532. DOI:10.1109/TTC.2018.2885306

[5] PAPAILIOPOULOS D S, DIMAKIS A G, CADAMBE V R. Repair optimal erasure codes through Hadamard designs[C]// 49th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2011. Monticello: IEEE, 2011: 1382. DOI: 10.1109/Allerton.2011.6120328

[6] DIMAKIS A G, GODFREY P B, WU Y, et al. Network coding for distributed storage systems[J]. IEEE Transactions on Information Theory, 2010, 56(9): 4539. DOI:10.1109/TIT.2010.2054295

[7] RASHMI K V, SHAH N B, RAMCHANDRAN K. A piggybacking design framework for read-and download-efficient distributed storage codes [C]//2013 IEEE International Symposium on Information Theory(ISIT). Istanbul: IEEE, 2013: 331. DOI: 10.1109/ISIT.2013.6620242

[8] RASHMI K V, SHAH N B, RAMCHANDRAN K. A piggybacking design framework for read-and download-efficient distributed storage codes[J]. IEEE Transactions on Information Theory, 2017, 63(9): 5802. DOI:10.1109/TIT.2017.2715043

[9] YUAN Shuai, HUANG Qin, WANG Zulin. A repair-efficient coding for distributed storage systems under piggybacking framework[J]. IEEE Transactions on Communications, 2018, 66(8): 3245. DOI: 10.1109/TCOMM.2018.2805347

[10] LI Guiyang, LIN Xing, TANG Xiaohu. An efficient one-to-one piggybacking design for distributed storage systems [J]. IEEE Transactions on Communications, 2019, 67(12): 8193. DOI:10.1109/TCOMM.2019.2941933

[11] 周悦，李贵洋，韩鸿宇，等. 一种均衡分配的修复校验节点的 Piggybacks 捎带设计[J]. 电子学报，2021，49(4)：812
ZHOU Yue, LI Guiyang, HAN Hongyu, et al. A balanced-allocation Piggybacks adding design for repairing parity nodes[J]. Acta Electronica Sinica, 2021, 49(4): 812. DOI: 10.12263/DZXB.20200056

[12] 张璐. 分布式存储系统中 piggybacking 框架的设计[D]. 西安：西安电子科技大学，2021
ZHANG Lu. Design of piggybacking framework in distributed storage system[D]. Xi′an: Xidian University, 2021. DOI:10.27389/d.cnki.gxadu.2021.001745

[13] SUN Rong, LI Xin, ZHANG Lu, et al. Distributed storage codes based on double-layered piggybacking framework [J]. IEEE Access, 2020, 8: 150447. DOI: 10.1109/ACCESS.2020.3002824

[14] SUN Rong, ZHANG Lu, LIU Jingwei. A new piggybacking design with low-repair bandwidth and complexity[J]. IEEE Communications Letters, 2021, 25(7): 2099. DOI: 10.1109/LCOMM.2021.3071855

[15] JIANG Zhengyi, HOU Hanxu, HAN Yunxiang, et al. An efficient piggybacking design with lower repair bandwidth and lower sub-packetization [C]//2021 IEEE International Symposium on Information Theory (ISIT). Melbourne: IEEE, 2021: 2328. DOI: 10.1109/ISIT45174.2021.9518037

[16] SHI Hao, HOU Hanxu, HAN Yunxiang, et al. New piggybacking codes with lower repair bandwidth for any single-node failure[C]// 2022 IEEE International Symposium on Information Theory (ISIT). Espoo: IEEE, 2022: 2601. DOI: 10.1109/ISIT50566.2022.9834881