xStripeMerge：基于纠删码存储的高效宽条带生成方法

郑美光，化韬斐，张心宇，胡志刚
（中南大学计算机学院，湖南 长沙 410083）

摘 要：为了解决纠删码存储系统的已有宽条带生成方法中扩容方案将产生大量宽条带生成带宽，合并方案受限于双条带的问题，提出了一种针对多条带合并下的宽条带生成问题的高效宽条带生成方法。定义了多条带合并过程的 2 个关键算子，并将宽条带生成问题建模为组合优化问题，提出了优先寻找具有小的奇偶校验块传输成本的窄条带组合方案的高效宽条带生成方法 xStripeMerge。实验结果表明，与目前最优的存储扩容方法相比，xStripeMerge 可以减少 75%宽条带生成带宽。xStripeMerge 的时间和空间复杂度远优于扩展的双条带合并方法，xStripeMerge 可以在更短的时间内获得与其性能相近的宽条带生成方案，并且 xStripeMerge 可以适用于大规模存储系统。

关键词：分布式存储；纠删码；宽条带；条带合并；奇偶块对齐
中图分类号：TP302.8
文献标志码：A
DOI: 10.11959/j.issn.1000−436x.2023217

xStripeMerge: efficient wide stripe generation approach based on erasure coding storage

ZHENG Meiguang, HUA Taofei, ZHANG Xinyu, HU Zhigang
School of Computer Science and Engineering, Central South University, Changsha 410083, China

Abstract: To address the issue of wide stripe generation in existing erasure coding storage systems, where storage scaling approaches resulted in a significant increase in wide stripe generation bandwidth and stripe merge scheme was constrained by dual-stripes, an efficient wide stripe generation approach was proposed for the wide stripe generation problem under multi stripes merging. Two key operators for the multi-stripes merging progress were defined, and the wide stripe generation problem was modeled as a combinatorial optimization problem. The efficient wide stripe generation approach xStripeMerge was proposed that prioritize the search for narrow-stripes combining schemes with small parity block transmission costs. Experimental results show that xStripeMerge can reduce the wide stripe generation bandwidth by 75% compared to the advanced storage scaling method. The time and space complexity of xStripeMerge is much better than that of the extended dual-stripes merge approach. xStripeMerge can get the wide stripe generation scheme with similar performance in a shorter period of time and it is also suitable for large-scale storage systems.

Keywords: distributed storage, erasure coding, wide stripe, stripe merge, parity align

0 引言

在采取差错预测等可靠性措施的情况下，分布式存储系统仍然会出现不可预期的故障，容错机制成为存储系统出现故障后避免立刻失效，并加速恢复正常工作状态的必要手段。纠删码技术和副本机制是存储系统用来提供数据容错的常用方法[1]。纠删码技术通过编码计算，在提供与副本机制等同的容错性能时需要的额外存储开销更少[2]。例如，在 Azure[3]中使用的传统三副本引入了 200%的存储冗

收稿日期：2023−09−01；**修回日期：**2023−10−24
基金项目：国家自然科学基金资助项目（No.62172442, No.62172451）；湖南省自然科学基金青年基金资助项目（No.2020JJ5775）
Foundation Items: The National Natural Science Foundation of China (No.62172442, No.62172451), Youth Science Foundation of Natural Science Foundation of Hunan Province (No.2020JJ5775)

余开销，而采用纠删码存储可以将存储冗余开销减少到 33%，并实现更高的可用性[4]。在众多的纠删码编码中，RS（Reed Solomon）码[5]是目前使用最广泛的纠删码，应用于谷歌的 ColossusFS[6]、HDFS（hadoop distributed file system）[7]等存储系统[4,8]中。RS 码 RS(k,m) 存储系统中，当原始存储文件丢失时，只要从存储节点上读取任意 k 个数据单元（数据块或奇偶块）即可恢复原始存储文件，其存储冗余度为 $\frac{n}{k}$（$n=k+m$）取决于其配置参数条带中数据块数量 k 和奇偶块数量 m。例如，Facebook f4[9]设置 k=10、m=4，其存储冗余度为 1.4×，QFS[10]设置 k=6、m=3，其存储冗余度为 1.5×。

近年来，一些研究工作为进一步减少存储冗余开始探索宽条带相关研究，即具有较多数据块和较少奇偶块的条带。例如，VAST 平台[11]提供了 RS(150,4)四数据块容错，其冗余度为 1.027×。对于 RS 码存储系统，其恢复成本将随着条带中数据块数量的增加而增加，重建宽条带中丢失的数据块将引发大量的数据传输开销。为此，一些工作探索了纠删码编码的参数更新问题，以实现存储系统的高访问性能和高存储空间利用率。

Yao 等[12]提出分层编码方法，思想是实际存储系统中根据数据的热度采用不同参数设置的纠删码存储数据。分层编码方法建议将新写入的数据块编码为小 k 值的窄条带，以获得高数据恢复性能。随着数据的老化和访问频率的降低，以窄条带存储的冷数据应当转换为大 k 值的宽条带存储以提高存储效率。当前已有关于从窄条带编码转换为宽条带的研究。存储扩展方案[13-14]通过扩展 s 个存储节点可将窄条带 RS(k,m) 重新编码为多种形式的宽条带 RS($k+s$,m)，但是该方法会产生大量的宽条带生成带宽。条带合并方法 StripeMerge[12]具有较低的宽条带生成带宽，但这种方法仅限于将 RS(k,m) 窄条带合并成 RS($2k$,m) 宽条带。因此，当存储扩展方案扩容的存储节点数量 $s>k$ 时，StripeMerge 生成的宽条带在保证相同容错下所引入的额外存储开销多于存储扩展方案。

基于此，本文提出了多条带合并的宽条带生成问题，设计了一种生成带宽少、具有扩展性的宽条带生成方案。本文定义了宽条带生成过程中的 2 个带宽密集型操作，并将宽条带生成问题建模为组合优化问题。在此基础上，本文提出了高效宽条带生成方法 xStripeMerge 来解决该组合优化问题。最后，本文实现了 xStripeMerge 原型并通过实验对 xStripeMerge 进行了性能评估。实验结果表明，xStripeMerge 在宽条带生成带宽方面明显优于当前已有的最优存储扩容方法，在保证相同带宽性能的条件下，时间效率明显优于扩展的条带合并方法。

1 相关工作

目前纠删码的相关研究大多关注减少实际系统存储中恢复故障块所需的数据传输带宽。一些工作通过设计新型纠删码以减少数据恢复开销，例如，Pamies-Juarez 等[15]设计了蝴蝶码，是一种最小存储再生码，具有较小的数据恢复开销，已在分布式存储系统（HDFS）[7]中实现了其原型。Ye 等[16]将蝴蝶码与 LRC（locally repairable code）混合实现了新的编码 Hybrid-RC，对编码所需的计算量与数据恢复所需的传输带宽实现了进一步优化。

一些工作通过改变存储系统中的条带布局以减少数据修复带宽[17-19]。Liu 等[18]针对纠删码中单一节点故障恢复问题提出了具有更高恢复并行度的条带放置方案。Hu[19]等结合已有的奇偶校验局部性和拓扑局部性纠删码编码方案提出了组合局部性宽条带编码方案，实现了宽条带的跨机架修复带宽和冗余度的权衡。

实际存储系统中的工作负载具有倾斜和动态特性[20]，倾斜特性体现为少部分数据为热数据被频繁访问，而大部分数据为冷数据几乎不被读取。动态特性反映在数据的热度会随时间变化。因此，一些工作考虑采用冗余转换的形式实现存储性能与数据恢复性能的权衡。Xia 等[21]通过使用 2 种不同参数的纠删码来适应工作负载的变化并在 HDFS 实现了 Product 码和 LRC 的原型 HACFS。基于数据热度的分析观察，Yao 等[12]提出了根据数据热度采用不同参数设置的纠删码存储数据，即采用窄条带存储热数据，采用宽条带存储冷数据，可以提高热数据的恢复性能和冷数据的存储效率，其所提 StripeMerge 方法大量减少了特定设置下 RS 码冗余转换，即 RS(k,m) 转换为 RS($2k$,m) 的数据传输。Wu 等[22]研究了条带在存储系统中的放置对 LRC 合并成本的影响，并设计了 2 种不同的条带放置方案以减少 LRC 条带合并中的带宽传输。而本文所关注的内容是如何在尽可能提升存储性能的同时实现 RS 码的高效冗余转换。

已知纠删码性能优化方法的对比如表 1 所示。

表 1　　已知纠删码性能优化方法的对比

方法	纠删码形式	应用	性能优化方向
设计新型纠删码[15-16]	单一	适合热数据存储系统	编码性能、数据恢复性能、存储性能
改变条带布局[17-19]	单一	适合热数据存储系统	数据恢复性能、数据更新性能
冗余转换[12,21-22]	多种	适合冷、热数据共存的存储系统	存储性能、数据恢复性能

2　理论基础

2.1　纠删码理论基础

与副本机制相比，纠删码因其较高的空间利用率和较强的容错性被广泛应用于大规模的存储系统。其技术原理如下：首先，将需要存储的文件划分为 k 个数据块，通过数据单元编码运算得到 m 个奇偶校验块；然后，将这 $k+m$ 个数据块分布式存储于 $n=k+m$ 个存储节点从而实现数据容错。每个数据块通常被配置为 64 MiB[23]或 256 MiB[24]以减轻 I/O 搜索开销。

根据是否为最大距离可分离（MDS, maximum distance separable）码[25]，纠删码可划分为两类。MDS 码的最小汉明距离等于 $m+1$，其中 m 为冗余存储块的个数。RS 码是目前存储系统中应用最广泛的 MDS 码。RS 码的编解码运算均是基于伽罗瓦域的运算。本文专注于采用范德蒙矩阵作为系数矩阵的 RS 码[26]，即条带中的每个奇偶校验块可由数据块通过线性编码得到，表示为

$$P_i=\sum_{j=1}^{k}A_{ij}D_j,1\leqslant i\leqslant m \tag{1}$$

其中，P_i 为条带中的奇偶校验块；D_j 为条带中的数据块；A_{ij} 为编码系数矩阵 $\boldsymbol{A}$ 的元素，$\boldsymbol{A}$ 是一个 $m\times m$ 的范德蒙矩阵。

2.2　宽条带生成

存储扩展方法[13-14]是目前大规模存储系统中解决宽条带生成问题的常用方法。存储扩展方法通过增加 s 个存储节点将 RS(k,m) 转换为 RS$(k+s,m)$ 从而扩展系统存储性能。图 1(a)展示了 NCScale[13]作为优秀的存储扩展方法，如何将分布于 4 个存储节点的 RS(2,2) 窄条带 $\{a,b,P_1,P_2\}$ 与属于其他条带的数据块 c、d、e、f 转换为分布于 8 个存储节点的 RS(6,2) 宽条带 $\{a,b,c,d,e,f,Q_1,Q_2\}$。通过位于 N_3 节点的奇偶块 $P_1=a+b$ 与数据块 c、d、e、f 编码可得到宽条带中的奇偶块 $Q_1=a+b+c+d+e+f$；通过位于 N_4 节点的奇偶块 $P_2=a+2b$ 与 N_3 节点传输的奇偶块差值 $\Delta=2^2c+2^3d+2^4e+2^5f$ 编码可得到宽条带中的另一奇偶块 $Q_2=a+2b+2^2c+2^3d+2^4e+2^5f$。扩充存储节点 N_5、N_6、N_7、N_8，并将位于同一节点的数据块 c、d、e、f 迁移到其他节点。从上述实例中可以看出存储扩展方法可将存储冗余度从 2×降低到 1.33×。但是，这种方法需要增加新的存储节点，而且在宽条带生成过程中会产生大量的数据传输开销。

StripeMerge[12]将每 2 个窄条带合并为一条宽条带。图 1(b)展示了 StripeMerge 如何将 RS(2,2) 编码的窄条带 $\{a,b,P_1,P_2\}$ 和 $\{c,d,P_3,P_4\}$ 转换为 RS(4,2) 编码的宽条带 $\{a,b,c,d,Q_1,Q_2\}$。通过位于 N_3 节点的奇偶块 $P_1=a+b$ 和奇偶块 $P_3=c+d$ 编码，得到宽条带中的奇偶块 $Q_1=a+b+c+d$；通过位于 N_4 节点的奇偶块 $P_2=a+2b$ 和奇偶块 $P_4=c+2d$ 编码，得到宽条带中的奇偶块 $Q_2=a+2b+2^2c+2^3d$。图 1(b)中存储冗余度从 2×降低到了 1.5×。在图 1(b)中，StripeMerge 通过 2 个窄条带合并的方式，条带编码转换过程中没有数据传输开销。相比于存储扩容方法 NCScale，StripeMerge 在解决宽条带生成问题时引发的数据传输带宽更少。但是，StripeMerge 目前只适用于将 RS(k,m) 转换为 RS$(2k,m)$，系统的存储冗余仅可以从 $\frac{k+m}{k}$ 降低至 $\frac{2k+m}{2k}$。

综上所述，无论是存储扩展方法还是目前已有的条带合并方案都无法在提升存储性能的同时实现高效的宽条带生成。

3　场景描述与问题建模

3.1　多条带合并下的宽条带生成

RS 码的冗余度为 $\frac{n}{k}$，提高数据块在条带中所占比例可以减少系统中的额外存储。基于此，本文通过合并多个窄条带生成一个宽条带来提高宽条带的存储性能。将 2 个窄条带合并为宽条带过程中，

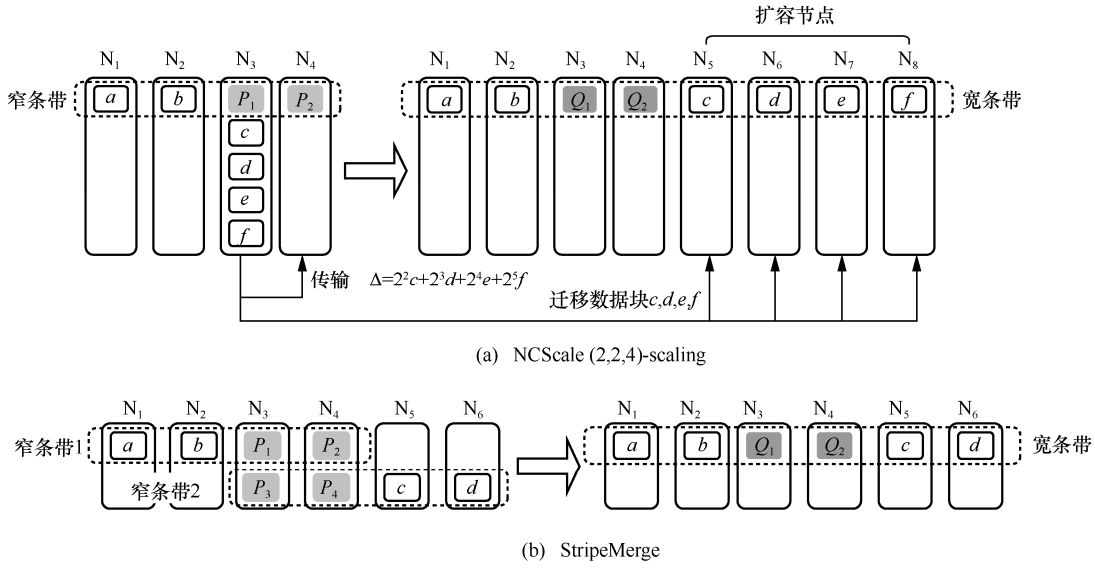


图 1　2 种宽条带生成方法

宽条带中的奇偶块可通过窄条带的奇偶块编码得到[12]。本文发现该性质对于多条带的合并仍然适用。基于式(1)，可以得到窄条带中的奇偶块编码为

$$\begin{bmatrix}P_{11}\\P_{21}\end{bmatrix}=\begin{bmatrix}1&1\\1&2\end{bmatrix}\begin{bmatrix}D_{11}\\D_{21}\end{bmatrix}$$
$$\begin{bmatrix}P_{12}\\P_{22}\end{bmatrix}=\begin{bmatrix}1&1\\1&2\end{bmatrix}\begin{bmatrix}D_{12}\\D_{22}\end{bmatrix}$$
$$\vdots$$
$$\begin{bmatrix}P_{1x}\\P_{2x}\end{bmatrix}=\begin{bmatrix}1&1\\1&2\end{bmatrix}\begin{bmatrix}D_{1x}\\D_{2x}\end{bmatrix}\tag{2}$$

宽条带中的奇偶块编码为

$$\begin{bmatrix}Q_1\\Q_2\end{bmatrix}=\begin{bmatrix}1&1\cdots&1\\1&2\cdots&2^{2x-1}\end{bmatrix}\begin{bmatrix}D_{11}\\D_{21}\\D_{12}\\D_{22}\\\vdots\\D_{1x}\\D_{2x}\end{bmatrix}\tag{3}$$

基于式(2)和式(3)可得

$$\begin{bmatrix}Q_1\\Q_2\end{bmatrix}=\begin{bmatrix}P_{11}\\P_{21}\end{bmatrix}+\begin{bmatrix}1^2P_{12}\\2^2P_{22}\end{bmatrix}+\cdots+\begin{bmatrix}1^{2(x-1)}P_{1x}\\2^{2(x-1)}P_{2x}\end{bmatrix}\tag{4}$$

通过上述分析，可以得到定理 1。

定理 1　在多条带合并生成宽条带的过程中，宽条带的奇偶块不需要通过传输窄条带中的数据块来计算，可以通过窄条带中的奇偶块编码得到。如果待合并的窄条带的奇偶块位于同一存储节点，那么可以利用该性质在该存储节点上本地计算得到宽条带中的奇偶块，从而可以大量减少多条带合并生成宽条带引发的数据传输开销。

3.2　合并过程的 2 个关键算子

本节解释了如何通过多个 RS(k,m) 窄条带的组合合并得到 RS(xk,m) 宽条带并定义了合并过程的关键算子。

定义 1　数据块主动迁移操作。为了将 x 个 RS(k,m) 条带合并成单个 RS(xk,m) 条带，条带的数据块数量从 k 增加到 xk。对于 RS 码存储系统，属于同一条带的数据块需要存储在不同节点上以保证单节点容错。待合并的窄条带中原隶属于不同条带的数据块在合并前满足单节点容错要求，在合并后可能位于同一存储节点而不满足容错要求。需要迁移此类数据块以确保合并后宽条带的数据块仍然位于不同的节点。本文将其定义为数据块主动迁移操作。

定义 2　奇偶块更新传输操作。多个窄条带合并为宽条带后，需要更新奇偶块。更新的奇偶块可以根据窄条带的奇偶块编码得到。为了保证更新后奇偶块仍然具有数据容错解码能力，需要将原本属于不同窄条带的奇偶块信息传输到同一节点来计算宽条带中的奇偶块。本文将其定义为奇偶块传输操作。

基于上述定义，多个 RS(k,m) 窄条带组合的宽条带生成带宽可以分为两部分，即数据块迁移和奇偶块传输所引发的数据传输带宽。RS(2,2) 窄条带 $\{D_1,D_2,P_1,P_2\}$、$\{D_3,D_4,P_3,P_4\}$、$\{D_5,D_6,P_5,P_6\}$ 合

并为RS(6,2)宽条带$\{D_1,D_2,D_3,D_4,D_5,D_6,Q_1,Q_2\}$的数据传输过程如图 2 所示。由于隶属于不同窄条带的数据块D_1和数据块D_3位于同一存储节点 N_2，因此需要将D_3数据块迁移到空余节点N_1，即图 2 中的线路①。由式(4)可得宽条带中更新的奇偶块$Q_1=P_1+P_3+P_5$、$Q_2=P_2+2^2P_4+2^4P_6$。因此将奇偶块P_1和奇偶块P_3传输到P_5所在的节点N_3，编码得到宽条带中的奇偶块Q_1，即图 2 中的线路②和线路④。将节点 N_4 本地计算得到的$\Delta=2^4P_6$传输到节点 N_6 编码得到宽条带中的奇偶块Q_2，即图中的线路③。

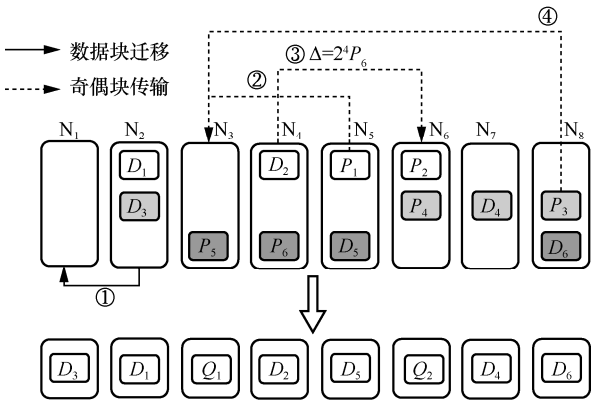


图 2　3 个 RS(2,2)窄条带合并为一个 RS(6,2)宽条带的数据传输过程

3.3　系统模型

针对分布式存储系统中的纠删码存储系统，考虑一个含有 n 个存储节点的存储系统，用集合$\{N_1,N_2,\cdots,N_n\}$表示。存储节点之间均存在无线通信链路。用户上传的文件初始默认为热数据，以窄条带RS(k,m)的形式存储于系统中。考虑目前已存储于系统的窄条带用集合$\mathrm{NS}=\{\mathrm{ns}_1,\mathrm{ns}_2,\cdots,\mathrm{ns}_l\}$表示。待数据老化为冷数据后，为减少系统保证容错所引入的额外存储，这些老化的窄条带将被合并为宽条带RS(xk,m)。多条带合并下的宽条带生成问题是如何以少量的数据传输代价将 l 个窄条带合并为$\lambda=\left\lfloor\frac{l}{x}\right\rfloor$个宽条带。$\lambda=\left\lfloor\frac{l}{x}\right\rfloor$个宽条带记为$\mathrm{WS}=\{\mathrm{ws}_1,\mathrm{ws}_2,\cdots,\mathrm{ws}_\lambda\}$。条带由原RS$(k,m)$码更新为RS$(xk,m)$码，存储冗余度由原来的$\frac{k+m}{k}$降为$\frac{xk+m}{xk}$。基于纠删码存储的条带合并系统模型如图 3 所示。

对于新生成的宽条带 ws_i，令生成 ws_i 前的 x 个窄条带组合记为ws_i^x，即宽条带 ws_i 由ws_i^x中 x 个窄条带合并后生成，所需的数据块迁移成本为

$$C_{\mathrm{data}}(\mathrm{ws}_i^x)=xk-\sum_{j=1}^{n}w_{\mathrm{data}}(N_j) \tag{5}$$

其中，k 为原窄条带中数据块的数量；$w_{\mathrm{data}}(N_j)$为

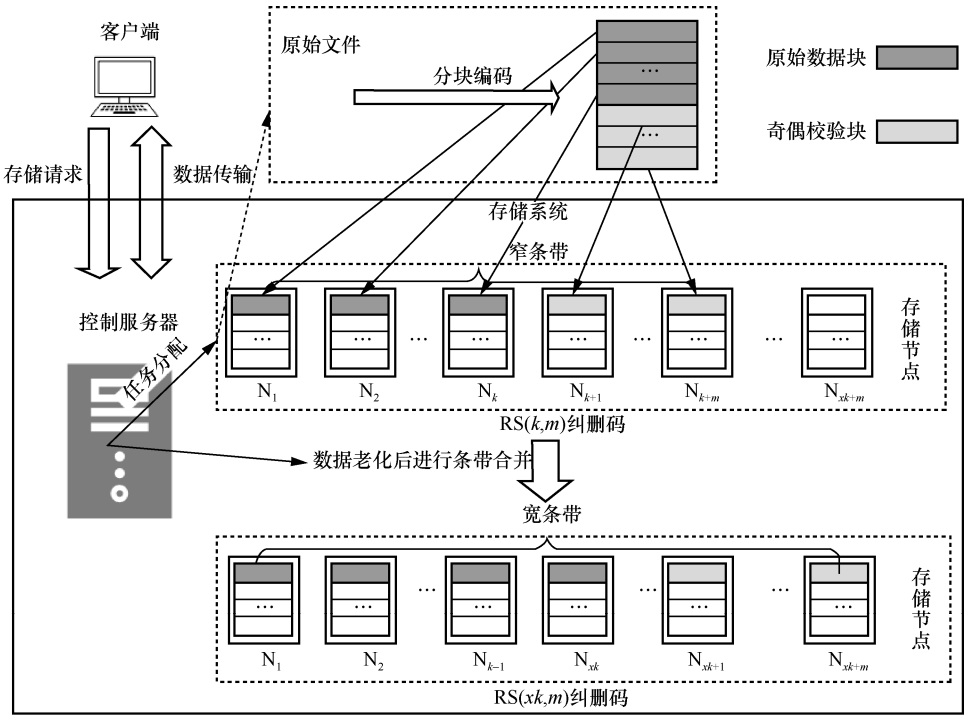


图 3　基于纠删码存储的条带合并系统模型

二进制变量，若 ws_i^x 中窄条带的数据块位于节点 j，则其值为 1，否则其值为 0。

组合 ws_i^x 生成宽条带 ws_i 的奇偶校验块传输成本为

$$C_{\mathrm{parity}}(\mathrm{ws}_i^x)=\sum_{p=1}^{m}\left(\sum_{j=1}^{n}w_{\mathrm{parity}\_p}(N_j)\right) \tag{6}$$

其中，$w_{\mathrm{parity}\_p}(N_j)$ 为二进制变量，若 ws_i^x 中窄条带的第 p 组奇偶块位于节点 j，则其值为 1，否则其值为 0。

基于式(5)和式(6)，可以得到 ws_i^x 生成宽条带 ws_i 的总数据传输带宽为

$$C(\mathrm{ws}_i)=C_{\mathrm{data}}(\mathrm{ws}_i^x)+C_{\mathrm{parity}}(\mathrm{ws}_i^x) \tag{7}$$

本文将多条带合并的宽条带生成问题建模为组合优化问题。优化目标为寻找具有最小宽条带生成带宽的条带组合方案。$o_j=\left\{\mathrm{ws}_{1j}^x,\mathrm{ws}_{2j}^x,\cdots,\mathrm{ws}_{ij}^x\right\}$，其中 $\bigcup_{i-1}^{\lambda}\mathrm{ws}_{ij}^x=NS$。设定集合 O 为全部窄条带的所有不同组合方案，则问题的目标函数定义为

$$\min\sum_{i=1}^{\lambda}C(\mathrm{ws}_i)\mathrm{ws}_{ij}^x\in o_j,o_j\in O \tag{8}$$

定理 2 寻找上述组合优化问题的最优解所需时间随着窄条带规模 l 呈指数级增加。

证明 对于 l 个窄条带，该组合优化问题的解集合 O 的大小为 $\dfrac{(\mathrm{C}_l^x\mathrm{C}_{l-x}^x\cdots 1)}{\left[\left(\dfrac{l}{x}\right)!\right]}$。从解集合 O 中寻找具有最小值的窄条带合并方案，最坏情况下需要至少 $\dfrac{(\mathrm{C}_l^x\mathrm{C}_{l-x}^x\cdots 1)}{\left[\left(\dfrac{l}{x}\right)!\right]}-1$ 次比较。由于该解集合的大小随着窄条带规模 l 呈指数级增加，因此寻找上述组合优化问题的最优解所需时间也会随着窄条带规模 l 呈指数级增加。

4 算法设计

虽然 StripeMerge 仅可将 RS(k,m) 转换为 RS($2k$,m)，但其中的贪心算法经扩展后可将 RS(k,m) 转换为 RS(xk,m)，以此解决多条带合并下的宽条带生成问题。本文将其记为 StripeMerge-G，其主要思想是给定当前存储系统中的一系列窄条带，计算 C_l^x 个窄条带组合的合并成本并降序排列，优先选择具有小合并成本的条带组合进行合并。贪心启发式算法的总时间复杂度为 $O\left(l^x x(k+m)\right)$，可以在多项式时间内找到宽条带生成问题的次优解，但该算法的空间复杂度为 $O(l^x)$。当需要合并的窄条带数量达到数千条时，StripeMerge-G 算法的内存消耗将达到数 TB。而且，随着 x 值增大，该算法的时间复杂度将大幅增加。因此，StripeMerge-G 在大规模条带合并中表现将不够理想。为此，本文提出了基于多条带合并启发式搜索的高效宽条带生成方法 xStripeMerge。

窄条带组合的合并成本包括数据块迁移成本和奇偶块传输成本两部分。对于具有小合并成本的窄条带组合，其数据块迁移成本和奇偶块传输成本也相应较小。对于数据块迁移成本较小的窄条带组合，考虑到数据块主动迁移性质，组合内不同窄条带的数据块应尽量存储于不同的存储节点。而对于奇偶块传输成本较小的窄条带组合，考虑到奇偶块更新传输特性，组合内不同窄条带的大部分奇偶块会存储于相同存储节点。寻找具有不同数据块存储位置的窄条带需要 C_l^2 次比较，而寻找具有相同奇偶块存储位置的窄条带仅需要 l 次遍历。基于上述分析，本文提出了 xStripeMerge，优先寻找奇偶校验块传输成本小的窄条带组合，从而快速找到总合并成本小的组合。

1) 奇偶块哈希表

搜索过程中，需要得到现有存储系统中窄条带的奇偶块位置以及利用窄条带奇偶块位置得到窄条带合并候选组合。本文沿用 StripeMerge 中的数据结构——奇偶块对齐哈希表[12]。奇偶块对齐哈希表存储了每个窄条带的奇偶块位置信息，

StripeMerge 中通过奇偶块哈希表直接寻找奇偶块位置相近的成对条带合并。本文通过该哈希表与后文提出的多条带合并搜索表找到每个条带的候选合并组合。

某存储系统包含 16 个存储节点，6 个 RS(4,4) 条带将其转换为 2 个 RS(12,4) 条带，系统现有的存储布局及所对应的部分奇偶块对齐哈希表如图 4 所示，其中，P_{ij} 表示条带 j 的第 i 组奇偶块，D_{ij} 表示条带 j 的第 i 个数据块。

奇偶块哈希表中每个关键字 key 指的是每个条

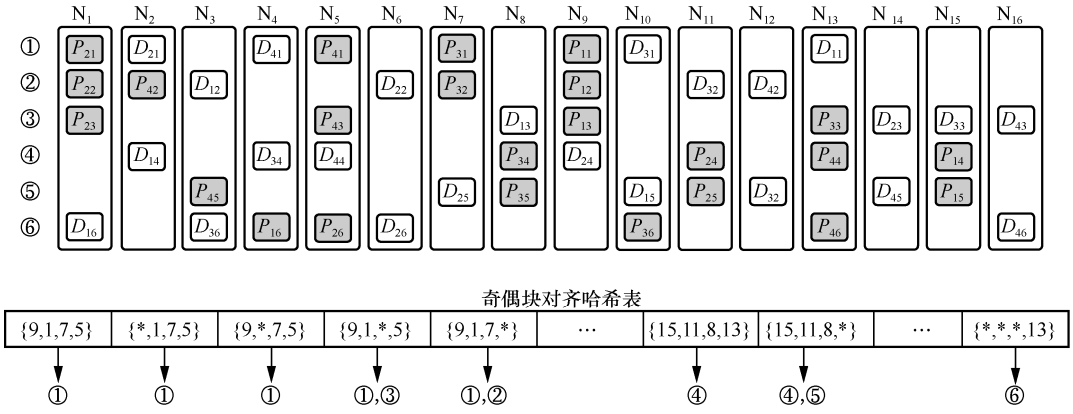


图 4　16 个存储节点存储 6 个 RS(4,4)窄条带

带任意 p 个奇偶校验块（$1 \leqslant p \leqslant m$）在 N 个节点上的位置，其值 value 是具有相应奇偶块位置的条带的索引列表。通过遍历已存储的每个条带奇偶块位置，可以得到构成该哈希表的键值对。以图 4 为例，条带①的第一个、第二个、第三个和第四个奇偶块分别存储于节点 N_9、N_1、N_7 和 N_5，可以生成四位置键值对{9,1,7,5}→①，三位置键值对{*,1,7,5}→①，{9,*,7,5}→①，{9,1,*,5}→①，{9,1,7,*}→①，以及对应的二位置和一位置键值对。其中条带②可以得到具有同样关键字 key 的三位置键值对{9,1,7,*}→②，因此奇偶块对齐哈希表中有{9,1,7,*}→①,②。

2) 多条带合并搜索表

利用窄条带奇偶块位置信息，可以寻找每个条带的合并候选匹配条带。本文假设候选匹配条带与当前条带均存在奇偶块对齐现象，即 2 个窄条带的奇偶块存储于同一存储节点。能够对窄条带合并优化的前提是存储系统有奇偶块对齐现象。将 x 个窄条带合并，对于条带 i，最好情况下为与之合并的 $x-1$ 个窄条带中每个窄条带的 m 个奇偶块均对齐，此时奇偶块传输成本为 0。在上述假设的条件下，最坏情况为每个窄条带仅有一个奇偶块（共 x 个奇偶块分块）与条带 i 的奇偶块对齐，此时合并成本为 $(x-1)(m-1)$。因此，本文构建多条带合并搜索表，寻找合并成本在 $[0,(x-1)(m-1)]$ 范围内的条带候选合并组合。

给定输入多条带合并的具体值 x 与窄条带 RS(k,m) 的奇偶块个数 m。多条带搜索表中的第一行为条带合并成本，范围为 $[0,(x-1)(m-1)]$。搜索表的第一列为条带间奇偶块位置处于同一节点的组数，范围为 $[1,m]$。搜索表内的其余单元格内数值为条带数量 q_p，即所需 p 组奇偶块对齐的条带数量 q_p，同一列单元格构成了相应合并成本 cost 下的搜索向量 $\{q_m,q_{m-1},\cdots,q_1\}$。例如，$x=3$，$m=4$ 下的搜索表 table(3, 4)如表 2 所示。cost = 1 的搜索向量为 $\{1,1,0,0\}$，表示要在合并成本为 1 的条件下进行窄条带合并，仅存在一组可能，即存在一个 4 组奇偶块对齐的条带且存在一个 3 组奇偶块对齐的条带时满足。

表 2　　搜索表 table(3,4)

条带/组	cost									
	0	1	2		3		4		5	6
4	2	1	0	1	1	0	0	0	0	0
3	0	1	2	0	0	1	1	0	0	0
2	0	0	0	1	1	1	0	2	1	0
1	0	0	0	0	0	0	1	0	1	2

多条带合并搜索表构建如算法 1 所示。使用一个 $m \times m$ 的单位矩阵初始化搜索表。通过递归方法得到对应 x 值下的多条带合并搜索表。算法 1 共需递归 $x-1$ 次，每次需要得到当前 x 下的 $m(xm-x-m+2)$ 大小的搜索表，因此算法 1 的时间复杂度约为 $O(x^2m^2)$。

算法 1　多条带合并搜索表构建

输入　x, m

输出　搜索表 table(x, m)

```
1)  function 构建搜索表 (X, M)
2)      if x == 2
3)          初始化搜表 C 为 m×m 的单位矩阵;
4)          return 搜索表 C;
5)      else
6)          构建搜索表(x − 1, m);
```

```
7)      for i = 0 : (x – 1) (m – 1)
8)        for j = 0 : i > m ? m : i
9)          if C[i – j]不为空
10)           获取 C[i – j]的搜索向量 V_k;
11)           V_k[j]++;
12)         end if
13)       end for
14)     end for
15)     return 搜索表 C;
16)   end if
17) end function
```

给定一系列的 RS(k,m) 窄条带集合以及参数 x。在 cost 从 0 到 $(x-1)(m-1)$ 的范围内，算法 2 依据奇偶块对齐现象搜索每个窄条带的匹配条带。对于条带 j，奇偶对齐搜索首先通过部分奇偶对齐哈希表找到任意 p（$1 \leqslant p \leqslant m$）个奇偶块对齐的条带索引，并以此构建条带 j 的搜索域（key 为 p，值为条带索引的哈希表）。然后，根据对应参数下的搜索表找到此时 cost 下的搜索向量，根据搜索域和搜索向量在常数时间内找到条带 j 的匹配条带。

xStripeMerge 如算法 2 所示。具体搜索过程包括两部分：奇偶块部分对齐搜索和贪心搜索。算法第 1)～13)行为奇偶块部分对齐搜索过程，通过奇偶块部分对齐搜索找到尽可能多的条带匹配组合。算法第 14)～17)行对剩余的未搜寻到合适组合的条带，调用贪心算法生成剩余的条带组合方案。

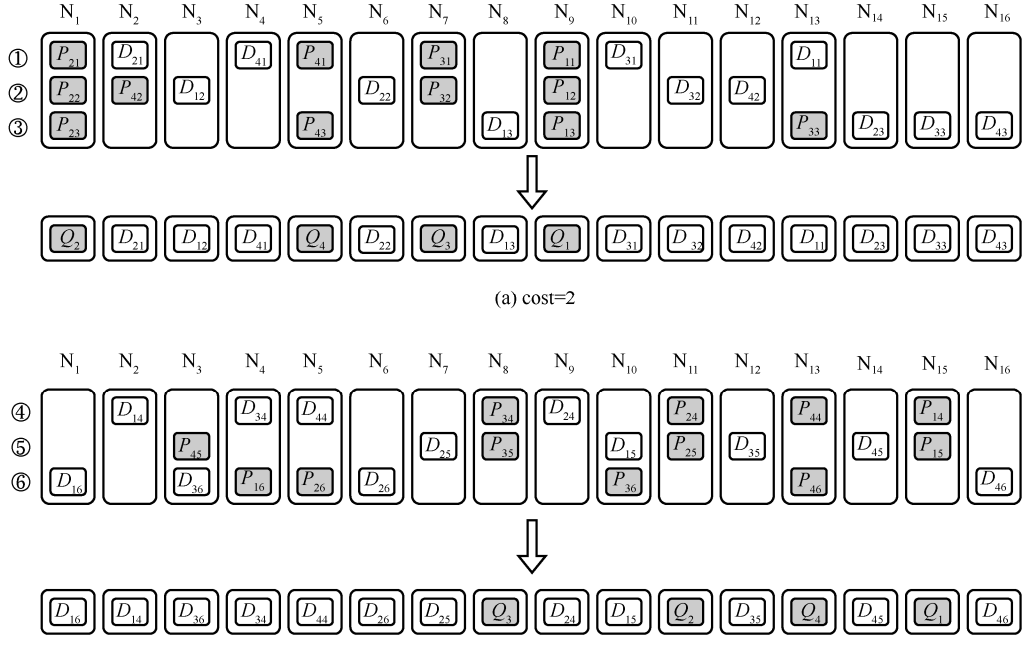
算法 2　xStripeMerge

输入　窄条带集合 $S:\{s_1,s_2,\cdots,s_l\}$, x

输出　合并的窄条带组合的集合 S'

```
1)  构建搜索表和部分奇偶对齐哈希表;
2)  奇偶块部分对齐搜索:
3)  for i = 0 : (x – 1) (m – 1)
4)    for j = 1 : l
5)      根据奇偶对齐哈希表构建条带 j 的搜索域;
6)      根据搜索表建立 cost 为 i 的搜索向量;
7)      依据搜索域和搜索向量寻找条带 j 的最小合并成本 c 的窄条带组合 S_j;
8)      if c ⩽ i
9)        将组合 S_j 加入集合 S';
10)       将 S_j 中的窄条带从 S 中删除;
11)     end if
12)   end for
13) end for
14) 贪心搜索:
15) if S != ∅
16)   S' = S' + 贪心算法(S);
17) end if
```

图 5 为图 4 实例的多条带合并启发式搜索的具



(a) cost=2

(b) cost=4

图 5　多条带合并启发式搜索的具体过程

体过程。算法从合并成本 $0 \leqslant \text{cost} \leqslant 6$ 搜索每个条带的对应匹配条带。当 $\text{cost}=2$ 时，此时通过搜索表可以得到目前的搜索向量为 $\{0,2,0,0\}$，即需要 2 个三组奇偶块对齐的条带与当前条带匹配。对于条带①，与三组奇偶块对齐的条带包含奇偶块位置均为 {9,1,*,5} 的条带③和奇偶块位置均为 {9,1,7,*} 的条带②，因此将条带①～③作为一个窄条带组合合并为一个宽条带。当 $\text{cost}=4$ 时，通过多条带搜索表可以得到搜索向量为 $\{0,1,0,1\}$，即需要一个三组奇偶块对齐的条带、一个一组奇偶块对齐的条带与当前条带匹配。对于条带④，与之三组奇偶块对齐的条带为奇偶块位置为 {15,11,8,*} 的条带⑤、与之一组奇偶块对齐的条带为奇偶块位置为 {*,*,*,13} 的条带⑥，因此将条带④～⑥作为一个窄条带组合合并为一个宽条带。至此完成图 4 中窄条带到宽条带的转换。

算法 2 中，奇偶对齐搜索阶段的时间复杂度是 $O\big(x(k+m)(x-1)ml\big)$，其中，l 表示条带数量，k 表示窄条带中的数据块数量，m 表示窄条带中的奇偶块数量。计算合并成本的时间复杂度为 $O\big(x(k+m)\big)$。由于条带数量 l 相比参数 k、m、x 要大得多，因此如果通过奇偶对齐搜索寻找到尽可能多的条带匹配，那么贪心算法的输入规模将会大大减少，整个算法的时间复杂度也会大大减少。这一结论也将在实验部分进行验证。

5　实验评估

为验证多条带条并启发式搜索方案，采用 C++ 搭建 xStripeMerge 系统，实验环境为 AMD Ryzen R7-5800H CPU, 16 GB 内存,WDC WDS100T2B0C-00PXH0 1 TiB 固态硬盘，操作系统为 Ubuntu16.04。

将本文所提 xStripeMerge 与扩展的 StripeMerge[12] 算法 StripeMerge-G 和 NCScale[13]，实现进行对比。评价指标为宽条带生成带宽和运行时间。

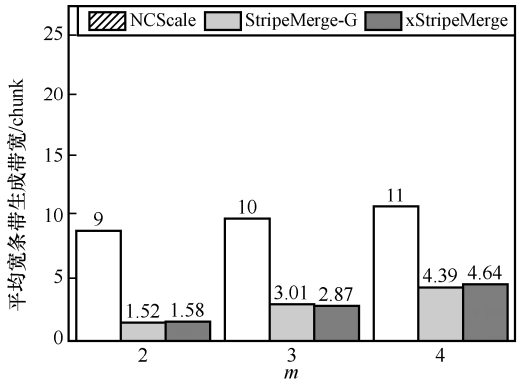
相比于目前已有的条带合并方法 StripeMerge[12]，xStripeMerge 存储同样大小的数据所需的额外存储开销更小。其原因可以用理论解释。RS 码存储数据的冗余度是由 $k+m$ 与 k 的比值决定的。StripeMerge 将 $\text{RS}(k,m)$ 转换为 $\text{RS}(2k,m)$，其冗余度从 $\dfrac{k+m}{k}$ 降低到 $\dfrac{2k+m}{2k}$。而 xStripeMerge 将 $\text{RS}(k,m)$ 转换为 $\text{RS}(xk,m)$，冗余度从 $\dfrac{k+m}{k}$ 降低到 $\dfrac{xk+m}{xk}$。xStripeMerge 需要利用贪心算法处理剩余无法在启发式搜索阶段得到合并方案的窄条带，在实验中发现 x=4 时需要贪心处理窄条带数量已经达到数百条，所需内存已经达到数 GB，因此目前 xStripeMerge 所取的 x 值的上限为 4。对于采用 RS(4,4) 的大规模存储系统，通过 StripeMerge 合并为宽条带后冗余度降低到 1.5×，而通过 xStripeMerge 合并为宽条带后可以最多将冗余度降低到 1.25×。相比于 StripeMerge，本文所提方法 xStripeMerge 可以减少 33%～50%的额外存储开销。

5.1　宽条带生成带宽

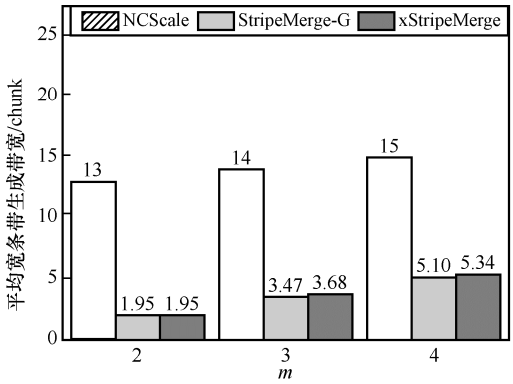
本文比较了 StripeMerge-G、NCScale 和 xStripeMerge 的平均宽条带生成带宽，即产生一个宽条带需要传输的数据块个数。由于贪心算法 StripeMerge-G 的空间复杂度为 $O(l^x)$，因此贪心算法的输入规模有限，即 x 值取值有限。本节进行了小规模（窄条带数量为 900 个）与大规模（窄条带数量为 10 000 个）2 种场景下的对比实验。小规模条带合并场景下的平均宽条带生成带宽比较如图 6 所示，其中，$4 \leqslant k \leqslant 10$，$2 \leqslant m \leqslant 4$，$x$=3。

总体看来，xStripeMerge 相比 NCScale 可以减少大量的条带生成带宽。例如，设定 $k=10$，$m=4$，$N=2(3k+m)$，xStripeMerge 相比 NCScale 可以减少 72%平均条带生成带宽。图 6(e)展示了 NCScale 和 xStripeMerge，StripeMerge-G 在参数设置为 $N=4(3k+m)$，$k=8$，$2 \leqslant m \leqslant 4$ 条件下的平均宽条带生成带宽。在 m 为 2 或 3 的条件下，其平均宽条带生成带宽相比 $N=2(3k+m)$ 小一些。其原因是节点数足够多，窄条带的数据块的位置分布更加分散，条带间数据块重叠的现象更不易发生。由于条带的合并成本由两部分决定，即数据块迁移成本和奇偶块传输成本，因此较分散的条带分布所需的数据块迁移成本会更小一些。

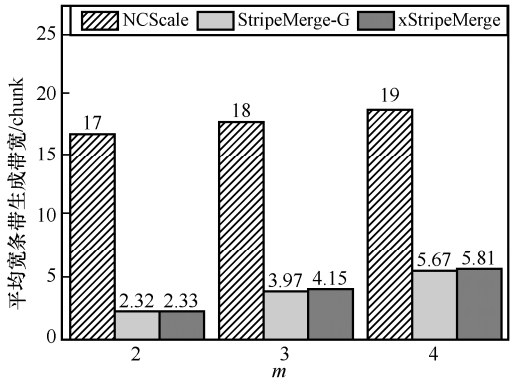
大规模条带合并场景下的平均宽条带生成带宽如图 7 所示。图 7(a)和图 7(b)分别展示了 x 为 3 和 4 下 NCScale 和 xStripeMerge 的平均宽条带生成带宽，其条带输入数均为 10×10^3。从图 7 可以看出，在大规模条带合并场景中，xStripeMerge 的性能表型仍然大幅优于 NCScale。例如，设定 $k=8$，$m=4$，xStripeMerge 相比 NCScale 可以减少 75.8%的平均条带生成带宽。StripeMerge-G 和 xStripeMerge 的性能



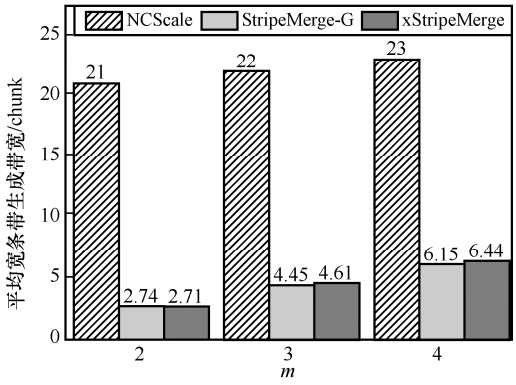
(a) N=2(3k+m), k=4



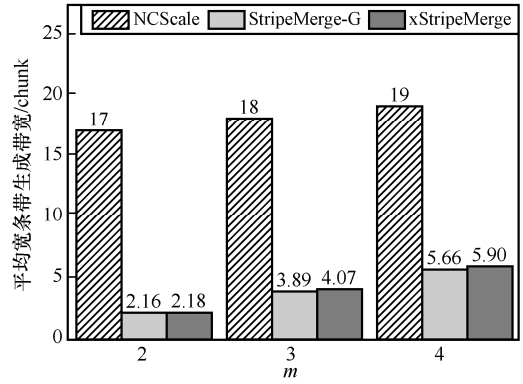
(b) N=2(3k+m), k=6



(c) N=2(3k+m), k=8

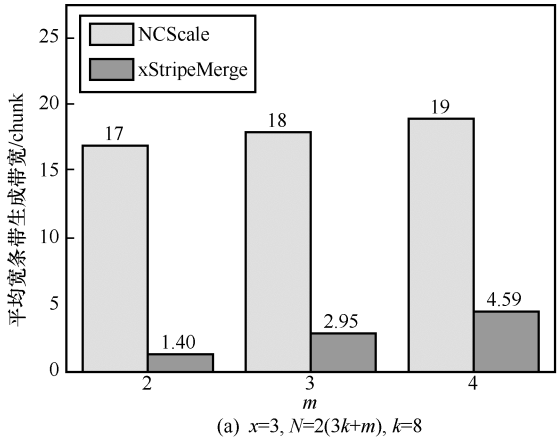


(d) N=2(3k+m), k=10

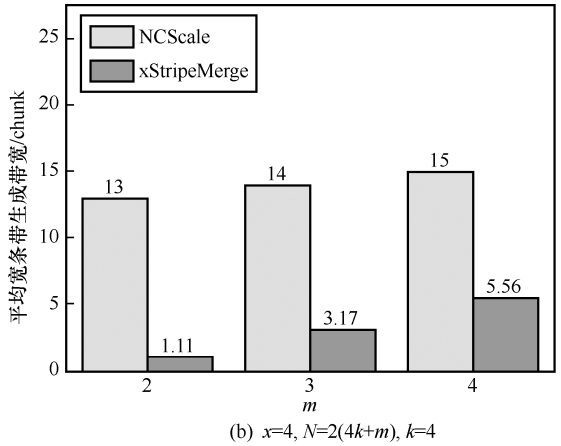


(e) N=4(3k+m), k=8

图 6　小规模条带合并场景下的平均宽条带生成带宽



(a) x=3, N=2(3k+m), k=8



(b) x=4, N=2(4k+m), k=4

图 7　大规模条带合并场景下的平均宽条带生成带宽

表现都优于 NCScale。StripeMerge-G 由于较高的空间复杂度，不适用于大规模条带合并场景。本文所提 xStripeMerge 在小规模和大规模条带合并场景下均有优异的表现。

5.2　运行时间

将窄条带合并为宽条带所需的时间由三部分构成，分别为算法生成传输计划时间（即算法运行时间）、通信时间和计算时间。由于 NCScale 生成的合并方案所产生的大量宽条带生成带宽会导致通信时间大幅增加，因此 NCScale 实际所需时间已经远多于其余 2 种方法。本节实验中仅比较了宽条带生成带宽相近的 StripeMerge-G 和 xStripeMerge 的算法运行时间。

实验设定条带数为[100,900]，$k=8$，$m=4$，节点数 $N=2(3k+m)$。2 种算法的实际运行时间如图 8 所示。从图 8 可以看出，当条带数取[100,900]时，而启发式贪心算法非常耗时，最多达到了 138 s，本文的 xStripeMerge 算法所用的时间不超过 3 s。

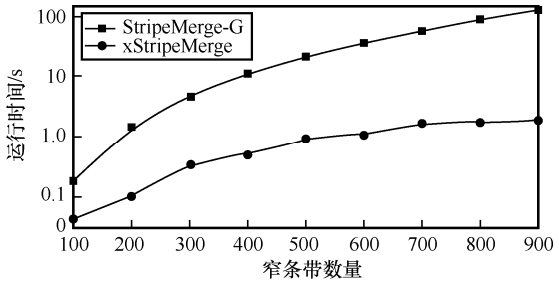


图 8　小规模条带合并场景下的算法运行时间

扩展的 StripeMerge 算法 StripeMerge-G 的总时间复杂度为 $O\left(l^x x(k+m)\right)$。而所提 xStripeMerge 的复杂度为 $O\left(x(k+m)(x-1)ml\right)$。因此，理论上 xStripeMerge 的时间性能也应该优于 StripeMerge-G。这与图 8 所示结果相符。

6　结束语

本文首先分析了目前宽条带生成方法中存在的问题，然后提出了纠删码中多条带合并下的宽条带生成并将其建模为组合优化问题。本文定义了多条带合并过程的 2 个关键算子即数据块迁移和奇偶块传输，并设计了基于多条带合并启发式搜索的高效宽条带生成方法 xStripeMerge 以解决上述组合优化问题。最后，本文实现了 xStripeMerge 原型并通过模拟实验证明，相比于存储扩展方法 NCScale，xStripeMerge 可以大幅减少大量宽条带生成过程中的数据传输开销。相比于扩展的 StripeMerge 算法 StripeMerge-G，所提 xStripeMerge 可以在更短的时间内获得与 StripeMerge-G 几乎具有相同宽条带生成带宽的条带合并方案，而且 xStripeMerge 可以应用于大规模条带合并场景。在今后工作中，将继续探索时间、空间复杂度更优异的算法以解决目前 x 选取的局限性问题。

参考文献：

[1] KADEKODI S, RASHMI K V, GANGER G R. Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk-reliability heterogeneity[C]//Proceedings of the 17th USENIX Conference on File and Storage Technologies. New York: ACM Press, 2019: 345-358.

[2] HU Y C, WANG Y S, LIU B, et al. Latency reduction and load balancing in coded storage systems[C]//Proceedings of the Symposium on Cloud Computing. New York: ACM Press, 2017: 365-377.

[3] CALDER B, WANG J, OGUS A, et al. Windows Azure storage: a highly available cloud storage service with strong consistency[C]//Proceedings of the 23th ACM Symposium on Operating Systems Principles (SOSP). New York: ACM Press, 2011: 143-157.

[4] ZHANG Y, LI H, LIU S, et al. PBS: an efficient erasure-coded block storage system based on speculative partial writes[J]. ACM Transactions on Storage, 2020, 16(1): 1-25.

[5] BRAKENSIEK J, GOPI S, MAKAM V. Generic Reed-Solomon codes achieve list-decoding capacity[C]//Proceedings of the 55th Annual ACM Symposium on Theory of Computing. New York: ACM Press, 2023: 1488-1501.

[6] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer System, 2013, 31(3): 1-22.

[7] SHVACHKO K, KUANG H R, RADIA S, et al. The Hadoop distributed file system[C]//Proceedings of IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). Piscataway: IEEE Press, 2010: 1-10.

[8] SHEN Z R, LEE P P C. Cross-rack-aware updates in erasure-coded data centers[C]//Proceedings of the 47th International Conference on Parallel Processing. New York: ACM Press, 2018: 1-10.

[9] MURALIDHAR S, LLOYD W, ROY S, et al. f4: Facebook's warm BLOB storage system[C]//Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation. New York: ACM Press, 2014: 383-398.

[10] OVSIANNIKOV M, RUS S, REEVES D, et al. The quantcast file system[J]. VLDB Endowment, 2013, 6(11): 1092-1101.

[11] KADEKODI S, SILAS S, CLAUSEN D, et al. Practical design considerations for wide locally recoverable codes (LRCs)[C]//Proceedings of the 21st USENIX Conference on File and Storage Technologies (FAST). Berkeley: USENIX Association, 2023: 1-16.

[12] YAO Q R, HU Y C, CHENG L F, et al. StripeMerge: efficient wide-stripe generation for large-scale erasure-coded storage[C]// Proceedings of IEEE 41st International Conference on Distributed

Computing Systems (ICDCS). Piscataway: IEEE Press, 2021: 483-493.

[13] ZHANG X Y, HU Y C, LEE P P C, et al. Toward optimal storage scaling via network coding: from theory to practice[C]//Proceedings of IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2018: 1808-1816.

[14] HUANG J Z, LIANG X H, QIN X, et al. Scale-RS: an efficient scaling scheme for RS-coded storage clusters[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(6): 1704-1717.

[15] PAMIES-JUAREZ L, BLAGOJEVIC F, MATEESCU R, et al. Opening the chrysalis: on the real repair performance of MSR codes[C]// Proceedings of the 14th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2016: 81-94.

[16] YE L Q, FENG D, HU Y C, et al. Hybrid codes: flexible erasure codes with optimized recovery performance[J]. ACM Transactions on Storage, 2020, 16(4): 1-26.

[17] 魏征, 窦禹, 高艳珍, 等. 一种基于条带的一致性散列数据放置算法[J]. 计算机研究与发展, 2021, 58(4): 888-903.
WEI Z, DOU Y, GAO Y Z, et al. A consistent hash data placement algorithm based on stripe[J]. Journal of Computer Research and Development, 2021, 58(4): 888-903.

[18] LIU C J, WANG Q, CHU X W, et al. ESetStore: an erasure-coded storage system with fast data recovery[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(9): 2001-2016.

[19] HU Y, CHENG L, YAO Q, et al. Exploiting combined locality for wide-stripe erasure coding in distributed storage[C]//Proceedings of the 19th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2021: 233-248.

[20] REN K, KWON Y, BALAZINSKA M, et al. Hadoop's adolescence[J]. VLDB Endowment, 2013, 6(10): 853-864.

[21] XIA M Y, SAXENA M, BLAUM M, et al. A tale of two erasure codes in HDFS[C]//Proceedings of the 13th USENIX Conference on File and Storage Technologies. Berkeley: USENIX Association, 2015: 213-226.

[22] WU S, DU Q P, LEE P P C, et al. Optimal data placement for stripe merging in locally repairable codes[C]//Proceedings of IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2022: 1669-1678.

[23] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system[C]//Proceedings of the nineteenth ACM Symposium on Operating Systems Principles. New York: ACM Press, 2003: 29-43.

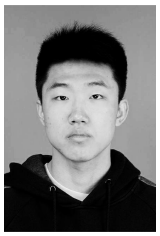
[24] RASHMI K V, SHAH N B, GU D, et al. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the Facebook warehouse cluster[C]//Proceedings of the 5th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley: USENIX Association, 2013: 8.

[25] 王意洁, 许方亮, 裴晓强. 分布式存储中的纠删码容错技术研究[J]. 计算机学报, 2017, 40(1): 236-255.
WANG Y J, XU F L, PEI X Q. Research on erasure code-based fault-tolerant technology for distributed storage[J]. Chinese Journal of Computers, 2017, 40(1): 236-255.

[26] ZHOU H, YANG Y H, LI W P. An erasure code-based approach to improve data recovery and update capability[C]//Proceedings of the International Conference on Mechanical, Electronic, Control and Automation Engineering. Paris: Atlantis Press, 2018: 401-409.

[作者简介]

郑美光（1983− ），女，江西新余人，博士，中南大学副教授、硕士生导师，主要研究方向为云边计算、云存储、联邦学习。



化韬斐（1998− ），男，山西介休人，中南大学硕士生，主要研究方向为存储编码、分布式存储系统等。

张心宇（1992− ），男，湖南浏阳人，中南大学博士生，主要研究方向为云存储、边缘计算和深度强化学习等。

胡志刚（1963− ），男，山西孝义人，博士，中南大学教授、博士生导师，主要研究方向为云边计算、机器视觉。