

基于数据对称打包的云计算并行核心失败校验缓解

闫 明¹, 王秀芬², 李 强¹

(1 天津理工大学 中环信息学院, 天津 300380;

2 天津农学院 计算机与信息工程学院, 天津 300384)

摘 要: 提出一种具有容错的云计算核心失败校验并行数据打包的前滚校验方法. 首先, 对云计算的核心失败校验算法进行介绍, 同时为避免数据溢出, 采用对称打包方式对输入进行并行计算, 并利用打包输出所包含的冗余特征, 提高连续核心故障的容错性能; 其次, 利用打包数据进行对应的通用矩阵乘法 (generic matrix multiplication, GEMM) 操作, 建立新的整数矩阵的乘法运算、卷积运算和故障恢复机制, 来缓解核心故障问题, 并对所提方案的计算复杂度进行理论分析; 最后, 通过在数据平均吞吐量、数据恢复率以及算法的计算时间三项指标上的对比实验, 验证了所提方法的有效性.

关键词: 对称打包; 云计算; 容错校验; 并行计算

中图分类号: TP391

文献标识码: A

文章编号: 1000—7180(2017)05—0073—06

Cloud Computing Parallel Core Failure Tolerance Verification Based on Data Symmetry Packing

YAN Ming¹, WANG Xiu-fen², LI Qiang¹

(1 College of Zhonghuan Information, Tianjin University of Technology, Tianjin 300380, China;

2 College of Computer and Information Engineering, Tianjin Agricultural University, Tianjin 300384, China)

Abstract: This paper here proposes a fault-tolerant cloud computing core failure check parallel data packing method. Firstly, the core of cloud computing failure check algorithm is introduced, at the same time to avoid data overflow, here use symmetric packing for parallel computing on the input and output using the redundant features contained in the package, which can improve the fault tolerance performance of continuous core fault; Secondly, the packet data corresponding to the GEMM operation, a new integer matrix the multiplication and convolution was construct to alleviate the core problem, and the computational complexity of the proposed scheme was analyzed in theory; Finally, the comparison experiments on the Amazon Web services optimization and image retrieval experiments show that the proposed method is effective.

Key words: symmetric packing; cloud computing; fault tolerance; parallel computing

1 引言

云计算集群为并行计算提供了可能, 与传统桌面多核系统计算相比可显著降低计算时间成本^[1]. 但是服务中断可能发生在不可预知的时间间隔内, 此外其他类型的中断, 如振动、功率或网络引起的性能下降, 经常出现在大型计算集群中, 这会严重影响并行计算性能^[2].

密集型程序的并行计算核心故障, 可通过回滚或前滚的方式进行缓解. 回滚方法^[3] 是对于检测到的故障, 通过设定周期检查点和重新计算进行故障缓解. 绝大多数的回滚方法是向后进行错误恢复 (Backward error recovery, BER)^[4-5], 对于计算故障通过系统状态定期存储和最后存储状态重新启动进行故障恢复. 几个 BER 研究表明^[6], 根据所需的核心失败等级, 可调动资源进行检查、系统状态存

储/恢复,并进行重新计算.该过程对于亿级系统是一个主要挑战.向前滚动方法可确保在不进行重新计算情况下,对故障核心的功能进行恢复,例子包括^[7]前向故障恢复方法(Forward error recovery, FER)^[8],这些方法的主要概念是对整个校验矩阵的行和列的乘积进行计算,并可用于输入矩阵或向量元素的校验^[9].如果核心故障被检测盗,那么这些额外的元素可用于求解 FER 线性方程组.前滚动方法相对于回滚方式具有更高的计算效率,并且计算容错性更好^[10-11].

本文所提方案包括一个新的整数矩阵的乘法运算和卷积运算,具体做法是对输入数据进行对称打包,并利用打包数据进行对应的 GEMM 操作.

2 问题描述

2.1 核心失败校验算法

考虑在 L 多核计算系统上的 L 并行矩阵乘积.每个核心的运算为 $M \times N$ 的矩阵 A_l ($0 \leq l \leq L$) 乘以 $N \times K$ 的内核流程矩阵 B ,从而使得多核系统乘以 L 后获得输出矩阵 R_l .这种情况会经常遇到,例如,在协方差矩阵计算,图像组的图像预测,以及分布式计算集群上运行高容量多媒体应用程序等.在所提方案中,如果 F 核心 ($1 \leq F < L$) 在设定时间内不返回结果,则表明 F 核心数据处理失败.

为便于进行故障恢复,基于 F 校验矩阵 $A_{c1}, \dots, A_{cF}$ 进行核心数据处理校验,校验矩阵的数目 F 与计算核心的失败数量有关,单一的简单校验矩阵可以原来对单一故障核心进行恢复:

$$R_l = A_l B, A_{c1} = \sum_{l=0}^{L-1} A_l \quad (1)$$

这个校验矩阵进行相同的 GEMM 操作, $R_{c1} = A_{c1} B$,这需要使用一个额外的计算核心.如果第 x 个核心出现失败 ($0 \leq x \leq L$),则可通过 $R_x = R_{c1} - \sum_{\forall l \neq x} R_l$ 对 R_x 进行恢复.如果 R_{c1} 的校验计算被忽略,那么该核心中产生数据处理失败.额外的校验通过多个矩阵 $A_0, A_1, \dots, A_{L-1}$ 的线性组合进行一个以上的失败校验.总的来说,为了容忍一个并行计算环境的核心失败, F 个其他核心预留计算线性校验矩阵的设定 $R_{cf} = A_{cf} B, 1 \leq f \leq F$.

2.2 数据打包计算

数值包装的概念最初提出是在多媒体应用中进行吞吐量尺度缩放,以及软错误容忍 GEMM 情况下的系统持续失败.打包处理过程中,将多个小动态

范围输入进行堆栈处理,利用整数打包因子 k 在标准的 32 位或 64 位表示中进行处理,从而避免数据溢出,实现对这些输入的并行计算.数据打包可以采取对称方式,对两个矩阵(或向量)的输入进行打包,或采取不对称方式,只对一个输入矩阵(或向量)进行打包.图 1 所示为两个向量的乘积的对称打包处理过程.

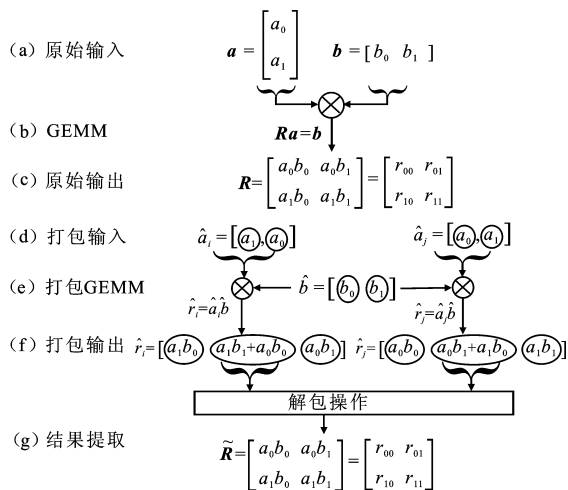


图 1 对称打包处理过程

在图 1 中,常规处理方式是执行 2×1 矩阵与 1×2 矩阵的相乘,同时在打包区域中,利用两个 1×1 向量相乘,确保以下打包过程完成: $b \rightarrow \hat{b}, a \rightarrow \hat{a}$, $\hat{a}_i = \hat{a}_i \hat{b}$, $\hat{r}_j = \hat{a}_j \hat{b}$.结果采取非打包形式进行输出.因此,在数据打包过程中,所需的计算总量相对于常规计算降低 50%.更一般的是,对于 $M \times M$ 整数矩阵与 $R = AB$ 的对称打包计算,打包过程会产生数据块 $\hat{A}_i, \hat{A}_j$ 以及 $\hat{B}$,其系数矩阵分别为 $0.5M \times M$ 和 $M \times 0.5M$,其计算形式可表示为

$$\begin{cases} \hat{A}_i = [(A_{\text{bot}}^T k) + A_{\text{top}}^T]^T \\ \hat{A}_j = [(A_{\text{top}}^T k) + A_{\text{bot}}^T]^T \\ \hat{B} = (B_{\text{top}} k) + B_{\text{bot}} \end{cases} \quad (2)$$

式中, $k \in N^*$, 并采取非打包形式计算 $\hat{R}_i = \hat{A}_i \hat{B}, \hat{R}_j = \hat{A}_j \hat{B}$. k 的取值取决于矩阵乘积的最大可能值,以确保数据恢复的准确性,却满足如下条件:

$$k > \log_2(\max |R_l|) + 1, 3k \leq W \quad (3)$$

其中,如果所有的处理过程是采用 32 位或 64 位整数表示的,那么 $W \in \{32, 64\}$;如果内核处理是采用单或双精度浮点表示的,那么 $W \in \{24, 52\}$.

3 基于卷积的 GEMM 并行打包方法

3.1 方案描述

考虑:(1) L 整数矩阵, $\mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_{L-1}$ ($L \geq 3$), 每个矩阵的维度是 $M \times N$, 必须通过一个 $N \times K$ 的整数核矩阵 $\mathbf{B}$ 进行处理;(2) L 整数输入信号向量 $\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_{L-1}$, ($L \geq 3$), 每个向量包含 N 个样本, 必须由 K 整数核心向量 $\mathbf{B}$ 进行过滤操作. 在这两种情况下, 每个操作是在 L 核集群上的不同的核心上进行操作. 这里提出了一种可以恢复所有 L 个 GEMM 操作结果的方法, 即使出现 $F = \lfloor L/3 \rfloor$ 连续核失败情形, 并且不需要额外的校验码输入. $F = \lfloor L/3 \rfloor$ 连续核失败恢复对于多核处理器计算集群非常重要, 可实现对多芯板故障连续失败问题进行解译. 为了实现这一点, 这里采取对称打包方式, 对两个输入和内核矩阵进行打包. 对称打包允许矩阵乘积的冗余溢出, 例如, 在图 1(f) 示例中的 K 位中间区域. 如本文所示, 如果构建适当, 这些冗余溢出可在核心故障中进行恢复.

3.2 算法步骤

步骤 1:(具有容错机制的 GEMM 打包方法) 对输入矩阵进行对称打包, 可获得 L 个打包后的输入矩阵 $\hat{\mathbf{A}}_0, \hat{\mathbf{A}}_1, \dots, \hat{\mathbf{A}}_{L-1}$, 其打包形式为

$$\hat{\mathbf{A}}_l = (\mathbf{A}_l \lll k) + \mathbf{A}_{(F+l) \bmod L} \quad (4)$$

式中, F 为要连续失败的核心数量, k 为打包因子. 例如, 对于 $L = 3$ 和 $F = 1$, 可得:

$$\begin{cases} \hat{\mathbf{A}}_0 = (\mathbf{A}_0 k) + \mathbf{A}_1 \\ \hat{\mathbf{A}}_1 = (\mathbf{A}_1 k) + \mathbf{A}_2 \\ \hat{\mathbf{A}}_2 = (\mathbf{A}_2 k) + \mathbf{A}_0 \end{cases} \quad (5)$$

对内核矩阵 $\mathbf{B}$ 进行打包, 该过程会产生 $N \times 0.5K$ 的矩阵 $\hat{\mathbf{B}}$:

$$\hat{\mathbf{B}} = (\mathbf{B}_{\text{top}} \lll k) + \mathbf{B}_{\text{bot}} \quad (6)$$

采用式(4)~(6)所示的打包因子 k , 会增加所有 $\hat{\mathbf{A}}_l$ 和 $\hat{\mathbf{B}}$ 的动态范围, 这里利用 64 位的表示方法对 $\hat{\mathbf{A}}_l$ 和 $\hat{\mathbf{B}}$ 进行构造.

步骤 2:(GEMM 打包计算) 通过利用 L 个 64 位的 GEMM 元胞在 L 个处理器核心上, 对 $M \times N$ 和 $N \times 0.5K$ 的两个矩阵进行相乘:

$$\forall l: \hat{\mathbf{R}}_l = \hat{\mathbf{A}}_l \hat{\mathbf{B}} \quad (7)$$

从而产生所有所需的结果, 以及一些“重复”的结果, 进行打包输出 $\hat{\mathbf{R}}_l$ 的数值表示. 图 2 所示为简单情况示例, 其为 1×1 与 1×2 矩阵的相乘, 其中 $L = 3, M = N = 1, K = 2$. 维度为 $M \times 0.5K$ 的输出矩阵 $\hat{\mathbf{R}}_0, \hat{\mathbf{R}}_1$ 和 $\hat{\mathbf{R}}_2$ 可分别表示为

$$\begin{aligned} \hat{\mathbf{R}}_0 &= \hat{\mathbf{A}}_0 \hat{\mathbf{B}} = (\mathbf{A}_0 \mathbf{B}_{\text{top}} \lll 2k) + [(\mathbf{A}_0 \mathbf{B}_{\text{bot}} + \mathbf{A}_1 \mathbf{B}_{\text{top}}) \lll k] + \mathbf{A}_1 \mathbf{B}_{\text{bot}} = (\tilde{\mathbf{R}}_{0, \text{top}} \lll 2k) + [(\tilde{\mathbf{R}}_{0, \text{bot}} + \tilde{\mathbf{R}}_{1, \text{top}}) \lll k] + \tilde{\mathbf{R}}_{1, \text{bot}} \end{aligned} \quad (8)$$

$$\begin{aligned} \hat{\mathbf{R}}_1 &= \hat{\mathbf{A}}_1 \hat{\mathbf{B}} = (\mathbf{A}_1 \mathbf{B}_{\text{top}} \lll 2k) + [(\mathbf{A}_1 \mathbf{B}_{\text{bot}} + \mathbf{A}_2 \mathbf{B}_{\text{top}}) \lll k] + \mathbf{A}_2 \mathbf{B}_{\text{bot}} = (\tilde{\mathbf{R}}_{1, \text{top}} \lll 2k) + [(\tilde{\mathbf{R}}_{1, \text{bot}} + \tilde{\mathbf{R}}_{2, \text{top}}) \lll k] + \tilde{\mathbf{R}}_{2, \text{bot}} \end{aligned} \quad (9)$$

$$\begin{aligned} \hat{\mathbf{R}}_2 &= \hat{\mathbf{A}}_2 \hat{\mathbf{B}} = (\mathbf{A}_2 \mathbf{B}_{\text{top}} \lll 2k) + [(\mathbf{A}_2 \mathbf{B}_{\text{bot}} + \mathbf{A}_0 \mathbf{B}_{\text{top}}) \lll k] + \mathbf{A}_0 \mathbf{B}_{\text{bot}} = (\tilde{\mathbf{R}}_{2, \text{top}} \lll 2k) + [(\tilde{\mathbf{R}}_{2, \text{bot}} + \tilde{\mathbf{R}}_{0, \text{top}}) \lll k] + \tilde{\mathbf{R}}_{0, \text{bot}} \end{aligned} \quad (10)$$

步骤 3:(结果更新) 对于 L 个打包输出矩阵 $\hat{\mathbf{R}}_0, \hat{\mathbf{R}}_1, \dots, \hat{\mathbf{R}}_{L-1}$, $L - F$ 矩阵即可满足所有 L 输出的恢复操作. 例如, 对于 $L = 3$ 和 $F = 1$ 情形, 任意 $\hat{\mathbf{R}}_0, \hat{\mathbf{R}}_1$ 和 $\hat{\mathbf{R}}_2$ 的两个输出矩阵可构建各种形式输出, 如图 2 所示. 例如, 假定 $\hat{\mathbf{R}}_0$ 和 $\hat{\mathbf{R}}_1$ 用来对 $\tilde{\mathbf{R}}_1, \tilde{\mathbf{R}}_2$ 和 $\tilde{\mathbf{R}}_0$ 进行恢复, 则所需公式为

$$\tilde{\mathbf{R}}_{0, \text{top}} = \tilde{\mathbf{R}}_0 \ggg 2k \quad (11)$$

$$T_0 = \tilde{\mathbf{R}}_0 - (\tilde{\mathbf{R}}_{0, \text{top}} \lll 2k) \quad (12)$$

$$T_1 = T_0 \ggg k \quad (13)$$

由于算法可利用普通的硬件进行整数表示, 那么 T_1 的所有负元素 (i, j) 大于或等于 $2k - 1$. 同时 $\tilde{\mathbf{R}}_{0, \text{top}}[i, j]$ 将会小于其正确值. 为了弥补算法影响, 首先定义中间矩阵 $\mathbf{E}_0$ 以存储 T_1 的签名值, 并调整 $\tilde{\mathbf{R}}_{0, \text{top}}$ 值过程如下:

S3-1: 对于 $\forall i, j$, 如果 $T_1[i, j] \geq 2^{k-1}$, 则设定 $\mathbf{E}_0[i, j] = T_1[i, j] - 2^k$, 转换为负数; 否则, 设定 $\mathbf{E}_0[i, j] = T_1[i, j]$, 不发生变化.

S3-2: 对于 $\forall i, j$, 如果 $\mathbf{E}_0[i, j] < 0$, 那么设定 $\tilde{\mathbf{R}}_{0, \text{top}}[i, j] \leftarrow (\tilde{\mathbf{R}}_{0, \text{top}}[i, j] + 1)$.

接下来, 进行类似的检查以提取 $\tilde{\mathbf{R}}_{1, \text{bot}}$ 的签名表示形式:

$$T_0 \leftarrow [T_0 - (T_1 \lll k)] \quad (14)$$

S3-3: 对于 $\forall i, j$, 如果 $T_0[i, j] \geq 2^{k-1}$, 那么设定 $\tilde{\mathbf{R}}_{1, \text{bot}}[i, j] = \tilde{\mathbf{R}}_{1, \text{bot}}[i, j] - 2^k$; 否则设定 $\tilde{\mathbf{R}}_{1, \text{bot}}[i, j] = T_0[i, j]$;

S3-4: 对于 $\forall i, j$, 如果 $\tilde{\mathbf{R}}_{1, \text{bot}}[i, j] < 0$, 那么设定 $\mathbf{E}_0[i, j] \leftarrow \mathbf{E}_0[i, j] + 1$.

类似的, 对 $\hat{\mathbf{R}}_1$ 执行与 $\hat{\mathbf{R}}_0$ 相似的过程, 以提取 $\tilde{\mathbf{R}}_{1, \text{top}}$ 和 $\tilde{\mathbf{R}}_{2, \text{bot}}$. 最后, 最后, 执行以下操作完成所有

结果的提取:

$$\begin{cases} \tilde{R}_{0,\text{bot}} = E_0 - \tilde{R}_{1,\text{top}} \\ \tilde{R}_{2,\text{top}} = E_1 - \tilde{R}_{1,\text{bot}} \end{cases} \quad (15)$$

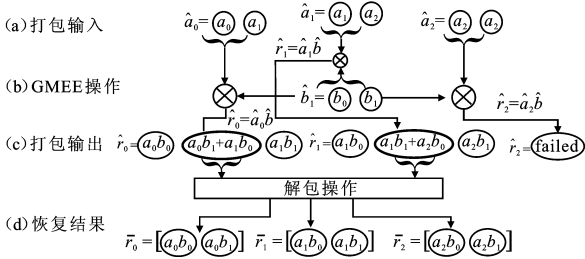


图2 整数矩阵乘积故障缓解实例

步骤4:(一维卷积前滚)同样的方法可以用来缓解核心故障问题,那么在 N 样本输入信号 a_l 与 K 样本核 b 可生产输出向量 r_l 形式为

$$r_l = a_l * b \Leftrightarrow r_l[i] = \sum_{k=0}^{K-1} a_l[i-k]b[k] \quad (16)$$

式中, $0 \leq i \leq N+K, 0 \leq l \leq L$. 具体而言,该算法将信号向量 a_l 进行打包,获得输入数据包 $\hat{a}_l$,同时卷积核 b 打包成 $\hat{b}$. 卷积形式为

$$\forall l: \hat{r}_l = \hat{a}_l * \hat{b} \quad (17)$$

然后,可采用任意形式的优化库对其进行优化,可获得 $(N + \frac{K}{2} - 1)$ 样本信号 $\hat{r}_0, \hat{r}_1, \dots, \hat{r}_{L-1}$. 此外, GEMM 的拆包过程遵循相同的计算流程,除了增加一个额外的求和运算. 例如,对于 $L = 3$,给定未打包输出 $\tilde{r}_{0,\text{top}}, \tilde{r}_{0,\text{bot}}, \tilde{r}_{1,\text{top}}$ 和 $\tilde{r}_{1,\text{bot}}$,其中“top”向量包含下标小于 $[0, N + \frac{K}{2} - 1]$ 的样本,而“bot”向量包含下标位于区间 $[\frac{K}{2}, N + K - 1]$ 的样本. 通过下面的联接操作可对输出信号进行恢复:

$$\forall l: \tilde{r}_l = [\tilde{r}_{l,\text{top}} 0_{K/2}] + [0_{K/2} \tilde{r}_{l,\text{bot}}] \quad (18)$$

式中, $0_{K/2}$ 表示 $1 \times \frac{K}{2}$ 的零向量.

4 计算复杂度理论分析

定理1 在不存在失败情况下,并行计算的 $N \times N$ 和 $N \times N$ 矩阵相乘,并假定一个 64 位操作相当于两个 32 位操作,则所提出的方法所需要的算术运算的数量是:

$$\frac{2NF + 5F - 8L - l}{2NL + 2NF - 2F} \times 100\% \quad (19)$$

该计算数量要小于传统校验方法所需要的算术

运算的数量.

证明:在所提出的方法中, L 输入矩阵 $A_0, A_1, \dots, A_{L-1}$ 的打包,同时 B 矩阵需要 $\frac{1}{2}N^2(2L+1)$ 的额外操作,而打包矩阵乘积需要 $\frac{L}{2}(2N^3 - N^2)$ 的计算操作. 在数据恢复阶段,需要 $\frac{7}{2}N^2(L-F)$ 数量操作在 $L-F$ 矩阵中进行输出提取. 因此,通过所有的算术运算总和进行倍增,可得所提方法的算术运算量为

$$Cx\{proposed\} = N^2(2NL + 8L - 7F + 1) \quad (20)$$

为缓解单核心失效或一组 F 连续核心失败,使用不加权的校验方法,共需要 $N^2(L-F)$ 数量的额外校验操作. 随后, $L+F$ 数量的 GEMM 需要进行计算,其对应的操作数量为 $(2N^3 - N^2)(L+F)$. 在这种情况下,算术校验运算复杂度为

$$Cx\{no\ checksum\ failures\} = N^2(2NF - 2F + 2NL) \quad (21)$$

根据式(20)、(21)推导结果,可知定理5所示的计算复杂度分析结果成立.

定理2 如果存在 F 个核心出现计算失败,并行计算的 $N \times N$ 和 $N \times N$ 矩阵相乘,并假定一个 64 位操作相当于两个 32 位操作,则所提出的方法所需要的算术运算的数量是:

$$\frac{2NF + 4F - 7L - 1}{2NF + 2NL - 3F + L} \times 100\% \quad (22)$$

在此情形下,该计算数量仍然要小于传统校验方法所需要的算术运算的数量.

证明:所提方法不论是否发生核心计算故障,其所需要的计算操作数量均为式(20)所示结果. 然而,如果出现一组连续的核心故障,校验法必须解决一个线性方程组来恢复丢失的数据. 假设失败的核与进行实际数据运算的核心相对应,输出结果不校验,则共需要 $N^2(L-F)$ 数量的数据恢复操作. 因此,在 F 个核心出现计算失败情况下,其校验操作数量为

$$Cx\{F\ checksum\ failures\} = N^2(2NF - 3F + 2NL + L) \quad (23)$$

根据式(20)、(23)的推导结果,可知定理5所示的计算复杂度分析结果成立.

5 实验分析

实验案例选取文献[12]所示的 AWS EC2 仿真案例,来自亚马逊购物平台. 实验硬件配置:处理器

为 Xeon E5-2666v3 2.9 GHz,操作系统为 Windows Server 2012,仿真软件为 Intel C++ 15.0.

5.1 数据块影响实验

本文采用对称打包方式对输入数据进行处理,由此可见数据块的大小对于算法的计算性能具有很大的影响,这里选取数据恢复率为 $P=90\%$ 情况下,算法随数据块大小变化的算法吞吐量、计算时间指标.数据块大小选取为 4~64 kB,校验轮数选取 $t=200,300,400$ 三种情况.实验对比如图 3、4 所示.

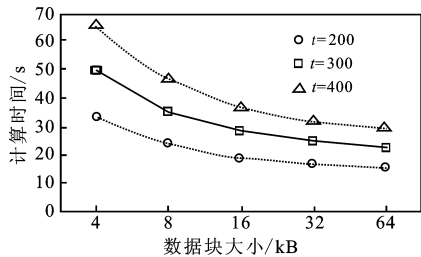


图 3 不同数据块大小下校验元的计算时间

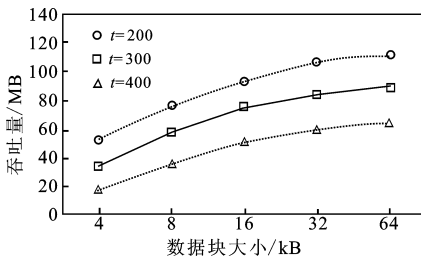


图 4 不同数据块大小下校验数据吞吐量

图 3 所示为数据块大小选取为 4~64 kB 情况下的算法计算时间变化情况.可见随着数据块大小的增加,算法的计算时间降低,这表明输入数据打包起到了简化数据操作的目的,但是同时也可以看出,随着打包数据的增大,其计算时间的降低幅度呈现逐渐降低的趋势,这主要是由于数据打包过程需要浪费一定计算时间所致.同时,可以看出随着校验轮数的增加,算法的计算开销是逐渐增大的.

图 4 所示为数据块大小选取为 4~64 kB 情况下的算法吞吐量变化情况.可见随着数据块大小大小增加,算法的吞吐量也随之增加,这表明输入数据打包起到了提高数据操作速率的目的,但是同时也可以看出,随着打包数据的增大,其吞吐量指标的增加幅度呈现逐渐降低的趋势,这主要是由于数据打包过程对算法执行过程起到一定反作用所致.同时,可以看出随着校验轮数的增加算法的吞吐量指标是逐渐降低的.

5.2 算法性能对比

为进一步验证算法的性能,将其后算法为单核

心校验方法^[13]、前滚校验算法^[14]和回滚校验算法^[9]进行对比.对比指标选取算法在 1~50 s 内的数据平均吞吐量、数据恢复率以及算法的计算时间均值.每种算法独立运行 100 次求评价指标的均值进行对比.吞吐量对比情况如图 5 所示,参数设定为: $L=16$,数据块大小为 32 kB.矩阵大小选取 100~10 000.

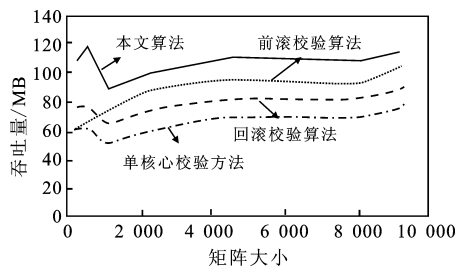


图 5 吞吐量对比结果

根据图 5 所示吞吐量对比结果可知,在算法计算吞吐量指标上,本文算法的吞吐量指标最大,表明算法在数据的处理速度上要优于选取的对比算法.同时可看出,前滚校验算法要优于回滚校验算法和传统的单核心校验算法.单核心校验算法的吞吐量指标最差,要远小于其他几种对比算法.

表 1 给出了对比计算方法的平均执行时间,参数设定: $L=16$,数据块大小为 64 kB. N 取值分为三种情况: $N=1\ 230, N=4\ 716$ 和 $N=9\ 364$.

表 1 平均执行时间对比				s
算法	$N=1\ 230$	$N=4\ 716$	$N=9\ 364$	
单核心方法	3 136	7 534	13 642	
回滚校验算法	987	1 357	1 893	
前滚校验算法	564	971	1 358	
本文算法	167	357	691	

根据表 1 可知,在计算时间上,几种算法均随着 N 取值的增加而不断增大,但是整体在计算时间对比上,本文算法的计算时间在几种算法中始终处于最小时间状态,这表明所提并行计算方法的有效性.同时可看出,单核心方法虽然采用了虚拟机并行方式,但是因为硬件的限制导致算法的计算时间很长,要远大于选取的对比算法.同时,可看出前滚校验算法在计算效率上要优于回滚校验算法.

表 2 给出了数据恢复率对比情况,参数设定: $L=16$,数据块大小为 64 kB. N 取值分为三种情况: $N=1\ 230, N=4\ 716$ 和 $N=9\ 364$.

表 2 数据恢复率对比情况 %

算法	N=1 230	N=4 716	N=9 364
单核心方法	82.3	79.6	75.3
回滚校验算法	91.6	88.7	84.6
前滚校验算法	93.6	89.4	87.9
本文算法	96.8	93.4	91.2

根据表 2 所示的对比情况可知,本文算法对于故障的恢复能力很强,在 $N=1\ 230$, $N=4\ 716$ 和 $N=9\ 364$ 三种情况下,能够始终保持在 90%以上.而对比算法中,前滚校验算法的故障恢复能力要优于回滚校验算法,两者均要优于传统的单核心校验方法,实验结果验证了算法的故障恢复能力.

6 结束语

本文提出一种具有容错的云计算核心失败校验并行数据打包的前滚校验方法,其本质是一种前滚校验算法,采用对称打包方式对输入进行并行计算,并利用打包输出所包含的冗余特征,提高连续核心故障的容错性能.同时利用打包数据进行对应的通用矩阵乘法操作,建立新的整数矩阵的乘法运算、卷积运算和故障恢复机制,来缓解核心故障问题,实验对比验证了算法的有效性.

参考文献:

[1] Liu Yanchen, Myung J Lee, Yanyan Zheng. Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system[J]. IEEE Transactions on Mobile Computing, 2015, 15(10): 2398-2410.

[2] 林伟伟,吴又泰. 面向云计算环境的能耗测量和管理方法[J]. 软件学报, 2016, 27(4): 1026-1041.

[3] Cole Freniere, Ashish Pathak, Mehdi Raessi, et al. The feasibility of amazon's cloud computing platform for parallel, GPU accelerated, multiphase-flow simulations[J]. Computing in Science & Engineering, 2016, 18(5): 68-77.

[4] Carlos Colman-Meixner, Chris Develder, Massimo Tornatore, et al. A survey on resiliency techniques in cloud computing infrastructures and applications[J]. IEEE Communications Surveys & Tutorials, 2016, 18(3): 2244-2281.

[5] 聂军. 基于 K-L 特征压缩的云计算冗余数据降维算法[J]. 微电子学与计算机, 2016, 33(2): 125-129.

[6] Xu Xiaolong, Cao Lingling, Wang Xinheng, et al. Resource pre-allocation algorithms for low-energy task scheduling of cloud computing[J]. Journal of Systems

Engineering and Electronics, 2016, 27(2): 457-469.

[7] Christian Napoli, Giuseppe Pappalardo, Giuseppe Marco Tina, et al. Cooperativestrategy for optimal management of smart grids by wavelet RNNs and cloud computing[J]. IEEE Transactions on Neural Networks and Learning Systems, 2016, 27(8): 1672-1685.

[8] 王宁,杨扬,孟坤,等. 云计算环境下基于用户体验的成本最优存储策略研究[J]. 电子学报, 2014, 42(1): 20-27.

[9] Li Kenli, Liu Chubo, Li Keqin, et al. A framework of price bidding configurations for resource usage in cloud computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(8): 2168-2181.

[10] Tulin Atmaca, Thomas Begin, Alexandre Brandwajn, et al. Performance evaluation of cloudcomputing centers with general arrivals andservice[J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(8): 2341-2348.

[11] Wu Zebin, Yonglong Li ,Antonio Plaza, et al. Parallel and fistributed fimensionality reduction of hyperspectral fata on cloud computing srchitectures[J]. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2016, 9(6): 2270-2278.

[12] Raazia Sosan, Choudhry Fahad Azim. RETRACTED ARTICLE: Mobile cloud computing: the taxonomy and comparison of mobile cloud computing application models[J]. Wireless Personal Communications, 2016, 89(4): 1435-1441.

[13] Marten J van Sinderen, Ivan I Ivanov. Editorial for special section of grid computing journal on "Cloud Computing and Services Science"[J]. Cluster Computing, 2015, 18(1): 453-455.

[14] Zhang Fujun, Gao Zunsheng, Ye Quanhui. Construcion of cloud platform for personalized information services in digital library based on cloud computing data processing technology[J]. Automatic Control and Computer Sciences, 2015, 49(6): 373-379.

作者简介:

闫 明 男,(1977-),硕士,副教授,高级工程师.研究方向为计算机应用技术、网络技术.
E-mail:yanminggreat@qq.com.

王秀芬 女,(1977-),硕士,副教授.研究方向为数学建模与计算机仿真.

李 强 男,(1979-),硕士,教授.研究方向为计算机应用技术、网络技术.