文章编号:1671-4229(2022)03-0014-15

基于分离逻辑的云存储系统并发正确性验证

王子豪，王捍贫*
(广州大学 计算机科学与网络工程学院，广东 广州 510006)

摘 要：云存储系统的应用日益广泛,其可靠性尤为受研究者关注。现有的基于块分离逻辑的验证方法能很好地描述和验证具备“文件 - 块 - 数据”三层架构的云存储系统的各类性质,文章在此基础上提出一种验证云存储系统并发正确性的方法,工作包括:①扩展云存储系统的建模语言,增加对线程间通信等并发行为的描述;②扩展断言语言,用于描述云存储环境下与并发执行和存储控制相关的系统性质;③提出一组霍尔规则,用于推导系统的性质。文章给出的验证实例能说明该方法的有效性。文章的工作扩展了块分离逻辑的适用范围,为 MapReduce 等框架下的并发程序正确性验证工作提供了理论基础。

关键词：分离逻辑；并发程序验证；云存储系统
中图分类号：TP 311.1　　**文献标志码**：A

A concurrent separation logic framework for cloud storage system

WANG Zi-hao, *WANG Han-pin**
(School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 510006, China)

Abstract: More and more applications rely on Cloud Storage Systems (CSSs) and the reliability of CSSs is attracting more attention. File-Block-Data structure is a common way of CSSs and natures of it can be described by the verification system based on Separation Logic with Blocks. This paper further proposes a system for reasoning about the concurrent management programs of CSSs, including: ① A Modeling Language to describe the concurrent actions of CCS management programs; ② An Assertion Language, so that verifiers can follow the communications and ownership transfers between the threads; ③ A set of Hoare Rules to reason about the system. There is an example to show how the system works. This work extends the verified objects of the Separation Logic with Blocks and it can be a possible method to validate the concurrent programs under concurrent frames like MapReduce in theory.

Key words: separation logic; concurrent program verification; cloud storage system

随着数据的快速增长,高性价比的海量存储系统已成为刚需,基于块的云存储系统又因其低成本、易扩展的特点而成为最经济可靠的选择[1]。市场研究报告表明,2020 年我国云存储市场规模已达 400 亿元[2]。随着公共云平台的推广,依赖于云存储服务的应用日益增加,云存储故障可能会造成意想不到的负面影响。例如,2016 年微软的云存储平台宕机以致部分用户在长达 8 天的时间内无法获得邮件服务[3];2018 年腾讯云存储发生故障,多重冗余失效,导致部分客户的数据灭失[4]。云存储系统的可靠性成为了研究热点。

验证云存储系统的正确性是提高系统可靠性的有效方法。形式化验证方法可以分析、证明程序的正确性,尽管代价略高,却能从逻辑角度保证程序中不存在特定的错误[5]。Reynolds[6] 提出的分离逻辑是霍尔逻

作者简介:王子豪(1997—),男,硕士研究生. E-mail:wangzh@e.gzhu.edu.cn
* 通信作者. E-mail:whpxhy@pku.edu.cn

引文格式:王子豪, 王捍贫. 基于分离逻辑的云存储系统并发正确性验证[J]. 广州大学学报(自然科学版),2022,21(3):14-28.

辑[7]的一种扩展，是描述和验证含地址操作和复杂数据结构的程序的有效工具[8]。文献[9]基于分离逻辑提出了块分离逻辑，用于描述云存储系统特有的存储结构和验证云存储管理程序的正确性，但需进一步考虑对系统并发正确性的验证方法。

并发程序的执行过程复杂，执行结果易受外界影响，其正确性验证一直是学界关注的热点。早在 1976 年 Owicki 等[10-11]便基于霍尔逻辑提出了“依赖 - 保障”方法，刻画共享内存冲突，验证并发程序的正确性。2007 年，在分离逻辑出现后，O′Hearn[12]进一步提出并发分离逻辑（Concurrent Separation Logic，CSL），认为并发正确性验证的难点在于不同线程共享存储区域所引起的冲突，同时给出了针对互不干扰的并发线程的正确性验证规则。在此基础上，CSL 进一步提出了资源所有权假设和分离性假设，规定一般的存储单元不可共享、可共享的存储单元需满足特定性质。上述规定避免了并发冲突的产生，使得 CSL 成为基于分离逻辑的并发程序正确性验证的基础方法。

为提升并发程序正确性验证方法的可用性，研究人员在 CSL 的基础上引入了不同的辅助机制。例如：

（1）资源权限。在 CSL 中，所有权假设已经为资源引入了权限机制。Dinsdale-Young 等[13]在此基础上提出了并发抽象谓词（Concurrent Abstract Predicate，CAP），用于描述指定共享空间的性质及其所允许执行的动作，并预先完成小模块的验证。CAP 细化了并发验证的粒度[14]。后续的改进包括 TaDA[15]、HOCAP[16]等验证方案，为线程访问共享资源和线程间通信提供了更灵活的描述方式。另有针对存储单元本身的计数式权限模型，能更直观地反映存储资源的使用状态。如 Bizjak 等[17]提出的 Iron 验证方案，为存储资源赋予分数量化每个进程所使用的存储资源，通过分数变化的追踪实现对失控存储单元的识别，帮助验证者发现内存泄漏错误。

（2）协议机制。协议机制即消息传递机制，主要用于描述线程间的交互行为。在 CSL 中，线程间的交互动作通过资源完成，交互的具体动作受限于对应的资源不变式。协议机制倾向于详细追踪线程间的交互过程，为验证者提供了更为灵活的描述方法。例如，Lei 等[18]提出了一种基于事件和通信代理的并发通信正确性验证模型，针对异步非阻塞通信程序给出了 2 组验证实例，证明了并发程序局部化、模块化验证的可行性。Hinrichsen 等[19]提出的 Actris 验证方案更可描述消息所附带的存储关系、数量关系等性质，同时支持互斥体等并发规范，并提出了 MapReduce 框架下并发程序正确性验证的初步思路。上述方案在不同程度上支持对历史状态的描述和推理[20]。

（3）幽灵状态。幽灵状态是描述系统性质时有助于确定真实执行进度但不影响程序实际执行的辅助部分，一种典型的实现方法是在程序或断言中引入不参与运算的变量并在验证过程中持续跟进。在幽灵状态机制下，验证者可以直接描述一些存储单元难以直观反映的系统性质而不必关心是否有相应的命令执行过程，基于此提出的方法有 Turon 等[21]提出的 GPS 验证方案和 Jung 等[22]提出的高阶幽灵状态等。Clochard 等[23]提出的幽灵状态监视机制则将程序正确性转化为执行程序过程中幽灵状态的正确性，验证者可以在不了解程序具体执行过程的情况下完成验证工作。幽灵状态机制能很好地与其他工具相结合，如前文提及的 Iron 验证方案中的权限分数便是幽灵状态的一种，Iris 验证方案[24]也为包括幽灵状态在内的系统状态描述方式提供了数学基础。

云存储系统不仅是一个单纯的并发程序，它与本地机器一同构成了独特的并发执行环境。在逻辑上，以固定大小的数据块为基本存储单元所构成的“文件 - 块 - 数据”3 层结构与分离逻辑所描述的“变量 - 可寻址内存”2 层结构有显著差异；在执行上，3 层存储架构的实现有赖于本地机器的 2 层存储架构。上述并发方案并未给出存储结构改变后的处理方式。为此，本文基于块分离逻辑提出一种云存储系统并发正确性验证方法，并给出相应的验证实例证明本文所提方法的有效性。

在存储资源控制上，本文通过扩展断言语言在块结构上实现了计数式权限机制，辅助验证者推理线程间竞争行为的性质。计数机制属于幽灵状态，能够直观反映可用存储空间的大小，可以更方便地描述云存储中“大小固定的数据块”以及本地机器上“不便扩展的内存资源”，实现 2 层结构与 3 层结构的统一。对块本身的计数限制并不影响块数量的扩增，云存储系统易扩展的特性和 CSL 原有的灵活程度不受减损。验证者也可以通过全局和局部资源计数的变化识别内存溢出和泄漏风险。

针对线程间交互，本文考虑异步通信环境和云存储系统对存储资源的管理需求，选择以信道作为描述进程间交互的工具，并相应地更新了建模语言、断言语言和状态更新规则。信道作为全局共享资源，同时支持数值信息的非阻塞传递以及存储资源使用权的显式转移。

本文将使用权转移视为辅助验证的标记之一,允许验证者以更直观的方式追踪交互细节。

本文第1节介绍预备知识和相关工作;第2节提出云存储管理系统的并发建模语言,当中新增了与信道相关的命令并完善了对系统状态的描述;第3节提出基于块分离逻辑的断言语言,在块断言中引入了计数机制,新增信道断言分量以描述线程间通信的性质;第4节提出一组验证规则,给出了与新命令相关的公理以及与信道相关的状态更新规则;第5节给出一组验证实例;第6节总结全文。

1 预备知识

霍尔逻辑和分离逻辑的特征都是形如

$$\{P\}C\{Q\}$$

的霍尔三元组。在部分正确性假设下,三元组成立意指在满足前置断言 P 的情况下执行程序 C,若能终止则得到的新状态必能满足后置断言 Q。分离逻辑在逻辑上将存储器划分为2层:被称为堆(Heap)的可变存储区和被称为栈(Store)的不可变存储区。其中堆以地址为索引,用于实现寻址操作;栈以变量名为唯一访问依据,功能与霍尔逻辑一致。这种结构可以很直观地描述指针等与地址相关的操作。

分离逻辑中的"分离"指一个堆可以被划分为多个互不相交的部分,多个互不相交的小堆也可以合取为一个堆。分离合取以符号"$*$"表示。在此定义下,形如

$$p * p$$

的式子将因$*$号两侧的子堆拥有相同地址的单元恒为假,即在一个有效的断言中相同地址的存储单元只能出现一次。这个要求解决了可变存储区潜在的别名问题。

分离逻辑同时提出局部推理规则(Frame Rule)

$$\frac{\{P\}C\{Q\}}{\{P * frame\}C\{Q * frame\}}。$$

局部推理规则使得验证者只需关注与命令直接相关的存储单元的局部状态,降低了验证复杂度。局部推理规则支持组合验证,为对局部状态的推理扩展至对全局状态的推理提供了路径。

1.1 基于分离逻辑的并发程序验证技术

并发分离逻辑(CSL)是基于分离逻辑的并发程序验证技术的重要基础,已衍生出大量并发程序验证工具[20,25]。

CSL首先就完全不共享存储单元的并发程序给出了推理规则,即互不干涉的并发推理规则,定义如下:

$$\frac{\{P_1\}C_1\{Q_1\} \quad \{P_2\}C_2\{Q_2\}}{\{P_1 * P_2\}C_1 \parallel C_2\{Q_1 * Q_2\}}。$$

其中,线程1对线程2不构成干扰是指线程1的执行不影响线程2中任一命令的前置条件,不改变其自由变量。

互补干扰条件不足以描述并发程序的行为。CSL进一步提出资源所有权假设和分离性假设:一般存储单元只可被一个线程使用,可被复用的存储单元应归属于某一资源组;在任意时刻,一个资源组最多被一个线程使用,且系统的所有存储单元可被线程和资源组完整划分;资源组被使用前后均需使对应的资源不变式成立。在此限制下,线程间的交互由资源组实现,交互的结果及其影响范围由资源不变式提前给定。使用资源时的验证规则定义如下:

$$\frac{\{P * RI_r\}C\{Q * RI_r\}}{\{P \wedge B\}\ \mathbf{with}\ r\ \mathbf{when}\ B\ \mathbf{do}\ C\ \mathbf{end\ with}\ \{Q\}}。$$

上述规则限制了进程间的交互,规避了并发执行可能带来的冲突。

锁式权限机制能够帮助控制并发冲突,分数式权限机制可以帮助验证者识别并发执行过程中资源分配不当的问题。例如,在Iorn方案[17]中存储资源获得的权限分数与线程无关,与资源大小无关,分数值仅取决于验证时执行分离和分离合取的次数。Iorn为系统初始堆的分数赋值为1,其后每次分离动作会将原堆的分数均分至划分所得的子堆上,而每次执行分离合取的所得的堆的分数为原子堆的分数和,即

$$\begin{gathered}\mathbf{emp}_\pi \dashv\vdash \mathbf{emp}_{\pi/2} * \mathbf{emp}_{\pi/2} \\ \mathbf{emp}_{\pi1} * \mathbf{emp}_{\pi2} \vdash \mathbf{emp}_{\pi1+\pi2}\end{gathered}。$$

权限分数不影响程序分配、释放、读写等命令的执行,仅反映验证过程。若权限分数出现损失、全局分数和不等于1,则证明验证过程遗漏了对部分可变存储区的控制,可能存在内存泄漏风险。验证者可以追溯局部分数的变化定位出错的代码段。

1.2 基于分离逻辑的云存储系统验证技术

云平台为并发程序提供了特殊的运行环境,其本身也是一个计算机程序。保障平台本身的正确性是验证平台上各类并发程序正确性的基础之一。

云存储系统中的块结构有以下功能和特点:块内存储具体数据、分散存放,块与块结合组成文件;用户对数据的读写和系统对结构的维护以块为最小单元;系统通过块定位存储数据的具体机器乃至具体位置。块本身具备了可寻址、可扩展的特性,且块与数据之间也存在"地址-数据"的映射。王捍贫等[9,26-28]就块结构的描

述、推理和验证做了一系列的研究工作。这些工作将分离逻辑原有的 2 层存储架构扩展为 Store、$Heap_B$ 和 $Heap_V$ 3 层结构,分别代表不可变存储区、可变的块存储区和可变的本地存储区。$Heap_B$ 和 $Heap_V$ 各自拥有堆所具备的分离属性,Store、$Heap_B$ 和 $Heap_V$ 3 者之间可构成“文件 - 数据块 - 数值”的 3 层体系,Store 和 $Heap_V$ 两者之间可以构成“变量 - 数值”的传统 2 层体系。

文献[9]提出的验证方法由建模语言、断言语言和规范语言共同构成。在建模语言中,该方法引入了与文件和块结构相关的变量和命令,改善了存储架构并相应地扩展了论域。有关表达式的语法详列如下:

$$e::=n\mid x,y,\cdots\mid e_1+e_2\mid e_1\times e_2\mid e_1 \bmod e_2\mid \#f\mid \#bk$$

$$fe::=\mathbf{nil}\mid f\mid fe\cdot bk\mid fe_1\cdot fe_2$$

$$bk::=n\mid b\mid f.e$$

$$be::=e_1=e_2\mid e_1\leqslant e_2\mid bk_1==bk_2\mid \mathbf{true}\mid \mathbf{false}\mid \neg\ be\mid be_1 \wedge be_2\mid be_1\vee be_2。$$

经扩增后的命令语法详列如下:

$$\begin{aligned}C::=\ &x:=e\mid x:=\mathbf{cons}(\bar{e})\mid x:=[e]\mid [e]:=e'\\ &\mid \mathbf{dispose}(e)\mid f:=\mathbf{create}(bk^*)\\ &\mid \mathbf{attach}(f,bk^*)\mid \mathbf{delete}f\mid b:=bk\\ &\mid x:=\{bk.e\}\mid b:=\mathbf{allocate}(\bar{e})\mid \mathbf{delete}bk\\ &\mid \mathbf{append}(bk,e)\mid f.e:=bk\mid C;C'\\ &\mid \mathbf{if}\ be\ \mathbf{then}\ C\ \mathbf{else}\ C'\mid \mathbf{while}\ be\ \mathbf{do}\ C。\end{aligned}$$

在断言语言部分,该方法沿用分离逻辑的断言语言描述与块无关的系统性质,引入块断言描述与块相关的系统性质,2 个分量共同构成全局断言以完整描述系统状态。其中,块断言具体语法如下:

$$\begin{aligned}\beta::=\ &\mathbf{true}_B\mid \mathbf{false}_B\mid \neg\ \beta\mid \beta_1\wedge\beta_2\mid \beta_1\vee\beta_2\\ &\mid \beta_1\rightarrow\beta_2\mid bk==bk_1\circledast\cdots\circledast bk_n\mid \forall x.\beta\\ &\mid \exists x.\beta\mid \forall b.\beta\mid \exists b.\beta\mid \forall f.\beta\mid \exists f.\beta\mid \mathbf{emp}_B\\ &\mid bk\mapsto\bar{e}\mid \beta_1*\beta_2\mid \beta_1-*\beta_2\mid fe=fe'\\ &\mid b_1\hookrightarrow b_2\mid bk_1==bk_2。\end{aligned}$$

在云存储环境下,并发程序之间不仅需要竞争云存储资源,还需要与本地机器上的其他线程竞争本地存储资源。即使将云存储资源视为可扩展资源,对于特定的数据或数据块的使用权却不可随意扩张,资源的竞争和调度受云存储管理系统控制。同时,本地内存资源有限、扩展不易,资源竞争带来的内存复用难以避免。现有工具对上述扩张限制的描述较为复杂,容易遗漏。

2　并发云存储管理系统建模语言

并发云存储管理系统的建模语言是 ML_{BCSS}[9] 的扩展,新增的元素主要用于描述和支持信道等并发机制。

2.1　信道机制设计

信道机制用于满足线程间的交互需求,主要是平台管理程序与其他任务线程的通信需求,包括传输数值和存储资源使用权的上收下发。本文以信道模拟和描述两者的通信过程:每个信道都是缓存队列,信道内分别对数值信息和块权限采用先进先出的交互方式;为确保写入成功,信道容量不设限制;以非阻塞方式读取,目标信道无消息可读是不改变系统状态、执行下一命令。

对于线程的执行而言,在读取动作完成前不可得知信道的具体状态、不可利用信道内的数值信息或存储资源权限;在读取动作完成后,被读取的信息出队、被读取的块权限归线程所有。线程向信道输出数值时,需要额外在存储器中作写入,直至该数值被读取后对应的存储单元方可被释放,输出数值是一个明显改变内存状态的操作。在转移数据块使用权时,涉及的是数据块的使用权而并未改变数据块的内容或者数据块与分布式文件之间的关系,即未对系统状态作实质性的改变。综上,本文新增一函数 $Store_C$,用于描述信道内的数值信息,定义如下:

$$Stores_C \triangleq CHVar\rightarrow Values*Values*=\{(n_1,\cdots,n_k)\mid n_i\in Valurs,k\in\mathbb{N}\}。$$

经信道转移的存储权限可用下式表示:

$$b.n.(l_1,\cdots,l_i),$$

式中包含 3 个分量,分别是块表达式 b,指代部分使用权被转移的块;一个不大于块 b 容量的整数 n,指代被转移的存储单元数量;一个长度不大于 n 的地址序列 $(l_1,\cdots,l_i)$、且序列中的本地地址应属于 b,列明被转移权限的非空闲单元所存储的具体数据。

2.2　建模语言语法

在表达式部分,本文沿用 ML_{BCSS} 所有的数值表达式、文件表达式、块表达式和布尔表达式,另新增一类信道表达式 che。在命令部分,新增与信道相关的命令、修改与数据块容量相关的语句。其中信道相关命令根据 2.1 小节的说明给定,块命令则配合计数式权限机制作相应修改。增改后的语法详列如下:

$$e::=n\mid x,y,\cdots\mid e_1+e_2\mid e_1\times e_2\mid e_1 \bmod e_2\mid \#f\mid \#bk\mid \$bk$$

$$fe::=\mathbf{nil}\mid f\mid fe\cdot bk\mid fe_1\cdot fe_2$$

$$bk::=n\mid b\mid f.e$$

$$be::=e_1=e_2\mid e_1\leqslant e_2\mid bk_1==bk_2\mid \mathbf{true}\mid \mathbf{false}\mid \neg\ be\mid be_1\wedge be_2\mid be_1\vee be_2$$

$che::=\varnothing\,|\,n\,|\,bk.n.\bar{e}\,|\,ch\,|\,che_1\cdot che_2$

$C::=x:=e\,|\,x:=\mathbf{cons}(\bar{e})\,|\,x:=[e]\,|\,[e]:=e'$

$|\,\mathbf{dispose}(e)\,|\,f:=\mathbf{create}(bk^*)$

$|\,\mathbf{attach}(f,bk^*)\,|\,\mathbf{delete}f\,|\,b:=bk$

$|\,x:=\{bk.e\}\,|\,b:=\mathbf{allocate}(n,\bar{e})\,|\,\mathbf{delete}bk$

$|\,\mathbf{append}(bk,e)\,|\,f.e:=bk\,|\,\mathbf{remove}(bk,e)$

$|\,C;C'\,|\,\mathbf{if}\ be\ \mathbf{then}\ C\ \mathbf{else}\ C'\,|\,\mathbf{while}\ be\ \mathbf{do}\ C$

$|\,ch:=\mathbf{newch}()\,|\,\mathbf{send}(ch,e)\,|\,x:=\mathbf{tryre}(ch)$

$|\,\mathbf{pass}(ch,bk,n,\bar{e})\,|\,\mathbf{get}(ch)\,|\,\mathbf{close}(ch)$

$|\,C_1\parallel C_2$。

新增的信道变量包括空信道、数值信息、存储资源使用权3种元素以及他们的自由组合,用于描述信道的状态。存储资源使用权一式的涵义与2.1小节所述一致。

新增语句主要涉及信道操作,可满足管理线程与任务线程间的通信需求。另外,新增的块容量限制将影响原有的块命令。增改后的命令释义详列如下:

- allocated,分配新的数据块并给定容量上限;
- append,数据块追加写,受块容量限制;
- delete,删除数据块并释放对应的存储单元;
- remove,释放数据块内指定的存储单元;
- newch,开设新的信道;
- close,停用没有待收取消息的信道;
- send,向信道传出数值消息;
- tryre,尝试从信道内取出数值消息;
- pass,向信道内传出数据块使用权;
- get,尝试接收数据块使用权。

2.3 机器状态描述

系统的状态由7个函数组成,其中包括LR_{BCSS}中使用的5个函数:由本地变量到本地存储地址的映射$Stores_V$、由数据块变量到数据块地址的映射$Stores_B$、由文件变量到一组数据块变量的映射$Stores_F$、由已使用的本地存储地址最终指向数值的映射$Heaps_V$、由已使用的数据块地址最终指向一组本地存储地址的映射$Heaps_B$。新增的2个函数$Stores_C$和Size分别表示信道的状态和数据块的容量上限。各个函数的具体定义如下:

$\mathrm{Values}\triangleq\mathbb{Z}\mathrm{Loc}$, $\mathrm{BLoc}\subseteq\mathbb{N}\in\mathrm{Values}$

$\mathrm{Var}\triangleq\{x,y,\cdots\}$ $\mathrm{BKVar}\triangleq\{b_1,b_2,\cdots\}$

$\mathrm{FVar}\triangleq\{f_1,f_2,\cdots\}$ $\mathrm{CHVar}\triangleq(ch_1,ch_2,\cdots)$

$Stores_V\triangleq Var\to Values$

$Stores_B\triangleq BKVar\to BLoc$

$Stores_F\triangleq FVar\to BLoc^*$

$Heaps_B\triangleq BLoc\to Loc^*$

$Heaps_V\triangleq Loc\to Values$

$Stores_C\triangleq CHVar\to Values^*$

$Size\triangleq BLoc\to\mathbb{N}$。

为方便起见,各类数值都限于整数范围内,各类地址限于非负整数范围内,则表示不包含任何地址的**nil**可以被定义为-1。

根据2.1小节给出的信道机制,数值信息的读写操作将改变存储器的状态;块权限的转移在执行时不改变存储器内的映射关系,在验证时影响各线程可用的存储区。因此,$Stores_C$是一个由信道变量指向数值队列的映射,而通过信道转移的数据块使用权仅用于辅助推理。

块的容量限制主要用于对云存储系统中块结构以及机器中有限内存资源的建模。在各类分布式存储系统中,数据块可以视为存储的基本单位,容量大多是固定的。如HDFS通常会将一个大文件划分为多个64 MB的数据块,TFS则会将多个小文件整合成为64 MB的数据块。在机器中,扩展内存资源通常需要停机后更新硬件才得以实现。模型中的数据块容量上限在数据块建立之初被指定,但在此后不可扩增,以表征云存储中固定大小的块结构和机器中内存不可增长的特性。在程序运行时,允许进程将自己所拥有的数据块作划分而不改变数据块的总容量,以支持对本地内存调度的建模。因此,Size是一个由已使用的数据块地址指向自然数的映射。

2.4 建模语言的指称语义

指称语义为语句和表达式在特定状态下的具体意义给出说明。机器的状态由一个七元组给定,但并非所有表达式或所有命令的释义都同时依赖于所有的状态函数。简便起见下文只列出释义所依赖的必不可缺的部分,给出本文新引入的表达式和语句的语义。

新增的信道表达式的语义给定如下:

$[\![n]\!](s_C)=n$;

$[\![\varnothing]\!](s_C)=()$;

$$[\![ch]\!](s_C)=\begin{cases}\varnothing & if\ ch\ with\ no\ int\ massage\\(n_1,\cdots,n_k) & if\ ch\ with\ k\ int\ massages\end{cases}$$

$[\![che_1\cdot che_2]\!](s_C)=(loc_1,\cdots,loc_i,loc'_1,\cdots,loc'_j)$,

$if\ [\![che_1]\!](s_C)=(loc_1,\cdots,loc_i)\ and$

$[\![che_2]\!](s_C)=(loc'_1,\cdots,loc_j)$。

其中,n为数值信息,ch为信道变量。

新增的数值表达式 $\$\ bk$ 表征地址为 bk 的数据块的容量。原有的表达式 $\#bk$ 可以代表该数据块内已使用的空间,应不大于 $\$\ bk$。其语义分别如下:

$[\![\#bk]\!](s_F)(s_B)(s_V)(h_B)(Size)=k$,

$where\ hB\ ([\![bk]\!]\sigma)=(loc_1,\cdots,loc_n)$

$and\ k<=[\![\$\ bk]\!](s_F)(s_B)(s_V)(h_B)(Size)$。

$[\![\$\ bk]\!](s_F)(s_B)(s_V)(h_B)(Size)=Size([\![bk]\!]\sigma)$。

与数据块有关的命令中,追加写操作需受数据块容量的限制、删除块时需一并释放存储内容实际占用的内存空间,以确保验证过程的内存资源限制与执行状态一致;新增的 remove 操作是用于模拟内存和分布式文件的擦写过程。有关命令的语义给定如下:

$[\![b:=\mathbf{allocate}(n,\bar{e})]\!]\sigma=(s_F,s_B[bloc/b],s_V,s_C,$
$h_B[(loc_1,\cdots,loc_i)/bloc],[h_V|loc_1:[\![e_1]\!]\sigma,\cdots,$
$loc_i:[\![e_i]\!]\sigma],Size[n/b])\ where\ bloc\notin dom(h_B)$
$and(loc_1,\cdots,loc_n)\notin dom(h_V)$;

$[\![\mathbf{delete}bk]\!]\sigma=(s_F,s_B,s_C,\ \overline{h_B}|(dom(h_B)-[\![bk]\!]\sigma),\ \overline{h_V}|$
$(dom(h_V)-[\![h_B(b)]\!]\sigma)$,
$Size[0/[\![bk]\!]\sigma])$;

$[\![append(bk,e)]\!]\sigma=(s_F,s_B,s_V,h_B[(loc_1,\cdots,loc_m,$
$loc_{m+1})/bk],[h_V|loc_{m+1}:[\![e]\!]],Size)$
$where\ h_B([\![bk]\!]\sigma)=(loc_1,\cdots,loc_m)$, $the\ term\ of\ sequence$
$(loc_1,\cdots,loc_m)\ belong\ to\ dom\ (h_V)$, $the\ block\ had\ not\ been$
$full\ m<Size([\![bk]\!]\sigma)\ and\ loc_{m+1}\notin dom(h_V)$。

$[\![\mathbf{remove}(bk,e)]\!]\sigma=(s_F,s_B,s_V,s_C,h_B[(loc_1,\cdots,loc_{i-1},$
$loc_{i+1},\cdots,loc_m)/[\![bk]\!]\sigma]$,
$\overline{h_V}|(dom(h_V)-loc_i),Size)\ where\ [\![e]\!]\sigma=i$,
$and\ h_B([\![bk]\!]\sigma)=(loc_1,\cdots,loc_m)(1\leqslant i\leqslant m)$,
$other\ wise\ [\![\mathbf{remove}(bk,e)]\!]\sigma=(s_F,s_B,s_V,s_C,h_B,h_V,Size)$
$where\ the\ state\ can\ not\ be\ changed\ legally$。

上式中,$f[i/x]$ 指用 i 取代原像 x 在函数 f 中的像后所得的函数,$\bar{f}|(dom(f)-\mathrm{A})$ 指将集合 A 从函数 f 的定义域中去掉后所得的函数。

信道机制中,块权限的转移用于辅助验证,对内存状态不作实质性的修改,有关命令对系统的影响由 3.2 小节说明。信道命令的语义给定如下:

$[\![\mathbf{pass}(ch,bk,n,\bar{e})]\!]\sigma=(s_F,s_B,s_V,s_C,h_V,h_B,Size)$,

$[\![\mathbf{get}(ch)]\!]\sigma=(s_F,s_B,s_V,s_C,h_V,h_B,Size)$,

$for\ passing\ the\ right\ will\ not\ change\ the\ state$。

$[\![ch:=\mathbf{newch}(\,)]\!]\sigma=(s_F,s_B,s_V,[s_C|ch:\varnothing],h_V,$
$h_B,Size)\ where\ ch\notin \mathbf{dom}(s_C)$;

$[\![send(ch,e)]\!]\sigma=(s_F,s_B,s_V,s_C[(loc_1,\cdots,loc_m,$
$loc_{m+1})/ch],[h_V|loc_{m+1}:[\![e]\!]\sigma],h_B)$;
$where\ s_V([\![ch]\!]\sigma)=(loc_1,\cdots,loc_m)\ and$
$loc_{m+1}\in \mathrm{Loc\text{-}dom}(h_V)$;

$[\![x:=\mathbf{tryre}(ch)]\!]\sigma=(s_F,s_B,s_V[n_1/x]$,
$s_C[(n_2,\cdots,n_m/ch)],h_V,h_B,Size)\ where\ s_C(ch)=(n_1,$
$\cdots,n_m)\ and\ m\in\mathbb{N}^+$, $otherwise$

$[\![x:=\mathbf{tryre}(ch)]\!]\sigma=(s_F,s_B,s_V,s_C,h_V,h_B,Size)$
$where\ s_C(ch)=\varnothing$。

3　断言语言

断言语言可以用于描述某特定状态下系统所具备的性质。文献[9]提出的断言语言 LR_{BCSS} 分别使用地址断言和块断言 2 个分离描述与块无关的系统特征以及愉快相关的系统特征。本文进一步描述存储资源所面临的限制,新增信道断言以描述线程间交互的性质,相应修改了全局断言。

3.1　块断言

与 LR_{BCSS} 相比,块断言在语法上主要是为数据块引入下标以表示当前可用的空闲空间。新增的 $remove$ 指令将块由只写不擦变为可擦写,块有可能完全空闲,需新增相应的语句描述。块断言的语法详列如下:

$$\begin{aligned}\beta::=\ &\mathbf{true}_B|\mathbf{false}_B|\neg\beta|\beta_1\wedge\beta_2|\beta_1\vee\beta_2\\&|\beta_1\to\beta_2|bk_1==bk_2|\forall x.\beta|\exists x.\beta|\forall b.\beta\\&|\exists b.\beta|\forall f.\beta|\exists f.\beta|bk==bk_1\circledast\cdots\circledast bk_n\\&|\mathbf{emp}_B|bk\mapsto_n(\bar{e})|bk\mapsto_n(\mathbf{emp})|\beta_1*\beta_2\\&|\beta_1-*\beta_2|fe=fe'|b_1\hookrightarrow b_2\text{。}\end{aligned}$$

块断言在语义上增加了关于块容量的说明。增改的后语义给定如下:

- $\$\ \sigma\vDash bk==bk_1\circledast\cdots\circledast bk_n iff\ [\![bk]\!]\sigma=$
  $[\![bk_1]\!]\sigma\cdot\cdots\cdot[\![bk_n]\!]\sigma,\ \sum([\![bk]\!]\sigma)=[\![bk]\!]\sigma$
  $and\ [\![bk_i]\!]\sigma\perp[\![bk_j]\!]\sigma for\ all\ the\ i,j\in\mathbb{N}$,
  $1\leqslant i<j\leqslant n$;
- $\$\ \sigma\vDash bk\mapsto_n(\bar{e})iff\ dom(h_B)=\{[\![bk]\!]\sigma\}$,
  $h_B([\![bk]\!]\sigma)=([\![e_1]\!]\sigma,\cdots,[\![e_k]\!]\sigma),[\![bk]\!]\sigma=$
  $k+n\ and\ forall\ 1\leqslant i\leqslant k,e_k\sigma\in dom(h_V)$;
- $\$\ \sigma\vDash bk\mapsto_n(emp)iff\ dom(h_B)=\{[\![bk]\!]\sigma\}$,
  $h_B([\![bk]\!]\sigma)=\varnothing,\ and\ [\![bk]\!]\sigma=n$。

上式未列出的 $bk_1==bk_2$ 和 $fe=fe'$ 语句在成立时可说明不同的块表达式指向同一块地址,不必强调两者容

量相同;$b_1 \looparrowright b_2$ 一式表示2块内容一致,其关注重点在于内容而非空间,语义不变。

通过显式的容量说明,验证者可以直观得知数据块或对应的内存区的容量能否满足程序运行所需、是否有溢出的可能。通过对各线程持有的存储单元数量求和,可以直观得知在验证过程当中是否有被遗漏的存储单元,避免内存泄露的风险。

带计数机制的块结构也可以描述本地机器上有限的内存资源。为满足其调度所需,块断言支持块的切分使用,即

$$bk \mapsto_n (emp) \dashv\vdash \exists n1, n2, n1 + n2 = n \wedge bk \mapsto_{n1} (emp) * bk \mapsto_{n2} (emp)。$$

通过上式,本地机器不必拘束于传统的2层结构,同时验证者可以使用信道机制描述本地机器的内存调度行为。

3.2 信道断言

信道断言主要包括表征信道内容的语句和基本的逻辑结构。对其他存储结构而言,由于同一时刻只能在一个线程中出现,期间不应出现2个相冲突的状态描述,在应用分离合取和分离析取时必然满足"互不相交"的要求。但对信道而言,通信内容至少有发送方和接收方2者可见,即同一时刻至少有2个不同线程的状态断言会包含同一信道的内容,应用分离合取和分离析取时可能会有冲突,需另外给出说明。由于涉及存储单元所有权的转移,这一部分断言同时依赖于不可变存储器 Store 以及可变存储器 $\mathrm{Heap_V}$ 和 $\mathrm{Heap_B}$ 的状态。信道断言的语法详列如下:

$$\begin{aligned}\gamma ::= &\ \mathbf{true}_C \mid \mathbf{false}_C \mid \neg\ \gamma \mid \gamma_1 \wedge \gamma_2 \mid \gamma_1 \vee \gamma_2 \\ &\mid \gamma_1 \to \gamma_2 \mid che \leftarrow\!\!\!\lhd che' \mid \forall x.\ \gamma \mid \exists x.\ \gamma \mid \forall b.\ \gamma \\ &\mid \exists b.\ \gamma \mid \forall f.\ \gamma \mid \exists f.\ \gamma \mid \mathbf{emp}_C \mid \gamma_1 * \gamma_2 \mid \gamma_1 - * \gamma_2。\end{aligned}$$

相应的语义给定如下:

- $\sigma \vDash \mathbf{true}_C$;
- $\sigma \vDash \forall x.\ \gamma$ *iff* $s_F, s_B, s_V(n/x), s_C, h_B, h_V, Size \vDash \gamma$ *foranyn* $\in \mathbf{Loc}$;
- $\sigma \vDash \exists b.\ \gamma$ *iff* $s_F, s_B(n/b), s_V, s_C, h_B, h_V, Size \vDash \gamma$ *for some* $n \in \mathbf{BLoc}$;
- $\sigma \vDash \forall f.\ \gamma$ *iff* $s_F(\bar{n}/f), s_B, s_V, s_C, h_B, h_V, Size \vDash \gamma$ *for any* $\bar{n} = (n_1, \cdots, n_k) \in \mathbf{BLoc}^*$;
- $\sigma \vDash \gamma_1 \to \gamma_2$ *iff if* $\sigma \vDash \gamma_1$ *then* $\sigma \vDash \gamma_2$;
- $\sigma \vDash che \leftarrow\!\!\!\lhd che'$ *iff* $[\![che]\!]\sigma = [\![che']\!]\sigma$ *and they are passing right of Blocks in the same way*;
- $\sigma \vDash \mathbf{emp}_C$ *iff* $dom(s_C) = \varnothing$;
- $\sigma \vDash \gamma_1 * \gamma_2$ *iff there exist* s_C^1, s_C^2 *with* $s_C = s_C^1 * s_C^2$ *that* $s_F, s_B, s_V, s_C^1, h_B, h_V, Size \vDash \gamma_1$ *and* $s_F, s_B, s_V, s_C^2, h_B, h_V, Size \vDash \gamma_2$。

  The function $s_C = s_C^1 * s_C^2$ *should be defined as follow*:

$$s_C(n) = \begin{cases} s_C^1(n),\ \mathit{if}\ n \in \{dom(s_C^1) - dom(s_C^2)\}; \\ s_C^2(n),\ \mathit{if}\ n \in \{dom(s_C^2) - dom(s_C^1)\}; \\ s_C^1(n), \mathit{if}\ \begin{matrix} dom(s_C^1) \cap dom(s_C^2)\} \\ and s_C^1(n) = s_C^2(n) \end{matrix} \end{cases}。$$

需要注意的是,尽管信道中的内容可能因为不同线程执行进度的差异而无法保证在每次执行中都能保持一致,但对某一特定信道而言,在同一时刻向不同线程所展示的数据流应当是一致的。因此,在对信道断言执行分离合取操作时,相同的信道应当包含内容完全相同的信息流。

3.3 全局断言

全局断言的形式是地址断言、块断言和信道断言组成的三元组

$$\langle \alpha, \beta, \gamma \rangle。$$

其中,块断言、信道断言分别由3.1小节和3.2小节给出。地址断言沿用自 LR_{BCSS},具体定义如下:

$$\begin{aligned}\alpha ::= &\ \mathbf{true}_V \mid \mathbf{false}_V \mid \neg\ \alpha \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \\ &\mid \alpha_1 \to \alpha_2 \mid e_1 = e_2 \mid e_1 \leqslant e_2 \mid \forall x.\ \alpha \mid \exists x.\ \alpha \\ &\mid \forall b.\ \alpha \mid \exists b.\ \alpha \mid \forall f.\ \alpha \mid \exists f.\ \alpha \mid \mathbf{emp}_V \\ &\mid e \mapsto e' \mid \alpha_1 * \alpha_2 \mid \alpha_1 - * \alpha_2。\end{aligned}$$

全局断言并非3个分量的逻辑求和。例如,若某个块出现在信道断言中,可变数据块存储器 $\mathrm{Heap_B}$ 的状态将由块断言和信道断言2者共同确定,简单地使用逻辑和联结2者并不能准确表示当中的划分关系,使用分离合取"*"联结也不能准确地描述当前数据块权限的归属。对机器状态而言,这个三元组是一个不可简单分割的整体。全局断言的语法详列如下:

$$\begin{aligned}\mathcal{P} ::= &\ \langle \alpha, \beta, \gamma \rangle \mid \mathbf{true} \mid \mathbf{false} \mid \neg \mathcal{P} \mid \mathcal{P}_1 \wedge \mathcal{P}_2 \\ &\mid \mathcal{P}_1 \vee \mathcal{P}_2 \mid \mathcal{P}_1 \to \mathcal{P}_2 \mid \forall x.\ \mathcal{P} \mid \exists x.\ \mathcal{P} \mid \forall b.\ \mathcal{P} \\ &\mid \exists b.\ \mathcal{P} \mid \forall f.\ \mathcal{P} \mid \exists f.\ \mathcal{P} \mid \forall ch.\ \mathcal{P} \mid \exists ch.\ \mathcal{P} \\ &\mid \mathbf{emp} \mid \mathcal{P}_1 * \mathcal{P}_2 \mid \mathcal{P}_1 - * \mathcal{P}_2。\end{aligned}$$

相应的语义给定如下:

- $\sigma \vDash \langle \alpha, \beta, \gamma \rangle$ *iff* $\sigma \vDash \alpha, \sigma \vDash \beta$, *and* $\sigma \vDash \gamma$;
- $\sigma \vDash \forall ch.\ \gamma$ *iff* $s_F, s_B, s_V, s_C(\bar{n}/ch), h_B, h_V, Size \vDash \gamma$ *for any* $\bar{n} \in \mathbf{Loc}^*$, *noweather how they pass the onwership of Blocks*;
- \$ $\sigma \vDash \exists ch.\ \gamma$ *iff* $s_F, s_B, s_V, s_C(\bar{n}/ch), h_B, h_V, Size$

$\models\gamma$ *for some* $\bar{n}\in\mathbf{Loc}^*$, *with a special way to pass the onwership of Blocks*;

– $\sigma\models\mathbf{emp}$ *iff* $dom(h_V)=\varnothing$, $dom(h_B)=\varnothing$, *and* $dom(s_C)=\varnothing$;

– \$ $s_F,s_B,s_V,s_C,h_B,h_V,Size\models\mathcal{P}_1*\mathcal{P}_2$ *iff there exists* h_H^1,h_H^2,s_C^1,s_C^2 *with* $h_H^1\#h_H^2$, $h_H=h_H^1*h_H^2$, *and* $s_C=s_C^1*s_C^2$ *such that* $s_F,s_B,s_V,s_C^1,h_B^1,h_V^1,Size\models\mathcal{P}_1$ *and* $s_F,s_B,s_V,s_C^2,h_B^2,h_V^2,Size\models\mathcal{P}_2$;

– \$ $s_F,s_B,s_V,s_C,h_B,h_V,Size\models\mathcal{P}_1-*\mathcal{P}_2$ *iff for any heap* h'_H *and communications status* s'_C, *if* $h_H\#h'_H$, $s_C*s'_C$ *exists and* $s_F,s_B,s_V,s'_C,h'_B,h'_V,Size\models\mathcal{P}_1$ *then* $s_F,s_B,s_V,s_C*s'_C,h_B*h'_B,h_V*h'_V,Size\models\mathcal{P}_2$。

4　证明系统

有效的霍尔三元组

$$\{P\}C\{Q\}$$

由前置条件 P、程序指令序列 C 以及后置条件 Q 组成。其中，前置条件和后置条件均为全局断言，程序指令序列遵循 2.2 小节给出的语法。本文考虑部分正确性断言，即对一个成立的霍尔三元组，要求：

(1)程序指令序列中所读写的变量或存储单元，必须被包含在前置条件 P 中或是在程序 C 中以 allocated 等语句合法分配所得。即程序对出现在其前置条件内的存储单元拥有完整的权限、对新存储单元自分配成功时起拥有完整权限，直至有关存储单元被剔出成立的全局断言后权限灭失；

(2)对于任何能令前置条件 P 成立的状态 σ，$\langle C,\sigma\rangle$ 是安全的，即在状态 σ 下运行程序 C 不会出现导致程序意外退出的故障；

(3)在任何能令前置条件 P 成立的状态 σ 下运行程序 C，若程序终止于一个新的状态 σ‘，则状态 σ’必须能令后置条件 Q 成立。

4.1　证明规则

本节主要为每一个语句给出一个基本的证明公理。这些公理仅依赖于语句本身，而与程序结构无关。不论涉及何种存储结构，公理都采用全局命题。第一部分是适用于原有命令的公理：

$$\{p\}\,\mathbf{skip}\,\{p\} \tag{A1}$$

$$\begin{array}{l}\{\langle x=x'\wedge\mathbf{emp}_V,\mathbf{emp}_B,\mathbf{emp}_C>\}\\ x:=e\\ \{\langle x=e[x'/x]\wedge\mathbf{emp}_V,\mathbf{emp}_B,\mathbf{emp}_C\rangle\}\end{array} \tag{A2}$$

$$\begin{array}{l}\{\langle x=x'\wedge e\mapsto x'',\mathbf{emp}_B,\mathbf{emp}_C\rangle\}\\ x:=[e]\\ \{\langle x=x''\wedge e[x'/x]\mapsto x'',\mathbf{emp}_B,\mathbf{emp}_C\rangle\}\end{array} \tag{A3}$$

$$\begin{array}{l}\{\langle x=x'\wedge\mathbf{emp}_V,\mathbf{emp}_B,\mathbf{emp}_C\rangle\}\\ x:=cons(e_1,\cdots,e_n)\\ \{\langle x\mapsto e_1[x'/x],\cdots,e_n[x'/x],\mathbf{emp}_B,\\ \mathbf{emp}_C>\}\end{array} \tag{A4}$$

$$\begin{array}{l}\{\langle e\mapsto -,\mathbf{emp}_V,\mathbf{emp}_C\rangle\}\\ [e]:=e'\\ \{\langle e\mapsto e',\mathbf{emp}_V,\mathbf{emp}_C\rangle\}\end{array} \tag{A5}$$

$$\begin{array}{l}\{\langle e\mapsto -,\mathbf{emp}_V,\mathbf{emp}_C\rangle\}\\ \mathbf{dispose}(e)\{\langle emp\rangle\}\end{array} \tag{A6}$$

$$\begin{array}{l}\{\langle\alpha,f=\mathbf{nil}\wedge\beta,\mathbf{emp}_C\rangle\}\\ f:=\mathbf{create}(bk_1,\cdots,bk_n)\\ \{\langle\alpha,\beta[(bk_1,\cdots,bk_n)/f],emp_C\rangle\}\end{array} \tag{A7}$$

$$\begin{array}{l}\{\langle\alpha,f=f'\wedge\beta,\mathbf{emp}_C\rangle\}\\ \mathbf{attach}(f,bk_1,\cdots,bk_n)\\ \{\langle\alpha[f'/f],\beta[f'\cdot(bk_1[f'/f],\cdots,\\ bk_n[f'/f])/f],\mathbf{emp}_C\rangle\}\end{array} \tag{A8}$$

$$\begin{array}{l}\{\langle\alpha,f=f'\wedge\beta,\mathbf{emp}_C\rangle\}\\ \mathbf{delete}(f)\\ \{\langle\alpha[f'/f],\beta[f'\cdot(\mathbf{nil}/f],\mathbf{emp}_C\rangle\}\end{array} \tag{A9}$$

$$\begin{array}{l}\{\exists x',\overline{l'}.\langle e=i\wedge x=x'\wedge\overline{l'}\hookrightarrow\overline{e'},bk\mapsto_n\overline{l'},\mathbf{emp}_C>\}\\ x:=\{bk.e\}\\ \{\langle x=l_i[x'/x]\wedge\overline{l'}[x'/x]\mapsto\overline{e'}[x'/x],\\ bk\hookrightarrow_n\overline{l'}[x'/x],\mathbf{emp}_C\rangle\}\end{array} \tag{A10}$$

$$\begin{array}{l}\{\langle\mathbf{emp}_V,b==b'\wedge\mathbf{emp}_B,\mathbf{emp}_C\rangle\}\\ b:=bk\\ \{\langle\mathbf{emp}_V,b==bk[b'/b]\wedge\mathbf{emp}_B,\mathbf{emp}_C\rangle\}\end{array} \tag{A11}$$

第二部分是与块命令相关的公理。由于本文引入了计数式机制，相关公理被改写如下：

$$\begin{array}{l}\{\langle\mathbf{emp}_V,\mathbf{emp}_B,\mathbf{emp}_C\rangle\}\\ b:=\mathbf{allocate}(\mathrm{n},\bar{\mathrm{e}})\{\exists\bar{\mathrm{l}}.<\bar{\mathrm{l}}\hookrightarrow\bar{\mathrm{e}},\\ \$\ \mathrm{b}=\mathrm{n}\wedge\mathrm{n}>|\bar{\mathrm{e}}|\wedge\mathrm{b}\rightarrow_{\mathrm{n}-|\bar{\mathrm{e}}|}\bar{\mathrm{l}},\mathbf{emp}_C\rangle\}\end{array} \tag{A12}$$

where $|\bar{e}|$ *indicates the counts of* $|\bar{e}|$

$$\{\exists \overline{l'}.\langle \overline{l'} \looparrowright \overline{e'}, bk \rightarrow_n \overline{l'} \wedge n>0, \mathbf{emp}_C\rangle\}$$

$$b:=\mathbf{append}(bk,e)$$

$$\{\exists \overline{l'},l. < \overline{l'} \looparrowright \overline{e'} * l \mapsto e, bk \rightarrow_{n-1} \overline{l'}\cdot(l), \tag{A13}$$

$$\mathbf{emp}_C\rangle\}$$

when the operation is successful.

$$\{\exists \overline{l'},n'. < \overline{l'} \looparrowright \overline{e'}, bk \rightarrow_n \overline{l'} \wedge bk==n',$$

$$\mathbf{emp}_C\rangle\}\ \mathbf{delete}(bk) \tag{A14}$$

$$\{\exists n'.\langle \mathbf{emp}_V, \mathbf{emp}_B \wedge \$\ n'=0, \mathbf{emp}_C\rangle\}$$

$$\{\exists \overline{l'}. < \overline{l'} \looparrowright \overline{e'} \wedge e=i, bk \rightarrow_n \overline{l'} \wedge \#bk=j$$

$$\wedge(\neg j<i), \mathbf{emp}_C>\}\ \mathbf{remove}(bk,e)$$

$$\{\exists \overline{l'}.\langle (l'_1,\cdots,l'_{i-1},l'_{i+1},\cdots,l'_j) \looparrowright \tag{A15}$$

$$(e'_1,\cdots,e'_{i-1},e'_{i+1},\cdots,e'_j),$$

$$bk \rightarrow_{n+1} (l'_1,\cdots,l'_{i-1},l'_{i+1},\cdots,l'_j), \mathbf{emp}_C\rangle\}$$

信道相关操作主要依据先进先出的原则给出相应的公理,同时考虑消息读取失败时的情形。有关基本公理详列如下:

$$\{\langle \mathbf{emp}_V, \mathbf{emp}_B, \mathbf{emp}_C\rangle\}\ ch:=\mathbf{newch}() \tag{A16}$$

$$\{\langle \mathbf{emp}_V, \mathbf{emp}_B, ch \leftarrowtail \varnothing\rangle\}$$

$$\{\exists ch'.\langle e=i, \mathbf{emp}_B, ch \leftarrowtail ch'\rangle\}\ \mathbf{send}(ch,e) \tag{A17}$$

$$\{\exists ch'.\langle e=i, \mathbf{emp}_B, ch \leftarrowtail ch'\cdot i\rangle\}$$

$$\{\exists ch',x'.\langle x=x', \mathbf{emp}_B, ch \leftarrowtail i\cdot ch'\rangle\}$$

$$x:=\mathbf{tryre}(ch) \tag{A18}$$

$$\{\exists ch',x'.\langle x=i, \mathbf{emp}_B, ch \leftarrowtail ch'[x'/x]\rangle\}$$

$$\{\exists ch',x'.\langle x=x', \mathbf{emp}_B, ch \leftarrowtail \varnothing\rangle\}$$

$$x:=\mathbf{tryre}(ch) \tag{A18a}$$

$$\{\exists ch',x'.\langle x=x', \mathbf{emp}_B, ch \leftarrowtail \phi\rangle\}$$

$$\{\exists ch',bk'_1,bk'_2,\bar{l},\overline{e'},\overline{e''}. < \bar{l} \looparrowright \overline{e'} \wedge |\bar{e}|=i$$

$$\wedge(l_{e1},\cdots,l_{e_i}) \looparrowright \overline{e''},$$

$$bk==bk'_1 \circledast bk'_2 \wedge bk \mapsto_{n'} \bar{l} \wedge$$

$$bk'_2 \mapsto_{n-i} (l_{e1},\cdots,l_{e_i}),$$

$$ch \leftarrowtail ch'\rangle\}\ \mathbf{pass}(ch,bk,n,\bar{e})$$

$$\{\exists ch',bk'_1,\bar{l},\overline{e'},\overline{e''}.\langle (l_{e1},\cdots,l_{e_i}) \looparrowright \overline{e''} -* \tag{A19}$$

$$\bar{l} \looparrowright \overline{e'}, bk==bk'_1, ch \leftarrowtail ch'\cdot$$

$$bk.n.(l_{e1},\cdots,l_{e_i})>\}$$

where $bk.n.(l_{e1},\cdots,l_{e_i})$ *defines a part of Block bk with n units that holding a i units Local Address Set* ($l \in \mathbf{Loc}^*$).

$$\{\exists ch'.\langle \mathbf{emp}_V, \mathbf{emp}_B, ch \leftarrowtail bk.n.\bar{e}\cdot ch'>\}$$

$$\mathbf{get}(ch) \tag{A20}$$

$$\{\exists ch',\overline{e'}.\langle \bar{e} \looparrowright \overline{e'} \wedge i=|\bar{e}|, bk \mapsto_{n-i} (\bar{e}),$$

$$ch \leftarrowtail ch'\rangle\}$$

$$\{\exists ch'.\langle \mathbf{emp}_V, \mathbf{emp}_B, ch \leftarrowtail \varnothing\rangle\}$$

$$\mathbf{get}(ch) \tag{A20a}$$

$$\{\exists ch'.\langle \mathbf{emp}_V, \mathbf{emp}_B, ch \leftarrowtail \varnothing\rangle\}$$

$$\{\langle \mathbf{emp}_V, \mathbf{emp}_B, ch \leftarrowtail \varnothing\rangle\}\ \mathbf{closech}(ch) \tag{A21}$$

$$\{\langle \mathbf{emp}_V, \mathbf{emp}_B, \mathbf{emp}_C\rangle\}$$

4.2 基本推理规则

推理规则依赖于特定的程序结构而成立。下列基本推理规则沿用自霍尔逻辑[7]、分离逻辑[6]、并发分离逻辑[12]以及 LR_{BCSS}[9]。

$$\frac{\{p\}C\{q\} \quad \{q\}C'\{r\}}{\{p\}C;C'\{r\}} \tag{R1}$$

$$\frac{\{p \wedge \mathrm{T}(be)\}C\{q\} \quad \{p \wedge \neg\, \mathrm{T}(be)\}C'\{q\}}{\{p\}\ \mathbf{if}\ be\ \mathbf{then}\ C\ \mathbf{else}\ C'\{q\}} \tag{R2}$$

$$\frac{\{p \wedge \mathrm{T}(be)\}C\{p\}}{\{p\}\ \mathbf{while}\ be\ do\ C\ \{p \wedge \neg\, \mathrm{T}(be)\}} \tag{R3}$$

$$\frac{\vDash p' \rightarrow p\{p\}C\{q\} \vDash q \rightarrow q'}{\{p'\}C\{q'\}} \tag{R4}$$

$$\frac{\{p\}C\{q\}}{\{p*r\}C\{q*r\}} \tag{R5}$$

where $\mathrm{Modify}_S(C) \cap \mathrm{FV}(r)=\varnothing$

$$\frac{\{p'\}C_1\{q'\} \quad \{p''\}C_2\{q''\}}{\{p'*p''\}C_1 \parallel C_2\{q'*q''\}} \tag{R6}$$

where $\mathrm{Modify}_S(C_1) \cap \mathrm{FV}(p'')=\varnothing$

and $\mathrm{Modify}_S(C_2) \cap \mathrm{FV}(p')=\varnothing$

上式中 FV (p) 为命题 p 中的自由变量集, $\mathrm{Modify}_S(C)$ 涵盖被程序 C 修改过的除信道变量以外的存储单元集合。T(be) 是将布尔表达式转换为全局断言的函数,定义如下:

$\mathrm{T}(e_1=e_2)=\langle e_1=e_2, true_\beta, true_\gamma\rangle$;

$\mathrm{T}(e_1 \leqslant e_2)=\langle e_1 \leqslant e_2, true_\beta, true_\gamma\rangle$;

$\mathrm{T}(bk_1==bk_2)=\langle \mathbf{true}_\alpha, bk_1==bk_2, \mathbf{true}_\gamma\rangle$;

$\mathrm{T}(\mathbf{true})=\mathbf{true}$; $T(\mathbf{false})=\mathbf{false}$;

$\mathrm{T}(\neg\, be)=\neg\, T(be)$;

$\mathrm{T}(be_1 \wedge be_2)=\mathrm{T}(be_1) \wedge \mathrm{T}(be_2)$;

$\mathrm{T}(be_1 \vee be_2)=T(be_1) \vee T(be_2)$。

在并发规则中要求的"互不干涉"原本指可能被任一程序修改的存储空间不得被另一程序读写。本文允

许以下例外：

(1)与信道相关的消息和权限传递。信道机制是在全局层面服务于线程间通信，如若受限则线程间再无交互的渠道；

(2)通过信道传递数据块使用权实现的复用。这部分复用对程序运行的影响可以通过对信道操作进行验证得以释明。

4.3 涉及信道的状态更新规则

在4.2节中，考虑到信道的内容可能会被多个线程知悉，且不要求线程关注所有信道的变化，本文对分离合取给出了新的限制。同理，在推理时验证者可以按需引入或剔除可能可用的信道。下述的推理规则给出了一些可以向语句的前后置条件引入新的信道状态的情景。

每个信道都可以传输数值和数据块使用权2种消息，2种消息均采用先进先出的队列模式，且2种消息是分别使用不同的命令完成出队入队操作。因此，在验证时只需保证同一种消息的时序能维持一致，而不必在所有消息队列上保证时序关系不变，即

$$ch \leftarrow ch' \cdot bk.n.\bar{e} \cdot n' \cdot ch'' \dashv\vdash ch \leftarrow ch' \cdot n' \cdot bk.n.\bar{e} \cdot ch'' \quad \text{(RC1)}$$

循此操作多次变换，则对任意信道都可以表示为

$$ch \leftarrow (n_1', \cdots, n_i') \cdot (bk_1.n_1.\bar{e}_1, \cdots, bk_j.n_j.\bar{e}_j)。$$

在线程间通信中，以接收信息后程序所执行的动作为标准，信道可作如下划分：

(1)单向传输消息，线程接收信息后不必根据该消息对外作出特定反应；

(2)双向响应消息，线程依据接收到的信息对外作出特定反应。

对于第一类行为，线程对外传出消息可能影响任何访问该信道的线程。尽管在读取消息前受信一方不知晓信道的具体状态，但对验证者而言，任一线程都有权获取该新消息。综上，对某一涉及信道操作语句的前置条件有：

$$\begin{aligned}&\langle \triangleright p * ch \leftarrow che \rangle \rightarrow^* \\ &\langle \triangleright p * ch \leftarrow che' \rangle \\ &\textit{iff there is a code } C \textit{ of other thread and} \qquad \text{(RC2)} \\ &\{\triangleright p' * ch \leftarrow che\}_i C \\ &\{p * ch \leftarrow che'\}_i \textit{ is valid}\end{aligned}$$

对于第二类行为，若某一线程在接收一个特定信息a后，能在不访问其他信道的前提下传出另一组特定信息b，则可认为传出信息a的进程期望获取信息b。即对某一涉及发信操作语句的后置条件为

$$\begin{aligned}&\langle p * ch \leftarrow che_1 \cdot che_2 \rangle \rightarrow^* \\ &\langle \triangleright p * ch' \leftarrow che_2' \rangle \\ &\textit{iff there is a code } C \textit{ of other thread and} \qquad \text{(RC3)} \\ &\{\triangleright \langle \alpha, \beta, \gamma * ch \leftarrow che_2 * ch' \leftarrow che_1' \rangle\}_i C \\ &\{\langle \alpha', \beta', \gamma * ch' \leftarrow che_1' \cdot che_2' \rangle\}_i \textit{ is valid}\end{aligned}$$

上式中的“▷”可用于信道操作的限制条件，表明在执行操作之前信道可能出现的状态，在有多个可能性时前后置条件可用下标区别各自的配对。出于限制信道混用带来的冲突考虑，在建模和验证期间可以限制线程对信道的读写权限，以维持“互不干涉”的状态。

在实际验证过程中，验证者可以主动区分不同信道的用途和使用者，以便追踪线程间通信行为对全局状态的影响。

5 程序验证实例

本节给出的实例由3个线程组成，分别为用户发出写请求、主节点分配任务和数据节点执行存储，为一个完整的写数据过程。本节首先给出实例及其说明：

(1)线程1. 用户端

CHVar：*appW*, *upD*;
Local Var List
BKVar：*Btemp*;
appW: = **newch**( );
upD: = **newch**( );
Btemp: = 0;
send(*appW*, 4);
while *Btemp* = = 0 **do**
Btemp: = **tryre**(*disW*);
end while;
send(*upD*, *Btemp*, 4, 3, 2, 1);

线程1有2次发信行为，分别为向主节点发送写请求和向数据节点提供具体数据，均为数值信息传递。另有一次等待主节点响应的收信过程，由于非阻塞条件下tryre命令不具备等待功能，需用循环语句辅助。

(2)线程2. 主节点

CHVar：*disW*, *disD*;

```
Local Var List
Var:i;
BKVar: Bpool;
disW: = newch();
disD: = newch();
Bpool: = 0;
i: = 0;
while true do
  i: = tryre(appW);
  if i > 0 then
    Bpool: = allocate(i);
    send(disW, Bpool);
    pass(disD, Bpool, i);
  end if;
  i: = 0;
end while;
```

线程 2 有 2 次发信行为,向用户端反馈可用数据块地址时是传递“地址”这一数值信息,令数据节点开辟块空间则是存储资源使用权的转移。

(3)线程 3. 数据节点

```
Local Var List
Var:len, j;
BKVar: Bdisk;
Bdisk: = 0;
j: = 0;
len: = 0;
while #Bdisk = 0 do
Bdisk: = get(disD);
end while;
while j = 0 do
  j: = tryre(upD);
end while;
if Bdisk = = j then
  while len < $ Bdisk then
    j: = tryre(upD);
    Bpool: = append(Bdisk, j);
    len: = len + 1;
  end while;
end if;
```

线程 3 有 2 组信息接收行为:首先从主节点获得数据块权限;其后从用户端获得需写入的数据、核对写入地址后完成操作。

后续分 3 部分给出各自线程的验证过程。系统的初始状态与 emp 等价。

1)用户端线程验证过程

① $\{\langle \mathbf{emp}_V, \mathbf{emp}_B, \mathbf{emp}_C \rangle\}$ (Given)

② $\{\langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) * \mathbf{emp}_B, \mathbf{emp}_C \rangle\}$ (Var)

$appW: = \mathbf{newch}();$

③ $\{\langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) * \mathbf{emp}_B,$
$appW \hookleftarrow \varnothing \rangle\}$ (A16)

$upD: = \mathrm{newch}();$

④ $\{\langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) * \mathbf{emp}_B,$
$appW \hookleftarrow \varnothing, upD \hookleftarrow \varnothing \rangle\}$ (A16)

$Btemp: = 0;$

⑤ $\{\langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge Btemp = 0,$
$appW \hookleftarrow \varnothing, upD \hookleftarrow \varnothing \rangle\}$ (A11)

$\mathbf{send}(appW, 4);$

⑥ $\{\langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge Btemp = 0,$
$appW \hookleftarrow 4, upD \hookleftarrow \varnothing \rangle\}$ (A17)

⑦ $\{\langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge Btemp = 0,$
$appW \hookleftarrow 0, disW \hookleftarrow \varnothing, upD \hookleftarrow \varnothing \rangle\}_1$ (⑰⑱, RC2)

⑧ $\{\exists bd \neq 0. \langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge Btemp = 0, appW \hookleftarrow 0, disW \hookleftarrow bd, upD \hookleftarrow \varnothing \rangle\}_2$ (㉗ – ㉙, RC3)

此处对信道状态的更新依赖于主节点的验证过程。验证式⑦由 RC2 推出,是其他线程对信道操作的影响在全局的体现。验证式⑧依赖于主节点的响应行为,在后续部分会详细列出对应的条件。由主节点的验证过程可知,验证式⑧的首次成立不会早于验证式⑦的首次成立。

此后为一循环,验证式⑦及⑧均可以成为循环体执行前成立的状态。现以验证式⑦为基础考虑第一次循环体执行,有:

⑨ $\{\langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge Btemp = 0,$
$appW \hookleftarrow 0, disW \hookleftarrow \varnothing, upD \hookleftarrow \varnothing \rangle\}_1$ (⑦)

$Btemp: = \mathbf{tryre}(disW);$

⑩ $\{\langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge Btemp = 0,$
$appW \hookleftarrow 0, disW \hookleftarrow \varnothing, upD \hookleftarrow \varnothing \rangle\}_1$ (A18a)

由于信道内无消息可读,在非阻塞通信的要求下,读操作不改变任何系统状态。同时由于验证式⑨与验证式⑩等价,可以构成循环不变式。

再考虑以验证式⑧为前置条件的执行过程:

⑪ $\{\exists bd \neq 0. \langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge$
$Btemp = 0, appW \hookleftarrow 0, disW \hookleftarrow bd, upD \hookleftarrow \varnothing \rangle\}_2$ (⑧)

$Btemp := \mathbf{tryre}(disW)$;

⑫ $\{\exists \mathrm{bd} \neq 0. \langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge$ $Btemp = bd, appW \leftarrowtail 0, disW \leftarrowtail \varnothing, upD \leftarrowtail \varnothing \rangle\}_2$ (A18)

可见验证式⑫的成立已经打破循环条件,应为执行循环语句的最终结果。需要特别注意的是,此处的通信仅传递了数值信息,地址为 bd 的块权限并未随之交予用户端。

后续可有如下验证:

⑬ $\{\exists \mathrm{bd} \neq 0. \langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge Btemp = bd, appW \leftarrowtail 0, disW \leftarrowtail \varnothing, upD \leftarrowtail \varnothing \rangle\}$ (⑩,⑫,R3)

$\mathbf{send}(upD, Btemp, 4, 3, 2, 1)$;

⑭ $\{\exists \mathrm{bd} \neq 0. \langle \mathbf{emp}_V, Btemp \mapsto_0 (\mathbf{emp}) \wedge Btemp = bd, appW \leftarrowtail 0, disW \leftarrowtail \varnothing, upD \leftarrowtail bd \cdot 4 \cdot 3 \cdot 2 \cdot 1 \rangle\}$ (A17)

至验证式⑭成立时,用户端已经确认主线程响应了写入请求并已通过信道 upD 向数据节点提交了待写入的数据,即目标块的地址 bd 以及待写入的数列(4,3,2,1)。

2)主节点程验证过程

⑮ $\{\langle \mathbf{emp}_V, \mathbf{emp}_B, \mathbf{emp}_C \rangle\}$ (Given)

⑯ $\{\langle i = - \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) * \mathbf{emp}_B, \mathbf{emp}_C \rangle\}$ (Var)

$disW := \mathbf{newch}()$;

⑰ $\{\langle i = - \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) * \mathbf{emp}_B,$ $\mathrm{disW} \leftarrowtail \varnothing \rangle\}$ (A16)

$disD := \mathbf{newch}()$;

⑱ $\{\langle i = - \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) * \mathbf{emp}_B,$ $disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing \rangle\}$ (A16)

$Bpool := 0$;

⑲ $\{\langle \mathrm{i} = - \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = 0, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing \rangle\}$ (A11)

$i := 0$;

⑳ $\{\langle i = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = 0, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing \rangle\}$ (A15)

㉑ $\{\langle i = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = 0, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_1$ (③,RC2)

㉒ $\{\langle i = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = 0, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail 4 \rangle\}_2$ (⑥,RC2)

初始化各变量后,线程进入循环状态。参考用户端线程的验证过程,循环体内可有验证过程如下:

㉓ $\{\langle i = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = -, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_1$ (㉑)

㉔ $\{\langle i = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = -, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail 4 \rangle\}_2$ (㉒)

$i := \mathbf{tryre}(appW)$;

㉕ $\{\langle \mathrm{i} = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = -, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_1$ (㉓,A18a)

㉖ $\{\langle i = 4 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = -, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_2$ (㉔,A18)

if $i > 0$ *then*

㉗ $\{\langle i = 4 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = -, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_2$ (㉖,TRUE)

$Bpool := \mathrm{allocate}(i)$;

㉘ $\{\exists \mathrm{bd} \neq 0. \langle \mathrm{i} = 4 \wedge \mathbf{emp}_V, Bpool \mapsto_4 (\mathbf{emp}) \wedge$ $Bpool = bd, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_2$ (A12)

$\mathrm{send}(disW, Bpool)$;

㉙ $\{\exists \mathrm{bd} \neq 0. \langle \mathrm{i} = 4 \wedge \mathbf{emp}_V, Bpool \mapsto_4 (\mathbf{emp}) \wedge$ $Bpool = bd, disW \leftarrowtail bd, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_2$ (A17)

$\mathbf{pass}(disD, Bpool, i)$;

㉚ $\{\exists \mathrm{bd} \neq 0. \langle \mathrm{i} = 4 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = bd, disW \leftarrowtail bd, disD \leftarrowtail bd.4.(\phi),$ $appW \leftarrowtail \varnothing \rangle\}_2$ (③,RC2)

㉛ $\{\exists bd \neq 0. \langle i = 4 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = bd, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_2$ (⑫,㊻,RC2)

else

㉜ $\{\langle i = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = -, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_1$ (㉕,FACSE)

skip;

㉝ $\{\langle i = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = -, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_1$ (A1)

end if;

㉞ $\{\langle i = 0 \wedge \mathbf{emp}_V, Bpool \mapsto_0 (\mathbf{emp}) \wedge$ $Bpool = -, disW \leftarrowtail \varnothing, disD \leftarrowtail \varnothing,$ $appW \leftarrowtail \varnothing \rangle\}_1$ (㉝)

㉟ $\{\exists bd\neq 0.\langle i=4\wedge \mathbf{emp}_V, Bpool\mapsto_0(\mathbf{emp})\wedge$
$Bpool=bd, disW\leftarrow\varnothing, disD\leftarrow\varnothing,$
$appW\leftarrow\varnothing\rangle\}_2$（㉛）

$i:=0;$

㊱ $\{\langle \mathrm{i}=0\wedge \mathbf{emp}_V, Bpool\mapsto_0(\mathbf{emp})\wedge$
$Bpool=-, disW\leftarrow\varnothing, disD\leftarrow\varnothing,$
$appW\leftarrow\varnothing\rangle\}_1$（㉞，A2）

㊲ $\{\exists bd\neq 0.\langle i=0\wedge \mathbf{emp}_V, Bpool\mapsto_0(\mathbf{emp})\wedge$
$Bpool=bd, disW\leftarrow\varnothing, disD\leftarrow\varnothing,$
$appW\leftarrow\varnothing\rangle\}_2$（㉟，A2）

㊳ $\{\langle i=0\wedge \mathbf{emp}_V, Bpool\mapsto_0(\mathbf{emp})\wedge$
$Bpool=-, disW\leftarrow\varnothing, disD\leftarrow\varnothing,$
$appW\leftarrow\varnothing\rangle\}$（㊱，㊲）

上列过程中，验证式㉗至验证式㉙说明了主节点对客户端写请求的响应，期间不涉及额外的信道操作，符合规则 $RC3$ 的要求，结果可用于更新客户端的状态性质。至验证式㉚成立时，主节点已经将新分配的数据块使用权送出，即使块变量 $Bpool$ 仍存有对应的块地址 bd，可用的空闲空间仍为 0。除此之外，$\mathrm{Heap_B}$ 中再无可用空间，块变量 $Bpool$ 所存储的具体数值可以暂不考虑，故循环体执行结果如验证式㊳所示。比对循环体执行前的系统状态可知验证式㊳构成循环不变式；进一步考虑循环推理规则 R3 的应用，可知验证式㊳为执行主节点线程所得的最终结果。

3）数据点程验证过程

㊴ $\{\langle \mathbf{emp}_V, \mathbf{emp}_B, \mathbf{emp}_C\rangle\}$（Given）

㊵ $\{\langle len=-\wedge j=-\wedge \mathbf{emp}_V, Bdisk\mapsto_0(\mathbf{emp})$
$*\ \mathbf{emp}_B, \mathbf{emp}_C\rangle\}$（Var）

$Bdisk:=0;$

㊶ $\{\langle \mathrm{len}=-\wedge \mathrm{j}=-\wedge \mathbf{emp}_V, Bdisk\mapsto_0(\mathbf{emp})$
$\wedge Bdisk=0, \mathbf{emp}_C\rangle\}$（A11）

$j:=0;$

㊷ $\{\langle \mathrm{len}=-\wedge \mathrm{j}=0\wedge \mathbf{emp}_V, Bdisk\mapsto_0(\mathbf{emp})$
$\wedge Bdisk=0, \mathbf{emp}_C\rangle\}$（A2）

$len:=0;$

㊸ $\{\langle \mathrm{len}=0\wedge \mathrm{j}=0\wedge \mathbf{emp}_V, Bdisk\mapsto_0(\mathbf{emp})$
$\wedge Bdisk=0, \mathbf{emp}_C\rangle\}$（A2）

㊹ $\{\exists bd\neq 0.\langle len=0\wedge j=0\wedge \mathbf{emp}_V,$
$Bdisk\mapsto_0(\mathbf{emp})$
$\wedge Bdisk=0, upD\leftarrow bd\cdot 4\cdot 3\cdot 2\cdot 1\wedge$
$disD\leftarrow bd.4.(\varnothing)\rangle\}$（⑭，㉚，RC2）

初始化各变量后，数据节点进入循环，等待主节点的写权限分配。与其他线程的循环等待相似，有效的读信道行为对应下列验证过程：

㊺ $\{\exists bd\neq 0.\langle len=0\wedge j=0\wedge \mathbf{emp}_V, Bdisk\mapsto_0$
$(\mathbf{emp})\wedge Bdisk=0, upD\leftarrow bd\cdot 4\cdot 3\cdot 2\cdot 1\wedge$
$disD\leftarrow bd.4.(\varnothing)\rangle\}$（㊹）

$Bdisk:=\mathrm{get}(disD);$

㊻ $\{\exists bd\neq 0.\langle len=0\wedge j=0\wedge \mathbf{emp}_V, Bdisk\mapsto_4$
$(\mathbf{emp})\wedge Bdisk=bd, upD\leftarrow bd\cdot 4\cdot 3\cdot 2\cdot 1\wedge$
$disD\leftarrow\varnothing\rangle\}$（A20）

即验证式㊻为循环执行完毕后的状态。数据节点完整接收了来自主节点的存储资源使用权限，对应的地址 bd 与主节点分配的地址以及用户端收到的反馈一致。下一循环体接收来自用户端的信息，即

㊼ $\{\exists \mathrm{bd}\neq 0.\langle \mathrm{len}=0\wedge \mathrm{j}=0\wedge \mathbf{emp}_V, Bdisk\mapsto_4$
$(\mathbf{emp})\wedge Bdisk=bd, upD\leftarrow bd\cdot 4\cdot 3\cdot 2\cdot 1\wedge$
$disD\leftarrow\varnothing\rangle\}$（㊻，R3）

$j:=\mathbf{tryre}(upD);$

㊽ $\{\exists \mathrm{bd}\neq 0.\langle \mathrm{len}=0\wedge \mathrm{j}=\mathrm{bd}\wedge \mathbf{emp}_V,$
$Bdisk\mapsto_4(\mathbf{emp})\wedge Bdisk=bd,$
$upD\leftarrow 4\cdot 3\cdot 2\cdot 1\wedge disD\leftarrow\varnothing\rangle\}$（A18）

用户端按时序提交了写入地址和 4 个待写入的数据共 5 条数值信息。在执行过程中，数据节点无法得知信道内尚未被读取的信息状态，但能在读取地址之后确认循环条件已不再成立。故验证式㊽为循环执行后的最终结果。

后续的验证过程有：

㊾ $\{\exists bd\neq 0.\langle len=0\wedge j=bd\wedge \mathbf{emp}_V,$
$Bdisk\mapsto_4(\mathbf{emp})\wedge Bdisk=bd,$
$upD\leftarrow 4\cdot 3\cdot 2\cdot 1\wedge disD\leftarrow\varnothing\rangle\}$（㊽，R3）

if $Bdisk==j$ **then**

㊿ $\{\exists bd\neq 0.\langle len=0\wedge j=bd\wedge \mathbf{emp}_V,$
$Bdisk\mapsto_4(\mathbf{emp})\wedge Bdisk=bd,$
$upD\leftarrow 4\cdot 3\cdot 2\cdot 1\wedge disD\leftarrow\varnothing\rangle\}$（㊾，TRUE）

其后是一个以 len 为循环变量的循环体。为方便起见，对循环体验证前先对验证式㊿进行等价变换，具体过程如下：

(51) $\{\exists bd\neq 0, \bar{l}, \bar{e}.\langle len=0\wedge j=-\wedge$
$\bar{e}=(4,3,2,1)\wedge \bar{l}\mapsto(e_1,\cdots,e_{len}),$
$Bdisk\mapsto_{4-len}(\bar{l})\wedge Bdisk=bd,$
$upD\leftarrow(e_{len+1},\cdots,e_4)\wedge disD\leftarrow\varnothing\rangle\}$（㊿）

(52) $\{\exists bd\neq 0, \bar{l}, \bar{e}, l'.\langle len=l'\wedge j=-\wedge$
$\bar{e}=(4,3,2,1)\wedge \bar{l}\mapsto(e_1,\cdots,e_{l'}),$
$Bdisk\mapsto_{4-l'}(\bar{l})\wedge Bdisk=bd,$

$upD \hookleftarrow (e_{l'+1},\cdots,e_4) \wedge disD \hookleftarrow \varnothing \rangle\}$（51）

$j := \text{tryre}(upD)$;

53 $\{\exists bd \neq 0, \bar{l}, \bar{e}, l'. \langle len = l' \wedge j = e_{l'+1} \wedge$
$\bar{e} = (4,3,2,1) \wedge \bar{l} \mapsto (e_1,\cdots,e_{l'})$,
$Bdisk \mapsto_{4-l'} (\bar{l}) \wedge Bdisk = bd$,
$upD \hookleftarrow (e_{l'+2},\cdots,e_4) \wedge disD \hookleftarrow \varnothing \rangle\}$（A18）

$Bpool := \mathbf{append}(Bdisk, j)$;

54 $\{\exists bd \neq 0, \bar{l}, l_n, \bar{e}, l'. \langle len = l' \wedge j = e_{l'+1} \wedge$
$\bar{e} = (4,3,2,1) \wedge \bar{l} \cdot l_n \mapsto (e_1,\cdots,e_{l'},e_{l'+1})$,
$Bdisk \mapsto_{4-l'-1} (\bar{l} \cdot l_n) \wedge Bdisk = bd$,
$upD \hookleftarrow (e_{l'+2},\cdots,e_4) \wedge disD \hookleftarrow \varnothing \rangle\}$（A13）

$len := len + 1$;

55 $\{\exists bd \neq 0, \bar{l}, l_n, \bar{e}, l'. \langle len = l' + 1 \wedge j = e_{l'+1} \wedge$
$\bar{e} = (4,3,2,1) \wedge \bar{l} \cdot l_n \mapsto (e_1,\cdots,e_{l'},e_{l'+1})$,
$Bdisk \mapsto_{4-l'-1} (\bar{l} \cdot l_n) \wedge Bdisk = bd$,
$upD \hookleftarrow (e_{l'+2},\cdots,e_4) \wedge disD \hookleftarrow \varnothing \rangle\}$（A2）

56 $\{\exists bd \neq 0, \overline{l_n}, \bar{e}, l'. \langle len = l' + 1 \wedge j = e_{len} \wedge$
$\bar{e} = (4,3,2,1) \wedge \overline{l_n} \mapsto (e_1,\cdots,e_{len})$,
$Bdisk \mapsto_{4-len} (\overline{l_n}) \wedge Bdisk = bd$,
$upD \hookleftarrow (e_{len+1},\cdots,e_4) \wedge disD \hookleftarrow \varnothing \rangle\}$（55）

比对循环体执行前成立的验证式52和执行后的验证式56，可知除循环变量 *len* 自增以外，其他数量关系和性质维持不变，验证式56可以作为循环不变式使用。可有后续验证：

57 $\{\exists bd \neq 0, \overline{l_n}, \bar{e}. \langle len = 4 \wedge j = e_4 \wedge$
$\bar{e} = (4,3,2,1) \wedge \overline{l_n} \mapsto (\bar{e}), Bdisk \mapsto_0 (\overline{l_n})$
$\wedge Bdisk = bd, upD \hookleftarrow \varnothing \wedge disD \hookleftarrow \varnothing \rangle\}$（56，R3）

end if;

58 $\{\exists bd \neq 0, \overline{l_n}, \bar{e}. \langle len = 4 \wedge j = e_4 \wedge$
$\bar{e} = (4,3,2,1) \wedge \overline{l_n} \mapsto (\bar{e}), Bdisk \mapsto_0 (\overline{l_n})$
$\wedge Bdisk = bd, upD \hookleftarrow \varnothing \wedge disD \hookleftarrow \varnothing \rangle\}$（49，57，TRUE）

至此可知，用户端发起的写请求得到了响应，数据节点在主节点赋予的权限范围内执行写入，被写入的块的地址与主节点向用户端反馈的块地址相同，被写入的数据与用户端提交的数据一致，写功能正确。该实例涵盖了循环、通信的重要的验证环节，可以说明本文所提方法的有效性。

6　小　结

本文基于分离逻辑提出了一种适用于并发云存储系统正确性验证的系统方法，在建模语言中引入新的命令和表达式以描述线程间交互行为、扩展断言语言，以准确描述资源受限的执行环境，并基于此完善了推理规则。验证实例说明本文提出方法的可行性。

本文所提逻辑能直观描述存储容量的限制、信道机制能描述权限的转移过程，帮助验证者验证存储管理行为的正确性，识别潜在的溢出和泄漏风险，满足云存储系统的验证所需。本文扩展了块分离逻辑的验证范围，所提方法也可推广到云存储环境下的普通并发程序的正确性验证上，为实现 MapReduce、Spark 等并发框架下的程序验证提供了理论上的可能性。

未来的工作包括：①降低信道机制的复杂度，改变现有通过验证者主动规避冲突的状态，提升其对时序关系的描述精度；②完善所用幽灵状态的数理基础；③探究验证规则的可表达性、相对完备性；④扩展并发正确性验证的适用范围；⑤设计辅助验证工具和算法。

参考文献：

[1]　刘洋. 云存储技术 - 分析与实践[M]. 北京：经济管理出版社，2017.

[2]　艾瑞咨询. 2021 年中国基础云服务行业发展洞察报告[J]. 产品可靠性报告，2021(10)：50-57.

[3]　腾讯科技. 微软再现宕机事故 部分用户 9 天无法使用电邮[EB/OL].（2016-01-28）[2022-01-31]. https：//tech. qq. com/a/20160 128/004262. htm.

[4]　腾讯云. 关于用户“前沿数控”数据完整性受损及腾讯云补偿措施的说明[EB/OL].（2018-08-06）[2022-01-31]. https：//tech. sina. com. cn/it/2018-08-06/doc-ihhhczfc3930310. shtml.

[5]　王戟，詹乃军，冯新宇，等. 形式化方法概貌[J]. 软件学报，2019，30(1)：33-61.

[6]　Reynolds J C. Separation logic：A logic for shared mutable data structures[C]//Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science. Piscataway：IEEE，2002：55-74.

[7]　Hoare C A R. An axiomatic basis for computer programming[J]. Communication of the ACM，1969，12(10)：576-580.

[8]　王捍贫，张博闻. 分离逻辑的技术基础与研究现状[J]. 广州大学学报(自然科学版)，2019，18(2)：1-9.

[9]　Wang H，Jin Z，Zhang L. Reasoning about cloud storage systems[C]//Proceedings of the 3rd IEEE International Conference on Data Science in Cyberspace. Piscataway：IEEE，2018：107-114.

[10] Owicki S, Gries D. An axiomatic proof technique for parallel programs I[J]. Acta Informatica,1976,6:319-340.

[11] Owicki S, Gries D. Verifying properties of parallel programs: An axiomatic approach[J]. Communications of the ACM. 1976,19(5):279-285.

[12] O'Hearn P. Resources, concurrency, and local reasoning[J]. Theoretical Computer Science,2007,375(1/2/3):271-307.

[13] Dinsdale-Young T, Dodds M, Gardner P, et al. Concurrent abstract predicates[J]. Lecture Notes in Computer Science: ECOOP 2010,2010,6183(1):504-528.

[14] 秦胜潮,许智武,明仲.基于分离逻辑的程序验证研究综述[J].软件学报,2017,28(8):2010-2025

[15] Pedro da R P, Dinsdale-Young T. TaDA: A logic for time and data abstraction[J]. Lecture Notes in Computer Science: ECOOP 2014,2014,8586(1):207-231.

[16] Svendsen K, Birkedal L, Parkinson A M. Modular reasoning about separation of concurrent data structures[J]. Lecture Notes in Computer Science: ESOP2013,2013,7792(1):169-188.

[17] Bizjak A, Gratzer D, Krebbers R, et al. Iron: Managing obligations in higher-order concurrent separation logic[J]. Proceedings of the ACM on Programming Languages,2019,3:1-30.

[18] Lei J, Qiu Z, Shao Z. Trace-based temporal verification for message-passing programs[C]//Proceedings of the 2014 Theoretical Aspects of Software Engineering Conference. Piscataway:IEEE, 2014:10-17.

[19] Hinrichsen J K, Bengtson J, Krebbers R. Actris: Session-type based reasoning in separation logic[J]. Proceedings of the ACM on Programming Languages,2020,4:1-30.

[20] Brookes S D, O'Hearn P W. Concurrent separation logic[J]. ACM SIGLOG News, 2016,3(3):47-65.

[21] Turon A, Vafeiadis V, Dreyer D. GPS: Navigating weak memory with ghosts, protocols, and separation[C]//Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications. New York: ACM,2014: 691-707.

[22] Jung R, Krbbers R, Birkedal L. Higher-order ghost state[C]//Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming. New York:ACM,2016:256-269.

[23] Clochard M, Marché C, Paskevich A. Deductive verification with ghost monitors[J]. Proceedings of the ACM on Programming Languages, 2020, 4:1-26.

[24] Jung R, Swasey D, Sieczkowski F,et al. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning[C]//Proceedings of the 42nd Annual ACM SIGPLANSIGACT Symposium on Principles of Programming Languages. New York: ACM,2015: 637-650.

[25] O'Hearn P. Separation logic[J]. Communications of the ACM,2019, 62(2):86-95.

[26] 金钊, 王捍贫, 张博闻, 等. 基于分离逻辑的云存储系统验证[J]. 计算机学报,2020,43(12):2227-2240.

[27] Jin Z, Zhang B, Zhang L, et al. An adaptation-complete proof system for local reasoning about cloud storage systems[J]. Theoretical Computer Science,2022,903:39-73.

[28] 张博闻,金钊,王捍贫,等.一种基于分离逻辑的块云存储系统验证工具[J].软件学报,2022,33(6):2264-2287.

【责任编辑:陈 钢】