软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(10):2993−3013 [doi: 10.13328/j.cnki.jos.006026]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于着色 Petri 网的 HDFS 数据一致性建模与分析*

乔嘉林[1], 黄向东[1], 杨义繁[1], 王建民[1], 吴 凯[2]

[1](清华大学 软件学院 大数据研究中心,北京 100084)

[2](新疆金风科技股份有限公司,新疆 乌鲁木齐 830026)

通讯作者: 黄向东, E-mail: huangxdong@tsinghua.edu.cn

摘 要: HDFS 分布式文件系统作为 Apache Hadoop 的核心组件之一,在工业界得到了广泛应用.HDFS 采用了多副本机制保证数据的可靠性,但是由于多副本的存在,在节点失效、网络中断、写入失败时可能会导致数据不一致.与传统文件系统相比,HDFS 被认为其数据一致性有所降低,但用户并不知道何时会出现不一致的情况,目前也没有相关工作对其一致性机制进行验证说明.当数据存在一致性问题时,会增加上层应用的不确定性.可见,对数据一致性的研究十分必要.HDFS 的软件规模庞大,且在分布式环境下运行,针对这些特点,采用了着色 Petri 网建模和状态空间分析的方法,完成了以下工作:(1) 使用着色 Petri 网对 HDFS 的读写流程建立模型,该模型详细刻画了 HDFS 内部各个组件的功能及相互协作的机制;(2) 基于着色 Petri 网模型,使用状态空间工具分析了 HDFS 的数据层一致性和操作层一致性,验证并详细说明了 HDFS 出现不一致的条件;(3) 在分析过程中,提出了“时点重复读”的操作层一致性验证方法,采用串行化的重复读策略降低了状态空间复杂度.基于以上工作,针对 HDFS 上层应用的开发给出建议,帮助提高应用的数据一致性.此外,在建模过程中提出的建模技巧给基于 CPN Tools 工具分析其他系统提供了借鉴.

关键词: HDFS;一致性;建模;着色 Petri 网;CPN Tools

中图法分类号: TP311

中文引用格式: 乔嘉林,黄向东,杨义繁,王建民,吴凯.基于着色 Petri 网的 HDFS 数据一致性建模与分析.软件学报,2021,32(10):2993−3013. http://www.jos.org.cn/1000-9825/6026.htm

英文引用格式: Qiao JL, Huang XD, Yang YF, Wang JM, Wu K. HDFS data consistency modelling and analysis based on colored Petri net. Ruan Jian Xue Bao/Journal of Software, 2021,32(10):2993−3013 (in Chinese). http://www.jos.org.cn/1000-9825/6026.htm

HDFS Data Consistency Modelling and Analysis Based on Colored Petri Net

QIAO Jia-Lin[1], HUANG Xiang-Dong[1], YANG Yi-Fan[1], WANG Jian-Min[1], WU Kai[2]

[1](School of Software, Tsinghua University, Beijing 100084, China)

[2](Xinjiang GoldWind Sci & Tech Co., Ltd., Urumqi 830026, China)

Abstract: As one of the core components of Apache Hadoop, the Hadoop distributed file system (HDFS) has been widely used in the industry. HDFS adopts a multiple replicas mechanism to ensure data reliability, which may incur inconsistency because of node failure, network partition, and write failure. HDFS is considered to have reduced data consistency compared to traditional file systems, which is difficult for users to understand when there will be inconsistent. At present, there is no relevant work to verify the consistency mechanism. When the data is inconsistent, it will increase the uncertainty of the upper applications. Thus, research for data consistency model is required. The large scale of HDFS makes the analysis more difficult. Code reading, abstracting, colored Petri net modeling, and state-space analysis are conducted to comprehend the system. The works are listed as the following. (1) Colored petri nets are used to model HDFS’s process of reading and writing files, the model describes the functions of inner components and their cooperation

* 基金项目: 国家自然科学基金(71690231, 61802224)
Foundation item: National Natural Science Foundation of China (71690231, 61802224)
收稿时间: 2018-11-14; 修改时间: 2019-07-02, 2019-09-16, 2020-01-18; 采用时间: 2020-02-27

mechanism in detail. (2) Data layer consistency and operation layer consistency of HDFS are analyzed with state-space tools based on a colored Petri net model, figuring out data consistency guaranteed by the system. (3) A time point repeatable read method is proposed to verify operation layer consistency and serial repeatable strategy is utilized to decrease state-space complexity. Based on the contribution above, the directions for HDFS application development are proposed, helping to improve the data consistency. The CPN modeling method and technique are applicated in the analysis of other distributed information systems.

Key words: HDFS; consistency; modelling; colored Petri net; CPN tools

Hadoop 是提供分布式存储和计算的一套开源软件系统[1,2],通过多年的发展,Hadoop 已经成为分布式存储计算领域事实上的行业标准,得到了广泛的应用.Hadoop 分布式文件系统(Hadoop distributed file system,简称 HDFS)是为 Hadoop 提供文件存储支撑的重要组成部分,是谷歌文件系统(Google file system)[3]的开源实现,它能够基于普通商用机器集群,提供高吞吐量、高可靠性及高可用性的存储服务.

HDFS 的设计目标主要有两方面:一是最大化系统吞吐率,二是提供可靠、高可用的服务.根据 CAP 定理(CAP theorem)[4],系统在一致性(consistency,即所有节点在同一时间访问相同数据的特性)、可用性(availability,即保证请求能收到回应的特性)和分区容忍度(partition tolerance,即在信息丢失或部分节点失效的情况下继续提供服务的特性)这 3 种特性上必须做出取舍,不可能同时满足.HDFS 为了达到设计目标,优先满足可用性,其次考虑分区容忍度,牺牲了一致性[5,6].

HDFS 的弱一致性具有重要的研究价值,因为它不能保证在同一时刻不同节点访问相同数据,这会给使用系统服务的上层应用带来设计的复杂性.为了在进行应用开发时对数据不一致进行恰当的处理,就必须深入理解 HDFS 的一致性模型.

为了研究 HDFS 的数据一致性,必须要克服如下 3 个技术难点:首先,HDFS 系统规模庞大,如何将与一致性分析相关的代码筛选出来是一个难点;其次,作为分布式系统,HDFS 是由位于不同服务器上的多个组件组成的,节点自身需要完成哪些工作、需要响应哪些请求、节点之间如何进行消息传递、如何进行协同操作,是分析分布式系统的重点问题;最后,分布式系统难以验证,如何建模系统状态并进行一致性检查,是一项巨大的挑战.

为了解决以上问题,本文基于着色 Petri 网[7]实现了 HDFS 数据一致性分析.研究对象为 hadoop-1.0.1,代码包为 org.apache.hadoop.hdfs,共包含 182 个 Java 文件,33 965 行 Java 代码,文献[7]中的数据结构建模及读写流程建模均与此代码相对应.研究过程如下:(1) 通过阅读源码对写文件和读文件两个流程中的重要的数据结构、本地操作以及系统组件之间的协作进行分析;(2) 使用 CPN Tools[8]建立着色 Petri 网模型,准确刻画 HDFS 在数据一致性方面的算法流程;(3) 使用 CPN Tools 的状态空间分析工具进行数据一致性分析.本文分析的目标是寻找 HDFS 系统不一致状态的发生条件及其后果,思路是用模型模拟系统的运行,并分析各中间结果的状态.

本文关注两种层面的一致性场景:(1) 数据层的一致性,即关注每个状态下同一份数据的多个副本之间是否一致;(2) 操作层的一致性,即关注当同时有多个客户端并发读一份数据时,是否能读到相同的数据.研究方式为暂停所有写操作的流程,并执行多个读操作,比较这些读操作返回结果是否一致.

本文的主要贡献包括:

(1) 使用着色 Petri 网对 HDFS 的读写流程建立模型.详细刻画了 HDFS 内部各个组件的功能及相互协作的机制;

(2) 使用状态空间工具分析了 HDFS 的数据层一致性和操作层一致性.提出面向一致性分析的数据模型简化方法,在满足一致性检验要求的前提下简化模型;提出“时点重复读”的操作层一致性判定方法;

(3) 使用 CPN Tools 对实际系统进行了完整的分析,分析了 HDFS 存在的一致性问题,对上层应用开发提出了实用的建议,可以增强上层应用的一致性,并且为分析其他分布式系统的一致性提供了可参考的方法.

本文第 1 节介绍相关工作,包括 HDFS 系统建模、着色 Petri 网和 CPN Tools.第 2 节分析 HDFS 的核心数据结构并对其建模.第 3 节介绍 HDFS 的读写文件流程,并详细介绍了建模网络结构.第 4 节介绍状态空间工具集的功能,使用该工具集对 HDFS 的一致性进行分析,从数据层一致性和操作层一致性这两个维度,分别在无错

条件和有错条件下进行分析,总结系统在各种条件下提供的一致性保证.第 5 节总结本文工作.

1 相关工作

1.1 HDFS相关研究

HDFS 作为开源分布式计算平台 Hadoop[1]的核心组件之一,在海量数据管理方面得到了广泛应用.雅虎在2012年已经在4个数据中心部署了42 000个Hadoop节点;Facebook在同年部署的Hadoop集群能够处理100PB的数据,并且数据量还在以每天 0.5PB 的速度增长[9].文献[10]列举了 Hadoop 在多种分布式计算场景下的应用情况,由此可见其应用范围之广.除此之外,还有很多公司提供商业版的 Hadoop 及相关支持,如 Cloudera、IBM、MapR、EMC 和 Oracle 等[9].同时,HDFS 系统复杂度较高,1.0.0 版本的 Hadoop 系统源代码有 52.6 万行,包含 2 353 个 Java 类,规模庞大,并且是分布式的系统设计,给数据一致性研究带来了较大困难.

HDFS 的原型系统 GFS[3]从设计者的角度进行说明,认为系统提供了较松的一致性(relaxed consistency),在不同条件下提供差异化的一致性保证,对于写文件操作,成功的串行写能保证数据一致且有确定的顺序,成功的并行写能保证数据一致但不保证确定的顺序,出现失败的操作不提供一致性保证.文献[11]在HDFS上层封装了新的文件系统接口,针对接口进行测试以观察其数据一致性模型,认为 HDFS 提供文件操作的顺序一致性,即从单个文件来看,所有操作执行的结果等效于这些操作按照某一个顺序序列执行的结果.但是,该顺序是由系统程序指定而非受用户控制.同时,对于整个文件系统来说并不是顺序一致的.Facebook 在使用 HDFS 构建新系统时,认为它提供了强一致性保证,并且可以避免数据恶化[12].从上述分析可知:就 HDFS 的一致性问题缺少系统性的研究和分析,没有能够形成共识.在研究方法上也以基于模型的理论推断以及系统外部行为分析为主,缺少基于源代码的分析案例.

1.2 大数据系统建模

国内外很多学者[13−15]对大数据系统从软件工程的角度进行建模,分析系统运行轨迹,并给出了有效的优化方向.文献[16]从 HDFS 文件系统的架构入手,通过对模型各个组成部分的分析,总结出了其在云计算领域中的应用优势以及存在的问题.由于整个工作和分析在架构层面进行,难免会丢失一些 HDFS 实现中的细节.文献[17]使用 UML 对分布式文献系统进行了建模,该方法易于理解,但由于无法在基于 UML 图的模型上进行仿真和数学分析,因此,该模型无法分析 HDFS 的一致性问题.文献[18]为了分析 HDFS 集群在节点频繁迁入迁出时的运行状态,使用了着色 Petri 网,分别对客户端、命名节点(namenode)、数据节点(datanode)进行了读写流程建模,通过 CPN Tools 对模型进行仿真,在集群动态变化的情况下,读写操作的成功率随副本数量发生变化.文献[18]发现:随着副本数量的增多,写操作成功率逐渐降低,这是由于写操作需要在全部副本都执行成功;而在集群动态变化时,任意节点都可能随时宕机,因此副本越多,写失败的概率越大.为了增加写操作成功率,作者修改了写操作成功的条件,只需有一个副本的写操作执行成功即可,从而提升了写操作的成功率.该工作提出的模型没有对副本的数据值进行建模,从而缺失了对副本是否一致的建模,因此无法用于 HDFS 的一致性分析.但是一致性是本文的重点研究目标,该工作与本文的研究目标不同,但是研究方法具有一定的相似性,均通过 Petri 网建模进行系统行为分析,具有一定的借鉴意义.以上对 HDFS 的建模和分析工作大多无法精细到各个节点内部的实现机制,难以得到数据一致性等问题的结论.此外,传统的建模工具得到的静态模型无法直接进行仿真,大大降低了所建立的模型的实用性.文献[19]对 HDFS 进行了建模,并对数据块多个副本的值进行了建模和检查,但未得到数据副本不一致产生原因的准确原因.本文在此基础上进行了细粒度建模,并进行了更深入的分析和论证,明确了 HDFS 无法满足读操作一致性的发生场景和原因.

1.3 Petri网概述

Petri 网是由 C.A.Petri 提出的一种图形化的建模工具[20].Petri 网适用于对并发、异步、分布式、并行、非确定性和随机的信息处理系统进行建模,在工作流、分布式系统分析等领域有着广泛的应用.简单的图形化表示是 Petri 网的重要特性之一,仅使用库所、变迁、弧和令牌这 4 种简单的元素,即可对复杂的系统进行表达.

在简单的图形化表示方法之外,Petri 网还具有严格的数学定义、执行语义和用于分析的数学理论,提供了丰富的模型分析工具.文献[21]是 Petri 网领域研究的综述,是 Petri 网理论发展中的一次重要总结.

Petri 网的正则定义如下.

Petri 网(Petri net,简称 PN)是一个五元组 $PN=(P,T,F,W,M_0)$,其中,$P=\{p_1,p_2,\dots,p_m\}$表示库所(place)的有限集合,库所中包含若干令牌,作为资源对象;$T=\{t_1,t_2,\dots,t_n\}$表示变迁(transition)的有限集合;$F\subseteq(P\times T)\cup(T\times P)$表示弧(arc)的集合;$W:F\to\{1,2,3,\dots\}$表示权重函数;$M_0:P\to\{0,1,2,\dots\}$表示初始标记.$P\cap T=\varnothing$并且 $P\cup T\neq\varnothing$.

Petri 网的发射规则定义如下.

(1) 对一个变迁 t 来说,当且仅当每个输入库所 p,都标记了最少 $w(p,t)$个令牌,这个变迁才是使能的.其中,$w(p,t)$表示由 p 指向 t 的弧的权重;

(2) 使能的变迁可以发射(fire)或不发射;

(3) 当发射一个使能的变迁 t 时,从 t 的每个输入库所 p 中删除 $w(p,t)$个令牌,在 t 的每个输出库所 p 中添加 $w(t,p)$个令牌.其中,$w(t,p)$表示由 t 指向 p 的弧的权重.

图 1 所示为 Petri 网发射规则的示意图,输入库所的弧权重分别是 2 和 1,输出库所的弧权重为 2,发射导致变迁的输入库所分别减少两个和一个令牌,输出库所增加两个令牌.

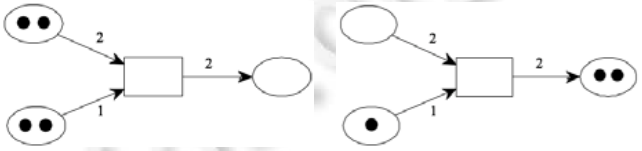

Fig.1 An example of firing an action

图 1 Petri 网发射规则示意

1.4 着色Petri网概述

Petri 网虽然提供了简单的建模元素,但其表达能力较低.为了进一步提高 Petri 网的建模效率,学界提出了多种高级 Petri 网模型,包括随机 Petri 网(stochastic Petri net,简称 SPN)[22]、着色 Petri 网(colored Petri net,简称 CPN)[7,23]等.

着色 Petri 网在 Petri 网模型的基础上添加了颜色集、颜色函数、弧表达式函数、守护函数和初始化函数等元素.在围绕令牌着色提出一系列改动之后,模型的表达能力得到了很大程度的扩展:令牌通过着色,可以表达更丰富的语义信息;弧表达式函数对令牌进行选择;另外,还有变迁守护函数对发射条件的控制,使模型的流程控制能力更强,可以实现复杂的分支选择和控制逻辑.因为着色 Petri 网与 Petri 网相比具有更强的表达能力,所以在对同一个系统进行建模时,着色 Petri 网能够以相对更少的模型元素进行相同语义的建模.在本文中,我们使用着色 Petri 网对系统进行建模.

1.5 CPN Tools概述

CPN Tools[24]是一套着色 Petri 网的编辑、仿真和分析工具,从 2000 年到 2010 年,由丹麦奥胡斯大学(Aarhus University)的 CPN 组负责研发.2010 年,该工具转交荷兰埃因霍温科技大学的 AIS 组继续开发和维护.CPN Tools 的主要功能包括:(1) 提供了着色 Petri 网的图形化建模工具,能够方便、快捷地建立模型;(2) 提供了着色 Petri 网模型的分析引擎,支持模型仿真、监控和状态空间分析等多种分析工具.

CPN Tools 采用 ML 语言(ML language)[25]作为模型的定义语言,ML 语言由爱丁堡大学的 Milner 提出,是一种函数式编程语言.本文使用 ML 语言进行系统建模及一致性检查.由于 CPN Tools 对 Petri 网的符号作了进一步的修改,以增强表达能力和可视化效果,如在库所中嵌入文字以及双线椭圆来代表折叠的 Petri 网等,因此包含了不同于标准 Petri 网的符号.为了能够更好地对系统行为加以介绍,本文使用 CPN Tools 中的模型图进行解释说明.

2 数据结构建模

本文首先介绍 HDFS 基本架构,并对 HDFS 中的一些数据结构进行建模,数据结构建模的任务是将 HDFS 的数据结构代码进行抽象,以着色 Petri 网模型的语言进行表达.本节我们不给出具体建模代码,仅使用文字描述建模对象内容.

2.1 HDFS基本架构

HDFS 作为分布式文件系统,由多个不同组件协同完成系统功能.了解系统架构,是建模和分析的基础[26].HDFS 采用主从模式的系统架构,如图 2 所示,它由 3 种不同的组件构成,即名字节点、数据节点和客户端,其主要功能简单介绍如下.

- 名字节点(namenode)是 HDFS 的核心组件和主节点,每个 HDFS 集群只有唯一的名字节点.该节点集中管理文件系统结构、文件源信息、文件与文件块的对应信息、文件块分布信息以及活动数据节点信息等.名字节点的主要功能包括:处理客户端对文件元数据的读写请求,维护文件系统树状结构;管理数据节点,提供数据节点的注册、注销等功能;维护文件块的分布信息,在处理读写请求时提供文件块的具体位置信息;
- 数据节点(datanode)是 HDFS 的从节点,每个 HDFS 集群通常有多个数据节点.数据节点是实际存储文件数据的地方,以文件块(block)为单位进行数据的存取.数据节点的主要功能包括:处理来自于客户端或其他数据节点的写操作请求,将文件块存储到本地文件系统中;处理来自于客户端或其他数据节点的读操作请求,将存储到本地的文件块读出并返回;在名字节点处注册,保持活动状态,并定时上传本地存储的文件块信息;
- 客户端(client)是部署在应用端的组件,提供对 HDFS 文件系统的访问接口,允许同时有多个客户端与系统进行连接.客户端的主要功能包括:处理来自上层应用的文件元数据操作,与名字节点交互完成元数据的管理操作;处理来自用户的读写文件请求,与名字节点交互更改或读取文件元数据,与数据节点交互提交或读取文件块,并将操作结果返回给上层应用.

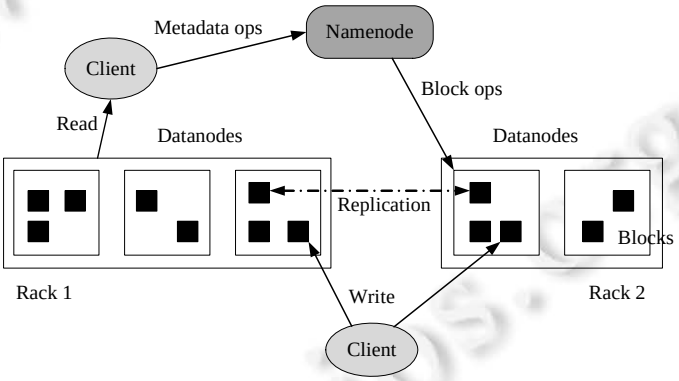

Fig.2 System architecture of HDFS

图 2 HDFS 系统架构

2.2 颜色集

着色 Petri 网用令牌(token)表示对象,用令牌的颜色来区分对象的类型.对 HDFS 的数据结构进行建模,就是使用 ML 语言对颜色集进行定义,并用颜色集与 HDFS 的数据结构建立对应的关系.CPN Tools 中的颜色集分为两种,即简单颜色集和复合颜色集:简单颜色集是系统预先定义的几种基本颜色集,包括布尔颜色集、整数颜色集、索引颜色集、枚举颜色集等;复合颜色集是由其他颜色集组合派生出来的颜色集,包括列表颜色集、乘积颜色集、记录颜色集、并颜色集等[7].

2.3 HDFS数据结构建模

HDFS 文件系统是按照树形结构组织而成的.所有文件在文件系统树中就可以用一个节点来唯一对应,该节点称为该文件的 inode.一个文件包含两部分:文件的源信息和文件实际存储的数据.在 HDFS 中,文件的源信息存储在名字节点的 inode 中,文件的数据以文件块的形式存储在数据节点上.文件块默认的容量是 64MB,每个文件会根据数据量的大小存储在一个或多个块中,由此产生了从文件到文件块的一对多映射,本文称这种映射关系为第 1 关系.HDFS 作为分布式系统考虑了节点失效问题,其解决方案就是使用数据副本:当存储某文件块的一些数据节点失效时,由未失效的数据节点上的文件块的副本对外提供服务,保持服务的可用性.由此产生了从文件块到数据节点的一对多映射,本文称这种映射关系为第 2 关系.

重要的数据结构及其包含的属性如下所示.

- 文件(FILE):文件名;
- 文件块(BLOCK):块 ID、文件块大小、时间戳;
- 第 1 关系(FILE_INODE):文件名、文件块的列表、是否正在构建、正在写的客户端 ID、正在写的最后一个文件块的数据节点的列表(仅当该文件处于构建状态,客户端 ID 和正在写的数据节点列表才有意义);
- 第 2 关系(LOCATED_BLOCK):文件块、数据节点列表(在写入数据时,按照该列表中的顺序依次发送写请求执行写入);
- 副本位置(STORED_BLOCK):所在数据节点、文件块;
- 租约(LEASE):客户端 ID、文件列表(租约是文件的写锁,写锁不排斥读操作.每个文件只有一个租约,客户端需要定期更新租约,否则租约管理器会回收租约.一个客户端同时可拥有多个文件的租约).

3 读写流程建模

3.1 写流程概述

HDFS 的写文件流程由客户端(client)、名字节点(namenode)和数据节点(datanode)共同完成.一次完整的写文件流程一共分为 6 步,如图 3 所示.

- 第 1 步,客户端向名字节点发送创建文件请求,该请求中只包含文件的元数据,名字节点记录文件的元数据;
- 第 2 步,每次客户端需要写新的文件块时,就向名字节点发送追加该文件块请求,名字节点为此文件块分配多个数据节点,并将当前最后一个文件块(即正在写入的文件块)所在的数据节点记录在名字节点的文件 inode 中,最后将该文件块的标识符和数据节点返回给客户端;
- 第 3 步,客户端发送文件内容给目标数据节点:文件内容以数据流的形式发送给第 1 个数据节点,数据节点接收到数据流后,首先在本地存储文件块,并将该数据流转发给下一个数据节点.值得注意的是:对于一个文件块 *A*,假设其要被写入到数据节点 *a*、*b*、*c* 中,名字节点首先记录 *A* 应该存储在节点 *a*、*b*、*c* 中,由于 Hadoop 采用了副本链式传播机制,客户端先将文件块 *A* 发送给 *a* 节点,再由 *a* 传递给 *b*,*b* 传递给 *c*.此时可能出现如下状态:*a* 已写完文件块,*b*、*c* 还未写完文件块.我们称此状态的文件块为未写完的文件块,并将所有副本都写完的文件块称为已写完的文件块;
- 第 4 步,完成写入的数据节点在接收到下游节点的结果后,整合结果并返回给上一个数据节点,直到返回给客户端;
- 第 5 步,在任意一个数据节点写文件块成功后,都会向名字节点发送成功接收文件块通知.名字节点此时会保存该文件块存储的位置,并更新第 2 关系的映射;
- 第 6 步,客户端向名字节点发送关闭文件请求.名字节点会释放相关的资源,写流程结束.

当写入的文件包含多个文件块时,则会循环执行第 1 步到第 5 步,直到所有文件块都写完.在此循环过程中,第 1 步客户端发送的信息为申请新文件块,而非创建新文件.

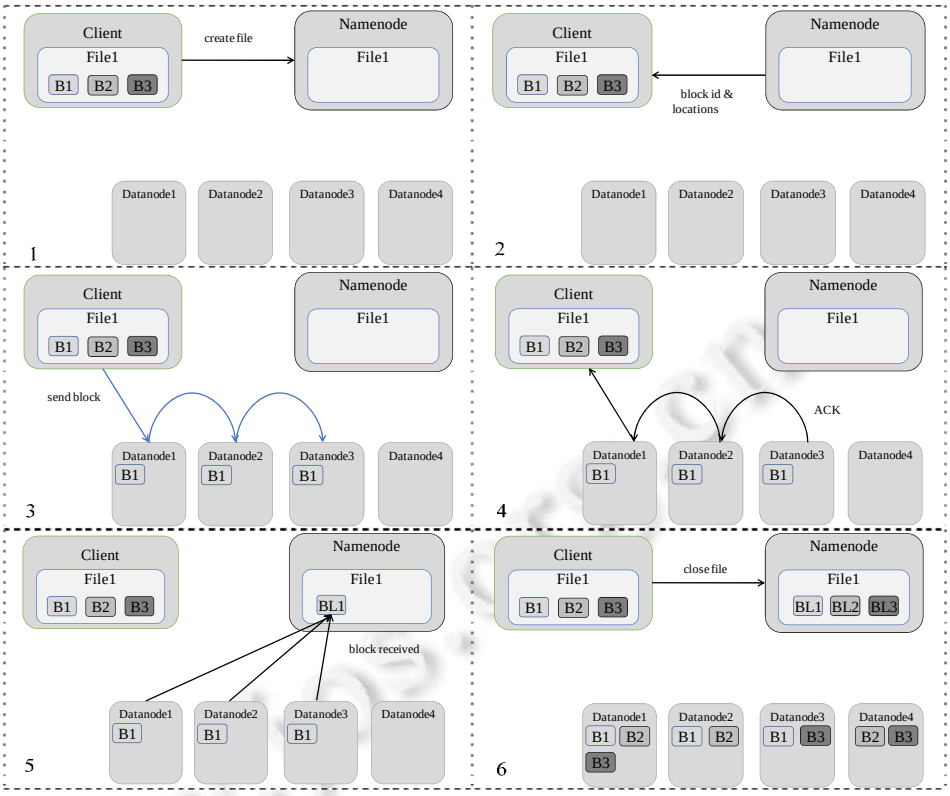

Fig.3 Writing process of HDFS

图 3 HDFS 写文件流程示意图

3.2 写流程建模

3.2.1 写流程模型概述

写流程的着色 Petri 网模型框架如图 4 所示,主要包括两个子流程:(1) 名字节点处理流程 NamenodeWrite; (2) 数据节点处理流程 DatanodeWrite.

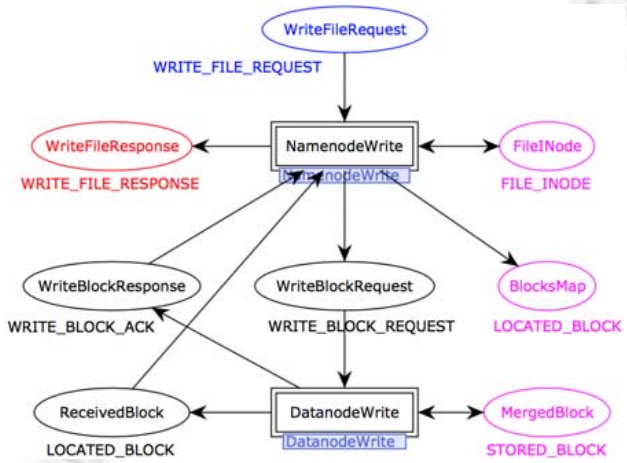

Fig.4 Writing process model

图 4 写文件模型

写流程的输入是客户端发起的写文件请求(库所 WriteFileRequest),输出是写文件请求的回复(库所 WriteFileResponse).输出包括第 1 关系(库所 FileINode)和第 2 关系(库所 BlocksMap)、数据节点存储的文件块信息(库所 MergedBlock).另外 3 个库所是写流程的中间结果,分别是发送给数据节点的写文件块请求(库所 WriteBlockRequest)、数据节点发给名字节点的接受块报告(库所 ReceivedBlock)、数据节点发给客户端的写文件块回复(库所 WriteBlockResponse).

写流程如下:上层应用的请求作为系统输入到达客户端,客户端联系名字节点创建文件元信息并申请文件块,根据返回结果构造写文件块请求(NamenodeWrite 子流程);该请求发送到数据节点并写入数据,数据节点返回给客户端写文件块回复(DatanodeWrite 子流程);客户端关闭文件并释放资源,向上层应用返回写文件结果(NamenodeWrite 子流程).

3.2.2 NamenodeWrite 流程

NamenodeWrite 是客户端向名字节点写入文件元信息的流程.图 5 所示为 NamenodeWrite 的模型,该流程包含两个子流程:打开文件并获取存储位置的子流程 OpenFileGetWriteLocations 和处理写文件块结果的子流程 ProcessWriteBlockResult.该流程的输入有:写文件请求(库所 WriteFileRequest)、写文件块回复(库所 WriteBlockResponse)、接收文件块通知(库所 ReceivedBlock).该流程的输出有:写文件块请求(库所 WriteBlockRequest)、第 2 关系(库所 BlocksMap)、写文件回复(库所 WriteFileResponse).既是输入又是输出的是第 1 关系(库所 FileINode),中间结果是租约(库所 Lease)和等待回复的写文件块请求(库所 WaitWriteFile).

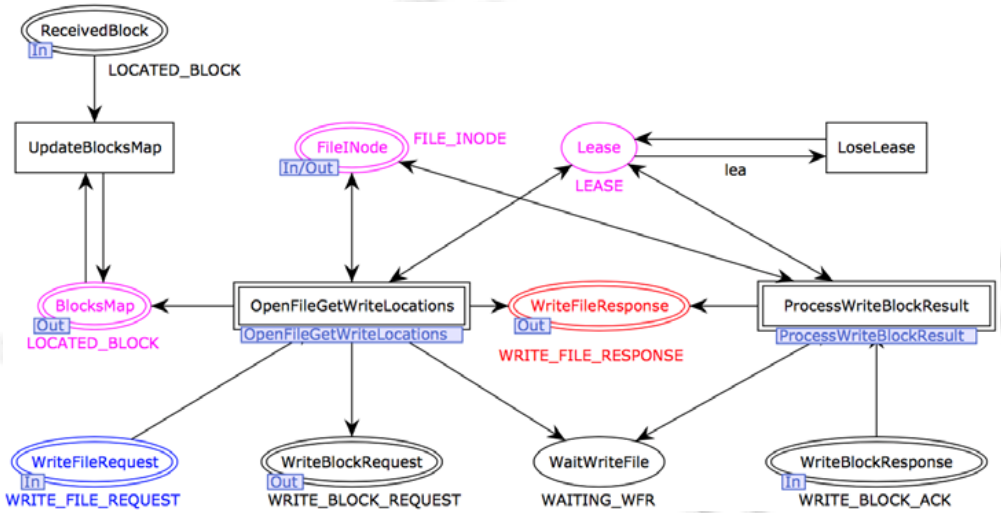

Fig.5 NamenodeWrite model

图 5 NamenodeWrite 模型

NamenodeWrite 流程为:客户端接收到写文件请求,联系名字节点创建文件,创建过程中会检查并更新第 1 关系、第 2 关系和租约,生成写文件块请求(由 DatanodeWrite 流程处理)和等待回复的写文件块请求,或由于失败直接产生写文件回复(由 OpenFileGetWriteLocations 子流程处理).写文件块回复由 ProcessWriteBlockResult 子流程处理.接收文件块通知由 UpdateBlocksMap 变迁更新第 2 关系.

在文件写入流程中,客户端需要定时地向名字节点发送心跳维护租约,流程中对写入流程中出现错误(如网络超时、节点失效、本地写磁盘失败等)建模为一次租约失效,由 LoseLease 变迁模拟.

3.2.3 OpenFileGetWriteLocations 流程

OpenFileGetWriteLocations 流程如图 6 所示,输入是写文件请求(WriteFileRequest),输出包括写文件回复(WriteFileResponse)、等待回复的写文件请求(WaitWriteFile 库所,记录了当前写成功的文件块数)、写文件块请

求(WriteBlockRequest 库所)和第 2 关系(BlocksMap 库所).既是输入又是输出的是租约(Lease 库所)和第 1 关系(FileINode 库所).中间数据包括创建文件请求(CreateFile 库所)、获得后续文件块请求(GetFollowingBlock 库所)、文件块计数器(BlockCounter 库所)和复合中间结果(WFB 库所).

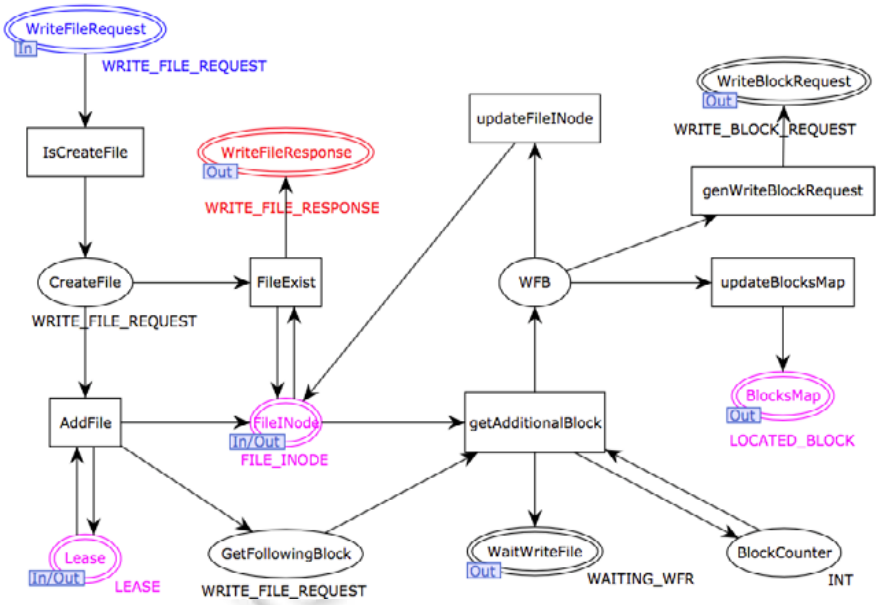

Fig.6 OpenFileGetWriteLocations model

图 6 OpenFileGetWriteLocations 模型

主要流程为:根据写文件请求,生成创建文件的请求(IsCreateFile 变迁),如果该文件已经存在,生成创建失败的结果返回(FileExist 变迁);否则检查并获取文件租约,在第 1 关系添加文件元数据(AddFile 变迁),生成获取后续文件块请求,getAdditionalBlock 变迁负责处理获取文件块的请求,根据文件块计数器和第 1 关系,生成等待回复的写文件请求、复合中间结果(更新第 1 关系的信息、更新第 2 关系的信息和写文件块请求),更新第 1 关系的信息由 updateFileINode 变迁处理,更新第 2 关系的信息由 updateBlocksMap 变迁处理.写文件块请求由 genWriteBlockRequest 变迁处理.

3.2.4 DatanodeWrite 流程

DatanodeWrite 流程是客户端向数据节点上写文件块过程的建模.如图 7 所示,该流程的输入是写文件块请求(WriteBlockRequest 库所),输出是写文件块回复(WriteBlockResponse 库所)、接收文件块通知(ReceivedBlock 库所)、更新数据节点存储的文件块(MergedBlock 库所).中间结果包括本地写文件块请求(WritingBlock 库所)、未合并的文件块(UnmergedBlock 库所)、失败的写文件快回复(WriteLocalFail 库所)、更新写文件块回复(WriteBlockAck 库所)、成功的写文件块回复(WriteLocalSuccess 库所).

主要流程为:

- Receive 变迁的输入和输出相同,作为标识,ReceiveAndTransfer 变迁模拟需要写入的每个数据节点接受到请求的行为(下文我们用“写入链”表示需要按需写入的数据节点链表).包括生成本地写文件块请求(WritingBlock 库所)和发给下一节点的写文件块请求(WriteBlockRequest 库所).本地写入成功后会产生一个通知名字节点接受文件块信息;
- Fail 变迁模拟了本地写文件块失败的情况.写文件块回复(WriteBlockAck 库所)是写文件块过程中链条下方的节点向上一节点发送的回复.genLastFAck 变迁处理写链条最后一个节点本地写失败的情况,直接产生写文件块失败回复.genMidFAck 变迁用来生成失败的写文件块回复,包括中间节点的回复和写

入链第 1 个节点的回复.该变迁的输入是本地失败的写文件回复(WriteLocalFail 库所)和下游节点的写文件块回复.isFirstAck 变迁将 WriteBlockAck 库所中第 1 个数据节点的回复转移到 WriteBlockResponseOut 库所中,作为返回给客户端的输出;

- Success 变迁表示成功执行本地写入,产生未合并的文件块.由于一个文件块可能由多次写文件的请求的数据拼接而成,AddNew 和 Merge 变迁共同完成文件块的新增或合并过程.genLastSAck 变迁处理写入链最后一个节点的情况,并通知名字节点存储成功.genMidSAck 变迁处理中间节点接收到写入链下游成功回复的情况,向写入链上游返回成功的回复.genMidFAck2 变迁处理本地写成功但写入链下游节点写失败的情况,返回失败的回复给写入链上游节点.

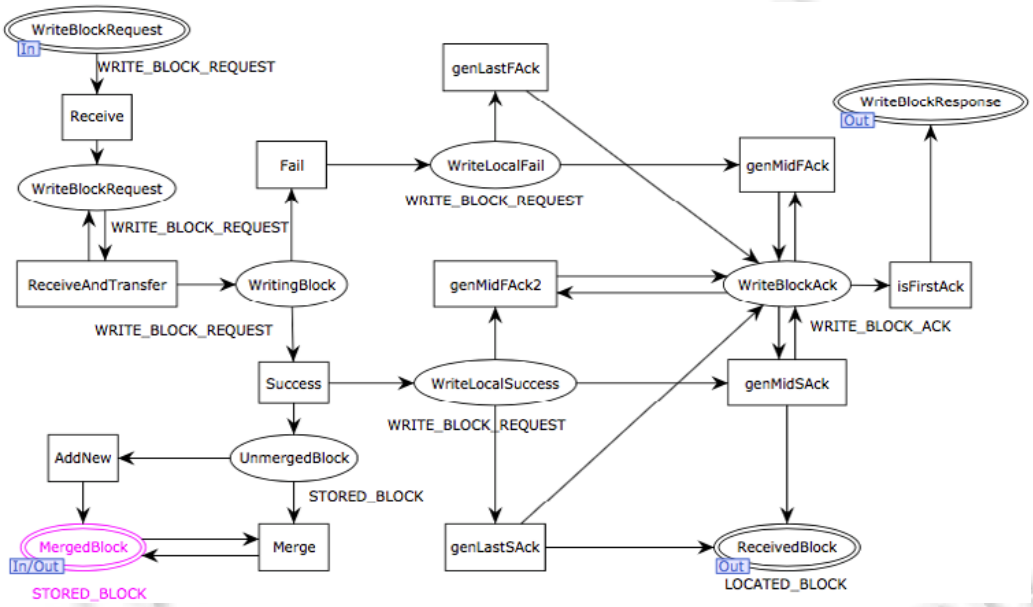

Fig.7 DatanodeWrite model

图 7 DatanodeWrite 模型

3.2.5 ProcessWriteBlockResult 流程

ProcessWriteBlockResult 流程是处理写文件块回复的流程,如图 8 所示,输入是 DatanodeWrite 流程返回的写文件块回复(WriteBlockResponse 库所),输出是写文件回复(WriteFileResponse 库所),既是输入又是输出的是等待回复的写文件请求(WaitWriteFile 库所)、租约(Lease 库所)和第 1 关系(FileINode 库所).中间数据包括需要删除的写文件块请求(FailedWriteToRemove)、失败的写文件请求(FailedWrite 库所)和成功的写文件请求(WriteFileSuccess 库所).一个文件通常包括一个或多个文件块,所以一个写文件请求会生成一个或多个写文件块请求,每次成功的写文件请求会等待所有写文件块请求都返回成功才可以返回,但一旦出现失败的写文件块回复,就可以直接返回失败的写文件回复.

主要流程为:获得写文件块回复和等待回复的写文件请求,如果是成功的写文件块回复,WriteFileSuccess 变迁会更新已经接受到的回复数量(WaitWriteFile 库所).如果回复的数量等于等待回复的数量,且所有回复都是成功的,则产生成功的写文件请求.WriteFileFail 变迁处理失败的写文件块回复:删除 WaitWriteFile 库所中的令牌,停止等待;输出到 FailedWrite 库所,进入失败写操作的处理流程;输出到 FailedWriteToRemove 库所,作为该文件写失败的标记.SuccessCloseFile 和 FailCloseFile 变迁处理成功和失败的写文件块回复.其输出包括:删除客户端对该文件持有的租约;修改文件状态 FileINode.产生成功或失败的写文件回复,输出到 WriteFileResponse 库所.

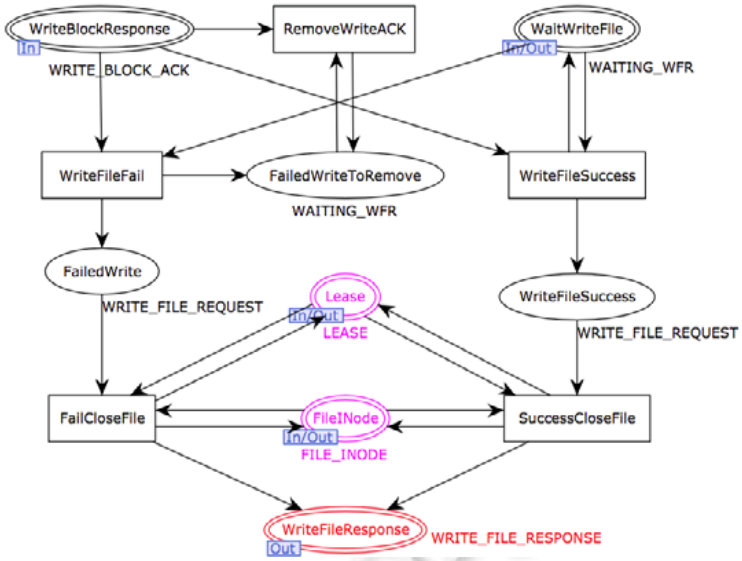

Fig.8 ProcessWriteBlockResult model

图 8 ProcessWriteBlockResult 模型

3.2.6 写流程建模小结

总的来说,写流程是将客户端接收的写文件数据,转化为 HDFS 内部数据结构的过程.主要的内部数据结构就是第 1 关系映射(FileINode)、第 2 关系映射(BlocksMap)和数据节点存储的文件块数据(MergedBlock),这些内部数据结构是读流程得以执行的基础.

3.3 读流程概述

读流程可以分为 4 个主要步骤,如图 9 所示:第 1 步是打开文件,由客户端向名字节点发送打开文件请求,同时请求获得所有文件块的位置信息;第 2 步是名字节点将所有已写完和未写完的文件块所在的数据节点返回给客户端;第 3 步是对于每个文件块,客户端依次发送读文件块请求到其已知的数据节点;第 4 步是文件块传输,由接收到读文件块请求的数据节点发起,将文件块数据发送给客户端.

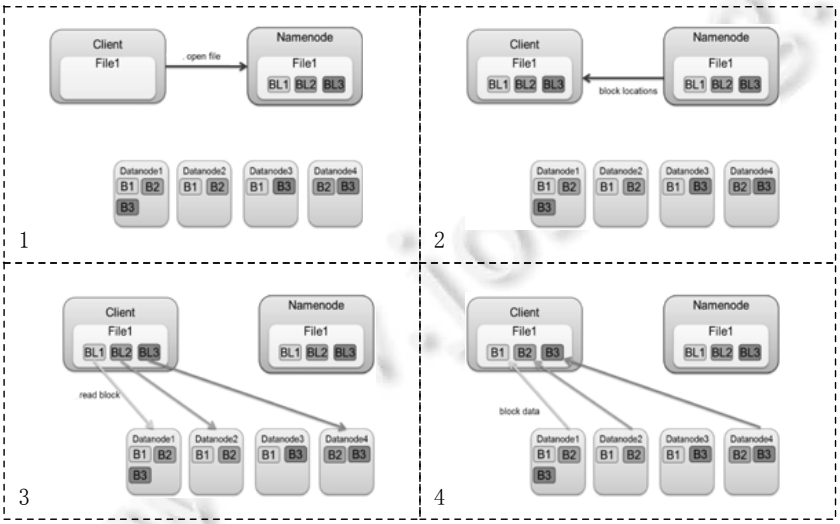

Fig.9 Reading process of HDFS

图 9 HDFS 读文件流程

值得注意的是:在 HDFS 中,在同一文件的读请求和写操作之间不存在互斥关系,即 HDFS 允许客户端读取正在写入的文件,此时,读流程在代码中会访问名为 INodeFileUnderConstruction 的类,其中的 target 字段记录了当前未写完的文件块所在节点.为了增强系统的可用性,读操作可以读取任意一个副本,因此只要有一个副本正常工作,就可以提供读服务.对于未写完的文件块,HDFS 的读操作仍然采用了读取任意一个副本的策略,并从上述 target 字段中任意选择一个数据节点进行读取.于是,当不同客户端读取未写完的文件块的不同副本时,这些客户端将读到不一致的结果.

若要改进 HDFS 来保证强一致性,则对于这种未写完的文件块可采用特殊的策略:要么对客户端屏蔽未写完的文件块,要么记录每个文件块在不同节点上的完成情况,并将未写完的文件块的读请求统一转发至已经写完的副本.

3.4 读流程建模

3.4.1 读流程模型概述

读流程模型如图 10 所示,主要包含两个子流程:客户端与名字节点交互的子流程 NamenodeRead 和客户端与数据节点交互的子流程 DatanodeRead.读流程的输入是上层应用发送的读文件请求(ReadFileRequest 库所)、第 1 关系(FileINode 库所)、第 2 关系(BlocksMap 库所)和数据节点存储的文件块(MergedBlock 库所),流程的输出是读文件回复(ReadFileResponse 库所).中间结果包括读文件块请求(ReadBlockRequest 库所)和读文件块回复(ReadBlockResponse 库所).

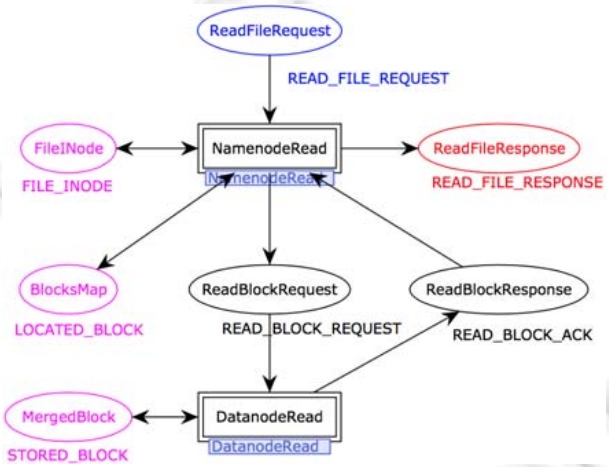

Fig.10 Model of reading process

图 10 读文件模型

主要流程为:接收到读文件请求,名字节点根据文件名检查 FileINode,查询该文件拥有的文件块,再根据文件块检查 BlocksMap,并将文件块和位置信息返回给客户端,客户端根据返回的结果生成读文件块的请求(NamenodeRead 子流程);读文件块请求发送到数据节点,数据节点将数据返回给客户端(DatanodeRead 子流程);客户端合并返回的读文件块请求,生成读文件回复,返回给上层应用(NamenodeRead 子流程).

3.4.2 NamonodeRead 流程

NamenodeRead 流程是客户端从名字节点获取文件块和位置信息,并以此构造读文件块请求的流程,该流程如图 11 所示.输入为:读文件请求(ReadFileRequest 库所)、第 1 关系(FileINode 库所)和第 2 关系(BlocksMap 库所)、读文件块回复(ReadBlockResponse 库所).输出包括读文件块请求(ReadBlockRequest 库所)和读文件回复(ReadFileResponse 库所).中间结果包括等待回复的读文件请求(WaitingRFR 库所)和需要删除的失败读请求(FailedReadToMove 库所).

NamenodeRead 流程的步骤如下:从流程外部接收到读文件请求,如果文件不存在,直接返回文件不存在类

型的读文件回复(FileNotExist 变迁);如果文件存在,则通过 OpenFileGetReadLocations 查找名字节点中存储的文件块和位置信息,生成读文件块请求和等待回复的读文件请求;ReadBlockSuccess 变迁根据读文件块回复更新等待回复的读文件请求,如果已经获得了所有的回复,则组成读文件回复返回给上层应用.如果收到了失败的读文件块回复,则触发 ReadBlockFail 变迁返回失败的读文件回复,同时产生等待回复的读文件请求在 FailedReadToMove 库所中,交给 RemoveReadAck 变迁删除后续到达的读文件块回复.

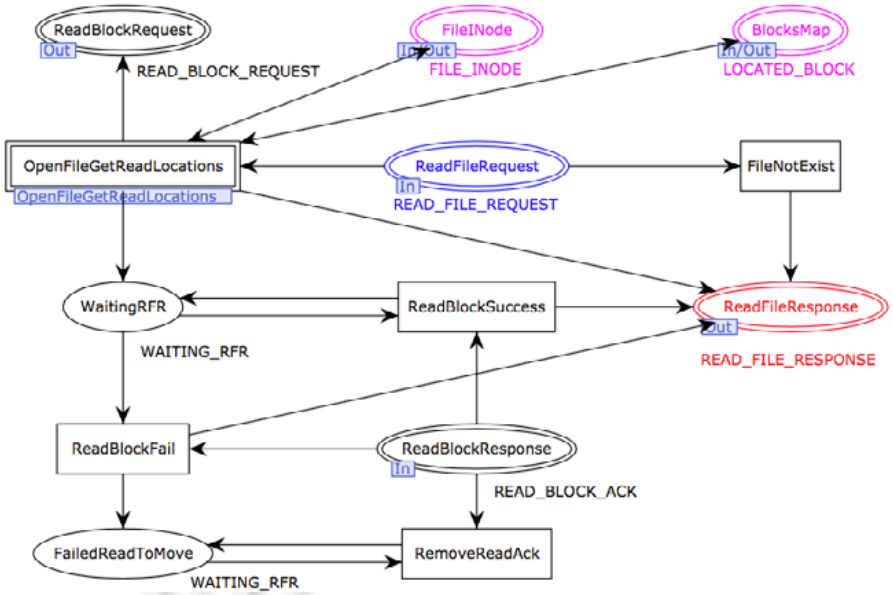

Fig.11 NamonodeRead model

图 11 NamenodeRead 模型

读操作流程中所有的变迁设置了特定的优先级,优先级均比写要高.这是为了模拟在写操作进行到任意步骤时发起两次读操作请求的情况下,在写流程状态不再继续时(例如,模拟实际场景中写入负载过高导致写入延迟过大)优先完成两次读操作.

3.4.3 OpenFileGetReadLocations 流程

OpenFileGetReadLocations 流程是根据文件名从名字节点获取文件块和存储位置的具体过程.该流程如图 12 所示,输入包括读文件请求(ReadFileRequest 库所)、第 1 关系映射(FileINode 库所)和第 2 关系映射(BlocksMap 库所).该流程的输出包括读文件块请求(ReadBlockRequest 库所)、等待回复的读文件请求(WaitingRFR 库所)和读文件回复(ReadFileResponse 库所).中间结果包括正在查找文件块及位置的读文件请求(RFR_BLOCKS_DATANODES 库所)、查找完全的文件块及位置的读文件请求(RFR_BLOCKS_DATANODES_FULL 库所).

主要流程如下:GetBlockList 变迁接收读文件请求,检查 FileINode 获取该文件的文件块列表,然后通过 GetLocFromBlocksMap 变迁从 BlocksMap 中逐个查找文件块列表中每个文件块的位置.如果在 BlocksMap 中没有查到,并且正在查找的是正在构建的文件的最后一块,则触发 GetLocFromTargets 变迁将 FileINode 正在写入的文件块所在数据节点列表中选取一个作为该文件块的存储位置.

通过上述查找过程后会产生 3 种结果:一是生成完整的包含所有文件块和位置的信息,则触发 HasAllLoc 变迁生成读文件块请求(对每一个文件块生成一个请求)和等待回复的读文件请求;第 2 种结果是没有查找到所有文件块的位置,则触发 LocNotFound 变迁生成失败的读文件回复;第 3 种是文件为空,触发 NoBlock 变迁生成失败的读文件回复.GenReadBlockRequest 变迁根据查找完全的文件块及位置的读文件请求生成等待回复的读文件请求和读文件块请求.

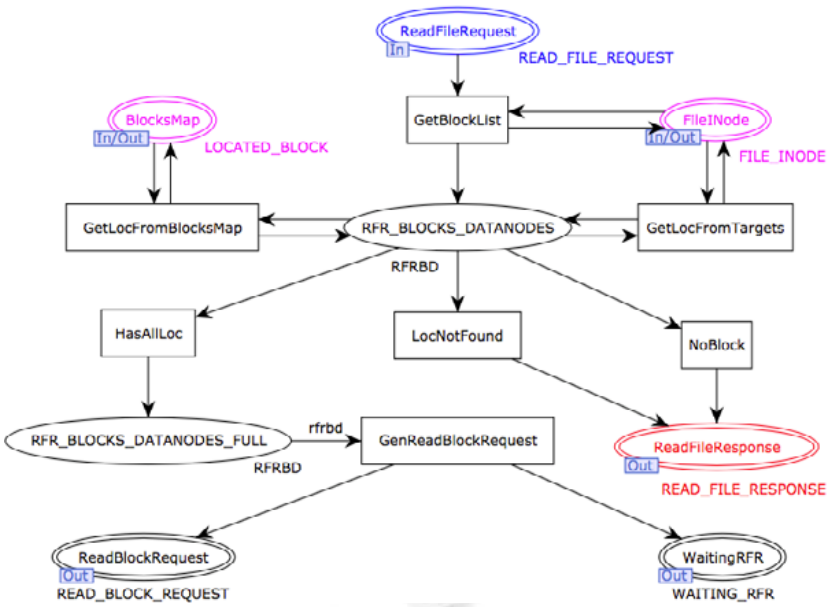

Fig.12 OpenFileGetReadLocations model

图 12 OpenFileGetReadLocations 模型

3.4.4 DatanodeRead 流程

DatanodeRead 流程是数据节点处理读文件块请求过程的建模,如图 13 所示.

模型的输入是读文件块请求(ReadBlockRequest 库所)、数据节点存储的文件块(MergedBlock 库所),输出是读文件块回复(ReadBlockResponse 库所).

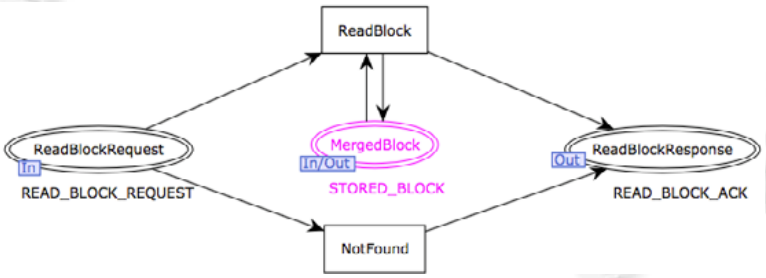

Fig.13 DatanodeRead model

图 13 DatanodeRead 模型

读文件块回复包括文件块内容.ReadBlock 变迁在 MergedBlock 中查找文件块,并构造读文件块回复.

NotFound 变迁用于处理未找到文件块的情况.

3.5 本节小结

本节介绍了 HDFS 读写流程建模过程中,各种库所和变迁的含义和功能.将读写流程综合,就组成了 HDFS 的读写模型,如图 14 所示.图的左边是写文件流程,右边是读文件流程.

该模型是后续进行一致性分析的重要基础.

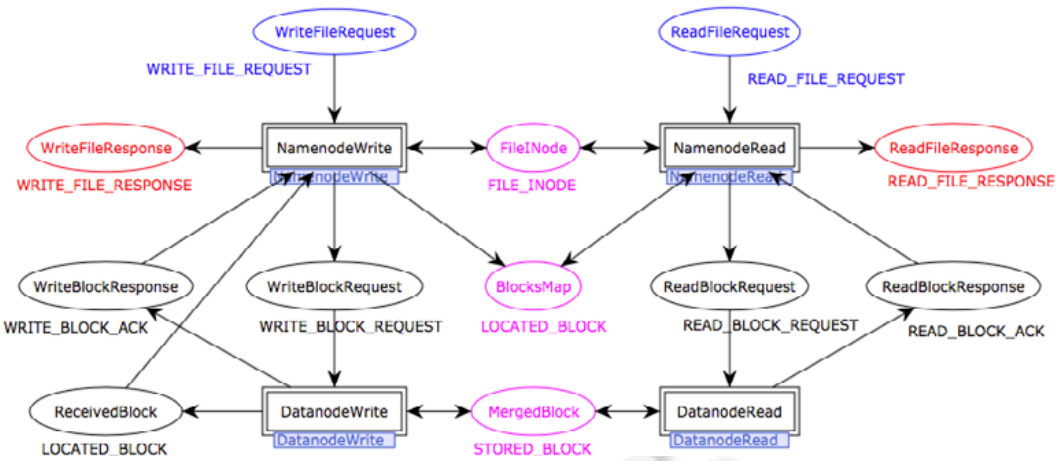

Fig.14 Reading/writing process model of HDFS

图 14 HDFS 读写模型

4 系统一致性分析

本文研究的一致性包括两个层面:数据层一致性和操作层一致性.如果任意时刻存储在多个节点上的数据副本完全相同,则称系统满足数据层一致性.如果任意时刻多次读同一数据返回的结果是完全相同的,则称系统满足操作层一致性.每个层面的一致性分析都包括两种情况:文件块被正确写入磁盘和存在(由网络传输超时、节点失效、磁盘故障等引起的)本地写文件块失败.本文分别称这两种情况为无错情况和有错情况.

4.1 分析工具介绍

我们首先介绍 Petri 网的状态.Petri 网的状态由令牌在库所的分布决定,一个确定的分布即为一个状态.

Petri 网的状态空间表达了系统运行时所有可能的状态的集合,本文用来分析系统的一致性.我们使用 CPN Tools 中的状态空间计算工具计算上述 HDFS 模型的状态空间.CPN Tools 提供了一套基于状态空间分析的工具,包括网络状态空间计算、状态空间报告、状态图生成、状态空间查询等.使用这些工具能够很容易地生成状态空间并作进一步分析,对研究系统一致性提供了有力的支持.

下面通过一个简单的着色 Petri 网模型为例,来介绍状态空间工具使用的方法.模型如图 15 所示,该图有两条路径:一是触发 *T*1,使令牌进入 *P*3 库所;二是依次触发 *T*2、*T*3,使令牌进入 *P*3 库所.

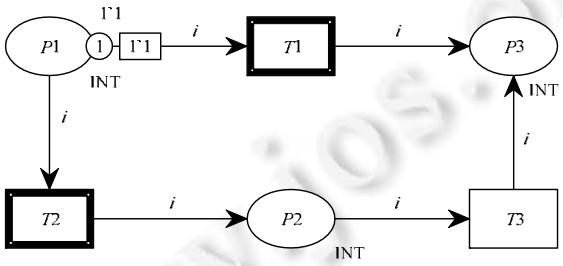

Fig.15 An example of state space

图 15 状态空间样例

使用 CPN Tools 提供的状态空间计算工具,能够计算网络的全部状态空间.图 15 的状态图如图 16 所示,该状态图包括 3 个状态节点.状态节点显示的内容包括:状态节点编号、该状态节点下所有库所中容纳的令牌信息.弧显示的内容包括:弧编号、该弧对应的状态变迁所触发的变迁以及触发条件.在状态图中对库所和变迁的

引用采用“页名库所名或变迁名”的格式(在 CPN Tools 中,本文各模型图中的双线矩形内的文字即为页名).状态图中没有输入弧的节点称为初始状态,图中左上角的状态 1 即为初始状态.没有输出弧的节点称为终结状态,图中左下角的状态 2 即为终结状态.可以查看每个状态中各个库所的令牌,并可用来进行系统分析.如图中状态 2 仅库所 *P*3 中有 1 个令牌,其他库所中无令牌.

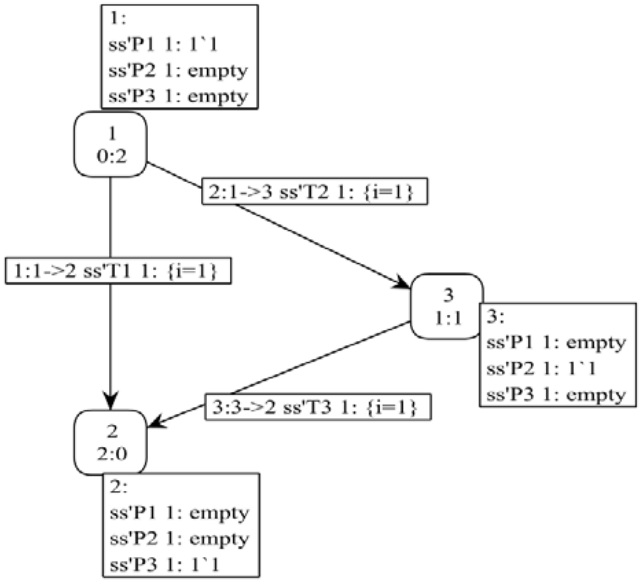

Fig.16 An example of state space graph

图 16 状态空间图样例

因为示例网络较为简单,因此可以画出完整的状态空间图.但是,对于复杂模型,可能会有成千上万个状态空间节点,全部画出就变得不可行.但是,CPN Tools 提供了状态空间的查询功能,可以通过 ML 语言编写检查规则,并在计算状态空间时自动检查满足规则要求的状态节点.

4.2 实验设置

初始状态为 1 个客户端向 3 个数据节点写入副本数为 2 的 1 个文件,文件包含 1 个大小为 64MB 的文件块.无错情况下,模型中的出错标记设置为假;有错情况下,模型中的出错标记变量设为真.分别进行了数据层无错、有错和操作层无错、有错情况下的一致性实验.

4.3 数据层一致性分析

给定系统约定的副本数,系统满足数据层一致性,当且仅当在任意状态下,一个文件块要么不出现,要么出现的次数为副本数.因此,在这一实验中,不一致状态节点的检查规则为:如果出现了一个文件块,但是出现的次数不等于副本数,则标记为不一致状态节点.

4.3.1 无错情况下的数据层一致性

无错情况下的数据层一致性实验中的状态空间报告见表 1.

Table 1 State space report of data layer consistency without error

表 1 无错情况下数据层一致性状态空间报告

统计项	个数
状态节点	49
终结状态	9
不一致状态	2

终结状态都是一致的.这说明在没有错误发生的条件下,系统虽然中间状态可能会发生不一致,但最终状态都是一致的.

本文以一个不一致的状态节点为例,分析出现不一致的条件.在该状态节点的库所中发现文件块存储的位置为 datanode3 和 datanode2,且只有 datanode3 上实际存储了该文件块,而 datanode2 上并没有.但在其他库所中可以看到表示发送给 datanode2 的写文件块请求的令牌.因此,这种不一致状态所表达的是:发往 datanode3 的文件块已在本地写成功,而由 datanode3 发往 datanode2 的文件块还在发送的过程中,两个文件块因为传播的过程有快有慢,所以出现数据不一致现象.当继续分析该状态节点的后继状态时,可以发现最终都会达到一致状态.

综合分析无错条件下的数据层一致性有以下结论.

(1) 系统会出现数据层的不一致,导致不一致的原因是多个副本之间传播的快慢程度不同;

(2) 经过足够时间的传播,系统最终会达到数据层的一致状态.

4.3.2 有错情况下的数据层一致性

有错情况下的数据层一致性实验中的状态空间报告见表 2.

Table 2 State space report of data layer consistency with error

表 2 有错情况下数据层一致性状态空间报告

统计项	个数
状态节点	211
终结状态	9
不一致状态	4

以其中一个不一致状态为例进行分析.在该状态节点的库所中发现 datanode2 实际存储了该文件块,但是写文件回复库所中的令牌状态是失败的.因此,这种不一致状态所表达的是:datanode2 的文件块写成功,而另一块文件写失败,导致整个操作的失败,所以出现不一致现象.由于不一致状态是终结状态,这种不一致并不会随着时间的推移而自动消失.

综合分析有错条件下的数据层一致性有以下结论.

(1) 系统会出现数据层的不一致,导致不一致的原因包括:多个副本之间传播的快慢程度不同;副本传播过程中出错,造成一部分副本没有写成功;

(2) 有错误发生时,即使经过足够时间的传播,系统最终也不会达到数据层的一致状态.

4.4 操作层一致性分析

操作层一致性必须通过读操作来实现,既要有写文件输入,又要有读文件输入.因此在该场景下,本文在模型中设置了两个客户端:一个客户端负责写入文件,一个客户端负责读文件.为了检查文件在写入过程中的不一致性对读操作是否可见,本文提出了“时点重复读”,即:在文件写入过程中的任意时点,暂停写操作流程,先后执行两次相同的读文件请求,比较两次读操作的结果是否相同.如果不同,证明在写文件过程中出现的不一致是对读操作可见的.在建模时设置所有读操作的优先级高于写操作,能够在不改变网络结构的前提下,实现这种操作的模拟.当需要进行时点重复读测试时,在写操作进行到任意步骤时发起两次读操作请求,即可在写流程状态不变的情况下完成两次读取.

两次读操作的局部模型如图 17 所示.在原有的模型之外,添加了两个库所 FirstRead 和 SecondRead,以及两个变迁 GenerateFirstRead 和 GenerateSecondRead,每个库所中只有一个令牌,用于表示读操作请求.

GenerateSecondRead 变迁的一条输入弧来自于库所 ReadFileResponse,这可以保证只有当第 1 个读请求完成后才开始发送第 2 次读请求.串行地执行两次读操作和并发地执行读操作产生的结果相同,因为读操作不会改变系统内部的数据结构状态.并且串行执行的状态空间规模远小于并发执行,可以简化分析.

操作层一致性的定义是任意时刻多次读同一数据的结果相同,因此在这一实验中,设置不一致状态节点的检查规则为:比较两次读操作返回的数据是否相同,将两次读请求结果不相同的状态节点标记为不一致状态节点.

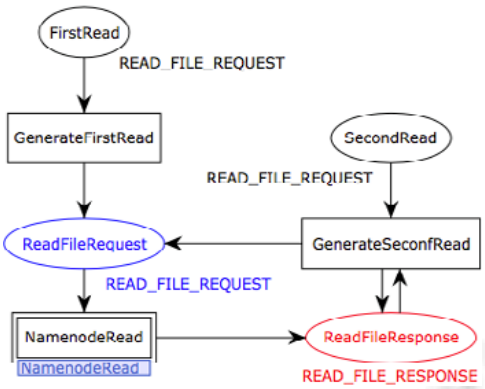

Fig.17 System read operation configuration of operation layer consisteny

图 17 操作层一致性分析系统读输入设定图

4.4.1 无错情况下的操作层一致性

无错情况下的操作层一致性实验中的状态空间报告见表 3.

Table 3 State space report of operation layer consistency without error

表 3 无错情况下操作层一致性状态空间报告

统计项	个数
状态节点	607
终结状态	40
不一致状态	8

所有不一致状态都是终结状态,因为只要读到两个结果就终结计算,并且只有存在两个结果时才有可能被判定为不一致状态.以一种不一致状态来分析原因:在第 1 关系中发现最后一个文件块的存储位置记录在 NameNode 中,记录显示该文件块应存储在 datanode1 和 datanode2 节点上.datanode1 的文件块已经写成功,而发送给 datanode2 的写文件块请求还未被处理.读文件回复库所则显示两次读操作分别访问了 datanode1 和 datanode2.因此,这种不一致状态表示的是:datanode1 节点上的文件块已经写成功,datanode2 节点上的文件块还处于发送状态,尚未被 datanode2 处理;而两次读文件操作分别访问了这两个数据节点,使得访问 datanode1 的读请求成功获得数据,而访问 datanode2 的读请求失败,从而产生了操作层不一致.

综合分析无错条件下的操作层一致性有以下结论.

(1) 系统会出现操作层的不一致,导致不一致的原因是多个副本之间传播的快慢程度不同;

(2) 经过足够时间的传播,系统最终会达到操作层的一致状态.

4.4.2 有错情况下的操作层一致性

有错情况下的操作层一致性实验中的状态空间报告见表 4.

Table 4 State space report of operation layer consistency with error

表 4 有错情况下操作层一致性状态空间报告

统计项	个数
状态节点	1 067
终结状态	81
不一致状态	16

以一个不一致状态为例分析原因:在该状态节点的库所中发现 datanode2 实际存储了该文件块,但 datanode3 写文件失败,读文件回复库所中显示两次读操作分别访问了 datanode2 和 datanode3,产生了一次成功一次失败的结果,因此出现了操作层的数据不一致.

综合分析有错条件下的操作层一致性,有以下结论.

(1) 系统会出现操作层的不一致,导致不一致的原因包括:多个副本之间传播的快慢程度不同;副本传播过程中出错,造成一部分副本没有写成功;

(2) 有错误发生时,即使经过足够时间的传播,系统最终也不会达到操作层的一致状态.

4.5 代价分析

本文通过阅读源码的方式对Hadoop进行人工建模,大约需要2~3周的时间.除人工建模方式以外,还可以通过系统日志分析实现自动建模,如对 Cassandra 的日志进行建模的工作[27].然而,对日志进行自动建模虽然效率高,但由于日志较多,建模结果可能包含一些不关心的状态,没有人工建模精准.人工建模和日志自动建模两种方法是时间与精准性的权衡,需要建模人员根据实际情况选择合适的建模策略.建模之后,即可使用 CPN Tools 计算状态空间.对于本文提出的模型进行计算耗时约 5 分钟,相比整体工作量来说,这一耗时是可以接受的.

4.6 本节小结

本节介绍了基于 CPN Tools 的状态空间分析工具集,并以该工具集为手段,以 HDFS 读写流程模型为样本,从数据层一致性和操作层一致性两个角度,详细分析了不一致的情况及其产生原因.本节实验包含了 HDFS 所有的不一致条件.

综合两种不一致性的分析可以发现:数据层不一致是操作层不一致产生的根本原因,只有数据层出现了不一致,才有可能被读操作观测到.操作层出现不一致的另一个原因是并发隔离机制不严格,因为 HDFS 允许在读文件的最后一块时从 FileINode 直接获取文件块存储位置,此时并不能保证每一个数据节点上都已经完成了写文件块操作,因此访问不同数据节点就可能导致不同的读操作结果(见表 5).

Table 5 HDFS data consistency analysis summary table

表 5 HDFS 数据一致性分析结果汇总表

场景	数据层一致性	操作层一致性
无错情况	1. 存在不一致,原因是多副本间数据传播速度不同; 2. 经过足够的时间,系统恢复一致	同左,且只发生在文件最后一块
有错情况	1. 存在不一致,原因是副本传播速度和副本写失败; 2. 如果出现写失败,则不会恢复到一致状态	同左,且只发生在文件最后一块

操作层不一致只发生在文件的最后一块,是因为最后一块的位置信息由名字节点分配,其他文件块位置信息由写成功的数据节点上报,因此能够保证数据的一致性.

HDFS 的文档对一致性的描述为:当一个客户端创建的文件关闭后,其他客户端可以立即看到完整的这一文件.本文的结论关注的则是在一个客户端创建文件的过程中,其他客户端读取该文件时的一致性,属于文档中未涉及的部分.

我们注意到:HDFS 提供了 FileStatus 的 *getLen*(·)接口来获取文件大小,该接口仅会返回已写完的文件块的总长度.然而,其提供的 *FSDataInputStream* 接口会将未写完的文件块也读取出来,因此,实际读取的大小可能大于 *getLen*(·)的返回值.由此可以对基于 HDFS 的上层应用开发提出建议:由于不一致出现在不同客户端读取同一个正在写入的文件这一情况下,因此,在应用层进行读操作时,通过对文件块的长度进行检测,来达到一致性和可用性的平衡.在读取文件之前调用 *getLen*(·)得到文件长度,若实际读取到的文件长度大于 *getLen*(·)的返回值,则说明读取到了未写完的文件块,此时可以根据应用需求选择是否将大于 *getLen*(·)返回值的部分数据舍弃掉:如果舍弃掉,则可以保证所有客户端读到的都一样;如果保留,则牺牲一致性而保证了高性能.由于该策略是在应用层进行的,不同的应用可以选择不同的控制策略,因此不会破坏 HDFS 的高可用性.由于该策略仅增加了一次 *getLen*(·)调用和一次长度判断,因此也不会影响应用层的读取性能.

5 总 结

本文基于着色 Petri 网开展 HDFS 数据一致性的相关研究,完成的主要工作有:

- 使用着色 Petri 网工具 CPN Tools 对 HDFS 的读写流程建立模型,该模型是一个无时序的着色 Petri 网模型,详细刻画了 HDFS 内部各个组件——名字节点、数据节点和客户端的内部数据结构、核心功能及相互协作的机制;
- 基于着色 Petri 网模型,使用状态空间工具分析了 HDFS 的数据层一致性和操作层一致性.如果任意时刻存储在多个节点上的数据副本完全相同,则称系统满足数据层一致性.如果任意时刻多次读同一数据返回的结果是完全相同的,则称系统满足操作层一致性.针对每一种一致性要求,都在无错条件和有错条件下展开状态空间分析,对不一致状态的产生原因及演化方向进行了详细的分析,明确了系统在各种条件下提供的一致性保障,本文研究包含了所有可能产生不一致的条件;
- 本文以副本一致性为例,对 HDFS 系统进行了分析.如果需要对 HDFS 系统的其他细节进行分析,则可以进行类似的建模.例如:如果需要进行 HDFS 的可用性、吞吐性能等分析,可以将本模型作为建模基础,修改或增加相关的模型元素和状态空间的检验方法,实现对其他特性的分析.此外,尽管本文是针对 HDFS 进行建模,但是本文的工作思路同样适用于其他分布式系统,具有较强的普适性.

后续研究工作包含两个方向:(1) 本文提出了一种在应用层避免读取数据不一致的解决方法,可以本文建模为基础,进一步分析 HDFS 中是否可以通过优化系统读写流程来提供一致的数据读取,为上层应用提供便利;(2) 本文的建模方法和技巧可以作为参考,应用到其他系统的建模分析工作中,尤其适用于对其他分布式系统的数据一致性进行建模,如分析 NoSQL 数据库的最终一致性等.

References:

[1] Hadoop. 2014. http://hadoop.apache.org/

[2] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM, 2008,51(1): 107−113.

[3] Ghemawat S, Gobioff H, Leung ST. The Google file system. ACM SIGOPS Operating Systems Review, 2003,37(5).

[4] Brewer EA. Towards robust distributed systems. In: Proc. of the PODC. 2000.

[5] Motamari P. Snapshotting in Hadoop distributed file system for hadoop open platform as service [MS. Thesis]. KTH, 2014.

[6] Hakimzadeh K, Sajjad HP, Dowling J. Scaling HDFS with a strongly consistent relational model for metadata. In: Proc. of the IFIP Int'l Conf. on Distributed Applications and Interoperable Systems. Berlin, Heidelberg: Springer-Verlag, 2014. 38−51.

[7] Jensen K. Coloured Petri nets and the invariant-method. Theoretical Computer Science, 1981,14(3):317−336.

[8] Ratzer VA, *et al*. CPN tools for editing, simulating, and analysing coloured Petri nets. In: Proc. of the Applications and Theory of Petri Nets. 2003. 450−462.

[9] Chen M, Mao SW, Liu YH. Big data: A survey. In: Proc. of the Mobile Networks and Applications. 2014. 1−39.

[10] Hadoop Wiki: Applications and organizations using hadoop. 2014. http://wiki.apache.org/hadoop/PoweredBy

[11] Valerio J, *et al*. Evaluating the price of consistency in distributed file storage services. In: Proc. of the Distributed Applications and Interoperable Systems. 2013. 141−154.

[12] Borthakur D, *et al*. Apache Hadoop goes realtime at Facebook. In: Proc. of the 2011 ACM SIGMOD Int'l Conf. on Management of Data. ACM, 2011.

[13] Huang XD, Wang JM, Qiao JL, Zheng LF, Zhang JR, Wong RK. Performance and replica consistency simulation for quorum-based NoSQL system Cassandra. In: Proc. of the Petri Nets. 2017. 78−98.

[14] Osman R, Piazzolla P. Modelling replication in NoSQL datastores. In: Proc. of the QEST. 2014. 194−209.

[15] Dipietro S, Casale G, Serazzi G. A queueing network model for performance prediction of apache Cassandra. In: Proc. of the VALUETOOLS. 2016.

[16] Wang F, Lei BH. Research on the service-aware model of NGN architecture and key technologies. Telecommunications Science, 2010(12):95−99 (in Chinese with English abstract).

[17] Wu YW, *et al*. Modeling of distributed file systems for practical performance analysis. IEEE Trans. on Parallel and Distributed Systems, 2014,25(1):156−166.

[18] Aguilera-Mendoza L, Llorente-Quesada MT. Modeling and simulation of Hadoop distributed file system in a cluster of workstations. In: Proc. of the Model and Data Engineering. 2013. 1–12.

[19] Yang YF. HDFS data consistency modelling and analysis based on colored Petri net [Ph.D. Thesis]. Beijing: Tsinghua University, 2014 (in Chinese with English abstract).

[20] Petri CA, Wrote; Green CF Trans. Communication with automata [Ph.D. Thesis]. Information System Theory Project, Applied Data Research Inc., Princeton N.J., 1966.

[21] Murata T. Petri nets: Properties, analysis and applications. Proc. of the IEEE, 1989,77(4):541–580.

[22] Molloy MK. Performance analysis using stochastic Petri nets. IEEE Trans. on Computers, 1982,100(9):913–917.

[23] Jensen K, Kristensen LM, Wells L. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. Int'l Journal on Software Tools for Technology Transfer, 2007,9(3-4):213–254.

[24] Westergaard M, Kristensen LM. The access/CPN framework: A tool for interacting with the CPN tools simulator. In: Proc. of the Applications and Theory of Petri Nets. 2009. 313–322.

[25] Milner R. A theory of type polymorphism in programming. Journal of Computer and System Sciences, 1978,17(3):348–375.

[26] Cai B, Chen XP. Insider of Hadoop Technology: In-depth Analysis of Hadoop Common and HDFS Architecture Design and Implementation Principles. Beijing: China Machine Press, 2013 (in Chinese).

[27] Huang XD. The formal modeling and optimization for data partitioning and replica consistency in distributed storage systems [Ph.D. Thesis]. Beijing: Tsinghua University, 2017 (in Chinese with English abstract).

附中文参考文献:

[16] 王峰,雷葆华.Hadoop 分布式文件系统的模型分析.电信科学,2010(12):95–99.

[19] 杨义繁.基于着色 Petri 网的 HDFS 数据一致性建模与分析[博士学位论文].北京:清华大学,2014.

[26] 蔡斌,陈湘萍.Hadoop 技术内幕:深入解析 Hadoop Common 和 HDFS 架构设计与实现原理.北京:机械工业出版社,2013.

[27] 黄向东.分布式存储系统数据分区与副本一致性形式化建模与优化[博士学位论文].北京:清华大学,2017.

乔嘉林(1993－),男,学士,主要研究领域为时序数据库,时序聚合索引,数据库副本技术.

黄向东(1989－),男,博士,助理研究员,CCF 专业会员,主要研究领域为数据库技术,大数据管理.

杨义繁(1989－),男,硕士,主要研究领域为数据库.

王建民(1968－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为大数据与知识工程:非结构化数据管理,业务过程与产品生命周期管理,数字版权与系统安全技术,数据库测试技术.

吴凯(1969－),男,高级工程师,主要研究领域为微特电机及控制电器专业,风力发电.