◎研究、探讨◎

分布式状态空间生成的设计与实现

郑 霄[1,2],李宏亮[1],吴 东[1],原 昊[1]
ZHENG Xiao[1,2],LI Hong-liang[1],WU Dong[1],YUAN Hao[1]

1.江南计算技术研究所,江苏 无锡 214083
2.解放军信息工程大学,郑州 450002
1.Jiangnan Institute of Computing Technology,Wuxi,Jiangsu 214083,China
2.PLA Information and Engineering University,Zhengzhou 450002,China
E-mail:uu88zheng@126.com

ZHENG Xiao,LI Hong-liang,WU Dong,et al.Design and implementation of distributed state space generation.Computer Engineering and Applications,2009,45(32):27-30.

Abstract: Parallelization of state space generation is an important technical method to deal with the state space explosion problem.A practical approach based on MapReduce framework is presented.It has the virtues of simpleness and easiness to use,which let the user need not caring about how to parallelize the state space generation algorithm,and that is where it differs from the existing distributed state space generation algorithms.Meanwhile,the manner that the MapReduce framework is used in this approach is also different from the common ones:It has the abilities of dynamically generating input files and repeatedly executing the state space generation process in need,while the common use of MapReduce is to one-off process vast amounts of data in-parallel.This approach has been implemented in a small-scale distributed environment and the experimental results show that:(1)distributed state space generation based on MapReduce does have the ability to analyze a model with large state space,and that (2)this method is quite applicable and scalable for a model with an increasing token number and a fixed place number, whose state space scales up mainly with the token number.

Key words: state space model;distributed state space generation;MapReduce;Hadoop

摘 要:状态空间生成的并行化是针对状态空间爆炸问题而提出的一种重要手段。提出了一种基于 MapReduce 的分布式状态空间生成方案,与现有的同类研究相比,它无需用户关心生成算法的并行化,具有简单易用性;与常规的 MapReduce 的用法相比,它增加了输入文件的自动生成和作业运行的自动循环控制。该方案已在小规模分布式环境下实现,实验结果表明:(1)基于 MapReduce 的分布式状态空间生成算法可以扩大模型的可求解规模;(2)对于状态空间规模的增长主要由托肯(token)数增加引起的一类模型,该算法具有良好的适应性和可扩展性。

关键词:状态空间模型;分布式状态空间生成;MapReduce;Hadoop
DOI:10.3778/j.issn.1002-8331.2009.32.009 **文章编号:**1002-8331(2009)32-0027-04 **文献标识码:**A **中图分类号:**TP302

1 引言

常用于计算机系统的设计与分析的状态空间模型一般有两种求解方法,模拟[1]和数值解析[2-3]。模拟求解是以概率论和统计学为基础,通过产生一定数量的模拟数据来获得具有一定置信度的估计值,因此其结果的精度会受到评估次数和单次评估的时间长度的影响;而且当进行多项指标评测或者评测指标发生改变时,需要重新进行模拟实验以获取结果。数值解析求解则先将系统模型转换成对应的连续时间马尔科夫模型(CTMC)[3],从而将问题的求解转换成向量矩阵乘解析,这种方法不但具有更高的求解精度,也可在无需增加额外工作量的前提下获得用户同时关注的多个度量指标。因此,数值解析方法对于需要多方面掌握系统特征和精确答案的用户来说更具吸引力。

然而,数值解析方法需要静态生成模型的状态空间,该过程的计算量和存储空间需求量会随着模型的规模快速增长,进而出现状态空间爆炸问题[4]。为了求解更大规模的模型,主要有两种思路:一种是尽量减少需要存储的状态空间,例如尽早舍

基金项目:国家重点基础研究发展规划(973)(the National Grand Fundamental Research 973 Program of China under Grant No.2007CB310900);国家高技术研究发展计划(863)(the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z117)。
作者简介:郑霄(1976-),男,博士研究生,主要研究方向为并行计算机系统高可用;李宏亮(1975-),男,博士,副研究员,主要研究方向为体系结构、高性能计算、容错计算等;吴东(1971-),男,博士,讲师,主要研究方向为可重构高性能计算;原昊(1984-),男,硕士研究生,主要研究方向为可重构高性能计算。
收稿日期:2008-12-05 **修回日期:**2009-02-16

弃无效状态分支、压缩状态信息量等,这方面的技术包括状态截取、偏序归约(partial-order reduction)、数据抽象(data abstraction)和对称归约(symmetry reduction)等[5-7];但存储空间压缩毕竟存在一个极限,因此这种思路难以从根本上突破可求解模型规模的限制。另一种思路则是并行化,即在存储能力更大、计算能力更强的分布式环境下并发完成状态空间的生成;由于分布式环境具有良好的可扩展性,这种思路在寻求求解更大规模模型的方法上更具挖掘潜力。目前,已有不少这方面的研究,例如:文献[7]主要讨论了分布式状态空间的生成算法,同时也强调了恰当选择分区函数的重要性;文献[8]探索了基于平衡二叉树的状态空间并行生成的自动化实现,并采用分类器概念替代了文献[7]中需要用户精心设计的分区函数,从而增强了从状态到节点的映射能力,同时也通过重映射功能实现了状态树的动态平衡;文献[9]主要从减少通信量的角度出发,建立了原状态描述数据与其索引的映射表并存入数据库,这样在判别状态是否存在和将状态传给其他处理器时只需使用状态的索引描述方式,从而将通信量几乎降低一个数量级。

该文探索了分布式状态空间生成的另一种实现,其优点在于:借助 Hadoop[10-11]的开源分布式并行编程框架,用户无需任何并行开发经验,只需利用其简单易用的编程接口(通过 Map 函数实现并发搜索多个已知状态的后继状态操作; 通过 Reduce 函数实现状态集归并和判断状态是否为新操作并输出搜索结果),即可实现状态空间生成的并行化。但该算法不同于直接采用 MapReduce 对大批量数据进行一次性处理并产生结果的常规用法,因为状态空间搜索是个递归过程,它需要对新状态重复执行后继状态搜索操作和判断状态是否为新的操作,为此除了需要设计 Map/Reduce 两个主要函数以外, 用户还需实现新输入文件的动态生成与新作业的自动提交等循环控制。

该方案已在小规模分布式环境下实现, 实验结果表明:(1)基于 MapReduce 的分布式状态空间生成算法可以扩大模型的可求解规模;(2)对于状态空间规模的增长主要由托肯(token)数增加引起的一类模型,该算法具有良好的适应性和可扩展性。

2 状态空间生成算法

状态空间生成算法的基本原理是:针对指定的模型,在给出初始状态的前提下, 按各种情况触发模型中的使能事件,从而生成所有可能出现的新状态;这些状态及其相互之间的生成关系即构成了模型的状态可达图。状态空间生成的算法流程如图1 所示。

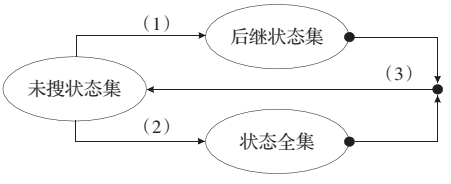


图 1 状态空间生成流程示意图

其中,未搜状态集用于存放模型的新状态,最初只含初始状态;状态全集用来存放模型最终所产生的全部状态;而后继状态集则用于临时存放由未搜状态推导出的后继状态。状态空间生成的过程就是在未搜状态集不为空的条件下循环执行下述三步操作的过程:

(1)从未搜状态集中取出"新"状态,搜索出它的所有后继状态;

(2)将搜索过的状态从未搜状态集中取出并放入状态全集;

(3)将后继状态集中的状态与状态全集进行比较,判断它是否已包含在状态全集中,若没有,说明是新状态,需要继续搜索它的后继,故将它加入未搜状态集。

在分布式算法中, 未搜状态集被分散到多台处理机上,然后并发执行后继状态的推导工作;状态全集按照事先设计的分区函数建立与处理机间的映射关系;根据该映射关系,后继状态先被传给相应的处理器,然后再判断它是否为新状态;所有的处理机完成各自的操作后,系统收集所有的状态,形成模型的完整状态空间。

3 MapReduce 算法与 Hadoop 平台简介

MapReduce 是 Google 公司针对大规模群组中海量数据处理而开发的一种分布式编程模型[10-11]。简单地讲,MapReduce 就是"任务的分解与结果的汇总",其中,"Map"的意思是将一个作业分解成为多个任务,而"Reduce"就是将这些任务处理的结果汇总起来,从而获得最后的分析结果。这两项功能主要通过两个函数来实现:Map 函数对输入集合中的所有成员做相同的处理,然后返回一个中间结果;Reduce 函数把所有 Map 输出的中间结果集中起来并进行分类和归纳。MapReduce 的设计主要是为了解决海量数据信息的处理, 但也可用于其他一般性的计算。

Hadoop是 Apache 开源组织开发的一个分布式计算开源框架, 采用 Java 编写, 目前在很多大型网站上都已经得到了应用,如 Amazon、Facebook 和 Yahoo 等等。Hadoop 的两大核心设计——MapReduce 和 HDFS(Hadoop Distributed File System),使其具备高效、可靠、经济易用与可扩展性好等特点,从而为没有并行编程经验普通用户进行分布式计算提供了方便的平台。

利用 Hadoop 平台提供的 MapReduce 框架实现分布式计算需要遵守 Hadoop 的相关 API 规范, 包括按照指定格式提供作业参数配置、自定义 Map 和 Reduce 函数、提供输入文件等。MapReduce 的输入、输出和中间结果均是以<*key*,*value*>键值对的形式存在,它们的 *key* 的类型可以有所不同;一条输入记录<k_1,v_1>经过 Map 函数的处理可以产生零到多条中间结果<k_i,v_i>,多个 Map 任务可以并发执行; 系统会自动收集各个 Map 任务的中间输出结果{<k_i,v_i>},并通过自带的 *Combiner*()函数对它进行分类, 形成若干具有相同 k_i 值的键值对集合, 然后通过 *partition*()函数将它们分配给若干 Reduce 任务,每个 Reduce 任务负责对所分得的键值对集合按 k_i 值归并,并输出最后的结果{<k_i,v_j>}。

4 算法设计与实现

4.1 变量与数据结构

该算法涉及到几个重要的变量与数据结构说明如下:

(1)状态空间模型:用于描述系统的结构特征与复杂行为,例如随机行为网模型[12],它由库所(place,用于条件描述)和行为(activity,用于事件描述)两类节点组成,而模型中的边描述了行为与库所间的关系和行为后果。每个 Map 操作在推导后继状态时都需要用到该模型,因此将它设置为全局共享。

(2)状态:指某一时刻所有库所的托肯(token)分布情况;若某个库所有使能边,当它含有托肯时,则可引发托肯迁移,从

而发生状态变迁。一个库所可能支持多条使能边,每条使能边的触发可以产生一个后继状态。

(3)状态全集:用于收集已出现的所有状态及每个状态出现的频率,为全局共享变量。

(4)未搜状态集:用于存放未被搜索的状态。每个 Map 操作都有一个局部未搜状态集,其元素在每个单循环操作的开始,从各自相应的逻辑输入块(logic inputSplit)中获得。

(5)循环控制变量:一个全局共享的辅助变量,用于控制当前 Map/Reduce 作业的循环次数。

4.2 Map 函数

算法中的每个 Map 任务主要负责搜索各自所分配的未搜状态的所有后继状态,并产生输出结果,其执行流程如下:

(1)对应的逻辑输入块中的所有状态记录读入本地的未搜状态集;

(2)对于本地未搜状态集中的每条状态记录,找出该状态中所有含有托肯的库所;

(3)对于每个有托肯的库所,按照模型的描述找出它所有的使能边,依次激活每条边,使源库所的托肯按规则转移到目标库所,从而完成一次状态变迁,产生一个后继状态;

(4)将后继状态输出到中间状态。

4.3 Reduce 函数

Hadoop 将各个 Map 的中间输出键值对<*key*,*value*>按 *key* 值隐式归类,Reduce 函数的任务是对每个类型的 *value* 值进行统计,然后将结果输出到状态全集。在状态搜索过程中,状态每出现一次,就会相应产生一个 *value* 值为 1 的键值对输出,因此该状态的 *value* 值总和即为它在状态遍历过程中出现的次数,而这个频率值在全部状态出现次数的比例即可反映出该状态的出现概率。

4.4 作业配置与循环控制

算法的实现需要按 Hadoop 要求提供作业的相关配置信息,包括作业名称、输出键值对中"键"与"值"的类型、输入文件的格式与路径、输出文件的格式与路径、最大 Map 任务数、最大 Reduce 任务数等。另外,为实现程序自动进入下一轮后继状态搜索的循环操作,除了在 Reduce 函数中将新状态写入下个循环的指定输入文件以外,主函数还需要在循环前设置循环标志,并在每次循环中完成作业的处理后,先判断指定的下个循环的输入文件是否为空,如果为空,说明没有新发现的后继状态,就意味着还需要进行下一轮的后继状态搜索操作,因此就将循环标志置为假。

图 2 阐述了分布式状态空间生成算法在 Hadoop 平台上的工作原理:

(1)首先,Hadoop 按照用户设定的 Map 任务数,将指定路径下的输入文件分割成相应数目的逻辑输入块,然后交给不同的 Map 任务去执行;

(2)对于自己所分配的逻辑输入块里的每条状态记录,每个Map 任务按照用户定义的 Map 函数找出它的所有后继状态,并将它们按指定的键值对格式输出到 collector 容器中,同时将搜索过的状态记录输出到状态全集;

(3)Reduce 任务统计出每个 *key* 的 *value* 值之和后输出到状态全集,同时通过比较判断出哪些后继状态是新状态,并将新状态写入一个新的输入文件;

(4)在输入文件内容不为空的情况下,循环执行(1)~(3)操作。

5 结果与分析

5.1 实验环境

该算法在 3 台普通微机上进行实验,每台机器的配置情况相同,均如表 1 所示:

表 1　单机配置情况

CPU	Intel Pentium3　1 GHz
内存	DDR 512 MB
操作系统	Redhat8.0

为保证实验的顺利进行,需要正确安装并配置 Hadoop 的多机运行模式,包括按照 XML 标准格式将这 3 台机器的名称

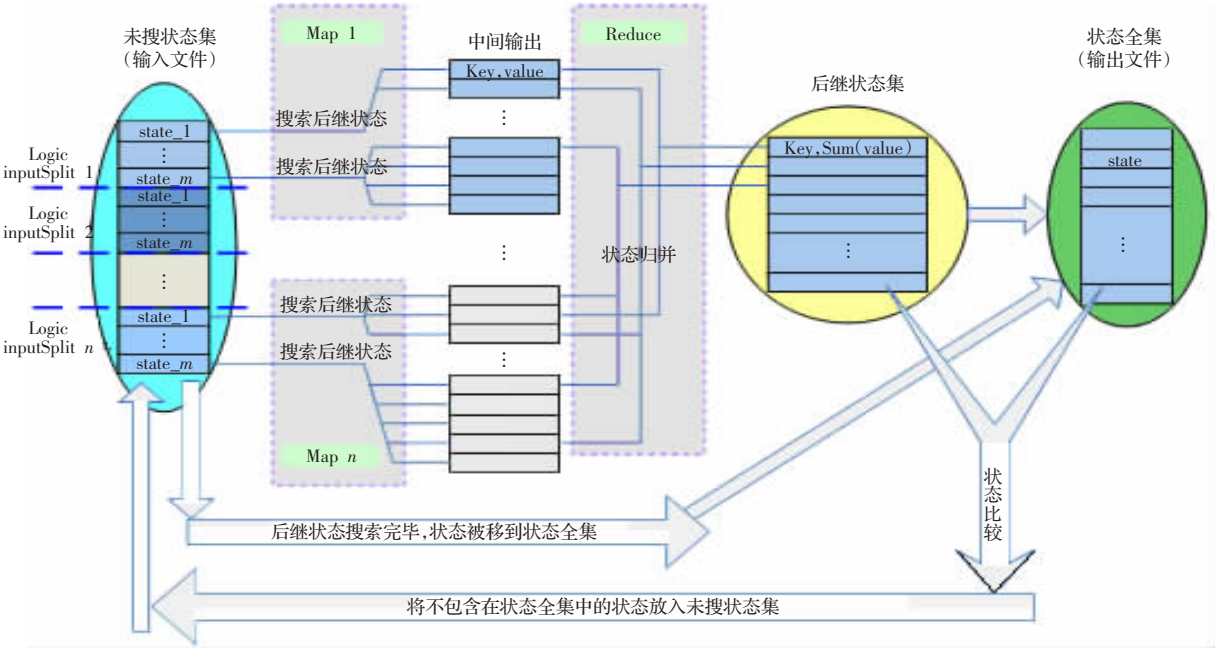


图 2　基于 MapReduce 的分布式状态空间生成的工作原理

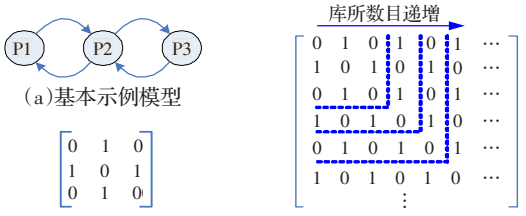
与 IP 地址添加到 hadoop-site.html 配置文件,设置机器间的无口令 ssh 登录等。

5.2　示例模型

为了便于实验的进行,选取如图 3 所示的一种示例模型。虽然模型比较简单,但它具备状态空间模型的基本特征,同时也易于扩大模型的状态空间。该模型的状态空间有两种扩大模式:

(1)保持模型的库所(place)数 p 不变、增加单库所的最大托肯(token)数 t;

(2)保持单库所的最大托肯数 t 不变,按照图 3(c)示意的规律来增加模型的库所数 p。



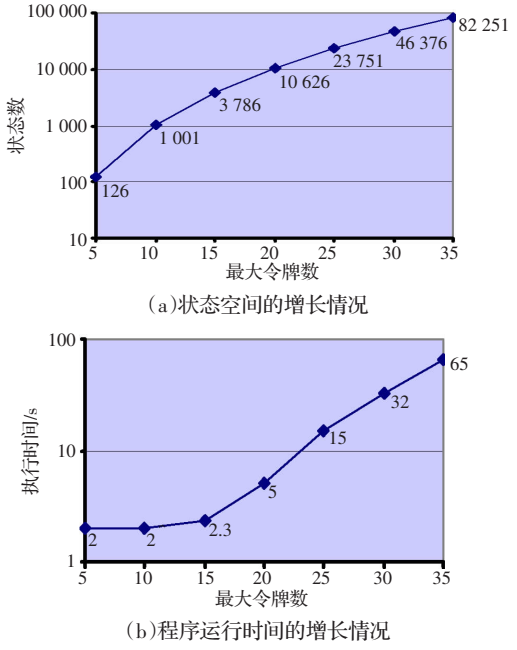
(a)基本示例模型　(b)基本示例模型对应的转换矩阵　(c)示例模型的转换矩阵递增规律

图 3　简单示例模型及其增长规律

该文从这两个方面来研究基于 MapReduce 的分布式状态空间生成算法对模型的状态空间膨胀的适应性。

5.3　实验结果与分析

图 4 和图 5 分别显示了这两种状态空间规模扩大模式下算法运行结果的变化情况。



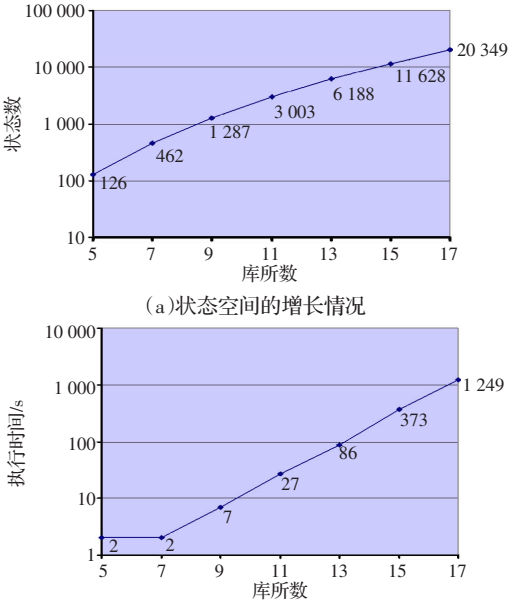
(a)状态空间的增长情况

(b)程序运行时间的增长情况

图 4　库所数 p=5 时模型的状态空间及其生成时间随最大托肯数增大的变化情况

在图 4 中,模型的库所数 p 固定为 5,最大托肯数 t 每次增加 5。图 4(a)反映了状态空间的增长情况,图 4(b)反映了程序执行时间的变化规律。由图 4 可以看出:

(1)随着最大托肯数 t 的增大,模型的状态空间快速膨胀:当 t=5 时,其状态数仅为 126;当 t=10 时(为原来的 2 倍),其状态数相应增加到 1 001 个,几乎是原来的 8 倍;当 t 再翻一倍增加到 20 时,其状态数相应地又翻了 10 倍,变为 10 625;当最大托肯数增加到 35 时,其状态空间膨胀到 82 251 个,增长了 600 多倍。这组数据证实了状态空间爆炸问题的严重性。



(a)状态空间的增长情况

(b)程序运行时间的增长情况

图 5　最大托肯数 t=5 时模型的状态空间及其生成时间随库所数增大的变化情况

(2)在同等 t 值增加速度条件下,算法的执行时间开始增加并不明显(例如:t 分别为 5、10 和 15 时,程序基本上都在 2 s 左右执行完成),而当 t>15 时,以后 t 值每增加 5,其执行时间都至少延长 1 倍。通过分析,产生该现象的原因是:前期的状态空间较小(状态数在数千以内),在分布式环境下,各 Map 任务的计算量并不大,此时算法的主要开销是由 Hadoop 本身的数据发布与收集造成的,这个时间相对固定,因此相差不大;后期状态空间达到一定程度,后继状态推导的计算任务占据了程序总时间的绝大部分,因此总时间受状态空间增长速度的影响较为明显。

图 5 显示了在最大托肯数 t 固定为 5、逐次增加模型的库所数 p(每次增加 2 个)时得到的运行结果,其(a)图和(b)图分别反映了状态空间和运行时间的增长情况。

与图 4 相比,图 5 的变化规律基本相似,但其状态空间和程序运行时间的增长速度有显著提高:

(1)在图 5(a)中,模型的库所数 t 每增加 2 个,其状态空间就增加 2 到 3 倍。在库所数 p 和最大托肯数 t 均为 5 时,模型共有 126 个状态,当 p 保持不变、t 增加到 15 时,模型的状态数增加到 3 786 个;当保持 t 不变、p 增加到 15 时,状态数则增加到 11 628 个。由此可见,p 值要比 t 值对模型状态空间的规模影响更大。

(2)在图 5(b)中,当 p 值较少时,程序的执行时间也基本上变化较小;当 p>9 时(对应的状态数也超过 1 000),程序几乎按照 p 值每增加 2、执行时间就翻 4 倍的速度增长;当 p>15 时,其执行时间需要 6 分多钟,是 p=5、t=15 时的执行时间的 162 倍。根据分析,这是由于库所数越大、模型的结构更复杂、每个状态的后继推导情况更多的原因造成的。

比较图 4(b)与图 5(b)可见,对于状态空间规模的增长主要由托肯(token)数增加引起的一类模型,该算法具有更好的适应性和可扩展性。

(下转 68 页)