文章编号: 1001- 2486(2010) 01- 0068- 06

涵盖 I/O 的广义存储一致性模型*

李 琼, 邓明堂, 杨学军
(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: 作为计算机体系结构核心问题之一的存储一致性研究主要是围绕 CPU 访存一致性问题展开的, I/O 设备 DMA 操作引发的存储一致性问题则一直处于研究的边缘。从 I/O 与存储体系结构一体化设计理念出发, 针对支持全局 DMA 访问的分布共享存储(DSM) 系统存储一致性问题, 研究广义存储一致性, 定义了涵盖 I/O 的广义程序概念, 建立了广义域存储一致性模型, 研究了基于广义域存储一致性模型的 Cache-Memory-I/O 一致性协议实现技术, 对于 DSM 系统实现全局共享 I/O 具有指导意义和参考价值。

关键词: I/O 体系结构; 广义存储一致性模型; 全局共享 I/O
中图分类号: TP391 文献标识码: A

A General Memory Consistency Model Included I/O Operations

LI Qiong, DENG Ming-tang, YANG Xue-jun
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: Memory consistency research, as one of the core issues of computer architecture, mainly focused on the CPU access memory, while the consistency issues of DMA operations is seldom studied. According to the idea of integrating the architecture design of I/O and memory, the paper defines the concept of general program included I/O operations, aimed at the consistency issues of DSM (distributed shared memory) systems with global DMA operations. The paper studies the general memory consistency model, and builds the general scope consistency model. Based on the general scope memory consistency model, the Cache-Memory-I/O consistency protocol and its implement technology are studied. All this may help the global shared parallel I/O system design for DSM system.

Key words: I/O architecture; general memory consistency model; global shared I/O

存储一致性模型是系统设计者与应用程序员之间的一种约定, 实际上是对共享存储系统中的多处理机的访存次序规定的约束, 是一种编程接口。为了保证共享存储系统上的应用程序的正确性, 系统必须首先有一套规则, 这套规则告诉程序员按照何种访存方式才能得到预期的结果, 编译器、操作系统和硬件系统则共同实现这一规则。

并行计算机系统结构中包含的存储层次越来越多, 层次间及层次内的副本越来越多, 缓存管理和数据一致性维护问题更为突出, 实现难度也越来越大。I/O 设备处于系统存储层次结构的底层, I/O 子系统的研究不能孤立进行, 即不能只局限于 I/O 子系统内部, 而应该与存储体系结构一体化设计。目前存储一致性研究主要是围绕 CPU 与存储器之间的数据一致性展开的, I/O 设备的一致性则一直处于研究的边缘, 将 Cache-Memory-I/O 整个系统作为数据一致性域的工作则更少。Cache-Memory 和 I/O 设备之间的数据一致性采用的一直是简单低效、粗粒度、通过操作系统过度干预的解决办法, 严重阻碍了 I/O 性能的提高。随着 I/O 瓶颈问题日益严峻, I/O 与存储体系结构设计一体化是必然的发展趋势, 研究涵盖 I/O 的存储一致性模型及实现技术具有重要意义。

1 涵盖 I/O 操作的存储一致性

存储一体化设计理念将 I/O 设备与主存、Cache 作为一个整体进行数据一致性维护, 因此必须考虑

* 收稿日期: 2009- 09- 09
基金项目: 国家自然科学基金资助项目(60621003)
作者简介: 李琼(1967—) , 女, 研究员, 博士。

从整个存储系统(包含 I/O 设备)的角度重新审视存储一致性模型。DMA 操作具有 CPU 干预少、并行性好、传输速度快等优点,是 I/O 设备的主要工作方式。DMA 可以与 CPU 同时访问存储器,因此可能引发存储不一致问题。如果不正确地处理 DMA 操作与存储器、Cache 的一致性关系,那么一段看似正确的代码就可能得到错误的结果。例如,处理器 $Processor_i$ 首先向硬盘控制器 $Disk\text{-}Controller_j$ 发起一次 DMA 操作,$Disk\text{-}Controller_j$ 进行 DMA 传输,直接向 $Processor_i$ 的存储器中写数据。在 DMA 传输期间,$Processor_i$ 一直等待 $Disk\text{-}Controller_j$ 的 DMA 完成中断信号,一旦收到该中断信号,$Processor_i$ 则开始读取新写入内存的数据。$Disk\text{-}Controller_j$ 在发送出最后一个数据后便认为自己的工作已完成,并发出中断信号。在多处理机系统中,如果 $Processor_i$ 和 $Disk\text{-}Controller_j$ 不在同一结点上,那么进行远程传输的时延可能会很长且无法预计。如果没有合适的同步机制,那么中断信号完全有可能在数据到达存储器之前先行到达处理器,使处理器读到数据旧值,从而引发错误。

1999 年,威斯康辛大学的 Mark D. Hill 等在存储一致性模型基础上提出了 I/O 体系结构的系统级框架 WIO(Wisconsin I/O Consistency)[1],试图为 I/O 建立统一的一致性描述方法。但 WIO 没有为 I/O 一致性建立规范的数学描述,且是分别从处理器和 I/O 设备独立构建偏序关系,因此建立的 I/O 一致性与存储一致性相对独立,其后的各种一致性模型也是简单地将传统一致性模型与一组固定的 I/O 操作偏序相加。SGI 公司在 CC-NUMA 系统研制时注重 I/O 与存储体系结构一体化设计,将 DMA 引发的存储一致性问题同 Cache 一致性协议进行了系统级融合,取得了很好的应用效果。

针对分布共享存储体系结构,我们定义全局 DMA 操作为 I/O 设备至远程分布共享主存的直接访问,建立支持全局 DMA 的广义存储一致性模型,设计支持全局 DMA 的 Cache-Mem-I/O 一致性协议,实现全局共享 I/O 系统结构,减少操作系统 I/O 服务开销,优化 I/O 性能。

2　广义程序模型

传统的存储一致性模型很难兼容 DMA 操作的原因是:这些一致性模型中的程序都像是在裸机上运行的,看不到操作系统的影子,同步操作全部由程序中的同步指令来指派,同步对象也都是体系结构抽象原语(lock 或 barrier 等);而且操作的概念也过于狭隘,仅仅包含 CPU 执行的带有程序员意志的"净指令",不包括系统事件、中断和 I/O 操作等。

DMA 的执行相对于一个应用程序来说有同步和异步两种方式。应用程序中对 I/O 设备的读写指令(如文件操作)会触发一个异常进而引发系统调用,再由操作系统调用驱动程序发起 DMA 操作。在此过程中,DMA 操作可以看作是由应用程序显式启动的,程序员可以预计到 DMA 启动的相对时机,此类 DMA 操作属于同步 DMA 方式。而有些 DMA 启动对应用程序来说是隐式的和异步的,如操作系统进行页面换入换出等虚存管理操作时引发的 DMA 操作,对于程序来说则是不可预见的。

为了与存储一致性兼容,DMA 相关设计做了妥协,例如从存储器中专门开辟出一块不可以缓存的(Uncacheable)区域用于 DMA,或者强迫所有的 DMA 都要先经过 Cache。这些解决方法都是简陋而低效的,为了设计高效的 DMA-存储一致性兼容模式,必须将 DMA 操作和程序指令纳入一个级别看待。为了建立一个与真实系统相符、涵盖 DMA 操作的存储一致性模型,有必要先建立一种涵盖范围更宽广的程序概念——广义程序模型。定义广义程序指令如表 1 所示。

DMA 操作的执行实际上是一系列访存操作的组合,为了表达上的统一,将 DMA 访存同 CPU 访存一样记为一条指令,将访存操作的操作数都改为操作区间,并统称为操作域。称由广义访存操作和广义同步事件构成的集合为广义程序指令集,用符号 V 表示。

传统的一致性模型中通常称串行程序中指令出现的顺序为程序序(Program Order),这是一个静态的、确定的全序序列,对于判断程序执行的正确性非常重要[2]。在顺序一致性模型中,只需对比指令的完成序和程序序是否一致就可以知道程序是否遵循了顺序一致性模型。静态的程序在执行过程中会发生很大改变,尤其在加入了 DMA 访存操作之后,原来数据一致性模型中的程序序概念不存在了,因为 DMA 操作的顺序和时机对程序员来说可能是不可预测的,程序的每一次运行都可能产生不同的访存序

列和系统事件。为此本文提出广义程序运行的概念。

表1 广义访存指令和广义同步操作

Tab.1 General access instructions and synchronous operations

操 作	操作说明	操 作	操作说明
G-LOAD	广义读操作,CPU 发起的读内存和 DMA 发起的读内存操作的合称	*G-Aquire*	获得广义锁,获得某段代码的独占执行权或独占运行资源,CPU 获得锁和 DMA 获得锁的总称
G-STORE	广义写操作,CPU 发起的写内存和 DMA 发起的写内存操作的合称	*G-Release*	释放广义锁,释放某段代码的独占执行权或独占运行资源,CPU 释放锁和 DMA 释放锁的总称
G-ACCESS	广义访存操作,广义读写操作的统称	*G-Synchron*	广义同步操作,广义锁操作的统称

定义1 广义进程运行。

一个(传统意义上的)进程的一次运行 p^r 是一个序结构< $V^r(p)$, $PO^r(p)$ >,其中 $V^r(p)$ 是进程运行期间的广义程序指令的集合,$PO^r(p)$ 是 $V^r(p)$ 上的一个全序关系,r 为此次进程运行的运行标号。

定义2 广义程序运行。

由 N 个(传统意义上的)进程 $p_1, p_2, \cdots, p_N$ 组成的程序 $PRG(p_1, p_2, \cdots, p_N)$ 的一次运行 PRG^r 是一个序结构< $V^r(PRG)$, $PO^r(PRG)$ >,其中,$V^r(PRG) = V^r(p_1) \cup V^r(p_2) \cup \ldots \cup V^r(p_N)$ 是程序 PRG 的广义程序指令集合,$PO^r(PRG) = PO^r(p_1) \cup PO^r(p_2) \cup \ldots \cup PO^r(p_N)$ 是 PRG 广义程序指令的流出序,r 称为此次程序运行的运行标号。

可以看出,进程的每次运行可能会定义出不同的序结构,为了讨论的方便,下面定义程序运行的等价性。程序运行等价的衡量标准是终态相等,引入 DMA 操作后,广义程序(段)运行的等价性定义如下:

定义3 等价运行。

两个广义程序(段) PRG_m, PRG_n 的两次运行 PRG_m^k, PRG_n^l 等价是指: PRG_m^k, PRG_n^l 在仅含 *G-LOAD* 和 *G-STORE* 指令的抽象 RISC 机器上产生的结果相等,其中运行结果是指所有存储器单元(和寄存器)的最终值以及所有 I/O 设备的最终状态。等价的广义程序运行构成的集合为运行等价类,并记 PRG^r 的运行等价类为 A^r。

在上文建立的广义程序模型中,如何判断一次运行是否正确呢?传统意义中的并行程序执行的正确性都有一个判断标准:若在多处理机环境下的一个并行程序的运行与同一程序在单处理机环境下的一个运行等价,则可以认为这个并行程序的运行是正确的[3]。这个判断标准中将多处理机映射到单处理机上,是为了实现并行程序的序列化,也就是强迫程序中的所有指令构成全序结构,且这个全序与每个进程的程序序的并集无圈。这个正确性判定标准也成为顺序一致性模型的核心思想,其中的关键在于通过去并行化和引入额外的序关系来保证程序无争端和无歧义地执行。本文借鉴上述思想定义广义程序运行的正确性和顺序一致性。

定义4 广义程序运行的正确性。

如果在多处理机环境下,涵盖 DMA 操作的广义并行程序的一次运行与同一程序在单处理机环境下以查询方式访问 I/O 设备的一次运行等价,则称该广义程序的运行是正确的,正确的运行所属的等价类称为正确运行等价类。

定义5 广义顺序一致性模型。

如果一个广义程序的一次运行满足下面两个条件,则称其符合广义顺序一致性模型:

(1) 任一处理器(或 DMA 控制器)的 *G-ACCESS* 指令都按照其在广义程序运行中流出的顺序完成;

(2) 在当前的 *G-ACCESS* 指令彻底完成之前不能开始执行下一条 *G-ACCESS* 指令。

广义顺序一致性模型比传统的顺序一致性模型更加严重地偏离了实际系统,因为它不但将并行处理机系统进行了去并行化,还将 DMA 和 CPU 的访存操作进行了串行化,实际上将 DMA 方式变成了查询方式,这样的一致性模型在实际系统中执行效率较低,因此有必要对其进行一定程度的松弛。

3　广义域一致性模型

顺序一致性模型中引入了很多序关系,其实对有些序关系进行弱化是不会引发程序运行错误的,而有些操作就必须按照某种固定的次序完成,否则就可能引发运行错误。所以要对顺序一致性进行松弛,就必须提取出那些对程序正确性有影响的序关系,并弱化其他序关系,给程序执行一定的自由度和优化空间。这些至关重要的序关系就是以下要分析的冲突访问对之间的关系。

定义 6　冲突访问。

两条 *G-ACCESS* 指令 $\hat{L}_i$, $\hat{L}_j$ $(i \neq j)$ 如果符合下列两个条件则称它们是冲突的,记为 $\langle \hat{L}_i \mid \hat{L}_j \rangle$ 或 $\langle \hat{L}_j \mid \hat{L}_i \rangle$,并称为一个冲突访问对:(1)它们的操作域的交集不为空;(2)至少有一条是 *G-STORE* 指令。

定义 7　冲突访问集合。

设 PRG^r 是广义程序 PRG 的一次运行,则记 $C(PRG^r) = \{(\hat{L}_i, \hat{L}_j) \mid (\hat{L}_i, \hat{L}_j \in V^r(PRG)) \cap (\langle \hat{L}_i \mid \hat{L}_j \rangle)\}$ 为 PRG^r 的冲突访问对集合。定义在 $C(PRG^r)$ 上的任一无圈的定序为 PRG^r 的一个候选执行,记为 $CR(PRG^r)$,其中定序是指对任一 $(\hat{L}_i, \hat{L}_j) \in C(PRG^r)$ 来说,$\hat{L}_i \xrightarrow{CR} \hat{L}_j$ 和 $\hat{L}_j \xrightarrow{CR} \hat{L}_i$ 有且仅有一个成立。

静态的程序序是反映程序员预期行为的唯一凭据,也是程序执行正确性分析的语义基础。因此狭义上的程序 PRG 的一个候选执行 $CR(PRG)$ 只要与程序序 $PO(PRG)$ 构成的并集无圈就可保证 $CR(PRG)$ 的正确性,其中并行程序的 $PO(PRG)$ 是将其映射到单处理机环境中的指令流出序。下面根据运行等价类来定义程序 PRG 的一次运行 PRG^r 的对等链。

定义 8　对等链。

设 $PRG^r = < V^r(PRG), PO^r(PRG) >$ 是程序 PRG 的一次运行,A^r 是其所属的运行等价类,若存在序结构 $\overrightarrow{PRG} = < \vec{V}^r(PRG), \overrightarrow{PO}(PRG) >$,使得 $\vec{V}^r(PRG) = \vec{V}^r(PRG)$ 且 $\overrightarrow{PRG}$ 为全序关系,则称 $\overrightarrow{PRG}$ 是 PRG^r 的一条链。特别地,如果还满足 $\overrightarrow{PRG} \in A^r$,则称 $\overrightarrow{PRG}$ 是 PRG^r 的一条对等链。

需要指出的是,PRG^r 的对等链可能不只一条,由 PRG^r 的所有对等链构成的集合是 A^r 的一个子集,称为对等类,记为 $\vec{A}^r$。如果对等链满足上节定义的正确性的要求,则称 $\vec{A}^r$ 为正确对等类。不难看出,对等链 $\overrightarrow{PRG}$ 实际上是广义程序运行 PRG^r 的等价串行执行。

定理 1　若 $CR(PRG^r)$ 为广义程序 PRG 的运行 PRG^r 的一个候选执行,$\vec{A}^r$ 是 PRG^r 的正确对等类,则候选执行 $CR(PRG^r)$ 是正确的充分条件是:在 $\vec{A}^r$ 中存在 PRG^r 的对等链 $\overrightarrow{PRG} = < \vec{V}^r(PRG), \overrightarrow{PO}(PRG) >$,使得 $CR(PRG^r) \cup \overrightarrow{PO}(PRG)$ 无圈。

定理 1 告诉我们,只要某个松弛的一致性模型满足 $CR(PRG^r) \cup \overrightarrow{PO}(PRG)$ 无圈,就可以保证程序的正确性。比如释放一致性模型、单项一致性模型(Entry Consistency)和域一致性模型(Scope Consistency)等。这些模型对程序的访存约束越松弛,性能优化的空间就越大,但对程序员的要求也越高。单项一致性模型和释放一致性模型类似,要求编程人员在临界区的开始和结束时分别使用 *Aquire* 和 *Release* 操作;不同之处在于,单项一致性模型要求每个共享变量都与某同步变量相关联,只有对同步变量的 *Aquire* 操作完成之后,相关的共享数据才得到一致性保证。单项一致性将每个共享变量与一个锁变量绑定在一起,解决释放一致性模型中所有同步都是全局同步的问题。但是,单项一致性模型给程序员编程增加了很大的负担,实现起来非常困难。域一致性模型[4]兼顾了单项一致性模型的性能优点和释放一致性模型的易编程性,它的核心思想是将锁变量与数据联系在一起,但不是给每个数据都加锁,而是给一个域上锁。

域释放一致性模型更符合包含 DMA 操作的共享存储系统,因为 DMA 操作具有地址连续、成批访问特点,这实际上是一种特性非常好的域,可以在不增加太多编译负担和硬件复杂度的前提下实现这一模型。因此,本文针对支持全局 DMA 操作的共享存储系统提出广义域一致性模型(General Scope

Consistency Model)。

在广义域一致性中,一致性域是存储器的一个局部视图,域内的数据受同一把锁保护,数据更新只对同一个域的指令可见,因此可以将一个一致性域看作由同一把锁保护的所有临界区的集合。每个一致性域都有一对"开域/关域"操作,其作用相当于获取锁/释放锁,当某个进程打开某个域时就称此进程产生了该域的一个段。段实际上是一个一致性域在给定进程中的一个临界区。域一致性不保证某个域中的指令能看到该域之外的数据更新。属于不同的一致性域的段的代码可以嵌套、交叉,而且两个段的执行也可以重叠和交叉。

为便于描述广义域一致性模型,定义一条 *G-STORE* 广义指令$\hat{L_j}$相对于域S执行完毕是指:$\hat{L_j}$处于一致性域S的一个段$S(i)$内,而且域S的段$S(i)$已经执行了关闭域操作。一条 *G-STORE* 指令$\hat{L_j}$相对于进程P执行完毕是指:$\hat{L_j}$所做的数据更新能被P感知到,即如果没有其他对同一单元的写操作,进程P读该单元将取回$\hat{L_j}$所写的值。

定义9 广义域一致性模型。

广义域一致性模型的规则可以概括为(括号内为相应的基于锁机制的定义):

(1)任何一个一致性域在任何时刻只能存在至多一个段,即只有该域所有的段均执行了关闭域操作之后,新的段才允许执行开域操作;

(2)若某个进程P要打开一个一致性域S(创建S的一个新段$S(i)$),则相对于域S已执行完毕的所有 *G-STORE* 指令必须也已经相对于进程P执行完毕(在处理器P或DMA执行 *G-Aquire*$(lock_i)$操作之前,所有已执行的相对于锁$lock_i$的 *G-STORE* 必须相对于处理器P或DMA执行完成);

(3)进程P被允许执行任意一条广义访存指令$\hat{L_i}$之前,进程P中所有$\hat{L_i}$之前的开域操作必须全部执行成功(在处理器P或DMA执行任意一条广义访存指令$\hat{L_i}$之前,所有在$\hat{L_i}$之前的 *G-Aquire*$(lock_i)$操作都已经完成)。

在上述定义中,第一个规则确保了同一个域的各个段间的互斥性,第二个规则确保了当一个进程P打开了一致性域S之后,之前相对于S域执行完毕的数据更新能立即对于进程P可见。依据这两个规则可得出以下推论:一个域的段在执行闭域操作之前,必须确保本段中所做的数据更新已在后续可能进入本域的进程上生效。第三个规则通过强迫域的打开操作在后续的访存指令开始执行之前完成,来确保这些访存指令访问的都是最新数据。

广义域一致性模型之所以比单项一致性模型的编程界面好,一个重要原因是:可根据释放一致性模型中已经存在的同步直接确定一致性域。例如,通过编译器识别、划分、记录一致性域,这是一种隐式的实现方法。也可以在程序中由程序员通过标号指明一致性域,利用所定义的域编制程序,这属于显式的实现方法。两种方法都需要硬件提供锁原语来保证锁变量的操作原子性。

4 广义域一致性模型实现与应用

我们遵循广义域存储一致性模型,在CC-NUMA系统设计时采用扩展的Cache一致性协议统一解决Cache-Memory-I/O的数据一致性和访问冲突问题。

在实现广义域一致性模型时,全局DMA写操作被拆分成对每个Cacheline的写操作,先获得Cacheline的独占访问权,然后给所有拥有此Cacheline副本的结点发送失效命令,收到失效命令的结点失效该副本并返回一个失效应答。拥有独占访问权的结点对失效应答进行计数,当收到了所有的失效应答后,此结点执行对Cacheline的更新操作,并向结点控制器提交一个改写完成反馈信号。当结点控制器收到此次DMA操作的所有Cacheline的反馈信号时,则认为此次DMA全部完成,并产生一个DMA完成中断信号给处理机。

在上述一致性模型实现方案中,获得锁和释放锁实际上是通过异常事件和DMA中断实现的。由于一次DMA时间很长、涉及的数据较多,为了确保程序的正确性,即便与线程相关的数据已经准备好也要等待中断信号的到来,才能使用数据。为了进一步松弛存储一致性约束,本文提出了基于DMA标号的

同步机制。系统在目录项中除了 Cacheline 的状态标识外还增加了一个 DMA 标号,用来表示一个处于临界区的 DMA 操作。由操作系统分配唯一的全局 DMA 标号(可循环使用),DMA 访问目的存储器时在修改 Cacheline 状态时也会将此标识填入域标号,表示对该 Cacheline 的写操作已经相对于域完成。同一个域的另一个段的访存操作检查目录项时,会读取 DMA 标号并和自己的域标识比较,若相等则访存操作可以执行,否则操作将被延迟。

在结点控制器芯片中设计全局并发 DMA 引擎,负责全局 DMA 请求的并发处理、域管理、请求合并、小粒度写合并、响应数据的生成等工作,提交请求给 Cache 一致性处理引擎,负责 Cacheline 的一致性维护。通过硬件支持全局 DMA 访问,实现多个 I/O 设备控制器与多个远程计算结点内存间的多流并发、直接数据传输,高效地实现全局共享 I/O 系统。

对于非共享 I/O 系统,并行文件系统服务一次文件读请求,在未命中缓存情况下读响应数据要经过 5 步传输路径和 3 次内存拷贝;而对于全局共享 I/O 系统,整个全局 DMA 过程只包含 2 步传输路径和 1 次内存拷贝,分析结果表明[5],基于广义域一致性模型的全局共享 I/O 系统具有显著的性能优势。

为了验证广义域一致性模型及 Cache-Memory-I/O 协议设计的正确性,我们一方面构建 Cache 协议模型进行了形式化验证,另一方面,基于 FPGA 原型系统和最终研制的 CC-NUMA 系统,采用多种验证工具和测试手段,充分验证了一致性模型及协议设计的正确性。在文件系统一级采用 IOR 等基准测试程序和应用程序,测试并行 I/O 带宽及可扩展性,实测结果表明,系统 I/O 吞吐能力和扩展能力强,实测并行 I/O 带宽高达 20.2GB/s,并行 I/O 带宽随着进程个数良好扩展[5]。

5　总 结

建立的广义存储一致性模型对于并行计算机 I/O 系统设计具有重要的指导作用,取得了很好的应用效果。基于广义域一致性模型的全局共享 I/O 系统在系统底层实现了单一存储空间和单一 I/O 空间,使其高层系统软件,包括并行文件系统、虚拟存储系统、并行检查点机制,以及用户应用程序均可从中受益,有效利用分布式共享内存和分布式共享 I/O 资源,获得更好的性能、更高的透明度、更高的可用性和更好的应用程序兼容性。

参 考 文 献:

[1] Mark D H, Anne E C, Manoj P, et al. A System-level Specification Framework for I/O Architectures[C]//The 11th SPAA, 1999: 27-30.

[2] Yang X J, Dai H D. Operating System-centric Memory Consistency Model—Thread Consistency Model[C]//The Fourth APPT'01, Germany, 2001: 12-15.

[3] 胡伟武. 共享存储系统结构[M]. 北京: 高等教育出版社, 2001: 1-35.

[4] Iftode L, Singh J P. Scope Consistency: A Bridge between Release Consistency and Entry Consistency[C]//The 8th SPAA, 1996: 10-18.

[5] Li Q, Pang Z B, Guo Y F, et al. A GPDMA-based Distributed Shared I/O Solution for CC-NUMA System[C]//The 9th Inter. Conf. for Young Computer Scientists, 2008: 172-177.