第 6 卷第 8 期
1995 年 8 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 6, No. 8
Aug. 1995

一种新的用于分布共享存储系统的存储器一致性算法*

邢 浩 沈美明 高耀清

(清华大学计算机系,北京 100084)

摘要 SVM 系统或称 DSM 系统是在基于分布存储器的多处理机上,实现物理上分布但逻辑上共享的存储系统.它集共享存储器易于编程和分布存储器可扩充性好于一体,为 MPP 计算机的使用带来了方便.本文首先介绍了 SVM 系统的数据一致性问题及其解决办法,然后提出了一种新的固定分布管理算法 NFDMA,并对此算法作了分析,最后与李凯的固定分布管理算法作了比较.

关键词 共享虚拟存储系统*,固定分布管理算法*,数据一致性.

近年来,并行处理技术越来越受到人们的重视.并行处理系统可以分为两类:紧耦合的基于共享存储器的多处理机和松耦合的基于分布存储器的多计算机.紧耦合的多处理机由多个处理器和共享存储器组成,处理器通过互连网络与共享存储器连接起来,每一个进程都可以访问全局存储器.这种体系结构对用户来说编程比较容易,其缺点是每个处理器都通过互连网络访问存储器,因而限制了处理器的个数.松耦合的多计算机由多个自带本地存储器的独立的处理机通过高速互连网络连接起来,处理机间通过消息传递(message_passing)方式相互通信,系统结构灵活,可扩充性好.这种系统的缺点是编程及任务动态分配困难,难以在处理机结点间传递复杂数据结构.

从 80 年代开始,人们考虑如何集两者之长,避两者之短,提出了共享虚拟存储(Shared Virtual Memory)的概念.共享虚拟存储系统又称分布共享存储系统(Distributed Shared Memory),这是在基于分布存储器的多处理机上,实现物理上分布但逻辑上共享的存储系统,其优点是编程容易、系统结构灵活、可扩充性好、系统价格低、有较好的软件移植性.因此,SVM 受到了越来越广泛的重视.

SVM 存储空间可以分成许多页,这些页分散在每个处理机的局部存储器 LM 上,每一个局部存储器都可以看作是 SVM 的 Cache.与多处理器的 Cache 相类似,SVM 系统也会产生数据一致性问题.有两类协议来处理这一问题:一类是写无效,另一类是写更新协议.由于

---

* 本文 1994－02－02 收到,1994－03－29 定稿

本项目受到国家自然科学基金的资助.作者**邢浩**,1970 年生,研究生,主要研究领域为并行与分布处理计算机系统.**沈美明**,1938 年生,副教授,主要研究领域为并行与分布处理计算机系统.**高耀清**,1960 年生,讲师,主要研究领域为分布与并行处理和多处理机体系结构,描述性语言,软件工程.

本文通讯联系人:邢浩,北京 100084,清华大学计算机系

写更新协议需要有专门的硬件支持,我们这里重点讨论写无效协议.使用写无效协议时,每一个数据项都有一个状态信息,指明该数据有效、共享、只读或可写.对于读操作,若数据有效则可立即返回;如果数据无效,则需发读请求读取数据.对于写操作,如果一个数据是有效的而且可写,则可立即返回;如果不可写,则需发无效请求,使其它拷贝无效,若是数据不在本地,还需发一个读数据请求.

由于共享数据在SVM系统中可以迁移,这就给数据的定位和访问带来了复杂性.多处理器的Cache一致性通常采用总线监测和目录表的方法.由于在SVM系统中,各处理机通过互连网络连接,采用目录表方法更为适宜.

1　一种新的固定分布管理算法NFDMA

1.1　一个共享虚拟存储模拟系统

清华大学计算机系于1992年5月研制成的并行图归约智能工作站[1],是一个典型的松耦合基于分布存储器的多计算机系统.它由SUN4工作站为宿主机以及16/32个IMS T800 Transputer为结点机组成的并行加速器构成,其上运行Genesys操作系统.我们的目标是在Genesys操作系统上层实现一个共享虚拟存储模拟系统.在这里我们只说明SVM模拟系统中所用的存储器一致性算法,考虑到本系统的处理机个数较多,采用固定分布算法比较合适,这样既可克服集中管理算法在结点数较多时的瓶颈问题,又比动态管理算法易于实现.我们在文献[1]提出的固定分布算法的基础上,提出了一种新的固定分布算法NFDMA.在介绍算法之前,先给出一些基本概念.

- 页占有者:当前拥有此页最新拷贝的结点.
- 页表:记录页的状态信息,包含访问方式:指出页当前可访问的方式;拷贝集:当前占有此页读拷贝的处理机号;锁:用于同一处理机上不同进程的多个页失效同步和远程页请求的同步.
- 无效原语:使页的拷贝无效.

1.2　一种新的固定分布管理算法NFDMA

在我们的系统中,所有的共享页以1K为单位,按Hash函数平均分配到各个结点上,每个结点管理一部分页.

$$H(p)=P \bmod N$$　　其中P为共享页的页号,N为结点个数.

为描述算法方便起见,我们把页的管理者称为manager,发请求的结点称为requester,当前或最近拥有某页写拷贝的结点称为owner,每页的owner只有一个,只有manager知道owner的位置.

页表至少应包含以下内容:access_type:该页所处的状态(下面有说明);owner:该页的占有者;copyset:拥有此页读拷贝的结点;lock:用于同步对页的访问.

owner和copyset仅管理者所有.

页状态有:nil:此页在盘上;uncached_remote:仅管理者拥有此页,其它结点均无此页.该状态仅对管理者有效;shared_remote:管理者和其它结点都拥有此页的拷贝;dirty_remote:其它结点占有此页,并且修改过,但管理者不占有此页.

读操作的算法如下:

```
read(P)    /* P为页号 */
  {
   If I am manager of P
     {
      if type=uncached_remote or shared_remote;
              读满足;
      if type=nil;
              读磁盘;
              type=uncached_remote;
      if type=dirty_remote;
              向 owner 结点发读请求读取该页;
              type=shared_remote;
     }
else    /* 普通结点 */
  {
   if type=shared_remote or dirty_remote;
           读满足;
   if type=nil;
           send a read-request to manager;
           type=shared_remote;
           如果不是从 manager 传来的 P,向 manager 发一确认消息;
  }
  }
```

管理者接到读请求的处理如下:

```
read_server()    /* 负责接收读请求 */
  {
   if type=uncached_remote or shared_remote;
           send P;
           type=shared_remote;
   if type=nil;
           读磁盘;
           send P;
           type=shared_remote;
   if type=dirty_remote;
           通知 owner 结点发拷贝给 requester;
           recv a copy of P; /* manager 也接收一份 P 的拷贝 */
           recv a confirmation; /* 从 requester 发来 */
           type=shared_remote;
  }
```

写操作的算法如下:

```
write(P)
  {
   If I am manager of P
     {
      if type=nil;
              读磁盘;
              type=uncached_remote;
      if type=uncached_remote;
              写满足;
      if type=shared_remote;
              向 copyset 中的结点发无效消息并接收回答;
              type=uncached_remote;
      if type=dirty_remote;
              向 owner 结点发写请求,并使 owner 的页无效;
              type=uncached_remote;
```

```
    }
    else    /* 普通结点 */
     {
      if type=dirty_remote:
            写满足;
      if type=nil or shared_remote:
            send a write-request to manager;
            若发者不是 manager,send a confirmation to manager;
            type=dirty_remote;
     }
    }
```

管理者接到写请求的处理如下:

```
write_server()    /* 负责接收写请求 */
  {
   if type=nil:
         读磁盘;
         send P;
         type=dirty_remote;
   if type=uncached_remote:
         send P;
         type=dirty_remote;
   if type=shared_remote:
         向 copyset 中除了 requester 的其它结点发无效消息并接收回答;
         type=dirty_remote;
         send P;
   if type=dirty_remote:
         通知 owner 结点发拷贝给 requester 结点,并使 owner 的页无效;
         recv a confirmation;    /* 从 requester 发来 */
  }
```

上面给出了读写算法的流程,这一算法实现了数据的严格一致性,即每次读取操作都返回该单元的最新值,任何时刻系统中的数据都是一致的.当一个处于 dirty_remote 状态的管理者接到其它结点来的读请求,管理者除了通知页占有者将页拷贝传给发请求结点外,同时自己也接收一份拷贝,这样,若其它结点再在此页发生读失效,就可以直接从管理者得到页拷贝.

当管理结点把其它结点来的读写请求向前传给 Owner 时,管理者结点都要等待发请求结点的确认消息.由于这两个操作之间的延迟较长,管理者在传递完读写请求后,将此页表项上锁,便立即返回处理其它请求;当管理者接到发请求结点的确认消息后,释放页表项,标志本次服务完成.

1.3 NFDMA 算法与文献[2]中固定分布算法的比较

NFDMA 算法中,所有的共享页在初始化时,都按 Hash 函数平均分到了各个结点上,结点负责管理分给它的页.在大多数情况下,页在其管理结点处都有拷贝,除非有另外的结点对此页作写访问.此外只要有结点对此页作读操作,管理结点又拥有了此页的拷贝.因此,管理者结点就象是它所管理的页的"家".在文献[2]的算法中,页完全分布在各个结点上,管理者只负责跟踪它所管辖的那些页的占有关系,并无 NFDMA 算法中的"家"的概念.

文献[2]的算法,在管理结点上出现一个页读失效需要2个消息:一个发给页占有者,另一个来自页占有者;在非管理结点上出现页读失效需要4个消息:一个给管理者,一个给占有者,一个来自占有者,一个用以确认.对于 NFDMA 算法,在管理结点上出现一个页读失效

也需要两个消息，这与文献[2]的算法一样；但是在非管理结点上出现页读失效时，若管理者有此页的拷贝，则仅需要2个消息：一个发给管理者，一个来自管理者；只有管理者无此页的拷贝时，才需4个消息. 对于写失效操作，也是如此. 两者的比较如表1所示.

表1 两种算法的消息传送次数比较

	文献[1]的算法	NFDMA 算法
管理者读写失效	2	2
非管理者读写失效	4	2 or 4

2 总 结

随着 MPP 计算机的发展，如何便于使用就成了一个迫切的问题，SVM 技术为解决这个问题提供了一个办法. 最近，商用的 SVM 系统开始出现，如 KSR－1、Cray 公司的 T3D 等. 因此，开展这方面的研究是很有意义的. 本文提出的一种新的固定分布数据一致性算法，将数据的状态与一致性算法加以结合，减少了页失效时消息的传送次数，提高了整个系统的性能.

参考文献

1 田新民等.《并行图归约智能工作站》设计报告. 清华大学计算机系技术报告，1992.

2 Li K, Hudak P. Memory coherence in shared virtual memory systems. ACM Trans. Computer Systems, 1989, 7 (4): 321－359.

A NEW MAINTAINING COHERENCE ALGORITHM FOR DISTRIBUTED SHARED MEMORY SYSTEM

Xing Hao Shen Meiming Gao Yaoqing

(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract A shared virtual memory (SVM) system can provide a virtual address space shared by all processors in a loosely－coupled distributed memory multicomputer. It combines the salability of network based architectures with the convenience of shared－memory programming. This paper, firstly discusses some issues for maintaining data consistency in SVM system, then proposes a new fixed distributed manager algorithm. Finally, this paper gives a analysis of NFDMA and compares to Li's scheme.

Key words Shared virtual memory system*, fixed distributed manager algorithm*, data consistency.