多核环境下负载均衡的并行离散事件全局调度机制*

陈莉丽[1,2],姚益平[2],蔡潞[3]
(1. 北京系统工程研究所 复杂系统仿真总体重点实验室,北京 100101;
2. 国防科技大学 计算机学院,湖南 长沙 410073;
3. 长安大学 信息工程学院,陕西 西安 710064)

摘 要:分析了多核环境下传统的离散事件时间弯曲并行系统的性能,针对其事件调度开销小和负载均衡能力强难以兼得的问题,提出了一种基于分布式队列的全局调度机制,设计了相应的数据结构和调度算法,大大减少了锁开销。通过大量实验对多核环境下几种典型离散事件系统并行策略的性能分析表明,本文提出的全局调度策略不仅事件调度开销小,而且回滚率大大降低,有效克服了传统策略回滚量较大或难以实现动态负载平衡的情况,并具备良好的可扩展性。

关键词:并行离散事件仿真; 时间弯曲协议; 并发; 多核; 多线程
中图分类号:TP391.9 **文献标志码**:A **文章编号**:1011-2486(2012)04-0108-06

A global schedule mechanism for PDES on multi-core environments

CHEN Lili[1,2], *YAO Yiping*[2], *CAI Lu*[3]
(1. Science and Technology on Complex System Simulation Laboratory, Beijing 100101, China;
2. College of Computer, National University of Defense Technology, Changsha 410073, China;
3. School of Information Management and Information Systems, Chang'an University, Xi'an 710064, China)

Abstract: The current trend in processor architecture design adopts the integration of multiple cores on a single processor. The tightly integrated processing cores in one chip with communication latencies substantially lower than those present in conventional clusters provide potential performance improvement especially for the fine-grained PDES. Thus, in the PDES domain, one of the research focuses is on modifying software platforms to efficiently utilize the computation resources of multi-core processors. The current dynamic load balancing technologies for PDES cannot reach the twin goals of good balance and low event-scheduling overhead. By taking advantage of multi-core architecture with shared memory address space and low communication, a global schedule mechanism based on a distributed event queue is proposed. Its specially designed data structures and algorithms reduced the cost of lock operations much. In comparison with the distributed event queue local schedule mechanism, the experiment results show that the distributed queue global schedule mechanism can effectively reduce the rollback rate and balance the workloads at a low event scheduling cost for Time Warp system on multi-core platforms.

Key words: PDES; Time Warp; Parallelism; Multi-core; Multi-thread

随着多核计算革命的兴起,通用的多核CPU成为主流的处理器。多核的优势在于其通信开销大大减少,这对分析仿真是一个极大的机遇。并行离散事件仿真(PDES)系统是一种应用广泛的自动并行离散事件系统[1-2]。在大部分PDES实现方式中,仿真对象与某个操作系统进程/线程是绑定的,并且事件的排队和执行在所属进程/线程内部进行,这里称之为分布式队列局部调度策略。这种策略下,除非提供额外的动态迁移机制,否则仿真对象和仿真事件的执行不能在操作系统进程/线程之间移动,这样会导致负载平衡问题。文献[3]指出在前瞻值(lookahead)很小时,无论是保守算法还是乐观算法都难以达到好的并行效果。

解决负载平衡有静态和动态两种方式。静态负载平衡是在仿真开始阶段根据应用特征选用特定的静态划分算法将仿真实体对象分发到各个处理节点上。静态划分的原则除了工作负载均衡外,还要使得节点间通信最小。在单机多核上减少并行节点间通信是为了尽量减少回滚的发生,如将关联紧密的仿真实体对象划分在一个组。虽然已有许多静态负载平衡的相关研究,并且在相关领域起到了重要作用,但其效果受限于应用特征。局部调度下的静态分发策略存在的另一个问

* **收稿日期**:2011-07-07
基金项目:国家自然科学基金资助项目(61170048);
作者简介:陈莉丽(1982—),女,广西桂林人,博士,E-mail:chenlili8209@nudt.edu.cn;
姚益平(1963—),男,教授,博士,博士生导师,E-mail:ypyao@nudt.edu.cn

题是可能出现并行性挖掘不够。例如,根据静态平衡划分算法将某应用的仿真实体对象划分为 6 个对象组,若在 4 核机器上运行,如果启动 6 个线程或进程作为并行节点,则线程或进程切换开销会影响性能,尤其是在事件粒度不大时这种切换开销会导致并行优势的丧失;如果启动 4 个线程或进程作为并行节点,则出现多个组分在同一个并行节点上,由于同一节点内的事件是顺序执行的,所以同一节点上的多个组之间潜在的并行性得不到挖掘。

为了解决上述问题,有很多学者进行了关于动态负载平衡的研究[4-5]。其中大部分策略是在仿真执行过程中动态地进行节点间的仿真实体对象迁移,这种策略需要通过计算或统计等方式确定迁移的时机,并且迁移过程中仿真必须停步,因而存在等待开销,平衡效果依赖于迁移时机生成算法的选择。

全局事件调度机制是实现最佳负载平衡的有效方式,但是全局调度带来的额外开销可能会大到抵消这种负载平衡所带来的好处。多核技术的优势与应用能为并行离散事件仿真领域的多个方面带来机遇,本文主要利用其通信开销剧减的优势,来研究如何降低全局调度的额外开销问题。

1　相关工作

目前,国内外的并行仿真平台在针对多核体系结构进行并行处理优化方面的研究处于初步探索阶段。

多核平台上的离散事件并行调度技术主要有两种。一种技术是采用基于分布式队列局部调度策略的多进程并行方式,进程间通过共享内存通信库或 MPI 通信协议交互。采用这种技术的平台主要有基于 C 语言开发的平台 ROSS-MPI[6] 以及 C++ 语言开发的基于对象建模的平台 musik[7] 和 Warp IV[8]。另一种技术是采用多线程并行方式,调度策略有分布式队列局部调度和集中式队列集中式调度两种。GTW[9]、ROSS-MTH[6] 和 ThreadedWarped[10] 都是多线程并行方式,但采用不同的调度机制。GTW 采用基于分布式队列局部调度策略的多线程并行方式。在多核处理器上,ROSS 可选用多线程、MPI 两种方式并行,其多线程方式采用与多进程方式相同的分布式队列局部调度策略。ThreadedWarped 是对 WARPED[11] 进行多核优化的版本,采用的是集中式队列、管理线程集中调度策略,该策略虽然能达到最佳的负载平衡,但由于多个工作线程需要进行大量竞争全局队列锁的操作,管理线程需占用一个节点资源,还要负责本机上所有工作线程的 GVT 计算和内存回收,随着线程增多、管理线程负载增大,可能造成性能瓶颈。

还有一种基于优先调度的 TW 协议动态负载平衡方案[12-13],这是一种介于分布式队列和集中式队列之间的在仿真过程中动态地切换对象组执行的调度策略,其基本思想是:将模型实例划分为 k 个 LP,$k > m$,m 为线程数目,直接以本地虚拟时间(Local Virtual Time,LVT)作为负载指标,线程从 LP 集合中调度 LVT 较小的 LP 进行执行,执行一段时间后,再调度其他 LP 进行执行。这种方案的特点是 LP 之间是以模型实例为元素的分布式队列,全局调度是以 LP 为元素的集中式队列。该方案需要根据运行环境及应用特点进行 LP 划分,并且需要根据应用特点确定切换的时机,这种方式对事件粒度大且 LP 间关联少的应用有效。

不同于以上技术,本文采用分布式队列全局调度的多线程并行方式,利用全局调度的优势达到负载平衡,通过分布式队列的结构来降低事件调度开销。

2　面向多核的 Time Warp 分布式队列全局调度策略

多核系统在硬件上与以往多处理器和多机系统的一个显著区别在于,各个处理核之间共享主存及最底层 cache。多核编程有多线程和多进程两种方式。对于多线程而言,多个线程之间共享数据地址空间,这使得节点线程间的事件和数据交互几乎不存在额外的通信开销。因此,本文的优化机制采用多线程来实现仿真事件在处理节点间的动态调度。

以组并行模式为例的全局调度分发机制如图 1 所示。在全局调度策略下,所有组(或实体对象、事件)并不从属于某个节点,而是根据各节点的忙闲情况动态调度的。

这里针对多核多线程 cache 失效和锁开销较大的特点,设计了满足数据局部性的基于分布式内存管理的分布式事件链表。图 2 是以两线程节点为例的分布式队列全局调度示意图,每个线程节点各自维持一个称为 nel 的线程专有事件优先级队列,各个线程的执行流程相同,不存在服务管理线程。nel 由按时戳排序的事件组成,包含以下三个指针:head 指针指向时间戳最小的事件,middle 指针指向调度器下一调度周期要检询的事件(即局部最小时戳未处理事件),tail 指针指向

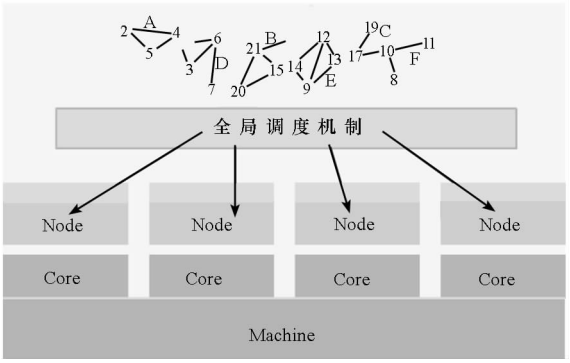

图 1　全局调度机制

Fig. 1　Global scheduling

时间戳最大的事件。所以 head 和 middle 指针(不含 middle 指针指向的事件)之间的队列部分相当于已处理事件队列,middle 和 tail 指针之间的队列部分相当于未来事件列表。

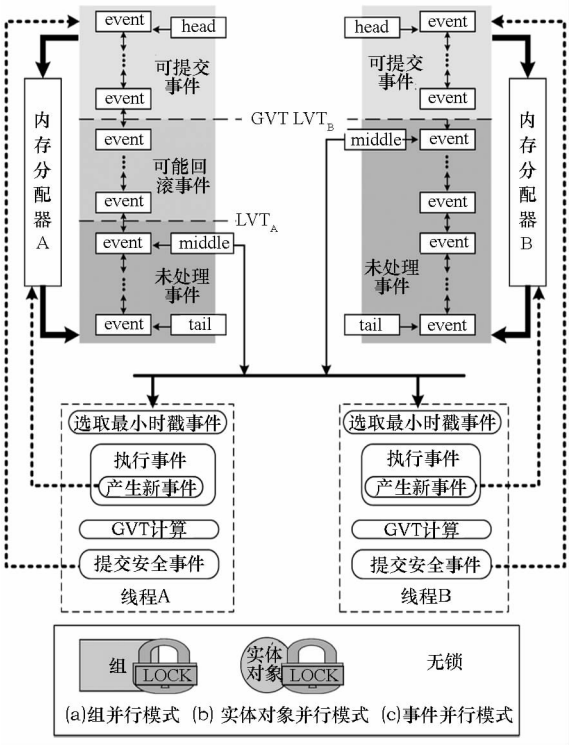

图 2　以两节点为例的分布式队列全局调度示意图

Fig. 2　Data structures and processing flow of DQ-GS mechanism for two thread nodes

下面先从事件的视图来介绍为了避免锁操作以及减少调度开销所采取的优化措施,然后给出了基于这些优化措施的全局虚拟时间(Global Virtual Time, GVT)计算算法。

(1) 事件的创建、回收

当一个线程节点在运行过程中要产生一个新事件时,该线程将从自己私有的内存分配器中分配一块内存给新事件,然后将该新事件加入到自己本地的 nel 中,这里将该线程称为该事件的所属线程。事件的执行可能由其他线程实施,但事件的删除及内存回收只能由其所属线程进行。如果需要撤销一个事件,只是将事件的无效标志位 Invalid 置为真,而不是真正释放事件内存。在整个运行过程中,事件的内存只在提交阶段释放,计算 GVT 后,由各节点提交各自事件链表中时间戳小于 GVT 的事件。因此,除了新事件时戳小于 middle 指针事件时戳时要加锁修改 middle 指针外,其他操作都无需加解锁操作。

(2) 全局调度

每次从所有节点的局部最小时戳未处理事件中选取一个全局最小时戳事件执行。被选取的事件并不从它的节点事件链表中删除,而是将事件的节点事件链表标志位 *nel_in_tag* 置为假,节点事件链表 middle 指针后挪,指向下一个待处理事件,这样可以避免事件被调度时从未处理队列删除的操作和事件执行完后插入提交队列的操作,从而减少全局锁占用时间。

(3) 事件的执行

如果采用的是组并行模式或实体对象并行模式,则在事件执行前尝试获取事件所属的组或实体对象的锁,如果成功获取锁则执行事件,否则将事件加入该组或实体对象的准备好事件队列 *ready_el* 中然后返回取下一个全局事件。放入 *ready_el* 中的事件将由占有该组或实体对象锁的线程节点执行完当前事件后再调度执行,待所有 *ready_el* 中的事件执行完后,线程节点释放锁返回到全局调度过程。

(4) 事件的回滚

上述的组并行模式或实体对象并行模式下每个组或实体对象都维持一个已处理事件链表以便回滚发生时访问。考虑到减少内存分配和回收操作,每个事件在整个系统中只有一份内存(一个内存对象),通过两对指针来记录事件在节点事件链表和已处理事件链表的位置。根据前面的设计思想,事件是在遍历节点事件链表时提交的,为了避免频繁地对事件所属组或实体对象的已处理事件链表的加解锁操作,事件在提交时并不将其从所属组或实体对象的已处理事件链表中删除。为了避免回滚已处理事件链表中的事件时访问到被节点事件链表释放了内存的事件,为每个事件添加一个 *pprev_ts* 字段记录该事件在已处理事件链表中的前一个事件的时间戳,只有当 *pprev_ts* 满足回滚条件时才访问前一事件,由于只有时间戳小于 GVT 的事件才会被释放,所以需要回滚的事件绝不会被释放,从而保证了程序的正确性,并

且避免了对链表的加解锁操作。

(5) 异步 GVT 算法

以事件为调度单位的全局调度方法相对于以实体对象为调度单位的全局调度方法的另一个好处是计算 GVT 时,在调度全局最小时戳事件(含回滚事件)时,将此全局最小时戳记录下来,然后由各节点报告各自正在运行的事件(包括执行的 *ready_el* 中的事件)产生的最小消息时戳值作为该节点的运行时 LVT(*runtime_LVT*)即可。异步方式计算 GVT 的步骤如算法 1 所示。其中全局变量 *wait_syn_flag* 是发起同步的标志位,全局变量 *pre_glbts* 记录发起同步时从所有线程事件队列选取的全局最小未来事件时间戳,全局变量 *syn_count* 作为同步计数器,各线程设置同步标志位 *syn_flag* 和提交标志位 *commit_flag*。线程节点的私有变量 *runtime_LVT* 记录该线程节点在执行事件过程中产生的新事件(消息)的最小时戳。

算法 1　分布式队列全局调度策略下的 GVT 计算

input: *GVT_BATCH*//控制 *GVT* 计算的频率
　　NUM_THREAD//并行节点数目
output: *GVT*//全局虚拟时间

```
LOCK(g_lock)
if th_para[my_id].syn_flag = true then
  th_para[my_id].syn_flag = false
  syn_count + +
  pre_glbts.reduce_to(runtime_LVT)
  if syn_count = NUM_THREADS then
    GVT ← pre_glbts;  syn_count ← 0;
    wait_syn_flag = false
    for u = 0 to NUM_THREADS - 1 do
      th_para[u].commit_flag = true
global_min_th_ts←MIN{ min_th_ts_1, min_th_ts_2, ⋯, min_th_ts_n }
sel_thread_ID←{ i | min_th_ts_i = global_min_th_ts }
GVTIntervalCounter + +
if GVTIntervalCounter ≥ GVT_BATCH
and wait_syn_flag = false then
  wait_syn_flag = true; GVTIntervalCounter ← 0
  pre_glbts ← global_min_th_ts
  for u = 0 to NUM_THREADS - 1 do
    th_para[u].syn_flag = true
LOCK(tpq[sel_thread_ID].middle_lock)
SelEvent ← tpq[sel_thread_ID].nel.peek_middle()
if SelEvent ≠ NULL then
  SelEvent - > nel_in_tag = false
  tpq[sel_thread_ID].nel.shiftdown_middle()
  min_th_ts_sel_thread_ID ←tpq[sel_thread_ID].nel.get_middle_ts()
  UNLOCK(g_lock)
  UNLOCK(tpq[sel_thread_ID].middle_lock)
  runtime_LVT←step _advance_optimistically(SelEvent)
else
  UNLOCK(g_lock)
  UNLOCK(tpq[sel_thread_ID].middle_lock)
  runtime_LVT←MAX_TIME
```

3　性能测试

3.1　实验环境及测试方案

测试平台为两路四核 2.53GHz Xeon Processors E5540,内存为 8GB,操作系统为 Kylin Server 3.1,GCC 版本为 4.1.2,锁方式采用 phread 旋转锁。测试用例采用并行离散事件仿真领域经典的 phold 测试程序[14]。

Musik 是由 Kalyan S. Perumalla 用面向对象语言开发的并行/分布仿真微内核,同一计算机上的多个进程通过设置的共享内存通信。本节在该 Musik 平台的基础上,将其改为同一计算机上为多线程执行,实现了分布式队列全局调度策略,将基于此优化机制的平台称为 G-MTH。同时为了方便与分布式队列局部调度策略做对比,还实现基于 Musik 原有局部调度策略的多线程版本,称为 D-MTH。实验同时与伦斯勒理工学院 Christopher D. Carothers 开发的以高事件率著称的基于 C 语言的高性能 Time Wrap 系统 ROSS 的多线程运行方式 ROSS-MTH 做了比较,这里采用的是 ROSS 的 2010 年更新的版本。

3.2　性能测试与分析

本节通过四个参数配置方案考察了多核环境下几种典型离散事件系统并行策略的性能以及影响性能的主要因素。

(1)性能随着 CPU 核数目增加的变化

参数设置为事件除生成随机目的地和时间戳外无其它操作,回滚事件除引擎开销外无其它工作负载,实体对象数目为节点数目的 4 倍,每个实体对象初始化 1 个事件,新事件的目的节点随机生成(局部率 = 1/节点数目),新事件与父事件的时戳差 $\triangle t$ 服从均值为 1 的指数分布(其中前瞻值 lookahead = 0)。从图 3 结果中可看出,全局调度方式随着 CPU 核数目的增加,单个事件的调度开销变化不大,体现了良好的可扩展性。

(2)性能随着事件粒度大小的变化

该测试核数目分别设置为 4 核和 8 核,事件粒度随横坐标变化,其它参数设置同前。从图 4

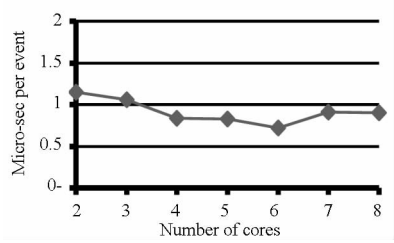

图 3 性能随着 CPU 核数目增加的变化

Fig. 3 Performance of G-MTH on the changing of cores

可看出,当事件粒度等同于约 120 个浮点运算指令时,4 核的性能开始超过串行性能;事件粒度等同于约 150 个浮点运算指令时,8 核的性能开始超过串行性能;到 520 个浮点运算指令时,8 核的性能开始超过 4 核的性能。并行相对串行的加速比随着事件粒度的增加趋向于等于核数目。

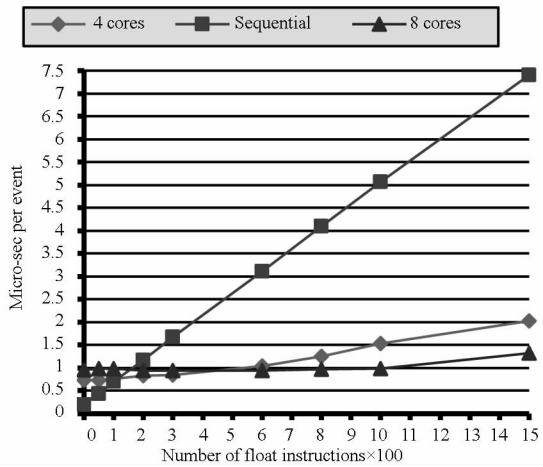

图 4 G-MTH 性能随事件粒度大小的变化

Fig. 4 Performance variation of G-MTH on the changing of event granularity

(3)回滚率以及性能随事件局部率(locality)的变化

组并行模式或实体对象并行模式下,组或实体对象内部的事件调度称为局部事件调度,如果是调度非本组或实体对象的事件则称为远程事件调度。事件局部率等于局部事件调度次数对总事件调度次数的比率,远程率等于远程事件调度次数对总事件调度次数的比率,局部率与远程率是互补的,两者相加等于 1。这里 phold 测试用例采用的并行模式是将实体对象等分为 NumGroup = 4 × nPE 份,nPE 等于并行节点的数目。Musik 和 ROSS 的 phold 测试用例中根据局部率计算目的仿真实体 ID 的算法采用文献[15]中的算法,测试的统计输出结果表明这种算法参数输入的局部率与实际的局部率不一致,经分析该算法的实际局部率($Locality_{actual}$)与参数输入的局部率($Locality_{input}$)满足以下关系:

$$Locality_{actual} = Locality_{input} + \frac{1 - Locality_{input}}{NumGroup}$$

所以本测试修正了该算法,修正后的算法如算法 2 所示。

算法 2 *PHOLD* 模型中根据设定的远程率选择目的对象

Input: *REMOTE_RATE*//远程率

Output: *DestLP*//选定的目标 *LP*

```
1: NLP⇐Total LPs in simulation
2: NLPperGroup⇐(NLP - 1) ÷ NumGroup + 1
3: OffsetLpid = (MyGroupNo + 1) × NLPperGroup
4: DestLp⇐ - 1
5: if RANDOM_DOUBLE(0..1)≤REMOTE_RATE then
6:     DestLp⇐RANDOM_INT(0,NLP - NLPperGroup)
7:     DestLp = DestLp + OffsetLpid
8:     if DestLp≥NLP then
9:         DestLp = DestLp - NLP
10: else
11:    DestLp⇐CurrentLp
```

图 5 的参数设置为核数目等于 4,事件局部率随横坐标变化,其他参数同测试(1)。从图5

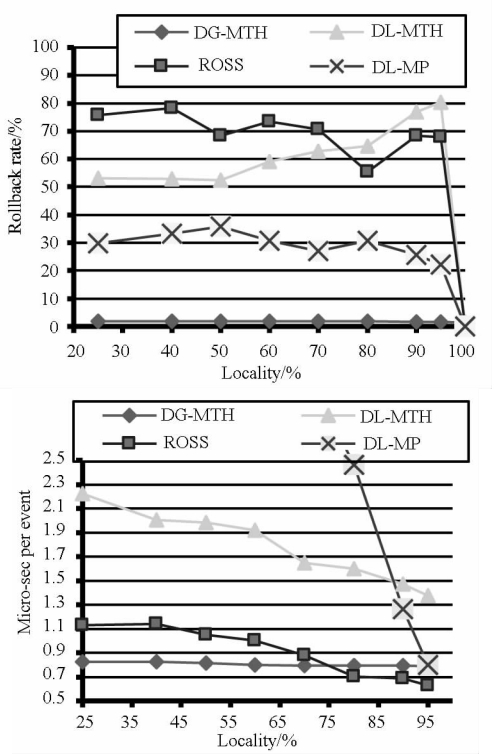

图 5 回滚率以及性能随 locality 的变化

Fig. 5 Rollback rates & performance variations on the changing of "locality"

可知,多线程局部调度方式 D-MTH 和 ROSS 的回滚率比多进程实现方式要高,并且在局部率超过 80% 时,回滚率都表现出与多进程并行所不同的上升趋势,只是在局部率为 100% 时骤减为 0,这

说明多线程局部调度在增加事件吞吐率的同时却增加了回滚率,而多线程全局调度方式G-MTH表现出极低的回滚率和良好的稳定性。

(4)回滚率随前瞻值和仿真事件密度的变化

图 6 的参数设置为核数目等于 4,前瞻值随横坐标变化,其他参数同测试(1)。从图 6 可知,全局调度方式相对于局部调度的回滚率和事件调度开销大大降低。

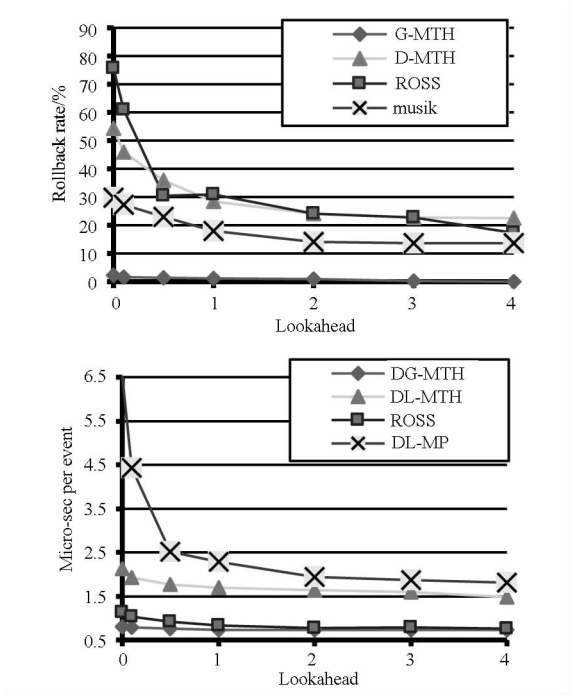

图 6　回滚率及性能随前瞻值的变化

Fig. 6　Rollback rates & performance variations on the changing of lookahead

4　结　论

本文对基于 Time Warp 的并行离散事件仿真系统在多核平台上的运行机制和性能进行了深入研究,提出了多核平台下的并行离散事件系统分布式队列全局调度算法。测试结果表明,该算法能有效降低回滚率且随着事件粒度增大性能明显优于以往算法。下一步工作将考虑引入无锁算法以及相关 cache 失效解决技术,以提高引擎性能,使其能够适用于更小粒度的并行系统。

参考文献(References)

[1] Fujimoto R M. Parallel and distributed simulation systems [M]. New York: John Wiley & Sons Inc, 2000.

[2] Jefferson D R. Virtual time [J]. ACM Transactions on Programming Languages and Systems, 1985,7(3):404 - 425.

[3] Carothers C,Perumalla K S. On deciding between conservative and optimistic approaches on massively parallel platforms [C]// Proceeding of the 2010 Winter Simulation Conference, Monterey,CA:Winter Simulation Conference,2010:678 - 687.

[4] Meraji S, Zhang W, Tropper C. A multi-state q-learning approach for the dynamic load balancing of time warp [C]// Proceedings of the 24th Workshop on Parallel and Distributed Simulation, Washington, DC, USA: IEEE Computer Society, 2010:1 - 8.

[5] Peschlow P, Honecker T, Martini P. A flexible dynamic partitioning algorithm for optimistic distributed simulation[C]// Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS 2007), San Diego, CA: IEEE Computer Society, 2007: 219 - 228.

[6] Carothers C, Bauer D, Pearce S. ROSS: a high-performance, low memory, modular time warp system[C]// Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS 2000), Washington, DC: IEEE Computer Society, 2000:53 - 60.

[7] Perumalla K S. μsik-A micro-kernel for parallel/distributed simulation systems [C]//Proceedings of the 19th Workshop on Parallel and Distributed Simulation (PADS 2005). Washington, DC: IEEE Computer Society, 2005: 59 - 68.

[8] Steinman J. Introduction to parallel and distributed force modeling and simulation[C] //Proceedings of the Spring 2009 Simulation Interoperability Workshop, 09S-SIW-041, 2009.

[9] Das S, Fujimoto R M, Panesar K, et al. GTW: a time warp system for shared memory multiprocessors [C]// Proceedings of the 26th Conference on Winter Simulation (WSC94), San Diego, CA: The Society for Modeling and Simulation International (SCS), 1994:1332 - 1339.

[10] Miller R J. Optimistic parallel discrete event simulation on a beowulf cluster of multi-core machines [D]. Cincinnati: University of Cincinnati, 2010.

[11] Martin D E, McBrayer T J, Wilsey P A. WARPED: a time warp simulation kernel for analysis and application development [C]//Proceedings of the 29th Hawaii International Conference on System Sciences, Washington, DC: IEEE Computer Society, 1996: 383 - 386.

[12] 苏年乐. 仿真模型可移植性规范的多核并行化研究[D]. 长沙:国防科学技术大学, 2010.
SU Nianle. Parallelization of simulation model portability specification on multi-core computer[D]. Changsha: Nation University of Defense Technology, 2003. (in Chinese)

[13] 苏年乐,李群,王维平,等. 基于多核平台的乐观并行离散事件仿真[J]. 系统仿真学报,2010, 22(4).
SU Nianle, LI Qun,WANG Weiping,et al. Optimistic parallel discrete event simulation based on multi-core platform[J]. Journal of System Simulation, 2010, 22(4). (in Chinese)

[14] Fujimoto R M. Performanc of time warp under synthetic workloads [C] // Proceedings of 1990 SCS Multiconference on Distributed Simulation. San Diego, CA: SCS, 1990:23 - 28.

[15] Bauer D W, Carothers C D, Holder A. Scalable time warp on blue gene supercomputers [C]//Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation, Washington, DC, USA: IEEE, 2009:35 - 44.