ISSN 1000-1239/CN 11-1777/TP
48(9): 1589-1596, 2011

大规模设备协同机制研究

荣晓慧 陈 峰 邓 攀 马世龙
(软件开发环境国家重点实验室(北京航空航天大学) 北京 100191)
(rongxh@nlsde.buaa.edu.cn)

A Large-scale Device Collaboration Mechanism

Rong Xiaohui, Chen Feng, Deng Pan, and Ma Shilong
(*State Key Laboratory of Software Development Environment* (*Beihang University*), *Beijing* 100191)

Abstract In the field of "Internet of things", area management and emergency rescue, the requirement of large-scale device collaboration is growing. Aiming at the large-scale and strict timing constraint of the large-scale device collaboration systems, in this paper, a two-level task model for large-scale device collaboration is presented. The task model is made up of collaboration task and device task called collaboration subtask. Based on this task model, a large-scale device collaboration mechanism described with Pi-calculus is given, which consists of task-level collaboration mechanism and subtask-level collaboration mechanism. In the task-level collaboration mechanism, aiming at the exclusivity of device resources, resource reservation is adopted to avoid the device access conflict among collaboration tasks. The subtask-level collaboration mechanism ensures the timing constraints among collaboration tasks and collaboration subtasks. It includes three parts: collaboration mechanism between task and subtask, collaboration mechanism among subtasks and collaboration mechanism based on time. Then the large-scale device collaboration mechanism's accuracy is proven theoretically. Finally, a large-scale device collaboration prototype system is designed and implemented. The results of simulation experiments on the prototype system show that the large-scale device collaboration mechanism can satisfy the performance requirements of large-scale and strict timing constraint in the large-scale device collaboration.

Key words large-scale device collaboration; collaboration task model; collaboration mechanism; timing constraint; Pi-calculus

摘 要 在物联网、区域管理和应急救援等领域,对大规模设备协同的需求越来越高.针对大规模设备协同中协同规模大和时序约束严格这两个特点,定义了包含协同任务和协同子任务的大规模设备协同两级任务模型,在此模型基础上利用Pi-演算给出了大规模设备协同中两级任务协同机制的描述,并从理论上证明了该机制的正确性.并实现了大规模设备协同原型系统,通过在原型系统上进行模拟实验,验证了该协同机制能够满足设备协同中的大规模性和严格的时序约束.

关键词 大规模设备协同;协同任务模型;协同机制;时序约束;Pi-演算

中图法分类号 TP393

近年来,通过IP技术实现了设备资源的共享和远程访问,极大地提高了很多国民经济重要行业中设备资源使用效率.但是,在区域管理、科学研究和应急救援等行业,不仅要实现设备的远程访问,更多

收稿日期:2009-12-26;**修回日期**:2010-09-25
基金项目:国家科技支撑计划基金项目(2006BAK12B11);软件开发环境国家重点实验室探索性自主研究课题(SKLSDE-2010ZX-06)

的是要求许多设备之间的协同工作,从而更好地满足这些行业的应用需求.例如,本文作者及其研究组参与的奥运中心区景观照明 IPv6 数字化网络控制系统项目中,对照明设备的控制不仅仅是简单的开灯关灯操作,而是利用大量照明设备的协同工作,产生具有艺术性的动态照明效果.本文称这类应用需求为大规模设备协同问题.大规模设备协同具有如下两个重要的特点:

1) 协同规模大.大规模设备协同的大规模性表现在两个方面:一方面,一次协同任务的执行需要部署在很大范围内的大量设备共同完成;另一方面,在大规模设备协同系统中存在大量的并行协同任务和设备资源访问任务.因此,如何保证在如此大的范围内对数量如此之多的设备进行协同工作,如何保证在大量并行协同任务的执行过程中无冲突地并发访问设备资源,这些是大规模设备协同机制研究和实现中的重大挑战和瓶颈问题.

2) 时序约束严格.在大规模设备协同中,由于存在大量的并行协同任务,除了要保证这些任务之间能够有效地共享各类资源,防止冲突和死锁,更重要的是保证这些协同任务之间以及任务与子任务之间的时序约束关系,因为任务间的时序性会直接影响设备协同的效果.因此,在大规模设备协同中对协同任务的时序约束要求非常严格.

根据上述分析,大规模设备协同机制研究是大规模设备协同研究的关键问题.因此本文针对大规模设备协同系统中任务多的特点,设计了两层设备协同任务模型,在此基础上提出了能够支持大规模、时序约束的协同机制,从理论上证明了该机制的正确性,通过在大规模设备协同原型系统上进行模拟实验验证了该机制的性能.

1 相关工作

设备系统的研究主要分为设备远程访问、支持用户协作的设备共享和设备间协同 3 类,下面介绍这 3 类研究的特点及其典型项目.

1) 设备远程访问:主要致力于设备远程共享,集成大量分布异构设备,方便用户使用,但是没有考虑协同问题.其代表性研究有美国能源部下一代因特网试验床计划资助、印第安纳州大学等多所高校共同完成的 XPort 项目[1],实现了对几台昂贵 X 射线结晶设备的远程访问以及这些仪器的使用规划、仪器操作、数据获取、筛选和分析等功能;美国国家自然科学基金资助的 CIMA(common instrument middleware architecture)项目[2],该项目通过普通仪器中间件对各类传感器提供统一封装,开发了一种适应于传感器数据传输的通信协议,利用传感器、视频服务器和计算机软件对实验仪器设备进行远程控制和数据访问.

2) 支持用户协作的设备共享:实现设备资源的对象化封装,用户通过操作物理设备对象进行协作,但是不支持动态流程建模和设备间的协同工作.其代表性研究有美国国家自然科学基金资助 NEESgrid[3] 项目,支持远程的共享实验设备和数据,并且为建模和仿真提供了合作空间,使得地震研究人员可以通过资源和设备的共享以及团队的协作计划、执行并发布他们的实验.该项目使得科学家们能够在实验进行时协同完成计划实验和进行远程观测.

3) 设备间协同:利用建模工具实现设备协同流程的定义,通过协同机制实现设备间协同流程的自动化.其代表性研究有欧盟委员会资助的 GRIDCC[4] 项目,基于网格中间件 globus 建立了一套地理上的分布式系统,能够远程控制和监测大量工作在不同环境下的复杂仪器,包括从地球物理台站监测地球状况所使用的传感器、欧洲电网的小发电机以及高能物理实验设备等,并采用了工作流技术实现了业务流程的建模以及设备协同.

在上述研究中,设备远程访问和支持用户协作的设备共享研究技术比较成熟,而设备间协同方面的研究相对较少.大规模设备协同是第 3 类研究中的挑战,但是,目前第 3 类研究有关设备间协同的项目中,一次设备协同流程中使用的设备数量不多,所以,由于对大规模性、同步性等性能要求高,目前有关大规模设备协同的技术研究和工程实践还很少.

2 Pi-演算简介

Pi-演算是 Milner 提出的以进程间的移动通信为研究重点的并发理论[5].目前 Pi-演算作为建模工具在计算机软件系统各研究方向上已有相关应用[6-7].Pi-演算可由 3 方面定义,分别为语法定义、操作语义和结构等价规则.

Pi-演算的语法定义:

$P::=0|\bar{y}\langle x\rangle.P|y(x).P|P+Q|P|Q|(\gamma x)P|[x=y]P|!P|A(y_1,y_2,\cdots,y_n)$.

Pi-演算的操作语义:

$$\text{PREFIX}:\frac{}{\alpha.P\xrightarrow{\alpha}P};$$

$$\text{SUM}:\frac{P\xrightarrow{\alpha}P'}{P+Q\xrightarrow{\alpha}P'};$$

$$\text{PAR}:\frac{P\xrightarrow{\alpha}P' bn(\alpha)\cap fn(Q)=\varnothing}{P|Q\xrightarrow{\alpha}P'|Q};$$

$$\text{COM}:\frac{P\xrightarrow{a\langle x\rangle}P' Q\xrightarrow{\bar{a}\langle u\rangle}Q'}{P|Q\xrightarrow{\tau}P'\{u/x\}|Q'};$$

$$\text{MATCH}:\frac{P\xrightarrow{\alpha}P'}{\text{if } x=xP\xrightarrow{\alpha}P'};$$

$$\text{RES}:\frac{P\xrightarrow{\alpha}P' x\notin\alpha}{(\forall x)P\xrightarrow{\alpha}(\forall x)P'};$$

$$\text{OPEN}:\frac{P\xrightarrow{\bar{a}\langle x\rangle}P' x\notin\alpha}{(\forall x)P\xrightarrow{\bar{a}\forall x}P'}.$$

Pi-演算的结构等价规则:

$P\mid Q\equiv Q\mid P,(P\mid Q)\mid R\equiv P\mid(Q\mid R)$,

$P+Q\equiv Q+P,(P+Q)+R\equiv P+(Q+R)$,

$(x)0\equiv 0,(x)(y)P\equiv(y)(x)P$,

$((x)P)\mid Q\equiv(x)(P\mid Q)$ if $x\notin fn(Q)$,

$x(y)P\equiv x(z)(\{z/y\}P)$ if $z\notin fn(P)$,

$(y)P\equiv(z)(\{z/y\}P)$ if $z\notin fn(P)$.

3 大规模设备协同任务模型

我们首先提出一种大规模设备协同任务模型,在此基础上研究大规模设备协同机制.将协同任务分为两级:协同任务和设备任务.协同任务由用户提交协同程序执行产生;设备任务即设备访问和控制,在协同任务执行过程中由协同任务触发产生.下面给出这两种任务的形式化定义:

定义 1. 协同任务.

设 $APro$ 是一个协同程序[8],对于给定 $APro$ 的输入 t_0,$APro$ 从时刻 t_0 开始的一次执行称为一个协同任务.

一个协同任务 $ATask$ 可以表示为 $ATask=(t_0,APro)$,其中 t_0 称为协同任务 $ATask$ 的任务初始时间(或提交时间),在 $ATask$ 执行过程中,$t\geqslant t_0$ 称为协同任务 $ATask$ 的任务当前时间,$APro$ 称为 $ATask$ 的任务脚本.

定义 2. 设备任务.

设 $SPro$ 是一个设备访问程序,$SPro::=sf$(其中,sf 为设备访问表达式),

$STask$ 是一个设备任务,$STask=(t_1,SPro)$(从时刻 t_1 设备访问表达式的执行).

一个设备任务是一个区域 a 的协同任务 $ATask$ 的子任务,如果该设备任务中涉及的设备 $\{s_{i1},\cdots,s_{im}\}\subseteq\{s_1,\cdots,s_n\}=a$,$n\geqslant 1$.

在下面大规模设备协同机制的介绍中,协同任务简称为任务,设备任务简称为子任务.

4 大规模设备协同机制

为满足大规模设备协同中规模大和时序约束严格的特点,大规模设备协同机制分为多协同任务间的协同控制(简称任务级协同机制)和协同任务中设备任务的协同控制(简称子任务级协同机制).下面使用 Pi-演算形式化描述这两级的协同机制,并证明该机制中协同流程的正确性.

4.1 任务级协同机制

任务级协同机制如图 1 所示,针对设备资源具有独占性的特点,当大规模设备协同系统中存在大量的并行协同任务时,采用资源预约机制来防止协同任务之间的设备访问冲突问题.

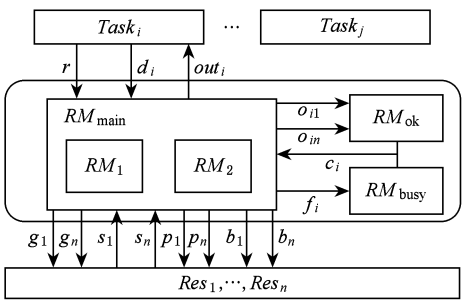


Fig. 1 Collaborative mechanism between tasks.

图 1 任务级协同机制

1) 任务级协同机制

$TCM=Task_i\mid\cdots\mid Task_j\mid ResMgr\mid Res_1\mid\cdots\mid Res_n$,

其中 r,out,d 为协同任务进程和设备调度进程之间的通道,在大规模设备协同系统中为上层任务预约设备和释放设备使用;o,c,f 为设备调度进程的内部通道,在系统中为设备调度决策使用;g,s,p,b 为设备调度进程和设备适配进程之间的通道,在系统中为与设备进行通信使用.其传递消息的含义如下.

① $Request$:设备请求消息,由协同任务通过 r(设备申请通道)发送给设备调度器;

② $Result$:设备分配结果消息(succ/fail),由设备

调度器通过 *out*(设备分配通道)返回给协同任务;

③ *Done*:设备使用结束消息,由协同任务通过 *d*(设备使用结束通道)发送给设备调度器;

④ *GetRes*:设备预约消息,由设备调度器通过 *g*(设备预约通道)发送给设备适配器;

⑤ *Msg*:设备状态消息(ok/busy),由设备适配器通过 *s*(设备状态通道)返回给设备调度器;

⑥ *Fmsg*:设备预约仲裁结果消息(succ/fail),由设备调度器中的仲裁器(进程 *RMok* 和 *RMbusy*)通过 *c*(仲裁结果通道)发送给设备调度主管理器(进程 RM_{main}).

另外,*o* 为肯定投票通道,*f* 为否定投票通道,*p* 为设备预约结果通道(succ/fail),*b* 为设备释放通道.

2) 协同任务进程

$Task_i = \bar{r}\langle Request_i\rangle.\ out_i(Result).([Result = \text{succ}]\tau.\overline{d_i}\langle Done\rangle.0 + [Result = \text{fail}]0)$,

假设 $Task_i$ 申请的设备为$(Res_m, \cdots, Res_n)$.

3) 设备调度进程

$ResMgr = (\gamma o_{i1}, \cdots, o_{in}, b, c_i)\, RM_{\text{main}} \mid RM_{\text{ok}} \mid RM_{\text{busy}}$,

其中:$RM_{\text{main}} = RM_1 \mid RM_2$;

$RM_1 = r(Request_i).(\overline{g_m}\langle GetRes\rangle.\ s_m(Msg_m).([Msg_m = \text{ok}]\overline{o_{in}} + [Msg_m = \text{busy}]\overline{f_i})) \mid \cdots \mid (\overline{g_n}\langle GetRes\rangle.\ s_n(Msg_n).([Msg_n = \text{ok}]\overline{o_{in}} + [Msg_n = \text{busy}]\overline{f_i}))$;

$RM_2 = c_i(Fmsg).([Fmsg = \text{succ}](\overline{out_i}\langle\text{succ}\rangle \mid \overline{p_m}\langle\text{succ}\rangle \mid \cdots \mid \overline{p_n}\langle\text{succ}\rangle).d_i.(\overline{b_m}\langle\text{bye}\rangle \mid \cdots \mid \overline{b_n}\langle\text{bye}\rangle) + [Fmsg = \text{fail}](\overline{out_i}\langle\text{fail}\rangle \mid \overline{p_m}\langle\text{fail}\rangle \mid \cdots \mid \overline{p_n}\langle\text{fail}\rangle))$;

$RM_{\text{ok}} = o_m \cdots o_n.\bar{c}\langle\text{succ}\rangle$;

$RM_{\text{busy}} = f.\bar{c}\langle\text{fail}\rangle$.

4) 设备适配进程

$Res_j = (g_j(GetRes).\tau.(\overline{s_j}\langle\text{ok}\rangle + \overline{s_j}\langle\text{busy}\rangle).(p_j(\text{succ}).\tau_{Work}.b_j(\text{bye}).\tau_{ReleaseDevice} + p_j(\text{fail}).\tau_{Cancel}))$.

4.2 子任务级协同机制

子任务级协同机制包括任务与子任务之间的协同机制、子任务之间的协同机制和基于时间的子任务协同机制.其中,任务与子任务之间的协同机制是当一个协同任务启动多个设备任务后,当所有设备任务都执行完成协同任务才能够继续,如图 2 所示;子任务之间的协同机制采用经典的信号量机制保证设备任务之间的同步,如图 3 所示;基于时间的子任务协同机制分为基于绝对时间协同和基于相对时间协同两种方式,但其基本原理一致,所以放在一起进行描述,如图 4 所示.下面分别描述子任务级的 3 种协同机制:

1) 任务与子任务间的协同机制

$ASCM = ATask \mid STask_1 \mid \cdots \mid STask_n$.

图 2 中 *s*,*wc* 为任务进程和子任务进程之间的通道,在系统中为协同任务和子任务通信使用;*e* 为异常处理进程和任务进程之间的通道,在系统中为任务异常处理使用.其传递消息的含义如下.

① *sTaskId*:子任务启动消息,由协同任务通过 *s*(子任务启动通道)发送给子任务,启动该子任务的执行;

② *cEnd*:子任务结束消息,由子任务通过 *wc*(子任务结束通道)发送给协同任务,通知协同任务其执行结束;

③ *abort*:子任务终止消息,由外部异常处理程序通过 *e*(子任务终止通道)发送给协同任务,通知协同任务某子任务执行被终止.

任务进程:

$ATask = (\overline{s_1}\langle sTaskId_1\rangle \mid \cdots \mid \overline{s_n}\langle sTaskId_n\rangle).(wc_1(cEnd_1) + Abort_1).\cdots.(wc_n(cEnd_n) + Abort_n).ATask'$,

其中,$Abort_i = \tau.e_i(abort)$.

子任务 *i* 进程:

$STask_i = s_i(sTaskId).\tau.\overline{wc_i}\langle cEnd_i\rangle$.

2) 子任务间协同机制

$SCM = STask_1 \mid \cdots \mid STask_n \mid SP$,

其中 *a*,*o*1,*r*,*w*,*s*,*o*2 为子任务进程和信号量管理进程间的通道,在系统中为子任务间的同步通信使用.其传递消息的含义如下.

① *semaphore*1:互斥信号量,由子任务通过 *a*(互斥信号量请求通道)发送给信号量管理器,请求占用该信号量;并通过 *r*(互斥信号量释放通道)通知信号量管理器该信号量使用完成;

② *semaphore*2:同步信号量,由子任务通过 *w*(同步信号量等待通道)发送给信号量管理器,等待

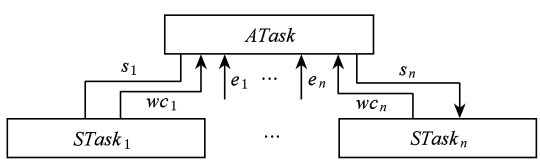


Fig. 2 Collaborative mechanism between task and subtasks.

图 2 任务与子任务间的协同机制

其他进程产生同步信号量;由子任务通过 s(同步信号量通知通道)发送给信号量管理器,产生同步信号量;

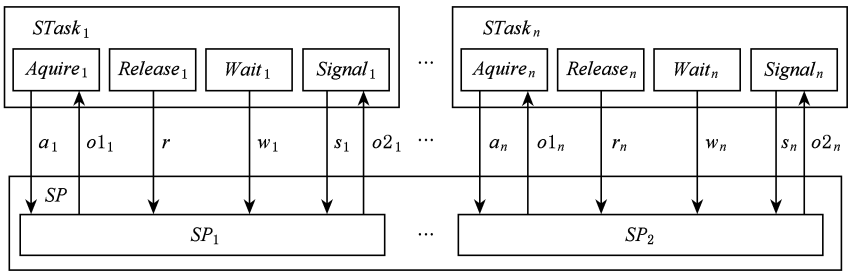


Fig. 3 Collaborative mechanism between subtasks.

图 3 子任务间的协同机制

另外,$o1$ 是互斥信号量通知通道,$o2$ 是同步信号量通知通道.

子任务进程:

$STask_i=(Aquire_i.\tau.Release_i+Wait_i+Signal_i).STask_i$,

其中,$Aquire_i=\bar{a}_i\langle semaphore1\rangle.o1_i$;

$Release_i=\bar{r}\langle semaphore1\rangle$;

$Wait_i=\bar{w}_i\langle semaphore2\rangle.o2_i(semaphore2)$;

$Signal_i=\bar{s}_i\langle semaphore2\rangle$.

信号量管理进程:

$SP=SP_1|\cdots|SP_n$;

$SP_i=a_i(semaphore1).(\overline{o1_i}.SP_i+r(semaphore1).\overline{o1_i}.SP_i)+w_i(semaphore2).\cdots.w_j(semaphore2).s_k(semaphore2).\overline{o2_i}\langle semaphore2\rangle.\cdots.\overline{o2_j}\langle semaphore2\rangle.SP_i$,

其中,k 不属于$(i,\cdots,j)$.

3) 基于时间的子任务协同机制

$$TSCM=STask|TP.$$

图 4 中 t,m,c,为子任务进程和时间进程之间的通道,在系统中为子任务的时间管理使用.其传递消息的含义如下.

① $Time$:子任务通过 t(相对时间通道)发送给时间管理器其继续执行的相对时间;

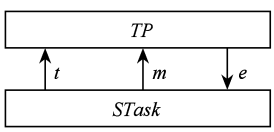


Fig. 4 Collaborative mechanism based on time between subtasks.

图 4 基于时间的子任务协同机制

② $Moment$:子任务通过 m(绝对时间通道)发送给时间管理器其继续执行的绝对时间;

另外,e 为子任务执行启动通道.

子任务进程:

$STask=(\bar{t}\langle Time\rangle+\bar{m}\langle Moment\rangle).e.\tau.0$.

时间进程:

$TP=(t(Time)+m(Moment)).\tau.\bar{e}.TP'$.

4.3 大规模设备协同机制验证

首先证明任务级协同机制 TCM 中的每个任务都可终止,即没有任务会死锁.

定理 1. TCM 可终止性. 对于任务级协同机制 $TCM=Task_i|\cdots|Task_j|ResMgr|Res_1|\cdots|Res_n$,每一个任务 $Task_i$ 有:$Task_i\xrightarrow{\bar{\alpha}}0$,其中:$\bar{\alpha}=r,out,d$.

下面证明两个任务 $Task_1$ 和 $Task_2$ 竞争资源 Res_i 和 Res_j 的情况,假设 $Task_1$ 使用资源 $Res_1,\cdots,Res_j$,$Task_2$ 使用资源 $Res_i,\cdots,Res_n$. 多任务竞争多资源情况的证明类似.

$$TCM=Task_1|Task_2|ResMgr|Res_1|\cdots|Res_n$$

$$\xrightarrow{\bar{r}\langle Request_1\rangle,\bar{r}\langle Request_2\rangle}out_1(res)([res=\text{succ}]\tau.\overline{d_1}.0+[res=\text{fail}]0)|(\overline{g_1}\langle GetRes\rangle.s_1(msg_{11}).([msg_{11}=\text{ok}]\overline{o_{11}}+[msg_{11}=\text{busy}]\overline{b_1}))|\cdots|(\overline{g_j}\langle GetRes\rangle.s_j(msg_{1j}).([msg_{1j}=\text{ok}]\overline{o_{1j}}+[msg_{1j}=\text{busy}]\overline{b_1}))|out_2(res)([res=\text{succ}]\tau.\overline{d_2}.0+[res=\text{fail}]0)|(\overline{g_i}\langle GetRes\rangle.s_i(msg_{2i}).([msg_{2i}=\text{ok}]\overline{o_{2i}}+[msg_{2i}=\text{busy}]\overline{b_2}))|\cdots|(\overline{g_n}\langle GetRes\rangle.s_n(msg_{2n}).([msg_{2n}=\text{ok}]\overline{o_{2n}}+[msg_{2n}=\text{busy}]\overline{b_2}))|RM_2|RM_{\text{ok}}|RM_{busy}|Res_1|\cdots|Res_n$$

$$\xrightarrow{\overline{g_1}\langle GetRes\rangle,\cdots,\overline{g_j}\langle GetRes\rangle,\overline{g_i}\langle GetRes\rangle,\cdots,\overline{g_n}\langle GetRes\rangle}$$

$out_1(res)([res=\text{succ}]\tau.\overline{d_1}.0+[res=\text{fail}]0)$
$|(s_1(msg_{11}).([msg_{11}=\text{ok}]\overline{o_{11}}+$
$[msg_{11}=\text{busy}]\overline{b_1}))|\cdots|(s_j(msg_{1j}).$
$([msg_{1j}=\text{ok}]\overline{o_{1j}}+[msg_{1j}=\text{busy}]\overline{b_1}))$
$out_2(res)([res=\text{succ}]\tau.\overline{d_2}.0+[res=\text{fail}]0)$
$|(s_i(msg_{2i}).([msg_{2i}=\text{ok}]\overline{o_{2i}}+$
$[msg_{2i}=\text{busy}]\overline{b_2}))|\cdots|(s_n(msg_{2n}).$
$([msg_{2n}=\text{ok}]\overline{o_{2n}}+[msg_{2n}=\text{busy}]\overline{b_2}))$
$|RM_2|RM_{\text{ok}}|RM_{busy}$
$|(\tau.(\overline{s_1}\langle\text{ok}\rangle+\overline{s_1}\langle\text{busy}\rangle)|(p_1(\text{succ}).Work.$
$b_1(\text{bye}).ReleaseDevice+p_1(\text{fail}).Cancel))|\cdots|$
$(\tau.(\overline{s_j}\langle\text{ok}\rangle+\overline{s_j}\langle\text{busy}\rangle)|(p_j(\text{succ}).Work.$
$b_j(\text{bye}).ReleaseDevice+p_j(\text{fail}).Cancel))$
$|(\tau.(\overline{s_i}\langle\text{ok}\rangle+\overline{s_i}\langle\text{busy}\rangle)|(p_i(\text{succ}).Work.$
$b_i(\text{bye}).ReleaseDevice+p_i(\text{fail}).Cancel))$
$|(\tau.(\overline{s_n}\langle\text{ok}\rangle+\overline{s_n}\langle\text{busy}\rangle)|(p_n(\text{succ}).Work.$
$b_n(\text{bye}).ReleaseDevice+p_n(\text{fail}).Cancel))$
$\xrightarrow{\overline{s_1}\langle\text{ok}\rangle,\cdots\overline{s_i}\langle\text{ok}\rangle,\overline{s_j}\langle\text{busy}\rangle,\overline{s_i}\langle\text{busy}\rangle,\overline{s_j}\langle\text{ok}\rangle,\cdots\overline{s_n}\langle\text{ok}\rangle}$
$out_1(res)([res=\text{succ}]\tau.\overline{d_1}.0+[res=\text{fail}]0)$
$|out_2(res)([res=\text{succ}]\tau.\overline{d_2}.0+[res=\text{fail}]0)$
$|(\overline{out_1}\langle\text{fail}\rangle|\overline{out_2}\langle\text{fail}\rangle|$
$\overline{p_1}\langle\text{fail}\rangle|\cdots|\overline{p_n}\langle\text{fail}\rangle)$
$|(p_1(\text{succ}).Work.b_1(\text{bye}).ReleaseDevice+$
$p_1(\text{fail}).Cancel)|\cdots|(p_n(\text{succ}).Work.$
$b_n(\text{bye}).ReleaseDevice+p_n(\text{fail}).Cancel)$
$\xrightarrow{\overline{out_1}\langle\text{fail}\rangle|\overline{out_2}|\overline{p_1}\langle\text{fail}\rangle|\cdots|\overline{p_n}\langle\text{fail}\rangle}0|$
$|Cancel|\cdots|Cancel\longrightarrow 0.$

然后证明在协同任务与子任务间协同机制 *ASCM* 中子任务是由协同任务启动，且协同任务等待其所有子任务执行完毕才继续执行. 若仍有在执行中的子任务，则协同任务一直等待，直到手动中止或超时处理，即证明任务与子任务之间存在时序性.

定理 2. *ASCM* 时序性. 对于任务与子任务间协同机制 $ASCM=ATask|STask_1|\cdots|STask_n$，任意一个协同任务 $ATask$ 都有 $ATask\xrightarrow{\bar{\alpha}}ATask'$，其中：$\bar{\alpha}=s_1,\cdots,s_n,(wc_1+e_1),\cdots,(wc_n+e_n)$.

下面证明子任务都执行完毕的情况，手动中止或超时处理的情况类似：

$ASCM=ATask|STask_1|\cdots|STask_n$
$\xrightarrow{s_1\langle sTaskId_1\rangle|\cdots\cdots|s_n\langle sTaskId_n\rangle}$
$(wc_1(cEnd_1)+Abort_1).\cdots.$
$(wc_n(cEnd_n)+Abort_n).ATask'$
$|\tau.\overline{wc_1}\langle cEnd_1\rangle|\cdots|\tau.\overline{wc_n}\langle cEnd_n\rangle$
$\xrightarrow{wc_1(cEnd_1),\cdots,wc_n(cEnd_n)}ATask'.$

最后证明基于时间的子任务协同机制 *TSCM* 能够在指定的时间执行相应的子任务，由于子任务的时间继承协同任务的时间且子任务由协同任务启动，因此子任务执行状态必是可达的.

定理 3. *TSCM* 状态可达性. 对于基于时间的子任务协同机制 $TSCM=STask|TP$，任意一个子任务 $STask$ 都有 $STask\xrightarrow{\bar{\alpha}}STask'$，其中 $\bar{\alpha}=t+m$.

下面分两种情况证明：

1) 指定相对时间 *time* 执行子任务.

$TSCM=STask|TP$
$\xrightarrow{\bar{t}\langle time\rangle}e.STask|\tau.\overline{e.TP}$
$\xrightarrow{\tau}e.STask|\bar{e}.TP$
$\xrightarrow{e}STask'|TP'.$

2) 指定绝对时间 *moment* 执行子任务.

$TSCM=STask|TP$
$\xrightarrow{\bar{m}\langle monent\rangle}e.STask|\tau.\bar{e}.TP$
$\xrightarrow{\tau}e.STask|\bar{e}.TP$
$\xrightarrow{e}STask'|TP'.$

5 原型系统及实验

为验证本文提出的协同机制能够满足大规模设备协同中的性能需求，设计并实现了原型系统并搭建了实验验证平台，其系统结构如图 5 所示.

大规模设备协同原型系统由大规模设备协同流程设计系统、大规模设备协同执行系统、大规模设备协同评估系统、通信服务器以及设备模拟器组成. 其中设备模拟器完成原型系统中设备功能的模拟；通信服务器实现与设备的通信；大规模设备协同流程设计系统提供图形化的协同流程设计界面，支持用户进行直观快捷的设备协同流程设计，用户设计的协同程序保存于大规模设备协同程序数据库中；大规模设备协同执行系统实现大规模设备协同流程的模拟执行，并根据执行情况将执行结果存入运营数据库；大规模设备协同评估系统对历史运营数据进行分析.

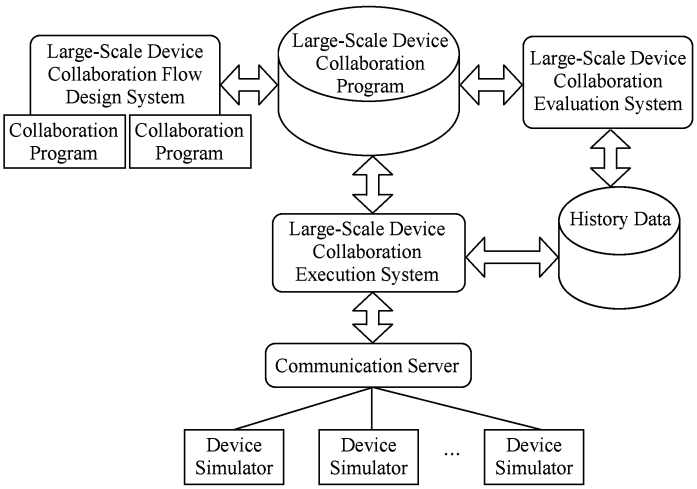


Fig. 5 Architecture of large-scale device collaborative prototype system.

图 5 大规模设备协同原型系统体系结构

5.1 实验环境

结合奥运中心区景观照明控制系统项目,以奥运中心区景观照明设备协同产生艺术照明效果为背景来验证本文提出的大规模设备协同机制的有效性.

实验验证平台中的大规模设备协同流程设计系统、大规模设备协同执行系统、大规模设备协同评估系统、通信服务器分别部署于独立的台式机,每个台式机的配置均为 Intel Pentium M 1.86 GHz 处理器,2 GB 内存,操作系统为 Windows XP SP3,JDK1.5;使用 10 台台式机部署照明设备模拟器,模拟 Panasonic IIU,型号:WR 3331-803;每台台式机模拟 100 部照明设备,台式机配置同上;大规模设备协同程序和运营数据存储于数据库服务器中,数据库服务器配置为 Intel Xeon 2.0 GHz,8 GB 内存,操作系统为 Redhat Server 5.0,数据库为 Oracle 10g;系统服务器、台式机之间通过百兆以太网互联,使用 IPv6 协议作为基础协议.

5.2 协同任务响应时间实验

在大规模设备协同系统中,当系统中存在多个并行执行的协同任务时,设备访问冲突检测及避免机制会造成协同任务的响应延迟,如果延迟过大会影响设备的协同效果,因此需要测试多协同任务在并行执行情况下的响应时间.

在协同任务响应时间实验中,从大规模设备协同程序数据库中随机选取 1,10,20,…,400,600,1 000 个协同程序并行执行,共进行 20 组实验. 为减少误差,每组实验运行 50 次,然后求平均值. 实验结果如图 6 所示:

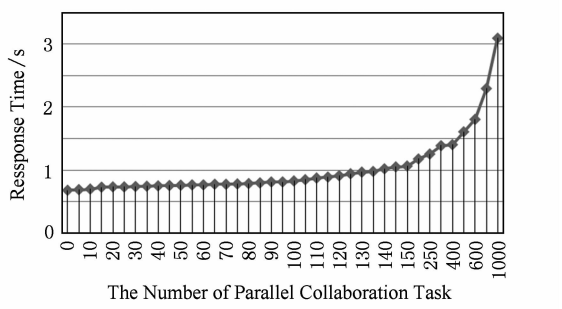


Fig. 6 Response time of collaborative tasks in parallel.

图 6 并行协同任务响应时间

图 6 中数据显示:当系统中并行协同任务数为 400 时,其平均响应时间小于 1.5 s,且平均响应时间增长稳定. 在大范围内的设备协同系统中,其响应时间能够满足协同效果的性能要求.

5.3 协同任务成功率实验

在大规模设备协同系统中,为保证设备协同效果,协同任务需要在一段时间内独占设备. 如果在规定时间内无法获得设备资源,则协同任务会由于执行超时而导致失败. 因此测试并行执行的协同任务数以及参与协同的设备数对大规模设备协同任务成功率的影响.

在协同任务成功率实验中,从数据库中随机选取 10,50,100,200 个协同程序并行执行,设置参与协同设备数为 2,5,10,…,50,100,…,800,900,1 000,共进行 76 组实验. 为减少误差,每组实验运行 50 次,然后求平均值. 实验结果如图 7 所示.

经过实验数据分析表明,大规模设备协同系统中协同任务的成功率受系统中的并行任务数与任务

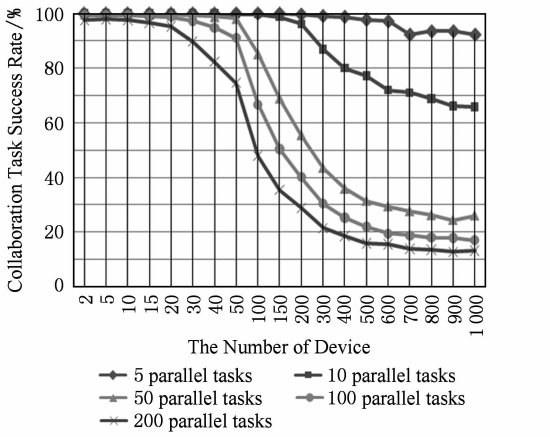


Fig. 7 Success rates of collaborative task.

图 7 协同任务成功率

中参与协同设备数的乘积影响,称其为大规模设备协同的系统负载.

从图 7 可以看出,在奥运中心区景观照明设备协同原型系统中,当系统负载达到 2 000 时,协同任务的成功率可接近 100%.实验结果表明,本文提出的大规模设备协同机制能够满足大规模设备协同系统中对大规模的要求.

6 结 论

本文针对大规模设备协同中规模大、时序约束严格的特点,结合对现有设备协同项目的分析比较,定义了大规模设备协同两级任务模型,在此模型上利用 Pi-演算给出了大规模设备协同机制的描述,理论上证明该机制是正确的.最后介绍了大规模设备协同原型系统的设计和实现,并在原型系统上进行模拟实验,验证了该机制能够满足大规模设备协同的性能要求.

基于本文提出的大规模设备协同机制实现的奥运中心区景观照明控制系统已经在北京奥运中心区部署使用,运行情况良好.

参 考 文 献

[1] Mcmullen D, Bramley R, Huffman J C, et al. The Xport collaboratory for high-brilliance X-ray crystallography [EB/OL]. 2000 [2009-12-26]. http://www.cs.indiana.edu/ngi/sc2000/index.html

[2] Devadithya T, Chiu K, Huffman K, et al. The common instrument middleware architecture: Overview of goals and implementation [C] //Proc of the 1st Int Conf on e-Science and Grid Computing. Piscataway, NJ: IEEE, 2005: 585-592

[3] Kesselman C, Butler R, Foster I, et al. NEESgrid system architecture version 1.1 [EB/OL]. 2003 [2009-12-29]. http://www.neesgrid.org/documents/NEESgrid_SystemArch_v1.1.pdf

[4] Macedo J, Pina A, Sá V, et al. GRIDCC Project D8.3 Project Final Report, with gender issues report appended [EB/OL]. 2008 [2009-12-29]. http://www.gridcc.org/documents/GRIDCC-WP8-D8_3-20080201-06-INF-Final_Report.pdf

[5] Milner R. Communicating and Mobile Systems: The Pi-Calculus [M]. Cambridge: Cambridge University Press, 1999

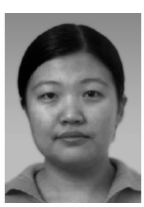
[6] Liao Jun, Tan Hao, Liu Jinde. Describing and verifying Web service using Pi-calculus [J]. Chinese Journal of Computers, 2005, 28(4): 635-643 (in Chinese)
(廖军, 谭浩, 刘锦德. 基于 Pi-演算的 Web 服务组合的描述和验证[J]. 计算机学报, 2005, 28(4): 635-643)

[7] Zhang Jing, Wang Haiyang, Cui Lizhen. Research on cross-organizational workflow modeling based on Pi-calculus [J]. Journal of Computer Research and Development, 2007, 44(7): 1243-1251 (in Chines)
(张静, 王海洋, 崔立真. 基于 Pi 演算的跨组织工作流建模研究[J]. 计算机研究与发展, 2007, 44(7): 1243-1251)

[8] Chen Feng, Rong Xiaohui, Deng Pan. The design of a large-scale area lighting scheme design language for Olympic Park [C] //Proc of 2009 Int Conf on Information Technology and Computer Science. Los Alamitos, CA: IEEE Computer Society, 2009

Rong Xiaohui, born in 1982. PhD candidate of Beihang University. Her main research interests include grid, large-scale device collaboration and collaboration performance evaluation.

Chen Feng, born in 1981. PhD candidate of Beihang University. His main research interests include grid, large-scale device collaboration and device collaborative language.



Deng Pan, born in 1983. PhD candidate of Beihang University. Her main research interests include grid, large-scale device collaboration and device model.

Ma Shilong, born in 1953. Professor and PhD supervisor of the College of Computer Science and Technology, Beihang University. His main research interests include grid, computation model in network, and logic and behavior in computing, etc.