异构集群系统中安全关键实时应用调度研究

朱晓敏[1)] 陆佩忠[2)]

[1)](国防科学技术大学信息系统工程重点实验室 长沙 410073)

[2)](复旦大学计算机科学技术学院 上海 200433)

摘 要 在集群系统中,为有安全需求的实时应用提供安全保障得到了广泛关注,但将实时应用的安全需求与调度算法相结合的研究并不多.文中提出了一种异构集群系统中安全关键实时应用的 2 阶段调度策略——TPSS.该策略综合考虑了任务的安全需求与时间限制.在 TPSS 的第 1 阶段,提出了一种自适应调度算法 DSRF,当系统负载较重时,DSRF 算法能在保证任务安全需求的基础上,通过降低新到任务和等待队列中任务的安全级别来提高任务的调度成功率.相反,当系统负载较轻时,DSRF 算法能在保证系统具有较高调度成功率的基础上充分利用任务在截止期前的空闲时间提高新任务的安全级别.在 TPSS 的第 2 阶段,提出了一种新的算法 FMSL,用来为所接收任务提供较为公平的安全服务,同时进一步提高了任务的整体安全级别.文中通过大量的模拟实验对 TPSS 策略与 DSRF 算法、SAEDF 算法和 RF 算法进行了比较.实验结果表明,TPSS 策略优于其它方法,使系统具有较强的安全性与灵活性.

关键词 调度;异构集群;实时;安全关键;截止期

中图法分类号 TP301 **DOI 号**: 10.3724/SP.J.1016.2010.02364

Scheduling for Security-Critical Real-Time Applications on Heterogeneous Clusters

ZHU Xiao-Min[1)] LU Pei-Zhong[2)]

[1)](*Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha* 410073)

[2)](*School of Computer Science, Fudan University, Shanghai* 200433)

Abstract Increasing attention has been directed towards the issue of security service for real-time applications with security requirements on clusters. In this paper, we propose a novel two-phase scheduling strategy TPSS which takes timing constraints and security needs of tasks into consideration. In the first phase, the authors propose a novel algorithm DSRF to schedule real-time tasks. When the system is in heavy burden, DSRF is able to degrade the security levels of new tasks and tasks waiting in local queues so as to enhance schedulability. On the contrary, when the system is in light burden, DSRF is capable of employing slack time to adequately improve the security qualities of new tasks. In the second phase, a new algorithm FMSL is proposed to minimize the difference of security levels of accepted tasks and further improve the security levels of these tasks on the whole, which degrades the probability of the applications being attacked. The authors compare TPSS, DSRF, SAEDF and RF by extensive simulation experiments. The experimental results indicate that TPSS significantly outperforms other algorithms and improves the security and flexibility of the cluster systems.

Keywords scheduling; heterogeneous clusters; real-time; security-critical; deadline

---

收稿日期:2008-04-26;最终修改稿收到日期:2009-10-19.本课题得到国家"八六三"高技术研究发展计划项目基金(2008AA7070412)、国家自然科学基金(60673082)、教育部全国优秀博士学位论文作者专项基金(200084)资助.**朱晓敏**,男,1979 年生,博士,讲师,主要研究方向为集群计算、实时系统、调度算法.E-mail: xmzhu@nudt.edu.cn. **陆佩忠**,男,1961 年生,博士,教授,博士生导师,主要研究领域为信息安全、高性能计算、通信.

1 引　言

近年来,集群技术发展飞快,在成本和体积迅速下降的同时,计算能力大幅度提升.对于计算密集型和数据密集型应用,采用集群技术是较为经济和可靠的手段[1].尤其是由不同计算能力的节点组成的异构集群在实际应用中得到了广泛应用.这是因为同一集群中的节点可能在不同时期购买,在计算能力上会有所差异,同时出于经济目的考虑,那些仍然具有一定计算能力的节点会和新购买的节点放在一起,从而构成了一个异构集群[2].目前,很多实时应用采用集群方式进行处理,例如,信号处理[3-4]、图像处理[5]、天气预报[6]等等.这些实时应用不仅要求运行结果正确,而且实时应用中的任务要在一定时间内完成[7].通常,实时系统分为硬实时系统[8]和软实时系统[9]两类.在硬实时系统(如病人监控系统、飞机控制系统)中,任何一个任务不能在截止期内完成都可能产生灾难性的后果;而在软实时系统(如视频传输系统、电话交换系统)中,一些任务不能在截止期内完成不会对系统产生太大的影响,但要尽量提高任务的调度成功率[7].本文所研究的安全关键实时应用运行在软实时系统上.

随着实时应用的不断发展,运行在集群上的许多实时应用不仅具有时限要求,而且还有安全保障要求[9].例如,在实时股票报价更新和交易系统中,每个用户的请求和企业后台应用程序的响应都有时间限制和安全需求,因此需要在用户与企业后台应用程序之间搭建集群来进行处理.但是由于集群执行的用户应用程序可以被大批用户访问,同时这些应用程序大多未经验证,因此,应用程序和用户都可能对集群构成威胁[10].例如,黑客可能利用应用程序漏洞控制集群,恶意用户也可能通过占用集群资源进行拒绝服务(DOS)攻击,甚至合法用户也可能通过篡改共享数据或执行大量循环操作以中断其它用户接受服务.然而,许多现存的集群计算环境还没有采用任何机制以应对安全威胁.因此,必须提供安全服务以保护运行于集群系统之上的安全性能要求较高的实时应用.

在集群计算环境中,调度算法对于提高实时应用的性能起着至关重要的作用[11].但是目前大多数调度算法仅仅考虑如何提高实时应用中任务的调度成功率,而未考虑应用的安全问题.本文提出一种 2 阶段调度策略 TPSS(Two-Phase Scheduling Strategy).该策略综合考虑了运行在异构集群上实时应用的安全需求与时间限制,能够根据系统状态动态调整调度目标.当任务到达速度较快、任务截止期较短或节点数较少等情况,使得系统负载较重时,任务的调度成功率作为调度的主要目标.相反,当系统负载较轻时,TPSS 能够在保证任务具有较高调度成功率的基础上利用任务截止期前的空闲时间提高任务的安全级别.需要注意的是,最小的安全级别能够满足用户的基本需求,高安全级别可以充分利用系统资源,使得系统的安全性更强.与此同时,TPSS 使得所接收任务具有较公平的安全服务,即在满足调度成功率不降低的前提下,任务之间的安全级别相差较小,从而降低了实时应用被攻击的概率.

本文第 2 节回顾相关的研究工作;第 3 节提出一种新的调度器模型和具有安全需求的任务模型;第 4 节介绍本文提出的 TPSS 策略和其中的 DSRF 算法与 FMSL 算法,并对这两种算法进行分析;第 5 节通过大量的模拟实验测试 TPSS 策略的性能;第 6 节总结全文.

2 相关工作

目前已有许多应用于集群系统的调度算法.但是多处理器系统中的多任务最优分配问题仍是个 NP 难题[12].因此,在实际应用中,通常采用尽可能接近最优的启发式(Heuristic)算法来解决这类调度问题[13-14].很多学者在这方面做了大量的工作.Branu等人对 11 种常用的启发式调度算法进行了评估[15].这些调度算法包括 OLB、UDA、Fast Greedy、Min-min、Max-min、Greedy、Genetic Algorithm(GA)、Simulated Annealing (SA)、GSA、Tabu 和 A*.文献[15]的实验结果表明,Min-min、GA 和 A* 具有较好的性能,但 Min-min 算法的负载均衡性相对较差.之后 Maheswaran 等人提出的 Sufferage 算法具有比 Min-min 算法较好的综合性能[16].Subramani 等人提出了一种相邻伙伴模式的启发式调度算法也具有较好性能[13].尽管以上这些算法具有较高的吞吐率和较好的负载均衡,但这些算法没有考虑任务的时间限制,不适合实时应用.尤其是 GA 和 SA 等采用人工智能的方法,由于这些方法的计算时间具有非常高的可变性,因此采用最长调度时间将极大增加系统延迟,甚至使系统变得不可调度[17].

实时任务调度算法一直是实时系统中的一个热

点研究问题.实时调度算法可以分为静态调度算法(static/offline)[18]和动态调度算法(dynamic/online)[2,19]两类.通常静态调度算法用于调度周期性(periodic)任务,而动态调度算法用于非周期(aperiodic)任务调度.本文考虑任务动态到达,因此采用的调度算法属于动态调度算法.同时,一些调度算法采用抢占式(preemptive)的调度方式[20-21],即任务在执行过程中可以被其它高优先级任务中断,而另一些调度算法采用非抢占式(Non-Preemptive)的调度方式[22-23].文献[22]指出,非抢占式的调度方式由于减少了任务之间的切换开销从而比抢占式的调度方式更为有效,尤其适合于软实时系统.由于本文考虑的安全关键实时应用运行在软实时系统上,因此本文提出的调度算法采用非抢占式的调度方式.此外,一些调度算法用于调度彼此存在依赖关系(dependent)的任务,这些任务之间的关系通常用由向无环图(DAG)来描述[2,19,24];而另一些调度算法用于调度不存在依赖关系的任务,即独立(independent)任务[3-4,23,25].本文采用的调度算法用于调度独立任务,因为有依赖关系的任务调度可以转换为独立任务调度[26].独立任务的调度算法可以采用两种模式:立即模式和批模式[16].所谓立即模式即任务到达后立即调度,而批模式采用当任务数量积累到一定数量时再进行调度.虽然采用批模式的调度方法可以积累较多的任务信息,使得某些调度较为有效,但是在实时系统中,尤其是任务截止期很短的情况下,采用批模式调度时,先到任务必须等待任务数达到一定量时才可被调度,这样就造成一些先到任务由于等待延迟而错失其截止期,降低了任务的调度成功率,因此本文的调度算法采用立即模式.

近年来,人们越来越关注于解决集群系统中的安全问题,为集群系统提供灵活有效的安全保障已经成为一个基本需求.例如,Son 等人提出了一种方法来折中安全质量以达到所需的实时要求[27];Yurcik 等人开发了一种通过监控方式管理集群系统安全的工具[28];Koenig 等人描述了一种集群安全工具 NVisionCC,用来监视集群中节点的处理过程并在异常情况下进行报警[29],等等.但是由于这些安全技术或策略没有考虑到应用的时间限制,因此不适合实时应用.

也有一些研究涉及到集群系统中实时应用的安全问题.例如,George 等人提出了一种并行控制协议以满足实时应用的安全需求[30].Ahmed 等人提出了一种安全最优并行控制协议来折中实时应用的安全需求和时间要求[31].一些学者研究如何将集群中的安全问题与调度算法结合在一起,从而提高运行在集群上的安全关键实时应用的安全性.Xie 等人提出了一种具有安全需求的调度策略 SAREC 和其相应的调度算法 SAEDF,用于提高集群系统中实时应用的安全性[32].但是,SAEDF 算法是一种贪婪算法,由于过于追求任务的高安全级别从而降低了任务的调度成功率,尤其是在系统负载较重时更为突出.同时,任务之间安全级别差异较大.之后 Xiong 等人在遗传算法的基础上提出了一种安全关键实时应用的调度算法,进一步提高了任务的整体安全级别[33].但是如前所述,这种人工智能的方法容易使系统变得不可调度,同时又采用了批模式的调度方式,使得对于实时性要求较高的应用不适用.此外,上面的调度算法都假设集群是同构的,即集群中所有节点具有相同的处理能力,限制了算法在异构集群中的扩展性.为此,本文提出一种用于异构集群系统中安全关键实时应用的调度策略,该策略能够根据系统运行状态动态调整调度目标,使得算法具有很强的自适应性,同时尽力为所接收的任务提供较为公平的服务.

3 系统调度模型

3.1 调度器模型

调度器模型可以分为两类:集中式(centralized)调度器模型和分布式(distributed)调度器模型[2].在集中式调度器模型中,由一个称之为中心调度器的处理器来收集全局调度信息,并做出调度决定.而在分布式调度器模型中,各自处理单元的调度程序根据局部范围的调度信息进行任务调度.与分布式调度器模型相比,集中式调度器模型具有两个显著优点:(1)通过对中心调度器的备份,便于实现容错处理;(2)实现比较容易.为实现 TPSS 的调度目标,本文在集中式调度器模型的基础上进行了改进,给出了一种新的调度器模型,如图 1 所示.该模型适合于具有安全需求且运行在异构集群系统上的实时应用.

在该调度器模型中,实时调度器为一全局调度器.当有一新任务到达时,实时调度器首先收集各节点上正在运行任务和局部队列中等待任务的反馈信息(feedback information),这些反馈信息包括正在运行任务的剩余执行时间、排队等待任务的等待时间等.然后根据实时调度器中的 DSRF 算法来决定

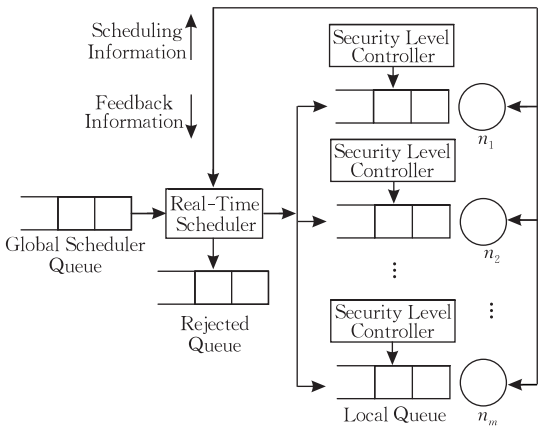

图 1　调度器模型

新任务是否可以被接收.如果新任务可以被接收,则将该任务发送到目的节点的等待队列中,否则发送到拒绝队列中.如果新任务被发送到目的节点后,实时调度器将任务调度信息(scheduling information)发送到该目的节点.这些任务调度信息包括所有等待队列中任务的执行顺序、为任务所提供的安全服务.需要注意的是,当一个新任务被发送到某一目的节点后,在同一节点的等待队列中,先于新任务到达的那些任务的执行顺序和为其提供的安全服务可能发生改变.之后,安全级别控制器——局部控制器根据 FMSL 算法对目的节点局部队列中任务的安全级别进行调整,使得任务具有较为公平的安全服务,同时进一步提高任务的整体安全级别.在调整过程中,等待队列中的所有任务满足其截止期且总体完成时间不延迟,因此保证了任务的调度成功率.

3.2　任务模型

令某一实时应用为一组实时独立任务的集合 $T=\{t_1,t_2,\cdots,t_n\}$. $N=\{n_1,n_2,\cdots,n_m\}$ 为节点集合,其中每个节点具有不同的处理能力.执行时间用矩阵 $\boldsymbol{E}=(e_{ij})_{n\times m}$ 表示,其中元素 e_{ij} 表示任务 t_i 在节点 n_j 上的执行时间. a_i 和 d_i 分别表示任务 t_i 的到达时间和截止期.令 $\boldsymbol{EST}=(est_{ij})_{n\times m}$ 为任务最早开始时间矩阵,其中元素 est_{ij} 表示任务 t_i 在节点 n_j 上的最早开始时间. $\boldsymbol{Z}=(z_{ij})_{n\times m}$ 为一二元矩阵, $z_{ij}=1$ 当且仅当任务 t_i 被分配到节点 n_j 上,否则 $z_{ij}=0$. $\boldsymbol{O}=(o_{ij})_{n\times m}$ 为任务执行顺序矩阵,在 $z_{ij}=1$ 时,元素 o_{ij} 表示任务 t_i 在节点 n_j 上的执行顺序.

一个系统可以采用多种不同类型的安全服务.这些服务类型包括保密服务、完整性服务、认证服务等等.我们将同一种类型的安全服务分在一组,在同一组内,多种安全服务可以提供同一类型的安全保证,但是由于采用的机制有所不同,所以安全质量有所不同.例如,SEAL、RC4 和 DES 都是加密算法,但是它们所提供的安全质量却不同.假设有 q 组安全服务 $G=\{g_1,g_2,\cdots,g_q\}$. $\boldsymbol{S}=(s_{ij})_{|s_i|\times q}$ 为安全级别矩阵,其中元素 s_{ij} 表示 g_j 组中安全服务 s_i 的安全级别, $0<s_{ij}\leq 1$, $|s_{ij}|$ 表示 g_j 组中安全服务的个数.每个任务可以选择不同组中的不同服务从而形成一个整体安全级别.假设 X_{ij} 表示任务 t_i 在节点 n_j 上所有可能的安全服务组合, x_{ij} 表示任务 t_i 在节点 n_j 上选择安全服务的一种组合,则任务 t_i 在节点 n_j 上的安全级别可以表示为

$$sl(x_{ij})=z_{ij}\sum_{l=1}^{q}w_l s_{kl},\ 0\leq w_l\leq 1,\ \sum_{l=1}^{q}w_l=1 \tag{1}$$

其中 w_l 表示 g_l 组的权重, s_{kl} 表示 g_l 组中安全服务 s_k 的安全级别.

在实际应用中,安全开销必须被定量地计算以使得调度更为有效.本文采用的安全开销模型类似于文献[32].由于侦听(snooping)、篡改(alteration)和哄骗(spoofing)是集群环境中最常见的 3 种攻击,而防范这 3 种攻击的有效方法就是提供保密服务(confidentiality service)、完整性服务(integrity service)和认证服务(authentication service).为不失一般性和便于分析比较,我们也考虑这 3 种类型的安全服务.与文献[32]不同的是,本文考虑了安全开销在异构环境中的差异.假设任务 t_i 需要组 g_l 中的第 k 个安全服务,如果 t_i 被分配到节点 n_j 上,则其安全开销可以表示为

$$c(x_{ij})=z_{ij}\sum_{l=1}^{3}c_{ij}(s_{kl}) \tag{2}$$

其中 g_1、g_2 和 g_3 分别提供保密服务、完整性服务和认证服务, p_j 表示节点 n_j 的处理能力.

表 1~3 给出了每种安全服务所对应的安全级别和它们相应的性能(在 90MHz 奔腾处理器上测定)[32],其中 μ_{k1}、μ_{k2} 和 μ_{k3} 分别代表在组 g_1、g_2 和 g_3 中第 k 个安全服务的性能.对于保密服务和完整性服务,安全计算开销取决于使用的加密算法(对于保密服务)、Hash 函数(对于完整性服务)、被保护的数据大小和节点的处理能力;而对于认证服务,安全计算开销则只依赖于所采用的认证技术和节点的处理能力.设 s_i 为任务 t_i 的数据大小, p_j 表示节点 n_j 的处理能力, p_b 表示基准节点的处理能力,则任务 t_i 在节点 n_j 的安全开销 $c(x_{ij})$ 可进一步表示为

$$c(x_{ij})=z_{ij}\left(\sum_{l=1}^{2}(s_i/\mu_{kl})+\mu_{k3}\right)\times p_b/p_j \tag{3}$$

表 1 保密服务的加密算法

加密算法	s_{kl}(安全级别)	u_{k1}/(KB·ms^{-1})
SEAL	0.08	168.75
RC4	0.14	96.43
Blowfish	0.36	37.5
Knufu/Khafre	0.40	33.75
RC5	0.46	29.35
Rijndael	0.64	21.09
DES	0.90	15
IDEA	1.00	13.5

表 2 完整性服务的 Hash 函数

Hash 函数	s_{kl}(安全级别)	u_{k1}/(KB·ms^{-1})
MD4	0.18	22.90
MD5	0.26	17.09
RIPEMD	0.36	12.00
RIPEMD-128	0.45	9.73
SHA-1	0.63	6.88
RIPEMD-160	0.77	5.69
Tiger	1.00	4.36

表 3 认证方法

Hash 函数	s_{kl}(安全级别)
HMAC-MD5	0.55
HMAC-SHA-1	0.91
CBC-MAC-AES	1.00

4 TPSS 策略

TPSS 策略包含两个算法:DSRF 算法和 FMSL 算法. DSRF 算法用于根据系统负载情况,自适应地调整调度目标,使得在系统负载较重时,任务具有较高的调度成功率;而在系统负载较轻时,能在保证任务具有较高调度成功率的基础上最大化任务的安全级别. FMSL 算法用于为所接收任务提供较为公平的服务,并且能够进一步提高任务的整体安全级别.

4.1 DSRF 算法

DSRF 算法源于 RF(Response First)算法[19]. RF 算法的核心思想是把任务分配到具有最早完成时间的节点上. 这种算法采用立即模式,是在异构集群系统中实时任务调度普遍采用的一种算法. 但是 RF 算法没有考虑应用的安全需求,同时不具备自适应性. 本文提出的 DSRF 算法包括 3 个步骤. 在步骤 1 中,新到任务被分配最高的安全级别,然后按照截止期最早最优先的策略插入到节点的局部队列中,选择具有最早完成时间的节点作为目的节点. 需要注意的是,这里的截止期最早最优先策略不是传统的 EDF 方法,因为 EDF 算法是抢占式的,而我们采用的这种策略只是对局部队列中的任务进行执行顺序的排队. 步骤 1 用来最大化任务的安全级别. 如果采用最高安全级别能够保证新任务的时限要求,同时执行顺序大于新任务的那些任务也满足其截止期,那么分配该任务;如果在任何节点上都不能满足上面的限制,那么转而执行步骤 2. 在步骤 2 中,降低新任务的安全级别,直到满足步骤 1 中所述的限制条件. 步骤 2 同样用来尽力提高新任务的安全级别. 当步骤 2 也不能将任务分配时,步骤 3 开始工作. 步骤 3 首先选择局部队列中任务安全级别之和最大的节点,然后采用轮询(round-robin)的方法降低局部队列中这些等待任务的安全级别,直到新任务能被分配,如果所有等待任务的安全级别都降到最低仍不能满足新任务或执行顺序晚于新任务的那些任务的时间限制,则放弃该任务,否则将新任务分配到该节点. 步骤 3 用来提高任务的调度成功率.

下面介绍 DSRF 算法的一些性质.

性质 1. 如果任务 t_i 被分配到节点 n_j 上,则 t_i 和 n_j 局部队列中执行顺序大于 t_i 的那些任务必须满足下面 2 个不等式:

$$est_{ij}+e_{ij}+c(x_{ij})\leq d_i \tag{4}$$

$$\forall t_k, o_{kj}>o_{ij}: est'_{kj}+e_{kj}+c(x_{kj})\leq d_k \tag{5}$$

其中, $est'_{kj}=est_{kj}+e_{ij}+c(x_{ij})$,任务 t_i 在节点 n_j 的最早开始时间 est_{ij} 可被计算为

$$est_{ij}=a_i+\sum_{o_{kj}<o_{ij},w_k=1}(e_{kj}+c(x_{kj}))+r_j \tag{6}$$

其中,a_i 为任务 t_i 的到达时间,r_j 为节点 n_j 上正在运行任务的剩余执行时间,$\sum_{o_{kj}<o_{ij},w_k=1}(e_{kj}+c(x_{kj}))$ 为节点 n_j 等待队列中执行顺序小于 t_i 的任务的执行时间(包括安全开销). 如果任务 t_k 在等待队列中,则 $w_k=1$,否则 $w_k=0$.

性质 2. 如果步骤 3 开始执行,那么步骤 3 所选择的节点 n_j 其局部队列中等待任务 t_l 的最早开始时间需重新计算.

(1) 如果任务 t_l 的执行顺序小于新任务 t_i 的执行顺序,即 $o_{lj}<o_{ij}$,则任务 t_l 的最早开始时间可被计算为

$$est'_{lj}=est_{lj}-\sum_{o_{kj}<o_{lj}}(c(x_{kj})-c(x_{kj})') \tag{7}$$

(2) 如果任务 t_l 的执行顺序大于新任务 t_i 的执行顺序,即 $o_{lj}>o_{ij}$,则任务 t_l 的最早开始时间可被计算为

$$est'_{lj}=est_{lj}-\sum_{o_{kj}<o_{lj}}(c(x_{kj})-c(x_{kj})')+e_{ij}+c(x_{ij}) \tag{8}$$

其中 $c(x_{kj})'$ 表示当任务 t_k 的安全级别降低后新的安全开销.注意,当等待任务的安全级别降低后,必须满足性质 1.

DSRF 算法的伪代码如算法 1 所示. DSRF 算法是一种自适应调度算法,在保证任务调度成功率的基础上尽力提高任务的安全级别.首先,新到任务被指定最高的安全级别(见第 2 行).如果采用最高的安全级别不能满足性质 1,降低新任务的安全级别直到满足性质 1(见第 3～17 行).如果新任务采用最小的安全级别仍然不能满足性质 1,选择局部队列中等待任务安全级别之和最大的节点(见第 23 行),采用轮询的方式降低这些等待任务的安全级别直到满足性质 1(见第 25～29 行).如果这些任务的安全级别都被降到最低仍不能满足性质 1,那么拒绝新任务,否则接收这个任务(见 40～45 行).

算法 1.　DSRF 算法.

```
1.  for each new task t_i do
2.      find←false; sl(x_ij)←max{sl}; /*初始化*/
3.    while sl(x_ij)!=min{sl} do
4.      earliestFinishTime←∞;   /*初始化*/
5.      for each node n_j in the cluster do
6.        Calculate r_j and Σ_{o_kj<o_ij, w_k=1} (e_kj+c(x_kj));
7.        Calculate the finish time f_ij:
            f_ij=est_ij+e_ij+c(x_ij);
8.        if f_ij<earliestFinishTime then
9.          earliestFinishTime←f_ij; findNode←j;
10.       end if
11.     end for
12.     if est_im+e_im+c(x_im)≤d_i and ∀t_k,o_km>o_im:
          est'_km+e_km+c(x_km)≤d_k (m←findNode)
          then
13.       find←true; break;
14.     else
15.       Degrade one security level in g_1, g_2 or g_3 by
          Round-Robin policy;
16.     end if
17.   end while
18.   if find==true then
19.     Allocate t_i to node findNode;
20.     Update earliest start time (EST) of tasks in
        the local queue of findNode;
21.   else
22.     degradeSum←0; S←∅; /*初始化*/
23.     Select node n_l on which the sum of security
        levels of tasks waiting in its local queue is
        largest;
24.     Put these tasks into setS;
25.     while S!=∅ do
26.       degradeSum++;
27.       for each task t_k on node n_l do
28.         if sl(x_kl)!=min{sl} then
29.           sl(x_kl)-=degradeSum;
30.           Calculate new EST according to Property 2;
31.           if est_im+e_im+c(x_im)≤d_i and ∀t_k,o_km>
                o_im:est'_km+e_km+c(x_km)≤d_k then
32.             find←true; break;
33.           end if
34.         else
35.           Remove t_k from S;
36.         end if
37.       end for
38.     if find==true then break; end if
39.     end while
40.     if find==true then
41.       Allocate t_i to node n_l;
42.       Update EST of tasks in the local queue of n_l;
43.     else
44.       Reject task t_i;
45.     end if
46.   end if
47. end for
```

定理 1.　DSRF 算法的时间复杂度为 $O(3k(mn+n^2))$,其中 m 为集群中的节点数,n 为一个节点局部队列中等待任务数,k 为安全级别个数.

证明.　计算新任务 t_i 在一个节点上的完成时间,其时间复杂度为 $O(n)$(见第 6～7 行).验证 t_i 是否能满足自身和在同一节点的局部队列中执行顺序大于 t_i 的那些任务的时间限制,其时间复杂度为 $O(n)$(见第 12 行).更新目标节点局部队列中任务的最早开始时间,其时间复杂度为 $O(n)$(见第 20 行).找到局部队列中等待任务的安全级别之和最大的节点(选中节点),其时间复杂度为 $O(m)$(见第 23 行).将选中节点等待队列中的任务复制到集合 S 的时间复杂度为 $O(n)$(见第 24 行).降低选中节点局部队列中任务的安全级别,其最坏时间复杂度为 $O(3kn^2)$(见第 25～39 行).如果任务 t_i 能被分配,则分配该任务并更新所分配节点局部队列中任务的最早开始时间,其时间复杂度为 $O(n)$(见第 41～42 行).其它行的时间复杂度均为 $O(1)$.因此 DSRF 算法的时间复杂度可被计算为

$$O(3k)(O(m)(O(n)))+3O(n)+O(m)+O(3nk^2)=O(3k(mn+n^2)).$$

证毕.

由于 n 为一个节点等待队列中的等待任务个数,所以 n 的值较小.同时,在实际应用中,k 和 m 也不是非常大的数,因此 DSRF 算法的时间复杂度

不高.

4.2 FMSL 算法

为使得所接收任务具有较公平的安全服务,同时进一步提高任务的整体安全级别,当一新任务被分配到某一节点后,FMSL 算法将对该节点上局部队列中的任务做进一步处理. 节点 n_j 上任务安全级别的公平性可用任务安全级别的标准差 FL_j 表示,FMSL 算法的目标即在不影响任务调度成功率的基础上最小化 FL_j.

FMSL 算法的核心思想是通过降低高级别任务的安全级别以提高低级别任务的安全级别. 当一新任务被分配到目标节点后,将安全级别相对最高和相对最低的任务分别拷贝到集合 $maxSet$ 和 $minSet$ 中. 降低 $maxSet$ 集合中任务的安全级别以提高 $minSet$ 集合中任务的安全级别,在这一过程中必须保证这些任务满足时间限制,且局部队列中这些任务的整体完成时间不被延迟.

下面介绍 FMSL 算法的一些性质.

性质 3. 如果 $t_p \in maxSet, t_q \in minSet$,当 $o_{pj} < o_{qj}$ 时,则必须满足下面的不等式:

$$est'_{qj} + c(x'_{qj}) \leq est_{qj} + c(x_{qj}) \tag{9}$$

其中 est'_{qj} 和 $c(x'_{qj})$ 分别为任务 t_q 的安全级别提高后新的最早开始时间和新安全开销.

$$est'_{qj} = est_{qj} + c(x_{pj}) - c(x'_{pj}) \tag{10}$$

其中 $c(x'_{pj})$ 为 t_p 的安全级别降低后新的安全开销.

执行顺序在任务 t_p 和 t_q 执行顺序之间的任务 t_b,其最早开始时间可被重新计算为

$$est'_{bj} = est_{bj} - (c(x_{pj}) - c(x'_{pj})) \tag{11}$$

执行顺序大于任务 t_q 执行顺序的任务 t_a,其最早开始时间可被重新计算为

$$est'_{aj} = est_{aj} - (est_{qj} + c(x_{qj}) - est'_{qj} - c(x'_{qj})) \tag{12}$$

图 2 给出了性质 3 的一个实例,其中 f_{ij} 表示任务 t_i 在节点 n_j 上的完成时间.

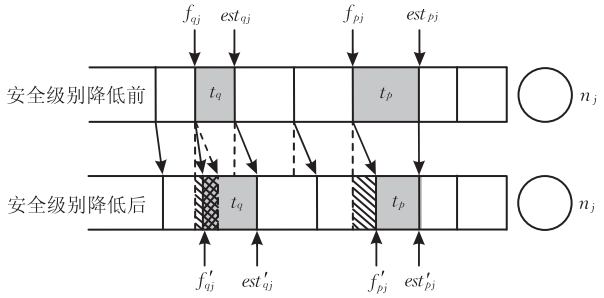

图 2 性质 3 实例

性质 4. 如果 $t_p \in maxSet$, $t_q \in minSet$,当 $o_{pj} > o_{qj}$ 时,则必须满足下面的不等式:

$$est_{qj} + e_{qj} + c(x'_{qj}) \leq d_q \tag{13}$$

其中 $c(x'_{qj})$ 为任务 t_q 的安全级别提高后新的安全开销.

执行顺序在任务 t_p 和 t_q 执行顺序之间的任务 t_b,当 t_q 的安全级别提高后,必须满足其截止期:

$$est'_{bj} + e_{bj} + c(x_{bj}) \leq d_b \tag{14}$$

其中 est'_{bj} 为新最早开始时间,可被计算为

$$est'_{bj} = est_{bj} + c(x'_{qj}) - c(x_{qj}) \tag{15}$$

任务 t_p 的最早开始时间计算方法与 t_b 一致,同时要满足 t_p 的新完成时间要小于等于原完成时间,以保证任务的调度成功率.

$$est'_{pj} + c(x'_{pj}) \leq est_{pj} + c(x_{pj}) \tag{16}$$

执行顺序大于任务 t_p 执行顺序的任务 t_a,其最早开始时间可被重新计算为

$$est'_{aj} = est_{aj} - (est_{pj} + c(x_{pj}) - est'_{pj} - c(x'_{pj})) \tag{17}$$

图 3 给出了性质 4 的一个实例.

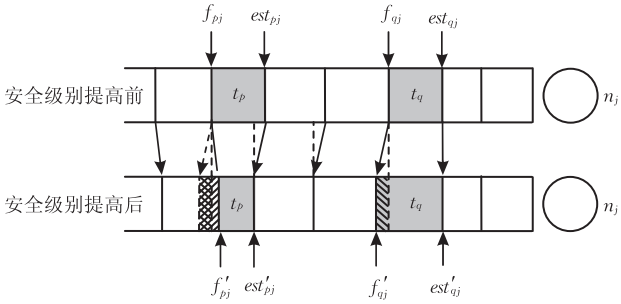

图 3 性质 4 实例

FMSL 算法的伪代码如算法 2 所示.

算法 2. FMSL 算法.

```
1.  while true do
2.    canAdjust←false; /*初始化*/
3.    Put tasks with maximal security level and minimal security level on n_j into maxSet and minSet, respectively;
4.    while maxSet!=∅ do
5.      Get a task t_p from maxSet;
6.      while minSet!=∅ do
7.        addLevel←−1; /*初始化*/
8.        Get a task t_q from minSet;
9.        Degrade security level of t_p one level;
10.       Calculate addLevel of t_q;
11.       if addLevel==0 then continue; end if
12.       if o_pj>o_qj then
13.         while addLevel>0 do
14.           if est_qj+e_qj+c(x'_qj)≤d_q and est'_bj+e_bj+
                c(x_bj)≤d_b then
```

```
15.          break;
16.         else
17.          addLevel−−;
18.         end if
19.        end while
20.        if addLevel!=0 then
21.         Update EST of tasks in the local queue by
            Property 4;
22.         Remove t_p from maxSet; remove t_q from
            minSet;
23.         canAdjust←true;break;
24.        end if
25.       end if
26.       if o_pj<o_qj then
27.        Update EST of tasks in the local queue by
           Property 3;
28.        Remove t_p from maxSet; remove t_q from
           minSet;
29.       end if
30.       Get next task from minSet;
31.      end while
32.      if canAdjust==true then break; end if
33.      Get next task from maxSet;
34.    end while
35.    if canAdjust==false then break; end if
36.  end while
```

FMSL 算法首先将节点局部队列中安全级别相对最高和安全级别相对最低的任务分别拷贝到集合 $maxSet$ 和 $minSet$ 中(见第 3 行).从集合 $maxSet$ 中任选一任务 t_p,并从集合 $minSet$ 中任选一任务 t_q,如果 t_p降低一个安全级别不能使得 t_q提高一个级别,那么从 $minSet$ 选择另外一个任务(见第 4～11 行).如果 t_p的执行顺序大于 t_q的执行顺序,当 t_q的安全级别被提高后,那么 t_q和执行顺序在 t_p和 t_q之间的那些任务不能错失其截止期(性质 4).如果性质 4 不能满足,降低增加的安全级别直到满足性质 4(见第 12～17 行).当 t_p和 t_q的安全级别调整后,将 t_p和 t_q分别从集合 $maxSet$ 和 $minSet$ 中移除,因为它们此时已经不具有相对最高和最低的安全级别(见第 22 行).如果 t_p的执行顺序小于 t_q的执行顺序,根据性质 3 更新局部队列中等待任务的最早开始时间并将 t_p和 t_q分别从集合 $maxSet$ 和 $minSet$ 中移除(见第 26～29 行).如果没有任何任务的安全级别可以调整,那么结束该算法(见第 35 行).

定理 2.　FMSL 算法的时间复杂度为 $O(n^2(k+n))$,n 为一个节点 n_j局部队列中的等待任务数,k 为安全级别个数.

证明.　找到节点 n_j局部队列中安全级别相对最高和相对最低的任务,其时间复杂度为 $O(n)$(见第 3 行).当 t_p的执行顺序大于 t_q的执行顺序时,降低任务 t_q提高的安全级别,以满足 t_q和执行顺序在 t_p和 t_q之间那些任务的时间要求,其最坏时间复杂度为 $O(k)$(见第 13～19 行).根据性质 4,更新 n_j局部队列中等待任务的最早开始时间,其时间复杂度为 $O(n)$(见第 21 行).当 t_p的执行顺序小于 t_q的执行顺序时,根据性质 3,更新 n_j局部队列中等待任务的最早开始时间,其时间复杂度为 $O(n)$(见第 27 行),因此 FMSL 算法的时间复杂度可被计算为

$$O(n)+O(n)(O(n)((O(k)+O(n))+O(n)))=O(n^2(k+n)).$$

证毕.

由于局部队列中的等待任务数 n 较小,同时在 $maxSet$ 和 $minSet$ 中的任务数还要远小于 n,因此 FMSL 算法的时间复杂度不高.

5　实验测试

本文通过大量的模拟实验结果来说明 TPSS 策略的性能.将其与 RF[19]、DSRF 和 SAEDF[32]的结果进行比较.为公平起见,将 RF 算法进行了略微修改,即它随机选择任务的安全级别.本文主要从以下几个方面比较了 TPSS、RF、DSRF 和 SAEDF 的性能:

(1)调度成功率(Guarantee Ratio,GR);

(2)安全级别均值(Security Level Average,SLA);

(3)安全级别标准误差(Security Level Standard Deviation,SLSD);

(4)整体系统性能(Overall System Performance, OSP,$OSP=GR*SLA/SLSD$).

5.1　模拟方法与参数

异构性可以分为节点异构性和任务异构性[16].本文在实验中充分考虑了节点和任务的异构性.下面给出实验的模拟方法:

(1)为了体现节点的异构性,用 p_j表示节点 n_j的处理能力.p_j为一正实数,p_j越大节点的处理能力越强.参数 $powerAverage$ 和 $powerSpan$ 分别表示节点的平均处理能力和处理能力的浮动范围.p_j均匀分布在 $powerAverage-powerSpan$ 和 $powerAverage+powerSpan$ 之间.

(2)h_i表示任务 t_i的处理难度.h_i为一正实数,h_i越大任务 t_i在同一节点上的执行时间越长.参数 $hardnessAverage$ 和 $hardnessSpan$ 分别表示所有任务的平均处理难度和处理难度的浮动范围.h_i

均匀分布在 $hardnessAverage - hardnessSpan$ 和 $hardnessAverage + hardnessSpan$ 之间，用以体现任务的异构性.

(3) 文献[16]将任务执行时间矩阵分为一致(Consistent)矩阵和不一致(Inconsistent)矩阵. 本文中的任务执行矩阵采用一致矩阵，即任务的执行时间与任务的处理难度成反比，与节点的处理能力成正比. 任务 t_i 在节点 n_j 上的执行时间 e_{ij} 可以表示为 $e_{ij} = baseTime \times (h_i / p_j)$. 参数 $baseTime$ 为一正实数.

(4) 任务 t_i 在节点 n_j 上的安全开销 $c(x_{ij})$ 充分考虑了任务异构性和节点异构性. $c(x_{ij})$ 的计算如公式(3)所示.

(5) 任务 t_i 的截止期可被计算为 $d_i = a_i + \max\{e_{ij}\} + \max\{c(x_{ij})\} + baseDeadline$. 其中 a_i 表示任务 t_i 的到达时间，$\max\{e_{ij}\}$ 表示 t_i 在所有节点上的最长执行时间，$\max\{c(x_{ij})\}$ 表示 t_i 在所有节点上的最大安全开销，$baseDeadline$ 为一随机正实数，$baseDeadline$ 越大，任务截止期越宽松.

(6) 任务 t_i 的到达时间 $a_i = a_{i-1} + intervalTime$，其中 a_{i-1} 表示 t_i 的前一个任务 t_{i-1} 的到达时间，$a_0 = 0$. 参数 $intervalTime$ 为一正实数，决定任务的到达速度.

(7) 任务 t_i 的数据大小 s_i 正比于 t_i 的处理难度. $s_i = baseSize \times h_i$. 参数 $baseSize$ 为一正实数.

表 4 给出了实验中的参数值.

表 4 实验参数表

参数	值(fixed)-(min, max, step)
节点数	(32)-(8,48,8)
任务数	(6400)
powerAverage	(700)
powerSpan	(400)-(50,600,50)
hardnessAverage	(300)
hardnessSpan	(100)
baseDeadline	(100)
baseTime	(60)
intervalTime	(1)-(0.5,1.5,0.1)
baseSize	50kB
安全服务权值	$w_1=0.2, w_2=0.5, w_3=0.3$

5.2 节点对性能的影响

本节通过一组实验来观察节点数对 TPSS、RF、DSRF 和 SAEDF 的影响. 实验结果如图 4 所示.

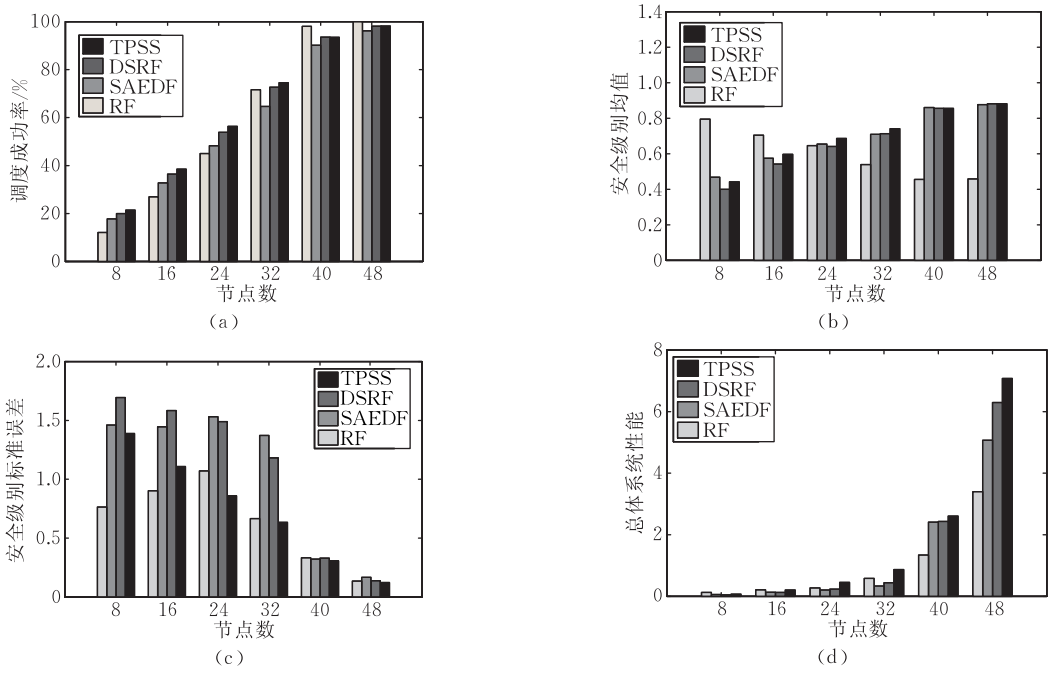

图 4 节点对性能的影响

图 4(a)显示了 DSRF 的调度成功率一直高于 SAEDF 的调度成功率，这是因为 SAEDF 在满足新任务和同一节点局部队列中任务截止期的前提下，始终为新任务选择尽可能最高的安全级别，这将导致新任务具有较长的执行时间，因此在系统负载较重时，晚到达的任务具有较晚的开始时间. 如果任务的开始时间延迟，那么任务错失其截止期的概率增大. 与 SAEDF 不同的是，当系统负载较重时，DSRF 将任务的调度成功率作为系统的主要目标. DSRF 通过降低局部队列中等待任务的安全级别，提高了系统的调度成功率. 需要注意的是，任务的安全级别降到最低仍在系统可接受的安全级别范围内. 从

图 4(a)中可以看出,TPSS 的调度成功率有时略高于 DSRF 的调度成功率,这是因为 TPSS 中的 FMSL 算法在调整局部队列中任务的安全级别时,有时会使得所有任务的新总体完成时间略早于原完成时间,因此后到任务的最早开始时间提前,使得任务的调度成功率略有增加(图 2、图 3 的例子也表明这一点).图 4(a)还显示了当节点数小于 24 时,RF 具有最低的调度成功率,但是当节点数大于 40 时,RF 具有最高的调度成功率,这是因为 RF 随机选取任务的安全级别,因此调度成功率不确定.

图 4(b)显示了当节点数小于 24 时,SAEDF 的安全级别均值高于 DSRF 的安全级别均值,这是因为 SAEDF 尽力提高接收任务的安全级别而未考虑系统的负载情况.相反,尽管 DSRF 的安全级别均值略低于 SAEDF 的安全级别均值,但是 DSRF 具有较高的调度成功率.从图 4(b)可看出,TPSS 的安全级别均值一直高于 SAEDF 的安全级别均值,这是因为 TPSS 算法在调整任务安全级别时,一个高安全级别任务被降低一个安全级别后,可以使得一个低安全级别任务提高一到多个安全级别,因此任务的整体安全级别提高了.当节点数大于 24 时,DSRF 具有和 SAEDF 基本相同的安全级别均值,这是因为当系统负载较轻时,DSRF 能够充分利用任务截止期前的空闲时间,提高任务的安全级别.因此 TPSS 具有很强的自适应性.

图 4(c)显示了 TPSS 的安全级别标准误差一直小于 DSRF 和 SAEDF 的安全级别标准误差.这可解释为 TPSS 中的 FMSL 算法在保证局部队列中任务时间限制的前提下,均衡任务的安全级别,使得任务具有较公平的安全服务.相反,DSRF 和 SAEDF 没有考虑这一点.当节点数较少时,系统负载较重,先到任务具有较高的安全级别,而晚到任务则需要通过降低其安全级别才可能保证被调度,因此任务的安全级别差异较大.随着节点数的增加,系统负载减轻,能够保证任务在接收的同时具有较高的安全级别,因此安全级别标准误差降低.由于 RF 随机选取安全级别,因此 RF 的安全级别标准误差与节点数不具有线性关系.

图 4(d)显示了 TPSS 的总体系统性能明显好于其它算法,平均高于 RF、SAEDF 和 DSRF 45.2%、53.9%和 67.7%.图 4(d)表明,当节点数较多时,TPSS 的整体性能增加显著,这意味着在系统负载较轻时,系统性能收益较大.

5.3　任务到达速度对性能的影响

本节我们通过一组实验来观察任务到达速度对对 TPSS、RF、DSRF 和 SAEDF 的影响.在本组实验中调整参数 *intervalTime* 从 0.5～1.5,步长为 0.1.实验结果如图 5 所示.

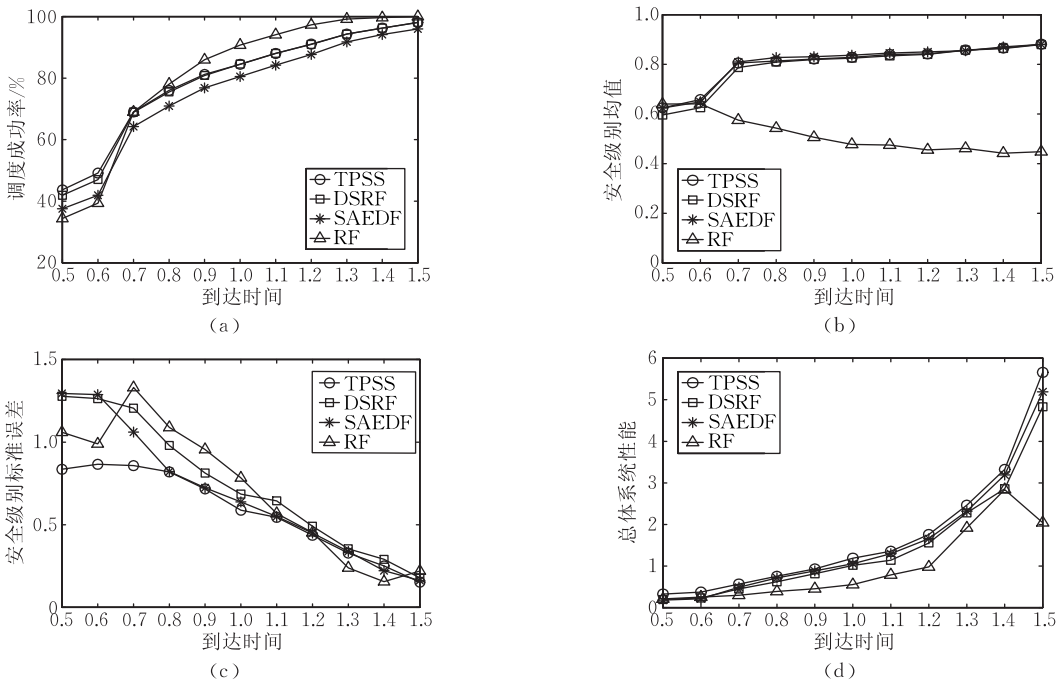

图 5　任务达到速度对性能的影响

图 5(a)表明,当到达时间较小时,任务的到达速度较快,使得在节点局部队列中的等待任务较多,所以一些晚到达任务由于具有较晚的开始时间而错失其截止期.随着到达时间值的增大,任务的到达速

度变慢,使得在节点局部队列中的任务较少,任务具有较早的开始时间,因此任务的调度成功率增加.从图 5(a)可以看出,TPSS 和 DSRF 相比于 SAEDF 具有较高的调度成功率,这是因为 DSRF 通过降低等待队列中任务的安全级别,减小了这些任务的运行时间,因此更多的任务因为具有较早的开始时间而被接收.相反,SAEDF 始终保持所接收任务的安全级别保持不变,因此一些任务由于具有较晚开始时间而错失截止期,不能被接收.尤其是当任务达到速度很快时,DSRF 具有更高的调度成功率,这是因为 DSRF 是一种自适应调度算法,当任务到达速度较快使得系统负载较重时,调度成功率是 DSRF 算法的主要目标.图 5(a)还显示,当到达时间值大于 0.7 时,RF 的调度成功率高于其它方法,这是因为 RF 随机选取任务的安全级别,从图 5(b)可以看出此时 RF 选择的安全级别均值最低,因此具有较高的调度成功率.

图 5(b)显示,当到达时间值小于 1.3 时,SAEDF 的安全级别均值略高于 DSRF 的安全级别均值,这是因为 SAEDF 过于追求当前新到任务的高安全级别,而没有考虑后到任务的调度成功率.相反,DSRF 算法采用自适应的调度策略,在系统负载较重时,通过降低节点局部队列中等待任务的安全级别提高了任务的调度成功率.当到达时间值大于 1.3 时,DSRF 和 SAEDF 具有相似的安全级别均值,这种情况是因为当任务到达速度较慢时,DSRF 算法不必通过降低任务的安全级别来接收新任务,此时提高任务的安全级别变成了 DSRF 的主要目标.

从图 5(c)可以看出,TPSS 相比 SAEDF 具有较小的安全级别标准误差,尤其是当任务到达速度较快时,这是因为 FMSL 算法起到的作用使得所接收任务间的安全级别差异较小.图 5(d)给出了整体的系统性能,TPSS 的整体系统性能明显好于其它算法,这说明对于非周期任务,TPSS 具有更强的灵活性和可靠性.

5.4 节点异构性对性能的影响

本节我们通过一组实验来观察节点异构性对 TPSS、RF、DSRF 和 SAEDF 的影响.实验中用参数 *powerSpan* 代表节点的异构性变化,*powerSpan* 增大,则节点的异构性增大,反之减小.实验结果如图 6 所示.

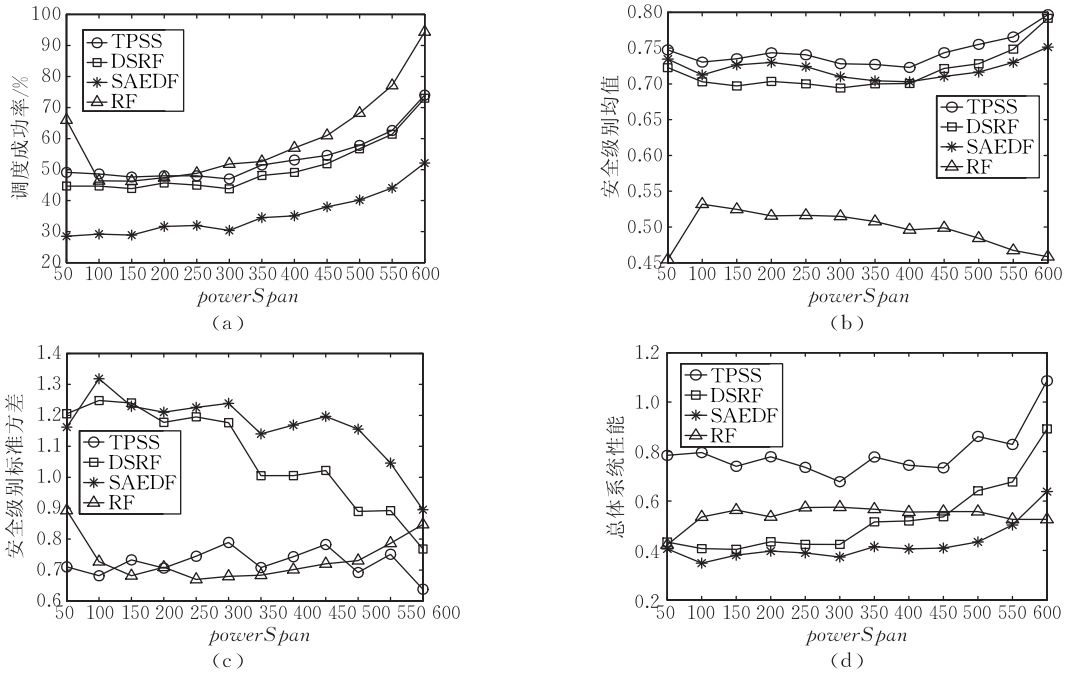

图 6 节点异构性对性能的影响

图 6(a)显示,随着节点异构性的增大,调度成功率有所增加,这是由于节点性能降低导致所接收任务的减少数目要小于节点性能提高导致所接收任务的增加数目.从图 6(a)中可以看出尽管节点的异构性变化较大,但 TPSS 的调度成功率一直高于 DSRF 和 SAEDF 的调度成功率.同时,在参数 *powerSpan* 的值为 50 时,RF 的调度成功率高于 TPSS 的调度成功率,但是此时,RF 随机选择的安全级别最低,因此具有较高的调度成功率.

图 6(b)显示了 TPSS、DSRF 和 SAEDF 的安全

级别均值明显高于 RF 的安全级别均值,这是因为 TPSS、DSRF 和 SAEDF 在调度过程中会优化任务的安全级别,而 RF 没有考虑到这一点.从图 6(b)中可以看出,当 *powerSpan* 小于 400 时,SAEDF 的安全级别均值高于 DSRF 的安全级别均值,但是 TPSS 相对于 SAEDF 却具有较高的安全级别均值,这是因为 TPSS 中的 FMSL 算法可使得任务的整体安全级别进一步提高.

从图 6(c)可是看出,TPSS 的安全级别标准误差小于 DSRF 和 SAEDF 的安全级别标准误差,因为 FMSL 算法通过调整任务的安全级别,减小了任务安全级别之间的差异.此外,图 6(c)显示了有时 RF 相对 TPSS 具有更低的安全级别标准误差,这是因为 RF 随机选取的安全级别均较低,因此安全级别的差异较低.

图 6(d)显示了整体系统性能,TPSS 的整体系统性能平均高于 RF、SAEDF 和 DSRF 的 OPS 48.3%、56.7%和 88.7%.因此 TPSS 适合于异构集群系统环境.

6　结　论

本文提出了一种异构集群系统中处理安全关键实时应用的调度器模型.在该调度器模型的基础上,提出了一种 2 阶段的调度策略——TPSS.在第 1 阶段,提出了一种自适应调度算法 DSRF,用于任务的实时调度.DSRF 能够根据系统的负载情况,自适应地调整调度目标.当系统负载较重时,DSRF 算法在满足系统基本安全保障的同时,将调度成功率作为调度的主要目标;当系统负载较轻时,DSRF 算法能够在保证系统具有较高调度成功率的基础上,充分利用任务截止期前的空闲时间提高任务的安全级别.在第 2 阶段,提出了一种用来为所接收任务提供公平安全服务的算法 FMSL.FMSL 算法在不降低任务调度成功率的基础上,通过减小节点局部队列中任务之间的安全级别差异来为任务提供较为公平的服务,从而降低了任务被攻击的概率.另外,FMSL 算法在调整任务安全级别的过程中能够进一步提高任务的整体安全级别.最后,通过大量的模拟实验对 TPSS、RF、SAEDF 和 DSRF 进行了比较,实验结果表明,TPSS 的性能优于其它算法,适合于异构集群系统环境,尤其是任务达到速度变化较大、节点动态加入或退出集群等情况,使得系统具有较强的安全性和灵活性.

参　考　文　献

[1] Hwang K, Xu Zhi-Wei. Scalable Parallel Computing: Technology, Architecture, Programming. USA: McGraw-Hill, 1998

[2] Qin Xiao, Jiang Hong. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. Journal of Parallel and Distributed Computing, 2005, 65(8): 885-900

[3] Zhu Xiao-Min, Lu Pei-Zhong. Scheduling of real-time signal processing in cluster-based software radio systems. Journal of Software, 2009, 20(3): 766-778(in Chinese)
(朱晓敏, 陆佩忠. 集群软件无线电系统中实时信号处理调度研究. 软件学报, 2009, 20(3): 766-778)

[4] Zhu Xiao-Min, Lu Pei-Zhong. Study of scheduling for processing real-time communication signals on heterogeneous clusters//Proceedings of the 9th International Symposium on Parallel Architectures, Algorithms, and Networks. Sydney, Australia, 2008: 121-126

[5] Klimeck G, McAuley M, Deen R, Oyafuso F, Yagi G, DeJong E M, Cwik T A. Near real-time parallel image processing using cluster computers//Proceedings of the 1st International Conference on Space Mission Challenges for Information Technology. Pasadena, California, USA, 2003: 13-16

[6] Chang Hsi-Ya, Huang Kuo-Chan, Shen Cherng-Yeu, Tcheng Shou-Cheng, Chou Chaur-Yi. Parallel computation of a weather model in a cluster environment. Computer-Aided Civil and Infrastructure Engineering, 2001, 16(5): 365-373

[7] Krishna C M, Shin K G. Real-Time Systems. USA: McGraw-Hill, 1997

[8] Atdelzater T F, Atkins E M, Shin K G. QoS negotiation in real-time systems and its applications to automated flight control. IEEE Transactions on Computers, 2000, 49(11): 1170-1183

[9] Beccari G, Caselli S, Zanichelli F. A technique for adaptive scheduling of soft real-time tasks. Real-Time Systems, 2005, 30(3): 187-215

[10] Pourzandi M, Gordon D, Yurcik W, Koenig G A. Clusters and security: Distributed security for distributed systems//Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid. Cardiff, UK, 2005: 96-104

[11] Zhang Yan-Yong, Sivasubramaniam A, Moreira J, Franke H. Impact of workload and system parameters on next generation cluster scheduling mechanisms. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(9): 967-985

[12] Ullman J D. NP-complete scheduling problems. Journal of Computer and System Sciences, 1975, 10(3): 384-393

[13] Subramani V, Kettimuthu R, Srinivasan S, Johnston J, Sadayappan P. Selective buddy allocation for scheduling parallel jobs on clusters//Proceedings of the IEEE International Conference on Cluster Computing. Chicago, USA, 2002: 107-116

[14] Vallee G, Morin C, Berthou J-Y, Rilling L. A new approach to configurable dynamic scheduling in clusters based on single system image technologies//Proceedings of the 17th International Parallel and Distributed Processing Symposium. Nice, France, 2003: 22-26

[15] Braun T D, Siegal H J, Beck N, Boloni L L, Maheswaran M, Reuther A I, Robertson J P, Theys M D, Yao B, Hensgen D, Freund R F. A comparison study of static mapping heuristics for a class of meta-tasks on Heterogeneous computing systems//Proceedings of the 8th Heterogeneous Computing Workshop. San Juan, Puerto Rico, 1999: 15-29

[16] Maheswaran M, Ali S, Siegel H J, Hensgen D, Freund R F. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing, 1999, 59(2): 107-131

[17] Beccari G, Caselli S, Zanichelli F. A technique for adaptive scheduling of soft real-time tasks. Real-Time Systems, 2005, 30(3): 187-215

[18] Topcuoglu H, Hariri S, Wu M Y, Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274

[19] He Li-Gang, Jarvis S A, Spooner D P, Nudd G R. Dynamic scheduling of parallel real-time jobs by modeling spare capabilities in heterogeneous clusters//Proceedings of the IEEE International Conference on Cluster Computing. Kowloon, Hong Kong, China, 2003: 2-10

[20] Palis M A. Online real-time job scheduling with rate of progress guarantees//Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks. Metro Manila, Philippines, 2002: 57-62

[21] Manimaran G, Murthy C S R, An efficient dynamic scheduling algorithm for multiprocessor real-time systems. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(3): 312-319

[22] Li Wen-Ming, Kavi K, Akl R. A non-preemptive scheduling algorithm for soft real-time systems. Computers and Electrical Engineering, 2007, 33(1): 12-29

[23] Zhu Xiao-Min, Lu Pei-Zhong. Multi-dimensional scheduling for real-time tasks on heterogeneous clusters. Journal of Computer Science and Technology, 2009, 24(3): 434-446

[24] Boyer W F, Hura G S. Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments. Journal of Parallel and Distributed Computing, 2005, 65(9): 1035-1046

[25] Baruah S K, Goossens J. Rate-monotonic scheduling on uniform multiprocessors. IEEE Transactions on Computers, 2003, 52(7): 966-970

[26] Ghosh S, Melhem R, Mosse D. Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems, 1997, 8(3): 272-284

[27] Son S H, Zimmerman R, Hansson J. An adaptable security manager for real-time transactions//Proceedings of the 12th Euromicro Conference on Real-Time Systems. Stokholm, Sweden, 2000: 63-70

[28] Yurcik W, Meng X, Kiyancla N. NVisionCC: A visualization framework for high performance cluster security//Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security. Fairfax, VA, USA, 2004: 133-137

[29] Koenig G A, Meng X, Lee A J, Treaster M, Kiyanclar N, Yurcik W. Cluster security with NVisionCC: Process monitoring by leveraging emergent properties//Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid. Cardiff, UK, 2005: 121-132

[30] George B, Haritsa J. Secure transaction processing in firm real-time database systems//Proceedings of the ACM SIGMOD International Conference on Management of Data. Tucson, Arizona, USA, 1997: 462-473

[31] Ahmed Q, Vrbsky S. Maintaining security in firm real-time database systems//Proceedings of the 14th Annual Computer Security Application Conference. Phoenix, Arizona, 1998: 83-90

[32] Xie Tao, Qin Xiao. Scheduling security-critical real-time applications on clusters. IEEE Transactions on Computers, 2006, 55(7): 864-879

[33] Xiong Sheng-Wu, Zhao Yong-Xiang, Xu Ning. SAREC-GA: A security-aware real-time scheduling algorithm with genetic algorithm//Proceedings of the 6th International Conference on Machine Learning and Cybernetics. Hong Kong, China, 2007: 3122-3127

ZHU Xiao-Min, born in 1979, Ph. D., lecturer. His current research interests include cluster computing, and real-time system.

LU Pei-Zhong, born in 1961, professor, Ph. D. supervisor. His current research interests include information security, high performance computing, and communication.

Background

This work is supported by the National Nature Science Foundation of China (60673082), and the Special Funds of Authors of Excellent Doctoral Dissertation in China (200084). Nowadays, increasing attention has been drawn towards the problem of security in the context of clusters. However, integrating security services with scheduling algorithms is an open issue for security-critical real-time applications on heterogeneous clusters. Most existing scheduling algorithms only strive to guarantee the timing constraints without considering the security needs for security-critical applications. At the same time, the scheduling algorithms for real-time applications with security requirements should be self-adaptive according to the system's burden and can provide higher quality security services on the promise of higher guarantee ratio. In this paper, a novel scheduling strategy TPSS is proposed to enhance the security levels for security-aware real-time applications on heterogeneous clusters. The TPSS strategy significantly improves the security quality of real-time applications on heterogeneous cluster, and guarantee the system with flexibility and reliability.