面向SOA关键型系统的混合关键度驱动自适应资源管理

张 奕 蔡皖东
(西北工业大学计算机学院 西安 710072)

摘 要 针对运行于开放、不确定环境下的SOA架构的多使命复杂关键型系统,提出了一种以混合关键度为驱动的动态分级自适应资源分配算法。它通过构造闭环反馈的分布式控制环,在线动态调整任务QoS,针对不同关键度的任务动态调整资源分配。该方法不但在统计意义上确保了关键型系统的实时性,而且实验证明该方法能有效提高系统在高负载条件下的实际资源利用率和降低关键任务的死线错过率,保证了关键型系统的高可靠性。

关键词 面向服务架构体系,自适应资源管理,安全/使命关键系统,关键度驱动,基于组件,服务质量
中图法分类号 TP393.07 **文献标识码** A

Mix-criticality Driven QoS Adaptive Resource Management for SOA-based Critical Systems

ZHANG Yi CAI Wan-dong
(School of Computer Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract SOA-based distributed and embedded safety & mission system(DESMCS) executes in open environments which have different dependability requirements. This paper presented a novel mix-criticality driven QoS adaptive dynamic resource management architecture, which provides middleware services for QoS- and criticality-based resource allocation and adaptation across heterogeneous computing nodes and communication networks. The architecture presented in this paper overcomes the disadvantages of traditional feedback-based static resource management techniques and provides layered mix-criticality based resources allocation services. Finally, experiments demonstrate that it enables the DESMCS to dynamically react to changing resource demands or resource availability and better resource utilization, improved dependability.

Keywords SOA, Adaptive resource management, Safety & mission critical system, Criticality-driven, Component-based, QoS

1 引言

关键型系统是指系统功能一旦失效将可能引起事故的系统[1]。关键型系统广泛应用于航空、航天、汽车、工业控制等重要领域,因此其可靠性备受关注。本文主要关注面向SOA的分布式嵌入式安全/使命关键系统(Distributed and Embedded Safety & Mission Critical System, DESMCS)。DESMCS是一种典型的通过有线或无线网络连接的大规模分布式实时嵌入式(Distributed Real-time and Embedded, DRE)系统。随着关键型应用的日益广泛和微处理器技术的不断革新,DESMCS功能日趋复杂,并执行于开放的操作环境,其系统运行的条件、输入的工作负载和可用资源都具有高度动态性、不确定性。因此,当前保证DESMCS高可靠性所面临的热点问题有:

(1)采用面向服务的架构(Service-oriented Architecture, SOA)设计当前大规模复杂DESMCS。

(2)如何在可用资源和输入工作载荷动态变化的情况下,获取端到端的服务质量(Quality of Service, QoS)。

(3)DESMCS中执行的使命具有混合关键度,针对不同关键度的使命需要采用不同的自适应策略,动态地调整使命执行参数:对于关键应用使命,系统提供的资源要考虑最坏情况下的载荷,以确保系统中关键使命的正确执行;对于非关键使命,要考虑动态资源分配,提高资源的利用率。既要保证系统的可靠性,又要保证系统性能的最大化。

(4)通过QoS可感知的自适应策略,使DESMCS系统组件能够在系统运行时自动添加和删除。

(5)传统DRE系统的资源管理紧耦合于特定的中间件平台,如果不重新设计中间件平台关键部分,就不能改进资源管理算法。当前大型复杂DESMCS由大量异构的节点连接组成,共享有限的资源,运行于动态、不确定的复杂环境。传统的静态QoS策略不支持不确定载荷,也不能保证任务请求的资源量随外界的变化而得到满足。

针对以上热点问题,本文提出了一种新的由混合关键度驱动的QoS可感知的动态分级自适应资源综合管理机制,主要有如下特点:

(1)针对采用SOA架构的复杂DESMCS,解决了传统应用和资源调配的静态耦合问题,将资源分配策略封装成服务并发布,随外界运行环境变化灵活选择服务的具体实现。

到稿日期:2010-10-28 返修日期:2011-01-17 本文受国家高技术发展计划(863)(2009AA01Z424)资助。
张 奕(1977—),女,博士生,讲师,CCF会员,主要研究方向为关键型系统及中间件研究,E-mail:zywait@nwpu.edu.cn;蔡皖东(1951—),男,博士,教授,博士生导师,主要研究方向为网络信息安全、高可信系统。

(2)基于反馈控制的动态闭环资源分配算法,更适合不确定性运行环境和变化的资源需求,保证系统性能的最大化。

(3)采用分级自适应资源管理策略,克服以往集中式资源管理算法的中心节点成为整个系统性能瓶颈的问题,保证DESMCS系统功能的可靠性。

(4)针对DESMCS使命关键度的不同级别,给出一种混合关键度驱动的动态QoS自适应策略,提供故障隔离和可组合性机制,在保证系统高可靠性的前提下,提高DESMCS的灵活性和整体性能。

2 相关研究

传统的DESMCS系统采用相对静态的开发和分析技术来实现资源的分配、调度和管理,其局限性如下:

(1)传统静态的方法面向封闭式的分布式实时系统,不适合于采用开放式的系统结构,运行于动态、不确定复杂环境下的新一代DESMCS。

(2)传统的资源管理算法都比较简单,资源按照被应用任务独占的方式分配,在整个系统运行过程中资源与应用任务之间的关系保持不变。

(3)整个系统设计是基于最坏情况所需资源来设计的,这使得系统在动态、不确定环境时缺乏灵活性,并造成大量的计算和网络资源浪费。

(4)系统维护成本极高。

而当前有关DRE系统资源管理方面的研究成果,主要分成以下3类:

(1)反馈控制调度(Feedback Control Scheduling, FCS)[2-7],满足开放系统结构的DRE系统严格端到端QoS需求。但FCS没有针对特定的安全/使命关键应用领域提供分层的服务,同时没有对当前流行的基于组件的集成控制提供有力的支持。

(2)资源分配和控制引擎(Resource Allocation and Control Engine, RACE)[8]和装箱算法[9],基于系统平均或最坏情况下的资源需求,为系统提供静态固定的资源分配算法。这种静态的开环资源分配算法不能满足系统动态变化的资源需求,更不适合于开放DESMCS不确定的复杂运行环境。

(3)Draco[10]针对资源有限的嵌入式系统提供了有效的动态资源管理算法,但没有针对DESMCS不同关键度的应用使命提供不同的自适应资源管理策略。

3 系统模型

DESMCS的某项具体工作由使命(mission)完成。使命是一个抽象的概念,表示具有优先级的一组组件(component)的集合。一个组件内包含多个应用对象,一个应用对象由变量和方法组成。应用对象的方法通过任务(task)来调度。使命的关键度(criticality)就是优先级,表示使命的相对重要程度。使命的关键度在DESMCS运行时是动态改变的。高关键度的使命比低关键度的使命需要更高的资源利用率和QoS需求。

假设DESMCS的资源集由$\{R_1,R_2,\cdots,R_l\}$组成。为了讨论简单起见,本文只讨论单一处理器资源(假设$l=1$)。因此,DESMCS由执行于n个处理器$P=\{p_i\mid 1\leqslant i\leqslant n\}$的$m$个任务$T=\{\tau_i\mid 1\leqslant i\leqslant m\}$组成。任意任务$\tau_i$由子任务集合$\{\tau_{ij}\mid 1\leqslant j\leqslant m_i\}$组成。子任务之间有严格的优先关系$\mathcal{R}_i\subseteq T\times T$,子任务$\tau_{ij}$只有在子任务$\tau_{ij-1}$执行完之后才能在处理器上执行[11]。任务$\tau_i$的所有子任务具有相同的执行速率$r_i$,因此可以通过调整第一个子任务的速率来动态调整任务的速率。

任务τ_i的QoS级别定义为QoS_i,τ_i可接受的最差QoS级别和可获得的最好QoS级别,分别用QoS_i^{min}和QoS_i^{max}描述,$QoS_i^{min}\leqslant QoS_i\leqslant QoS_i^{max}$。任务$\tau_i$的相对QoS级别用$Q_i$表示[12],$Q_i\in[0,1]$:

$$Q_i=\frac{QoS_i-QoS_i^{min}}{QoS_i^{max}-QoS_i^{min}} \tag{1}$$

DESMCS各种使命的重要程度各不相同(即任务具有不同的关键度),任务τ_i的关键度用w_i表示,并且任意任务的子任务具有相同的关键度。w_i越大,表示任务越重要,意味着任务需要更好的执行性能。因此,当$w_i>w_j$时,可以得到$Q_i>Q_j$。在多任务的DESMCS系统中,资源分配可能导致任务之间的不公平性。由文献[13]可知,公平资源分配的前提条件是

$$\frac{Q_1}{w_1}=\frac{Q_2}{w_2}=\cdots=\frac{Q_m}{w_m} \tag{2}$$

不同的任务τ_i具有不同的执行周期,即不同的执行速率,而不同的应用程序可以执行相同的任务。例如,航空系统是典型的DESMCS系统,其中的雷达管理子系统和导航识别系统都需要执行信号处理任务。不同的应用程序可以执行相同的任务,因此,面向不同应用的任务τ_i具有不同执行速率,在不会引起系统故障的前提下,速率动态调整范围用$[r_{min,i},r_{max,i}]$表示。根据端到端调度方法,假设任务的时限等于其周期[13]。文献[14]提供了确保任务速率满足可调度范围的方法。

4 基于闭环反馈的QoS自适应资源分配原理

4.1 资源利用和QoS级别

资源分配和QoS级别之间的关系由资源消耗函数描述[13]。由于本文只讨论处理器的CPU资源,因此任务τ_i的CPU利用率u_i表示为$u_i=f(Q_i)$。由于DESMCS运行于动态、不确定环境,τ_i的CPU资源利用率是动态变化的。t_k表示第k个采样周期,在时间间隔$[t_k,t_{k+1}]$的CPU利用率用$u_i(k)$表示:$u_i(k)=c_ir_i(k)Q_i$,其中,c_i表示任务τ_i的估计执行时间,$r_i(k)$表示任务τ_i在第k个采样周期的任务调用速率。而总的系统利用率定义为所有任务的利用率之和:

$$u(Q_1,Q_2,\cdots,Q_m)=\sum_{i=1}^{m}u_i(Q_i) \tag{3}$$

系统设计时的处理器利用率估计的设定值用$B=[B_1\cdots B_n]^T$表示。优化控制目标是动态选择任务速率,使每个处理器的利用率和设计时估计值相差的程度最小。利用率控制问题可以抽象为动态限制优化问题[15]:

$$\min_{\{r_i\mid 1\leqslant i\leqslant m\}}\sum_{j=1}^{n}|B_j-u_i| \tag{4}$$

约束条件为

$$r_i\in[r_{min,i},r_{max,i}]\ (1\leqslant i\leqslant m)$$

4.2 QoS自适应闭环反馈控制设计

在第k个采样周期,DESMCS的各任务当前QoS级别用$Q_1(k),Q_2(k),\cdots,Q_m(k)$描述。所有任务的平均QoS级别表示为[13]

$$\overline{Q}(k)=\frac{1}{m}\sum_{i=1}^{m}Q_i(k) \tag{5}$$

由于 DESMC 的系统负载和资源分配是动态变化的，因此，传统的反馈控制调度方法很难适用于 DESMCS。本文采用一种新的基于任务当前 QoS 级作为反馈信息的控制规则。

假设忽略系统错误，CPU 资源分配可由以下系统模型描述：

$$u_i(k+1)=u_i(k)+\alpha(\bar{Q}(k)-Q_i(k)) \tag{6}$$

当 $\bar{Q}(k)-Q_i(k)>0$，意味着在下一个采样周期任务需要在更高的 QoS 级执行；当 $\bar{Q}(k)-Q_i(k)<0$，意味着在下一个采样周期任务执行时，需要适度降低 QoS 级。其中，控制参数 α 的确定必须要满足以下两个条件：

• 可行性：对于每个采样周期，$u_i(k)$ 是正数（$i=1,2,\cdots,m$）。

• 可控性：在线闭环反馈控制循环的优化控制目标是系统错误趋向于 0，CPU 利用率趋向于常量。

$$\sum_{i=1}^{m}u_i(k+1)=\sum_{i=1}^{m}u_i(k)+\alpha\sum_{i=1}^{m}(\bar{Q}(k)-Q_i(k))=\sum_{i=1}^{m}u_i(k) \tag{7}$$

为了满足以上两个条件，将资源消耗函数 $f(Q_i)$ 的有限上下边界定义为 lb_i 和 ub_i：$0<lb_i\leqslant\frac{df(Q_i)}{dQ_i}\leqslant ub_i<\infty$。由文献[13]可知，在公平 QoS 管理基础上的资源分配，选取控制参数 α 满足 $0<\alpha\leqslant\min_{i\in\{1,\cdots,m\}}lb_i$。

5 SOA 关键型系统资源管理的体系结构

5.1 分级资源管理框架

面向 SOA 的典型安全/使命关键系统（如航空分布式实时系统、无人机的紧急情况反应系统、空中防御系统[13]）由大量的处理器节点组成，并通过高速通信接口连接。这些计算节点按照所提供的不同服务进行分类，和传统的某一节点属于特定应用的体系结构不同，所有可计算资源由系统应用所共享。因此，过去集中式的资源管理机制不适合当前这种大型的面向 SOA 的 DESMCS 系统。当系统的规模不断增加，集中式的资源管理算法的计算时间随着端到端应用任务增加，呈指数比例的增长[16]，集中式资源管理的中心节点成为性能瓶颈或单一的故障点。为了解决集中式资源管理的瓶颈问题，对系统资源进行资源分组，并采用分级自适应资源管理算法是一种有效途径，如图 1 所示。

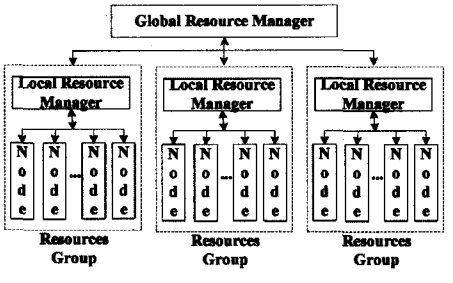

图 1 分级自适应资源管理框架

局部资源管理器（Local Resource Manager，LRM）和具体任务相关，当新任务到达或某任务完成时触发 LRM。LRM 专门维护和节点相关的局部元数据，元数据是动态或静态系统资源状态的数据集合。资源组对应一个子系统。LRM 管理一组资源的分配、QoS 配置和运行时的自适应调整。全局资源管理器（Global Resource Manager，GRM）负责 LRM 之间的协调，保证整个系统的正确操作，并能调节特定性能参数（例如服务的级别），以保证 DESMCS 的系统稳定性。

5.2 基于混合关键度 QoS 自适应资源管理机制

资源管理的目的就是要正确配置分布式系统使之能够提供充足的服务[17]。本文所提出的资源管理机制由分布在子系统上的 LRM 和分布于每个处理器上的 QoS 自适应动态资源管理框架组成，如图 2 所示。

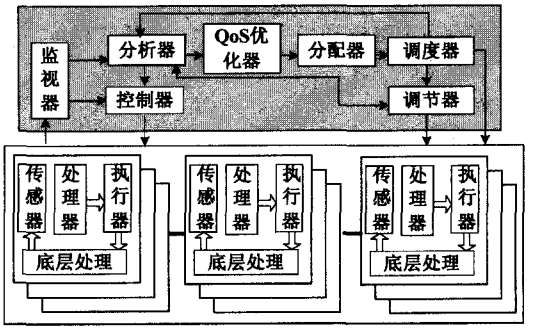

图 2 QoS 自适应动态资源分配和控制框架

QoS 自适应动态资源管理框架一方面要支持静态的资源保证，满足关键应用最差情况下的资源需求。另一方面，对于非关键应用需要提高资源分配的灵活性。为了满足混合关键度的 DESMCS 以上应用需求，本文采用基于反馈原理的闭环资源管理方法，设计 QoS 自适应动态资源管理框架。

（1）监视器：资源监视器跟踪不同系统资源的利用率，例如 CPU、内存和网络带宽。资源监视器为其它组件提供系统运行时的资源消耗信息。监视器同时跟踪关键 QoS 属性，例如端到端延迟、实时行为、资源利用条件、容错能力、可靠性等。LRM 根据所监视的相应 QoS 参数来确定当前的资源需求。

（2）分析器：根据系统的动态变化，产生新的满足 QoS 要求的资源分配计划。

（3）分配器：根据其它组件的资源需求和当前可用的系统资源为之分配资源。资源分配器基于反馈信息跟踪每个资源管理任务的资源需求，并实现不同的资源分配算法。

（4）调节器：对资源分配进行一定的调整，用以保证系统资源的可控性。

（5）控制器：根据不同的反馈控制算法和监视器提供的反馈信息，计算端到端的自适应决策，确保各种应用使命（特别是高关键度应用使命）的 QoS 需求得到满足。

（6）调度器：接收任务的死线和处理时间，根据指定的调度算法（例如最早死线优先调度、速率单调调度、基于紧急度调度、基于速率的最早时限优先调度）进行任务调度。

6 混合关键度驱动的分级自适应资源分配算法

随着 DESMCS 功能日趋复杂，不同关键度的应用具有各自不同的资源需求和 QoS 级别。为了保证 DESMCS 的高可靠性，必须采用关键度可感知的动态分级自适应资源分配算法，保证各级关键度应用的正确执行。当 LRM 接收到改变任务关键度的命令后，LRM 使用可行性分析寻找新的资源分配计划，满足任务改变关键度后的 QoS 需求。

任务的关键度由以下两个参数确定：平均请求处理成本和 QoS 参数。平均请求处理成本代表相对 QoS 级别 Q_i 的请求处理时间，用变量 $\cos t_i$ 表示。QoS 参数代表每个 QoS 级

别必须保证的平均延迟时间，用变量 la 表示：$w_i=\frac{\cos t_i}{la_i}$。

混合关键度驱动的动态分级资源分配算法由基于关键度的抢占服务实现，其中基础算法采用基于速率的最早时限优先(Rate-based Earliest Deadline, RBED)调度算法[18]，保证LRM满足高关键度任务的死线。当有新任务到达时，如果系统可用资源不充足，通过调整中低关键度任务的速率，从而保证DESMCS系统资源利用率最大化。具体算法实现如图3所示。

```
while(资源充足) do
  LRM分配任务执行;
    for i=1 to m do
      AdjustQoS(r_i, r_i)，满足 Q_i∈[0,1];
    end for
end while
while(资源不充足) do
  for i=1 to m do
    if(w_i==高关键度) then
      保证 QoS_i^min;
      不允许资源抢占;
    end if
    if(w_i<=可调度关键级) then
      允许资源抢占，保证资料利用率最大化;
      rate_adjust(i, w_i) do
        Δr_i=r_i'-r_min;
        if(w_i != w_i^old) then
          Δu_i=Δc_iΔr_iQ_i;
          w_i^old = w_i;
          return Δu_i;
        end if
      end if
    end for
end while
```

图3 分级自适应资源分配伪代码

7 仿真实验

为了验证算法的有效性，搭建仿真环境开展上述实验，对C++实现的任务发生器和在MATLAB环境下实现本文提出的混合关键度驱动的分级自适应资源分配算法进行仿真。假设DESMCS执行3个不同关键度的使命：声纳信号处理使命(关键度设定为5)、实时图像解码使命(关键度设定为10)、导弹防御使命(关键度设定为15)。由6个资源控制任务来完成系统资源管理的要求，利用RBED算法运行一个关键度为15的高关键度任务、两个关键度为10的中等关键度任务和3个关键度为5的低关键度任务。任务的仿真参数设置如表1所列。

表1 任务仿真参数

	1	2	3	4	5	6
lb_i	0.43	0.43	0.43	0.32	0.32	0.16
c_i	116	130	220	245	300	500
Q_i	0.32	0.32	0.32	0.64	0.64	0.96
w_i	5	5	5	10	10	15
$[r_{min,i}, r_{max,i}]$	[0,0.4]	[0,0.33]	[0,0.38]	[0,0.5]	[0,0.46]	[0,0.65]

假设DESMCS期望延迟是0.6s，由图4的实验结果看出，在采样循环数250附近，由于缺乏资源，关键度为5的低关键度使命暂停。当关键度为15的最高关键度使命出现时，基于关键度的抢占服务调整关键度为10使命的执行速率，保证最高关键度使命接收所需资源，最高关键度使命的延迟率得到了保障，从而DESMCS的可靠性得到了保证。

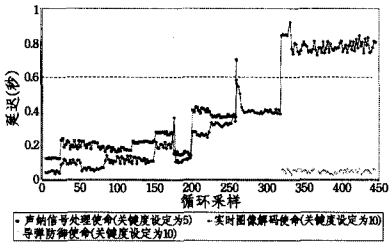

图4 不同使命QoS性能

由图5的实验结果看出，基于RBED的静态资源分配算法的处理器，利用率只有56%左右；而采用关键度驱动的动态自适应资源分配算法，处理器利用率为87%左右。在保证系统高可靠性的前提下，提高了DESMCS的灵活性和整体性能。

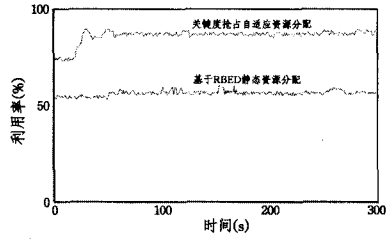

图5 性能曲线

结束语 运行于开放、不确定环境下的复杂DESMCS系统，由于其应用任务的多样性和执行时间的不确定性，导致了目前资源管理算法系统利用率低的问题。本文提出的基于闭环反馈机制的资源管理机制，采用混合关键度驱动的动态分级资源分配算法，通过基于关键度的抢占服务很好地满足了不同关键度任务动态变化的资源请求。实验结果证明，该动态算法不但在统计意义上满足了DESMCS的实时性，并且在兼顾QoS的基础上为DESMCS提供了更好的性能确保。

参 考 文 献

[1] Medikonda B S, Panchumarthy S R. A Framework for Software Safety in Safety-Critical Systems[J]. ACM SIGSOFT Software Engineering, 2009, 34(2): 1-9

[2] Cucinotta T, Palopoli L, Marzario L, et al. Adaptive reservations in a Linux environment[C]// Proc. 10th IEEE RTAS. Toronto, Canada, 2004: 238-245

[3] Abdelzaher T F, Stankovic J A, Lu Chen-yang, et al. Feedback Performance Control in Software Services[J]. IEEE Control Systems, 2003, 23(3)

[4] Lu Chen-yang, Stankovic J A, Gang Tao, et al. Feedback control real-time scheduling: framework, modeling, and algorithms[J]. Real-time Systems, 2002, 23(1/2): 85-126

[5] Thimm H, Klas W, Walpole J, et al. Managing Adaptive Presentation Executions in Distributed Multimedia Database Systems [C]// Proc. Intl. Workshop on Multimedia Database Management Systems. 1996

[6] Steenkiste P. Adaptation Models for Network-aware Distributed Computations[C]// 3rd Workshop on Communication, Architec-

ture, and Applications for Network-based Parallel Computing (CANPC99). Orlando, FL: IEEE Springer, Jan. 1999

[7] Bergmans L, Halteren A V, Pires L, et al. A QoS Control Architecture for Object Middleware[C]// 7th Intl. Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS 2000). Enschede, The Netherlands, LNCS 1905, Oct. 2000: 117-131

[8] Shankaran N, Roy N, Schmidt D C, et al. Design and performance evaluation of an adaptive resource management framework for distributed real-time and embedded systems[J]. EURASIP J. Embedded Syst., 2008

[9] de Niz D, Rajkumar R. Partitioning bin-packing algorithms for distributed real-time systems[J]. International Journal of Embedded Systems, 2006, 2(3/4): 196-208

[10] Mart'l P, Lin Cai-xue, Brandt S A, et al. Draco: Efficient resource Management for Resource-Constrained Control Tasks [J]. IEEE Transactions on Computers, 2009, 58(1)

[11] Sun J, Liu J W-S. Synchronization Protocols in Distributed Real-Time Systems[C]// Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '96). 1996

[12] Harada F, Ushio T, Nakamoto Y. Adaptive Resource Allocation Control for Fair QoS Management [J]. IEEE Transaction on Computers, 2007, 56(3): 344-357

[13] Juedes D, Drews F, Welch L, et al. Heuristic resource allocation algorithms for maximizing allowable workload in dynamic, distributed, real-time systems[C]// 12th Workshop on Parallel and Distributed Real-time Systems. 2004

[14] Lehoczky J P. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadline [C] // Proc. IEEE Real-time Systems Symp (RTSS '90). 1990

[15] Wang X, Jia D, Lu C, et al. DEUCON: Decentralized end-to-end utilization control for distributed real-time systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2007, 18(7)

[16] Shankaran N, Kinnebrew J, Koutsoukos X, et al. An Integrated Planning and Adaptive Resource Management Architecture for Distributed Real-time Embedded Systems [J]. IEEE Transactions on Computers, Special Issue on Autonomic Network Computing, 2009

[17] Ver'lssimo P, Rodrigues L. Distributed Systems for System Architects[M]. Norwell, MA, USA: Kluwer Academic Publishers, 2001

[18] Brandt S A, Banachowski S, Lin C, et al. Dynamic Integrated Scheduling of Hard Real-time, Soft Real-time and Non-Real-time Processes[C] // Proc. 24th IEEE Real-time Systems Symp (RTSS '03). Dec. 2003: 396-407

(上接第 78 页)

图 3 是归一化吞吐量随归一化检测时间的变化曲线，并比较了采用序贯检测与采用能量检测时的归一化吞吐量。其中“SD”代表序贯检测，“ED”代表能量检测。在图 3 中，归一化检测时间的取值范围并非 0 到 1，这是由于检测概率必须为 P_{d0} 的限制条件造成的。从图 3 中可以看出，对于相同的检测概率下限 P_{d0}，采用序贯检测的归一化吞吐量明显大于采用能量检测器的，这一现象也恰恰说明了序贯检测较能量检测的优越性。当 $P_{d0}=0.9$ 时，采用序贯检测器的最大归一化吞吐量 $\overline{R_{0\_SD\_Opt}}$ 比采用能量检测器的最大归一化吞吐量 $\overline{R_{0\_ED\_Opt}}$ 大 25%；当 $P_{d0}=0.95$ 时，最优的 $\overline{R_{0\_SD\_Opt}}$ 比最优的 $\overline{R_{0\_ED\_Opt}}$ 大将近 30%；当 $P_{d0}=0.99$ 时，最优的 $\overline{R_{0\_SD\_Opt}}$ 比最优的 $\overline{R_{0\_ED\_Opt}}$ 大 45%。

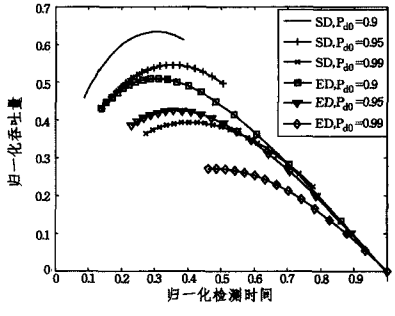

图 3 归一化吞吐量随归一化检测时间的变化曲线

结束语 在 CR 系统中，CR 用户需要检测主用户是否存在，若检测到主用户不存在，则可以接入主用户的频段进行数据传输。在参数设置时，CR 用户一方面要对主用户有足够的保护，另一方面要保证 CR 系统的吞吐量达到最大。本文研究了 CR 用户采用序贯检测时 CR 系统吞吐量的优化问题。文中首先分析了 CR 用户采用序贯检测器时系统的吞吐量，证明了在对主用户有足够保护的条件下，存在唯一最优的目标虚警概率和目标检测概率使得系统吞吐量最大，并给出最优参数设置的搜索方法。仿真结果验证了分析结果，并表明在 CR 系统中，采用序贯检测比采用能量检测可达到更大的吞吐量。

参 考 文 献

[1] Spectrum policy task force repot[R]. 02-135. Federal Communications Commission, Nov. 2002

[2] Haykin S. Cognitive radio: Brain-empowered wireless communications [J]. IEEE Journal Selected Areas in Communications, 2005, 23(2): 201-220

[3] Cabric D, Mishra, S M, Broderse R W. Implementation issues in spectrum sensing for cognitive radios [C]// Proc. Asilomar Conference on Signals, Systems, and Computers. USA: IEEE Press, 2004: 772-776

[4] Tandra R, Sahai A. SNR Walls for Signal Detection [J]. IEEE Journal of Selected Topics in Signal Processing, 2008, 2(1): 4-17

[5] Wald A. Sequential Analysis [M]. New York: Dover Publications, 2004: 1-224

[6] Choi K W, Jeon W S, Jeong D G. Sequential detection of cyclostationary signal for cognitive radio systems[J]. IEEE Transactions on Wireless Communications, 2009, 8(9): 4480 - 4485

[7] Yan Xin, Zhang Hong-hai, Rangarajan S. SSCT: A Simple Sequential Spectrum Sensing Scheme for Cognitive Radio [C]// Global Telecommunications Conference. Honolulu: IEEE Press, 2009: 1-6

[8] Nikhil K, Ahmed T. A performance study of novel sequential energy detection methods for spectrum sensing[C]// IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP). Dallas, Texas, USA: IEEE Press, 2010: 3090-3093

[9] Boyd S, Vandenberghe L. Convex optimization [M]. UK: Cambridge University Press, 2004: 1-716

[10] 陈宝林. 最优化理论算法[M]. 北京，清华大学出版社，2005: 1-468