软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(7):1440−1456 [doi: 10.3724/SP.J.1001.2011.03833]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

异构集群系统中具有 QoS 需求的实时任务容错调度*

朱晓敏+, 祝江汉, 马满好

(国防科学技术大学 信息系统工程重点实验室,湖南 长沙 410073)

Fault-Tolerant Scheduling for Real-Time Tasks with QoS Requirements on Heterogeneous Clusters

ZHU Xiao-Min+, ZHU Jiang-Han, MA Man-Hao

(Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: xmzhu@nudt.edu.cn

Zhu XM, Zhu JH, Ma MH. Fault-Tolerant scheduling for real-time tasks with QoS requirements on heterogeneous clusters. *Journal of Software*, 2011,22(7):1440−1456. http://www.jos.org.cn/1000-9825/3833.htm

Abstract: Fault-Tolerant scheduling, an effective means of improving a system's performance, plays a significant role in scheduling research. Despite the fact that fault-tolerant scheduling has been extensively proposed for real-time tasks on clusters, QoS (quality of service) requirements for some tasks have not been considered. This paper proposes a fault-tolerance scheduling algorithm FTQ (fault-tolerant QoS-based scheduling) for real-time tasks with QoS needs on heterogeneous clusters. FTQ adopts the primary/backup model and takes the timing constraints of tasks, QoS requirements of tasks, reliability of systems, and system resource utilization into account. FTQ can adjust the QoS levels of real-time tasks and the execution schemes of backup copies to improve system flexibility, reliability, schedulability, and resource utilization. The system reliability is quantitatively measured and combined into FTQ, which improves the system performance. Meanwhile, FTQ strives to advance the start time of primary copies and delay the start time of backup copies to make backup copies adopt passive execution scheme, or decrease overlapping sections of primary and backup copies as much as possible to improve resource utilization. FTQ adaptively adjusts the QoS levels of tasks and the execution schemes of backup copies to attain a higher flexibility. The overlapping technology of backup copies is employed. The latest start time of backup copies and its constraints are analyzed. Compared with NOFTQ and DYFARS, FTQ shows obvious superiority with a higher scheduling quality proven by a considerable number of simulated experiments.

Key words: heterogeneous cluster; real-time; scheduling; fault tolerance; heuristic

摘 要: 容错调度是调度问题中一个重要的研究内容,是提高系统可靠性的有效手段.目前已有很多集群系统中实时任务的容错调度算法,但是这些算法都没有考虑到任务的 QoS 需求问题.提出了一种异构集群系统中具有 QoS 需求的实时任务容错调度算法 FTQ(fault-tolerant QoS-based scheduling).该算法采用主版本/副版本(primary/backup,简称 PB)技术,综合考虑了任务的时间限制、任务的 QoS 需求、系统的可靠性和系统资源的利用率,能够自适应地

* 基金项目: 国家安全重大基础研究计划(973)(6136101); 国家高技术研究发展计划(863)(2008AA7070412)
收稿时间: 2009-10-14; 修改时间: 2009-12-28; 定稿时间: 2010-03-11

根据系统负载情况动态地调整任务的 QoS 级别和副版本的执行模式,从而提高了系统的灵活性、可靠性、可调度性和资源的利用率.对系统的可靠性进行了定量分析,并将其引入到容错调度算法中,提高了系统的可靠性.同时,在调度过程中尽量提前主版本的开始时间,推迟副版本的开始时间,以使任务的副版本采用被动执行模式或者使任务主版本和副版本的重叠部分尽量少,提高了资源的利用率.此外,采用了副版本重叠技术,并分析了副版本的最晚开始时间及其约束条件,提高了任务的调度成功率.通过大量的模拟实验,对 FTQ,NOFTQ 和 DYFARS 算法进行了比较.实验结果表明,FTQ 算法的性能优于其他方法,具有更好的调度质量.

关键词: 异构集群;实时;调度;容错;启发式

中图法分类号: TP311 **文献标识码:** A

实时性和可靠性是评价实时系统性能的两项重要指标,而容错技术是实现这两项指标的有效方法[1].集群系统在处理实时任务的过程中,不仅需要保证任务在规定的时限内完成,而且在系统出现故障的情况下,实时任务仍可以继续运行[2].

随着实时应用领域的不断扩大,许多硬实时应用和软实时应用都具有服务质量(QoS)需求.例如,Abdelzaher 等人描述了一种自动飞行控制系统[3],该系统用来模拟 F-16 战斗机的飞行过程.在该系统中,飞机的飞行控制任务包括导航(guidance)、控制器(controller)控制、慢速飞行(slow navigation)、快速飞行(fast navigation)和任务控制(task control).这些任务都必须在一定的时间内完成,以满足实时性.为了提高系统的稳定性,每项任务都可以通过更改任务的执行周期和执行时间来选择不同的 QoS 级别.不同的 QoS 级别可以提供不同的飞行质量.又如,在实时通信信号处理系统中,信号数据可以采用不同的处理算法进行解调或译码[4].通常,时间复杂度较高的信号处理算法具有较好的信号处理质量,即 QoS 级别,但需要较长的处理时间.而时间复杂度较低的信号处理算法正好相反.例如,对于分组 Turbo 码的译码,文献[5]中提出了一种接近最优的迭代算法,但是译码的时间复杂度非常高.为了在译码质量和时间复杂度之间加以折衷,许多学者进行了大量研究,并提出了多种具有不同时间复杂度的译码算法[6,7].

值得注意的是,上面的这些实时应用都具有很高的可靠性需求.由于自动飞行控制系统将被应用于军事作战中,系统一定要保证即便在硬件或软件出现故障时,飞行控制任务仍能在规定的时间内完成,否则将产生灾难性的后果.因此,飞行控制系统必须具有出错导向安全的特性.即,在出错时通过降级或其他措施保证飞机安全着陆.而对于实时信号处理系统,虽然某些信号数据未能在规定的时间内完成,不会导致灾难性的后果,但是,过时或不完整的数据对于用户来说可能变得没有价值,尤其是在当今的军事信息战中.

容错技术是提高系统可靠性的有效手段.通常,容错可以分为硬件容错和软件容错两种.硬件冗余是解决实时系统中永久性错误的有效途径[8].实时容错调度算法是一种采用软件方法解决容错问题的软件容错方法,该方法的优点是不需要额外的硬件代价来提高系统的可靠性[9].近年来,很多专家学者对集群系统中实时任务的容错调度问题进行了大量的研究.与此同时,也有不少基于具有 QoS 需求的实时任务调度算法.但是,还没有关于具有 QoS 需求的实时任务容错调度算法的相关研究,本文所进行的研究正是在这样的背景下提出来的.

本文将传统的实时容错调度问题拓展到具有 QoS 需求的实时容错调度问题,提出了一种动态的实时容错调度算法 FTQ(fault-tolerant QoS-based scheduling),用于调度异构集群系统中独立、非周期、有 QoS 需求的实时任务.

1 相关工作

容错技术在安全关键实时系统中占有非常重要的地位,直接关系到系统的可靠性.而容错调度算法可以通过分配任务的多个版本来实现容错,这是目前非常流行的容错技术[10].有效的容错调度算法能够极大地提高系统的性能,包括系统的可调度性、可靠性、灵活性等等.但是大量的研究表明,集群系统中的多任务调度算法是 NP 完全问题[11].也就是说,在集群调度中不存在最优的线性时间复杂度调度算法.因此,在实际应用中,通常采用接近最优的启发式方法来解决此类调度问题.

主版本/副版本(primary/backup,简称 PB)技术是目前最重要的软件容错方法.在 PB 方法中,一项任务的两个版本——主版本和副版本被分配到两个不同的节点上,同时,通过接受测试(acceptance test)来检查结果的正确性[12].目前,已有很多采用 PB 方法进行容错调度的文献.例如,Ghosh 等人提出了两种技术——释放(deallocation)和重叠(overloading).这两种技术的采用可以通过较少的开销进行容错,同时能够提高系统的可调度性.重叠是指多个副版本可以在同一个节点上的同一个时间槽上重叠,而释放是指当一个任务的主版本成功执行后,其对应的副版本将被删除以回收预留的系统资源[13].Manimaran 等人在文献[13]的基础上进行了改进,考虑了任务之间的资源限制,同时将处理器划分为若干组,从而保证在多个处理器出错的情况下能够实现容错[10].Al-Omari 等人研究了一种 PB 重叠方法,允许一项任务的主版本和其他任务的副版本进行重叠,从而提高了任务的调度成功率[14].上面提到的这几种方法的一个共同特点是,当且仅当在任务的主版本出错时,其副版本才开始执行,这种模式称为被动副版本(passive backup copy)模式.如果任务的副版本仅在主版本所在节点出错的时候执行,那么必须有足够的裕度留给副版本.也就是说,任务截止期和主版本结束的时间差必须大于副版本执行的时间,否则,副版本不能在任务截止期之前完成.前面提到的这几种方法都假设裕度至少大于任务副版本的执行时间,这种假设的限制比较严格,在系统比较繁忙的情况下不可能保证所有任务的裕度都大于任务的执行时间.

与被动副版本模式相对应的另外一种模式是主动副版本(active backup copy)模式,主动副版本模式允许主版本和副版本同时执行.Tsuchiya 等人提出了一种容错方法,该方法允许任务的主版本和副版本执行的开始时间可以不同[15].Yang 等人也提出了一种主动副版本模式的容错调度算法用以提高调度成功率[16],等等.但是,完全采用主动副版本模式的调度方式将会导致在任务裕度较大的情况下,副版本与主版本同时执行造成系统资源的浪费.之后,Al-Omari 等人研究了一种自适应的容错调度方法,该方法根据任务裕度和任务的出错概率计算主版本和副版本的重叠部分大小[17].但是,上面这些容错方法都只考虑了系统同构的情况,而没有考虑系统中处理器(节点)异构的情况,因此不适合异构集群环境.

近来,Qin 等人提出了一种异构系统中实时任务容错调度算法 eFRD[12].该算法对系统的可靠性进行了量化计算,从而在进行容错调度的同时提高了系统的可靠性.但是,该容错调度算法是一种用来处理有依赖关系的实时任务静态调度算法,而本文考虑的实时任务之间没有依赖关系,即是相互独立的.这是因为,有依赖关系的实时任务可以转化为相互独立的任务[18].另外,本文的容错调度算法用来处理动态到达的任务,属于动态调度算法.之后,Luo 等人提出了一种异构系统中可靠性驱动的实时容错调度算法,该容错调度算法动态地考虑了副版本的主动执行模式和被动执行模式,提高了系统的灵活性[19].但是,该算法没有考虑不同任务副版本之间的重叠问题,因此降低了调度成功率.最为重要的是,上面提到的所有容错调度算法都没有考虑为具有 QoS 需求的实时任务提供容错处理这一问题.

本文在研究了大量文献的基础上,提出了一种异构集群系统中具有 QoS 需求的实时任务容错调度算法,该算法综合考虑了任务的时间限制、QoS 需求、系统的可靠性、系统资源的利用率,能够自适应地根据系统负载情况动态地调整任务的 QoS 级别和副版本的执行模式,提高了系统的灵活性、可靠性、可调度性和资源的利用率.

2 容错调度模型

2.1 前提假设

本文所提出的实时容错调度模型是建立在以下前提假设之上的:

(1) 某一时刻系统中只有一个节点出现故障,在下一节点出现故障之前,那些因前一故障而执行失败的任务主版本通过运行任务副版本而可以正确结束;

(2) 在对任务进行容错调度时,系统能够及时诊断并报告节点故障,调度时不会将任务分配给出现故障的节点;

(3) 一个节点出现故障不会导致其他节点出现故障,即系统提供故障隔离机制;

(4) 任务的主版本和副版本相同,即在同一 QoS 级别下在同一节点上具有相同的执行时间.所有任务具有

相同的优先级,并具有相同的 QoS 需求范围.

2.2 调度器模型

调度器模型可以分为两类:集中式调度器模型和分布式调度器模型[12].与分布式调度器模型相比,集中式调度器模型有两个最明显的优点:1) 通过对中心调度器的备份,便于实现容错处理;2) 实现比较简单、容易[20].

本文采用集中式调度器模型.在该调度器模型中,所有实时任务动态地到达一个被称为中央调度器(central scheduler)的处理器.由这个中央调度器,任务被分配到集群中的各个节点进行处理.各个节点与调度器之间通过高速网络进行连接,节点和中央调度器并行运行.每个节点都有一个局部队列,被分配到该节点上的任务首先进入局部队列,等待节点进行处理.如果任务不能被调度,则被发送到拒绝队列中.为了实现本文的调度目标,在中央调度器中,实时控制器、QoS 控制器、可靠性控制器和副版本执行模式控制器将根据系统的负载情况尽量提高系统的调度成功率、动态调整任务的 QoS 级别、最大化系统的可靠性和动态调整副版本的执行模式,从而提高系统的灵活性、可靠性、可调度性和资源的利用率.该调度器模型如图 1 所示.

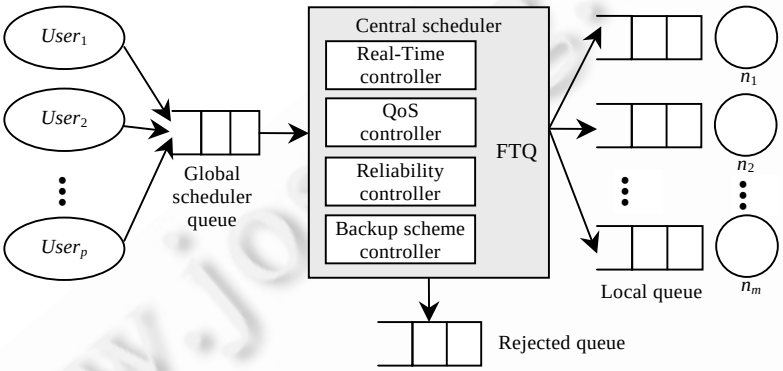

Fig.1 Scheduler model

图 1 调度器模型

2.3 任务模型

令某一实时应用为一组实时独立任务集合 $T=\{t_1,t_2,\ldots,t_n\}$,集合 T 中所有的实时任务都具有相同的容错需求.由于采用 PB 方法进行容错,因此任务 t_i 具有两个版本,分别为主版本 t_i^P 和副版本 t_i^B .$N=\{n_1,n_2,\ldots,n_m\}$为节点集合,由于集群为异构集群,因此,每个节点的处理能力有所不同.执行时间用矩阵 $E=(e_{ij})_{n\times m}$ 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.由于预测任务的执行时间是在调度算法之外的一个独立的研究问题[17],因此本文假设每个任务在不同节点上的执行时间已知.a_i 和 d_i 分别表示任务 t_i 的到达时间和截止期.f_i 和 f_i^B 分别表示主版本 t_i^P 和副版本 t_i^B 的完成时间.l_i 表示主版本 t_i^P 的裕度(laxity time),$l_i=d_i-f_i$,即主版本 t_i^P 在完成后距截止期的时间长度.

令 $EST=(est_{ij})_{n\times m}$ 为主版本 t_i^P 的最早开始时间矩阵,其中,元素 est_{ij} 表示主版本 t_i^P 在节点 n_j 上的最早开始时间. $LST^B=(lst_{ij}^B)_{n\times m}$ 为副版本 t_i^B 的最晚开始时间矩阵,其中,元素 lst_{ij}^B 表示副版本 t_i^B 在节点 n_j 上的最晚开始时间. $n(t_i^P)$ 和 $n(t_i^B)$ 分别表示主版本 t_i^P 和副版本 t_i^B 被分配到的节点.令 $Z=(z_{ij})_{n\times m}$ 为一个二元矩阵,$z_{ij}=1$ 当且仅当主版本 t_i^P 被分配到了节点 n_j 上,否则,$z_{ij}=0$.类似地, $Z^B=(z_{ij}^B)_{n\times m}$ 也为一个二元矩阵,其中, $z_{ij}^B=1$ 当且仅当副版本 t_i^B 被分配到了节点 n_j 上,否则, $z_{ij}^B=0$.$n(t_i^P)=j\Leftrightarrow z_{ij}=1, n(t_i^B)=j\Leftrightarrow z_{ij}^B=1$.如果主版本 t_i^P 分配到节点 n_j 上后,直到执行结束,节点 n_j 没有出现故障,则记 $o_{ij}=1$,否则,记 $o_{ij}=0$.

为不失一般性,$Q=\{q_1,q_2,\ldots,q_k\}$表示 QoS 级别集合,其中,$q_1<q_2<\ldots<q_k$.在进行容错调度时,应尽量提高任务的 QoS 级别,以充分利用系统资源,为任务提供更高级别的服务.需要注意的是,任务的最低 QoS 级别能够满足任务

的基本需求,高 QoS 级别可以使任务具有更好的服务质量.

X_i表示主版本 t_i^P 的所有可行调度,也就是说,在系统中有多个节点可以被选作候选节点处理某个任务.$x_i \in X_i$ 为 t_i^P 的调度选择.任务主版本 t_i^P 采用 x_i 调度的 QoS 级别可以表示为 $q(x_i)$.x_i 是一个可行调度需满足下面两个条件:1) 满足 t_i 的截止期 d_i,即 $f_i \leqslant d_i$;2) 满足 QoS 需求,即 $q_1 \leqslant q(x_i) \leqslant q_k$.在节点 n_j 上,主版本 t_i^P 采用 $q(x_i)$级别的任务执行时间表示为 $e_{ij}(q(x_i))$.相应地,副版本 t_i^B 采用 $q(x_i)$级别的任务执行时间可表示为 $e_{ij}(q(x_i)^B)$.

与之相对应的是 X_i^B,表示副版本 t_i^B 的所有可行调度,也具有与上面相似的性质,这里不再赘述.

如前所述,任务副版本的执行模式有两种,即主动副版本模式和被动副版本模式.本文将根据任务主版本裕度的大小,采用自适应的副版本执行方式.令 $s(t_i^B)$ 表示副版本 t_i^B 在节点 n_j 上的执行模式,则 $s(t_i^B)$ 可以表示为

$$s(t_i^B)=\begin{cases} passive, & \text{if } l_i \geqslant e_{ij}(q(x_i)^B) \\ active, & \text{if } l_i < e_{ij}(q(x_i)^B) \end{cases} \tag{1}$$

下面考虑任务副版本 t_i^B 的实际执行时间.t_i^B 的实际执行时间不仅与 t_i^B 的执行模式有关,还与 t_i^P 执行是否成功有关.t_i^B 在节点 n_j 的实际执行时间 re_{ij}^B 可以表示为

$$re_{ij}^B=\begin{cases} 0, & o_{ij}=1 \wedge s(t_i^B)=passive \\ 0<re_{ij}^B \leqslant e_{ij}(q(x_i)^B), & o_{ij}=1 \wedge s(t_i^B)=active \\ e_{ij}(q(x_i)^B), & o_{ij}=0 \wedge (s(t_i^B)=passive \vee s(t_i^B)=active) \end{cases} \tag{2}$$

式(2)表明,如果任务 t_i 的主版本 t_i^P 成功执行完毕,并且 t_i^B 以被动副版本模式执行,则在 t_i^P 执行完毕后,t_i^B 将从节点 $n(t_i^B)$ 上删除,那么 t_i^B 的实际执行时间 $re_{ij}^B=0$;如果任务 t_i 的主版本 t_i^P 成功执行完毕,并且 t_i^B 以主动副版本模式执行,则副版本的一部分会与主版本同时执行,因此 t_i^B 的实际执行时间大于 0 且小于其采用 QoS 级别为 $q(x_i)^B$ 的预计执行时间 $e_{ij}(q(x_i)^B)$;如果任务 t_i 的主版本 t_i^P 未成功执行,即 t_i^P 被分配到节点 $n(t_i^P)$ 上后,由于节点 $n(t_i^P)$ 出现了故障,其副版本 t_i^B 必定执行,且成功执行完成,则所执行时间为采用 QoS 级别为 $q(x_i)^B$ 的预计执行时间 $e_{ij}(q(x_i)^B)$.

2.4 可靠性模型

由于可靠性模型可以用来定量地评估系统的容错能力,所以本文在进行容错调度的研究过程中充分考虑了系统的可靠性.Srinivasan 等人给出了系统可靠性的定义,指出系统的可靠性是指在没有任何故障的情况下,系统能够正常执行任务的概率[21].

系统的可靠性依赖于系统的可靠性开销.假定集群系统中节点出现故障的过程是相互独立的,并且符合泊松过程,在系统无节点出现故障时,系统的可靠性开销可以表示为[21]

$$rc=rc(Z)=\sum_{j=1}^{m}\sum_{i=1}^{n}\lambda_j z_{ij} e_{ij} \tag{3}$$

其中,Z 表示任务的分配矩阵,λ_j 表示节点 n_j 的失效率.上面的系统可靠性是在节点未出现故障的情况下系统可靠性开销的计算方法,没有考虑到采用 PB 方法进行容错时系统的可靠性开销.同时,上面的模型没有反映任务具有 QoS 需求的情况,也没有考虑到在容错调度中副版本的执行模式.下面给出一种改进的可靠性开销模型,该模型考虑了采用 PB 方法的容错方式、任务的 QoS 需求和副版本的执行模式.

$$rc(Z)=\sum_{j=1}^{m}\sum_{i=1}^{n}\lambda_j z_{ij} o_{ij} e_{ij}(q(x_i)) \tag{4}$$

$$rc(Z^B)=\sum_{j=1}^{m}\sum_{i=1}^{n}\lambda_j z_{ij} re_{ij}^B=\sum_{j=1}^{m}\sum_{i=1,o_{ij}=1,s(t_i^B)=active}^{n}\lambda_j z_{ij} re_{ij}^B+\sum_{j=1}^{m}\sum_{i=1,o_{ij}=0}^{n}\lambda_j z_{ij} e_{ij}(q(x_i)^B) \tag{5}$$

$$rc=rc(Z)+rc(Z^B) \tag{6}$$

根据系统的可靠性开销,系统的可靠性 r 可被定义为[21]

$$r=e^{-rc} \tag{7}$$

从上面的定义可以看出,如果想提高系统的可靠性,则应尽量减小系统的可靠性开销.任务在分配过程中应选择使得系统可靠性开销尽量小的节点.即:在同等的执行时间下,节点的失效率越小越好;而在相同节点失效率的条件下,则执行的时间越小越好.这说明,任务应尽量分配给处理能力强、稳定而可靠的节点,从而提高整个系统的可靠性.

3 容错调度算法 FTQ

本文提出一种新的异构集群系统中实时任务容错调度算法 FTQ(fault-tolerant QoS-based scheduling),该算法综合考虑了实时任务的 QoS 需求、系统的可靠性、资源的利用率和实时任务的调度成功率.在给出 FTQ 算法之前,我们首先介绍 FTQ 算法的性质.

性质 1. 系统应尽量提高所接受任务的 QoS 级别,以提高任务的服务质量.任务 QoS 级别的大小取决于任务的开始时间、执行时间和截止期,即满足下面不等式:

$$\forall t_i \in T, est_{ij}+e_{ij}(q(x_i)) \leqslant d_i \tag{8}$$

$$\forall t_i \in T, lst_{ik}^B + e_{ik}(q(x_i)^B) \leqslant d_i \tag{9}$$

其中,$j \neq k$.公式(8)和公式(9)表明,如果一个任务能被接受,则任务的主版本和副版本必须能够满足任务的时间限制,即在截止期内完成.同时,公式(8)和公式(9)也表明,任务的主版本和副版本所选择的 QoS 级别可以不同.根据系统的负载情况,主版本的 QoS 级别可能大于或等于副版本的 QoS 级别,也有可能小于副版本的 QoS 级别.这种考虑极大地提高了系统的灵活性.而在传统的不考虑任务 QoS 需求的容错调度算法中,如果副版本不能以和主版本相同的处理时间来执行,则任务会被拒绝.所以,任务 QoS 级别的自适应变化提高了任务的调度成功率.

为了节约 CPU 资源,任务的副版本应尽可能地采用被动执行模式;如果采用主动执行模式,也应使副版本的实际执行时间尽可能地少.因此,任务的主版本应尽早被调度,而任务的副版本应该尽可能晚地被调度.任务 t_i 的主版本 t_i^P 在节点 n_j 上具有最早开始时间 est_{ij} 必须满足下面两个条件:1) 节点 n_j 上至少有一个空闲时间槽可以容纳 t_i^P ;2) t_i^P 的完成时间 f_i 必须小于等于 t_i 的截止期,即 $f_i \leqslant d_i$.

为不失一般性,在计算 est_{ij} 之前,假设任务 $t_{i1},t_{i2},\ldots,t_{iq}$ 已被分配到节点 n_j 上.这里的任务可以指一个任务的主版本,也可以指一个任务的副版本.在节点 n_j 上的空闲时间槽为 $[0,s_{i1}],[f_{i1},s_{i2}],\ldots,[f_{i(q-1)},s_{iq}],[f_{iq},\infty]$.

为了得到 t_i^P 的最早开始时间,必须从左到右扫描空闲时间槽,即从最早开始的时间槽开始,找到第 1 个符合条件的空闲时间槽 $[f_{ik},s_{i(k+1)}]$,满足下面不等式:

$$s_{i(k+1)}-\max\{a_i,f_{ik}\} \geqslant e_{ij}(q(x_i)) \tag{10}$$

因此,t_i^P 的最早开始时间 est_{ij} 可以计算为

$$est_{ij}=\max\{a_i,f_{ik}\} \tag{11}$$

任务 t_i 的副版本 t_i^B 的最晚开始时间不必一定选择空闲的时间槽,因为任务的副版本之间可以进行重叠.

下面分 4 种情况讨论在任务副版本重叠时,副版本的最晚开始时间和约束条件.

情况 1. 假设对于任意 $t_i \in T, t_k \in T$,如果 $l_i \geqslant e_{ij}(q(x_i)^B), l_k \geqslant e_{kj}(q(x_k)^B)$,即 $s(t_i^B)=passive, s(t_k^B)=passive$, 且 $n(t_i^P) \neq n(t_k^P), n(t_i^B)=n(t_k^B), t_i^B$ 先被分配到节点 $n(t_i^B)$ 上,t_k^B 后被分配到节点 $n(t_k^B)$ 上.图 2 给出了此种重叠调度时的一个实例.

在图 2 中,t_i^B,t_j^B 和 t_k^B 均以被动副版本模式执行,t_i^B 和 t_k^B 有重叠.假设 t_i^B 和 t_x 先被分配到节点 n_2 上,t_k^B 后被分配到节点 n_2 上.任务 t_x 可以有两种情况:1) 某个任务的主版本;2) 某个任务的副版本,但该任务的主版本与 t_k^P 不在同一节点上,则 t_k^B 的最晚开始时间可被计算为

$$lst_{kj}^B = d_k - st_{xj} - e_{kj}(q(x_k)^B) \tag{12}$$

其中,st_{xj} 表示任务 t_x 在节点 n_j 上的开始时间.

公式(12)表明,t_k^B 的最晚开始时间小于等于其截止期 d_k 与其执行时间的差.但在调度过程中,要尽量使 t_k^B 的结束时间接近其截止期,从而节约系统资源,提高任务的调度成功率.

情况 2. 假设对于任意 $t_i \in T, t_k \in T$,如果 $l_i \geqslant e_{ij}(q(x_i)^B), l_k < e_{kj}(q(x_k)^B)$,即 $s(t_i^B) = passive, s(t_k^B) = active$,且 $n(t_i^P) \neq n(t_k^P), n(t_i^B) = n(t_k^B)$,$t_i^B$ 先被分配到节点 $n(t_i^B)$ 上,t_k^B 后被分配到节点 $n(t_k^B)$ 上.图 3 给出了此种重叠调度时的一个实例.

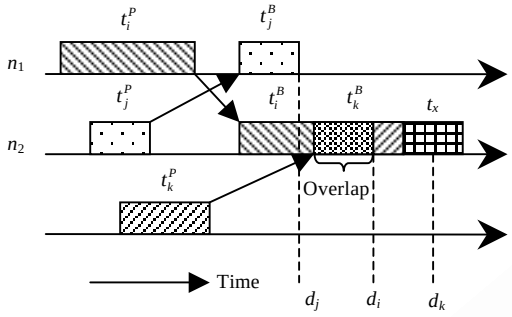

Fig.2 An example of case 1 for backup copy overlapping scheduling

图 2 副版本重叠调度情况 1 实例图

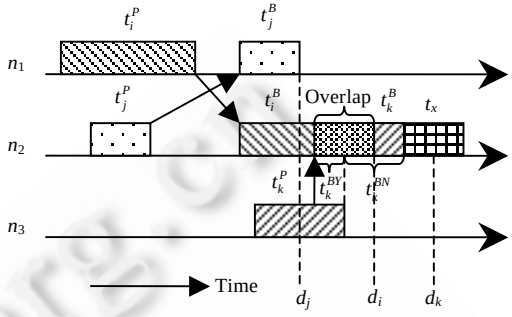

Fig.3 An example of case 2 for backup copy overlapping scheduling

图 3 副版本重叠调度情况 2 实例图

在图 3 中,t_i^B 和 t_j^B 以被动副版本模式执行,t_k^B 以主动副版本模式执行.t_i^B 和 t_k^B 有重叠.假设 t_i^B 和 t_x 先被分配到节点 n_2 上,t_k^B 后被分配到节点 n_2 上,任务 t_x 的情形如情况 1.副版本 t_k^B 的执行时间在逻辑上可以分成两部分 t_k^{BY} 和 t_k^{BN}.t_k^{BY} 为与主版本 t_k^P 重叠执行的时间,t_k^{BN} 为不与主版本 t_k^P 重叠执行的时间.t_k^B 最晚开始时间的计算方法与公式(12)相同,但必须满足下面的不等式:

$$lst_{kj}^B \geqslant f_k \tag{13}$$

公式(13)表示,t_k^B 的最晚开始时间 lst_{kj}^B 必须大于等于 t_k^P 的完成时间,即采用被动执行模式,否则不能实现容错.

定理 1. 如果任务 t_k 的副版本 t_k^B 可以与任务 t_i 的副版本 t_i^B 重叠,且 t_i^B 采用被动副版本执行模式,那么 t_k^B 的最晚开始时间一定大于等于 t_k^P 的完成时间 f_k,即采用被动执行模式.

证明:假设 t_k^B 的最晚开始时间可以小于 t_k^P 的完成时间 f_k,即 t_k^P 与 t_k^{BY} 同时执行,当系统中某一节点出现故障时,t_i^B 和 t_k^{BY} 重叠的部分只能有一个任务执行.由于 t_k^B 采用主动副版本执行模式,t_k^{BY} 与 t_k^P 一起执行.如果 t_i^B 对应的主版本 t_i^P 所在的节点 $n(t_i^P)$ 失效,那么,t_i^B 必须执行,而 t_k^{BY} 也必须执行,t_i^B 与 t_k^{BY} 又有重叠部分,出现在同一个节点上两个不同的任务在同一时间段内执行,因此假设不成立. □

情况 3. 假设对于任意 $t_i \in T, t_k \in T$,如果 $l_i < e_{ij}(q(x_i)^B), l_k \geqslant e_{kj}(q(x_k)^B)$,即 $s(t_i^B) = active, s(t_k^B) = passive$,且 $n(t_i^P) \neq n(t_k^P), n(t_i^B) = n(t_k^B)$,$t_i^B$ 先被分配到节点 $n(t_i^B)$ 上,t_k^B 后被分配到节点 $n(t_k^B)$ 上.图 4 给出了此种重叠调度时的一个实例.

在图 4 中,t_i^B 和 t_j^B 以主动副版本模式执行,t_k^B 以被动副版本模式执行.t_i^B 和 t_k^B 有重叠.假设 t_i^B 和 t_x 先被分配到节点 n_2 上,t_k^B 后被分配到节点 n_2 上,任务 t_x 的情形如情况 1.副版本 t_i^B 的执行时间在逻辑上可以分成两部分 t_i^{BY} 和 t_i^{BN}.t_i^{BY} 为与主版本 t_i^P 重叠执行的时间,t_i^{BN} 为不与主版本 t_i^P 重叠执行的时间.t_k^B 最晚开始时间的计算方法与公式(12)相同,但必须满足下面的不等式:

$$lst_{kj}^B \geqslant f_i \tag{14}$$

公式(14)表示,t_k^B 的最晚开始时间 lst_{kj}^B 必须大于等于 t_i^P 的完成时间,否则不能实现容错处理.

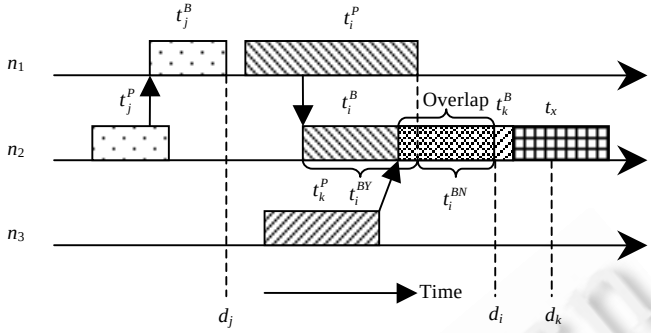

Fig.4 An example of case 3 for backup copy overlapping scheduling

图 4 副版本重叠调度情况 3 实例图

定理 2. 如果任务 t_k 的副版本 t_k^B 可以与任务 t_i 的副版本 t_i^B 重叠,且 t_i^B 采用主动副版本执行模式,t_k^B 采用被动副版本执行模式,那么 t_k^B 的最晚开始时间一定大于等于 t_i^P 的完成时间 f_i.

证明:假设 t_k^B 的最晚开始时间可以小于 t_i^P 的完成时间 f_i,即 t_i^P 与 t_i^{BY} 同时执行,当系统中某一节点出现故障时,t_k^B 和 t_i^{BY} 重叠的部分只能有一个任务执行.由于 t_i^B 采用主动副版本执行模式,t_i^{BY} 与 t_i^P 一起执行.如果 t_k^B 对应的主版本 t_k^P 所在的节点 $n(t_k^P)$ 失效,那么 t_k^B 必须执行,而 t_i^{BY} 也必须执行,t_k^B 与 t_i^{BY} 又有重叠部分,出现在同一个节点上两个不同的任务在同一时间段内执行,因此假设不成立. □

情况 4. 假设对于任意 $t_i \in T, t_k \in T$,如果 $l_i < e_{ij}(q(x_i)^B), l_k < e_{kj}(q(x_k)^B)$,即 $s(t_i^B) = active, s(t_k^B) = active$,且 $n(t_i^P) \neq n(t_k^P), n(t_i^B) = n(t_k^B)$,$t_i^B$ 先被分配到节点 $n(t_i^B)$ 上,t_k^B 后被分配到节点 $n(t_k^B)$ 上.图 5 给出了此种重叠调度时的一个实例.

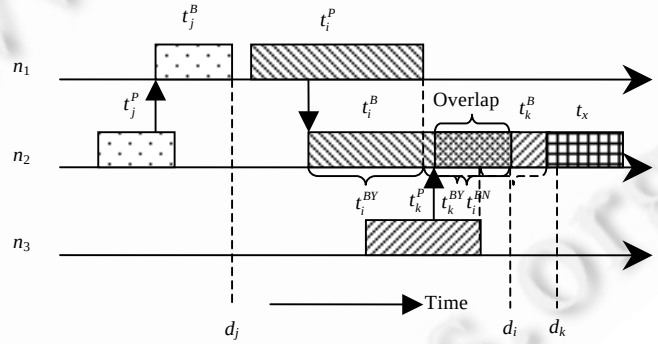

Fig.5 An example of case 4 for backup copy overlapping scheduling

图 5 副版本重叠调度情况 4 实例图

在图 5 中,t_i^B,t_j^B 和 t_k^B 均以主动副版本模式执行,t_i^B 和 t_k^B 有重叠.假设 t_i^B 和 t_x 先被分配到节点 n_2 上,t_k^B 后被分配到节点 n_2 上,任务 t_x 的情形如情况 1.副版本 t_k^B 的执行时间在逻辑上可以分成两部分 t_k^{BY} 和 t_k^{BN},如情况 2 所述;副版本 t_i^B 的执行时间在逻辑上可以分成两部分 t_i^{BY} 和 t_i^{BN},如情况 3 所述.t_k^B 最晚开始时间的计算方法与公式(12)相同,但必须满足下面两个不等式:

$$lst_{kj}^B \geqslant f_i \tag{15}$$

$$lst_{kj}^B \geqslant f_k \tag{16}$$

公式(15)和公式(16)表示,t_k^B 的最晚开始时间 lst_{kj}^B 必须大于等于 t_i^P 的完成时间,且 t_k^B 采用被动执行模式,否

则不能实现容错处理.

定理 3. 如果任务 t_k 的副版本 t_k^B 可以与任务 t_i 的副版本 t_i^B 重叠,且 t_i^B 采用主动副版本执行模式,那么, t_k^B 的最晚开始时间一定大于等于 t_i^P 的完成时间 f_i,且大于等于 t_k^P 的完成时间 f_k,即采用被动执行模式.

证明:欲证明上述定理,只需在定理 1 的基础上证明 t_k^B 的最晚开始时间一定大于等于 t_k^P 的完成时间 f_k.假设 t_k^B 的最晚开始时间可以小于 t_k^P 的完成时间 f_k,即 t_k^P 与 t_k^{BY} 同时执行.当系统中某一节点出现故障时, t_i^B 和 t_k^{BY} 重叠的部分只能有一个任务执行.如果 t_i^B 对应的主版本 t_i^P 所在的节点 $n(t_i^P)$ 失效,那么, t_i^B 必须执行.由于 t_k^B 采用主动执行模式,由于 t_k^{BY} 与 t_i^B 又有重叠, t_k^{BY} 也必须执行.这样,出现在同一个节点上两个不同的任务在同一时间段内执行,因此假设不成立. □

图 6 给出了 FTQ 算法主版本分配的伪代码.

FTQ 算法主版本分配的目标是在任务的主版本满足其截止期的前提下,将其尽量分配到使系统的可靠性达到最高的节点,并使得任务主版本的 QoS 级别尽可能地高.

```
1.  for each task t_i, schedule the primary copy t_i^P do
2.    find←FALSE, q_m←q_k;                                  /*Initialization*/
3.    while q_m≠q_1 do
4.      r←0; est←∞;                                         /*Initialization*/
5.      for each node n_j in the cluster do
6.        Calculate the earliest start time est_ij of task t_i on node n_j; /*Scan time slots*/
7.        if est_ij+e_ij(q_m)≤d_i (Property 1) then
8.          find←TRUE;                                      /*Find a feasible node*/
9.          Calculate system reliability r_j using Eq.(6);
10.         if r_j>r or r_j==r && est_ij<est then
11.           r←r_j; est←est_ij;                            /*Record the reliability and earliest start time*/
12.         end if
13.       end if
14.     end for
15.     if find==FALSE then
16.       Degrade one QoS level of q_m: q_m←q_{m-1};
17.     else
18.       break;
19.     end if
20.   end while
21.   if find==FALSE then
22.     Reject the primary copy t_i^P ;
23.   else
24.     Allocate t_i^P to node n_j where the system reliability of t_i^P on n_j is maximal;
25.   end if
26. end for
```

Fig.6 Pseudocode of primary copy allocation in FTQ algorithm

图 6 FTQ 算法主版本分配伪代码

FTQ 算法副版本分配的伪代码如图 7 所示.

FTQ 算法副版本分配的目标是在任务的副版本满足其截止期的前提下,将其尽量分配到使得系统的可靠性达到最高的节点,同时尽量采用被动执行模式,并使得任务副版本的 QoS 级别尽可能地高.

通过计算可知,FTQ 算法主版本和副版本分配的时间复杂度均为 $O(knq)$.因此,FTQ 算法的时间复杂度为 $O(knq)$.

```
1. for each task t_i whose primary copy t_i^P has been allocated, schedule backup copy t_i^B do
2.   find←FALSE, q_m←q_k;                    /*Initialization*/
3.   while q_m≠q_1 do
4.     r←0; lst←0; scheme←active;            /*Initialization*/
5.     for each node n_j except n(t_i^P) in the cluster do
6.       Calculate the latest start time lst_ij^B of t_i^B on node n_j using Eq.(13) based on Theorem (1);
7.       if lst_ij^B + e_ij(q_m^B) ≤ d_i (Property 1) then
8.         find←TRUE;                        /*Find a feasible node*/
9.         if lst_ij^B ≥ f_i then            /*Passive scheme*/
10.          scheme←passive;                 /*Find a node where t_i^B can execute in passive scheme*/
11.          Calculate system reliability r_j using Eq.(6);
12.          if r_j>r or r_j==r && lst_ij^B > lst then
13.            r←r_j; lst ← lst_ij^B;        /*Record the reliability and latest start time*/
14.          end if
15.        else                              /*Active scheme*/
16.          if lst_ij^B > lst then
17.            lst ← lst_ij^B;               /*Record the latest start time*/
18.          end if
19.        end if
20.      end if
21.    end for
22.    if find==FALSE then
23.      Degrade one QoS level of q_m: q_m←q_{m-1};
24.    else
25.      break;
26.    end if
27.  end while
28.  if find==FALSE then
29.    Reject the primary copy t_i^P and backup copy t_i^B;
30.  else
31.    Allocate t_i^B to node n_j where the system reliability of t_i^B on n_j is maximal if passive scheme
       or the latest start time is latest if active scheme;
32.  end if
33.end for
```

Fig.7 Pseudocode of backup copy allocation in FTQ algorithm

图 7 FTQ 算法副版本分配伪代码

4 实验测试

本文通过大量的模拟实验来测试 FTQ 算法的性能.进行模拟实验最大的优势在于,在大规模集群系统中进行测试不需要额外的硬件开销[22].为了测试 FTQ 算法的性能,本文将其与最近 Luo 等人提出的异构集群系统中可靠性驱动的实时容错调度算法 DYFARS[19]和本文提出的另一种算法 NOFTQ(non-overlapping fault-tolerant QoS-based scheduling)进行了比较.NOFTQ 与 FTQ 算法的不同在于,NOFTQ 没有考虑副版本之间可以重叠的问题.通过 FTQ 与 NOFTQ 的比较,目的在于分析采用副版本间可重叠技术导致系统性能变化的情况.为公平起见,将 DYFARS 算法进行了略微修改,即它随机选取任务的 QoS 级别.尽管修改后的 DYFARS 算法也提供了 QoS 需求,但是它们没有最大化任务的 QoS 级别.下面简要介绍一下 DYFARS 算法:(1) 尽量将任务主版本分配到可靠性开销最小的节点上;(2) 尽量使任务副版本采用被动执行模式;(3) 如果任务副版本只能采用主动执行模式,则尽量将任务副版本分配到可靠性开销最小的节点上.

本文主要从以下几个方面比较 FTQ,NOFTQ 和 DYFARS 的性能,其中整体系统性能评估方法类似文献[20]:(1) 调度成功率(guarantee ratio,简称 GR);(2) QoS 级别均值(QoS level average,简称 QLA);(3) 系统单位时间内可靠性开销均值(reliability cost average,简称 RCA);(4) 整体系统性能(overall system performance,简称

OSP,$OSP=GR^2\times QLA\times e^{(-RCA)}$).

4.1 模拟方法和参数

异构性可以分为节点异构性和任务异构性.节点异构性是指一个任务在不同节点上执行时间的差异;任务异构性是指一个节点上的不同任务执行时间的差异.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 是一个正实数,p_j 越大,节点处理能力越强.参数 *NodePowerAverage* 和 *NodePowerSpan* 分别表示所有节点的平均处理能力和节点处理能力浮动范围.p_j 均匀分布在 *NodePowerAverage–NodePowerSpan* 和 *NodePowerAverage+NodePowerSpan* 之间[23];

(2) h_i 表示任务 t_i 的处理难度,用以体现任务的异构性.h_i 是一个正实数,h_i 越大,任务 t_i 在同一节点上的执行时间越长.参数 *TaskHardnessAverage* 和 *TaskHardnessSpan* 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 *TaskHardnessAverage–TaskHardnessSpan* 和 *TaskHardnessAverage+ TaskHardnessSpan* 之间[23];

(3) 由于本文所提出的调度模型是一种通用模型,为不失一般性,我们给出一个抽象的 QoS 级别定义,只需满足高级别 QoS 任务需要较多的处理时间,而低级别 QoS 任务需要较少的处理时间,因此,我们假设 $0.1\leqslant q(x_i)\leqslant 1$ 为任务 t_i 的当前 QoS 级别,$q(x_i)\in[0.1,0.2,\ldots,1]$,步长为 0.1;

(4) 任务 t_i 在节点 n_j 上的执行时间 e_{ij} 可以表示为 $e_{ij}=q(x_i)\times BaseTime\times 10\times(h_i/p_j)$,其中,参数 *BaseTime* 为一个随机正实数;

(5) 任务 t_i 的截止期按如下公式计算:$d_i=a_i+\max(e_{ij})+TimeBaseDeadline$,其中,$a_i$ 是任务 t_i 的到达时间,$\max(e_{ij})$为任务 t_i 在所有节点上执行时间的最大值.参数 *TimeBaseDeadline* 是一个随机正实数,*TimeBaseDeadline* 越大,任务截止期越宽松;

(6) 每个节点的失效率 λ_j 均匀分布,单位是 1/1000000(每小时);

(7) 任务 t_i 的到达时间 $a_i=a_{i-1}+TimeInterval$,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 *TimeInterval* 为一个正实数,用于决定任务的达到速率.

表 1 给出了实验中的参数值.

Table 1 Lists of experimental parameters

表 1 实验参数列表

Parameter	Value (fixed)—(min, max, step)
Node number	(64)—(4,256,−)
Task number	(2048)
NodePowerAverage	(700)
NodePowerSpan	(360)—(160,400,40)
TaskHardnessAverage	(300)
TaskHardnessSpan	(120)
TimeBaseDeadline	(360)—(360,1440,180)
BaseTime	(60)
TimeInterval	(2)
NodeErrorRateSpan	(1.2,2.0)—(0.9,2.3,0.1)

4.2 节点数对性能的影响

本节我们将通过一组实验来观察节点数对 DYFARS,NOFTQ 和 FTQ 的影响.实验结果如图 8 所示.

图 8(a)显示了当节点数增加后,DYFARS,NOFTQ 和 FTQ 的调度成功率都随之增高.这是因为,节点数增加后,系统的处理能力增强,有更多的节点可以用来容纳更多的实时任务.从图 8(a)中可以看出,除了节点为 4 的这种极低系统资源的情况外,NOFTQ 和 FTQ 的调度成功率都高于 DYFARS 的调度成功率.这是因为,DYFARS 的 QoS 级别不能作调整,所以导致部分任务不能被接受;而 NOFTQ 和 FTQ 可以通过降低任务的 QoS 级别来提高任务的调度成功率.从图 8(a)还可以看出,FTQ 的调度成功率始终高于 NOFTQ.这是因为,FTQ 在 NOFTQ 的基

础上采用了副版本重叠技术,提高了系统资源的利用率,进而提高了系统的可调度性.图 8(a)中,FTQ 的调度成功率平均高于 DYFARS 和 NOFTQ,分别为 39%和 15.4%.

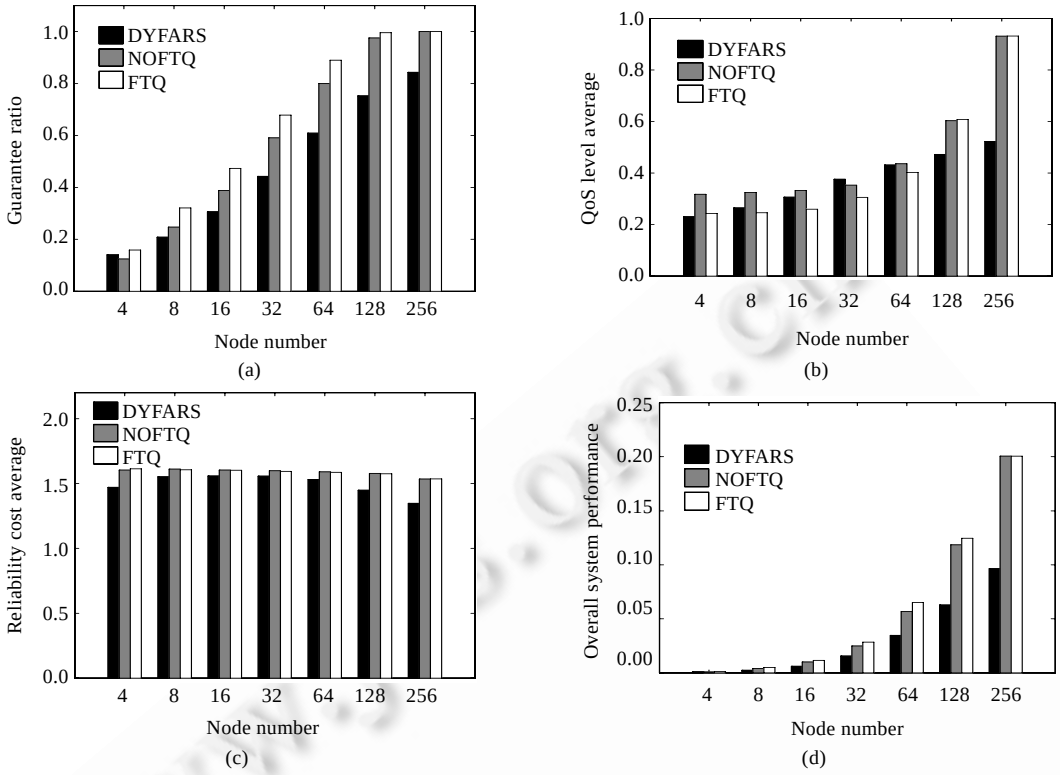

Fig.8 Performance impact of node number

图 8 节点数对性能的影响

由图 8(b)可以看出,在节点数较少的情况下,FTQ 的 QoS 级别较低.这是因为,在节点数较少时,系统的负载较重,FTQ 能够通过降低任务的 QoS 级别来提高任务的调度成功率;而当节点数目增加后,FTQ 的 QoS 级别随之增加,这表明,节点数增加后,系统的处理能力增强,FTQ 可以为任务提供更高的 QoS 级别,因此具有很强的自适应性.而在节点数小于等于 64 时,NOFTQ 的 QoS 级别高于 FTQ.这是因为,FTQ 在系统负载较重时,通过降低任务的 QoS 级别来提高任务的调度成功率.图 8(b)也显示出,DYFARS 的 QoS 级别均值随节点数的增加也有所增加,但它最高只能增加到 0.5 左右.这是因为,DYFARS 随机选取任务的 QoS 级别,那些 QoS 级别较高的任务在系统负载较重时被拒绝,因此 QoS 级别均值较小;而当系统负载变轻时,系统能够接纳 QoS 级别较高的任务,因此总体的 QoS 级别增加.但是,由于 DYFARS 在 0.1~1 之间随机地为任务选取 QoS 级别,因此,当系统负载较轻、所接收任务越多时,任务的平均 QoS 级别就越接近 0.5.而 NOFTQ 和 FTQ 可以根据系统负载情况自适应地调整任务的 QoS 级别,当节点数较多使得系统负载较轻时,QoS 级别均值最高可以达到 1.

图 8(c)显示了 DYFARS,NOFTQ 和 FTQ 的可靠性开销趋势.除了节点为 4 的这种极低的系统资源情况外,三者的可靠性开销都随着节点数的增加而减少.DYFARS 的变化较为明显,而 NOFTQ 和 FTQ 的可靠性开销略有减少.这是因为,节点的出错率均匀分布,当节点数增加后,高可靠性的节点数增加,而任务数不变,因此,任务会越来越多地被分配到可靠性比较高的节点上进行处理,使得系统的可靠性开销逐渐减小.NOFTQ 和 FTQ 的可靠性开销变化较小是因为它们能够充分利用系统资源,在节点数目较多、系统资源增加的情况下,NOFTQ 和 FTQ 能够自适应地提高任务的 QoS 级别,一直保持对系统资源的充分利用,因此单位时间内可靠性开销变化很小.从图 8(c)还可以看出,一般情况下,DYFARS 的可靠性开销最小,NOFTQ 的可靠性开销最大,FTQ 比 NOFTQ

的可靠性开销略小.这是由于,DYFARS 使任务在节点可靠性比较高的节点上执行的较多,在可靠性比较低的节点上执行的较少;而 FTQ 和 NOFTQ 对资源利用率较高,对可靠性较高节点的利用虽然多于对可靠性较低节点的利用,但相差不是很大.对资源的充分利用,也正是 FTQ 和 NOFTQ 的调度成功率和 QoS 级别较高的原因.而 FTQ 比 NOFTQ 的可靠性开销略少,是因为 FTQ 采用了副版本重叠技术,提高了可靠性较高节点的利用率,使任务更多地在这些节点上执行.

图 8(d)给出了随着节点数的增加,DYFARS,NOFTQ 和 FTQ 整体性能的变化情况.从图中可以看出,随着节点数的增加,FTQ 的性能显著增强,平均高于 DYFARS 和 NOFTQ,分别为 86%和 14.8%.

4.3 节点异构性对性能的影响

本节通过一组实验来观察节点异构性对 DYFARS,NOFTQ 和 FTQ 的影响.实验中用参数 *NodePowerSpan* 代表节点的异构性,*NodePowerSpan* 增大,则节点的异构性增大(处理能力强的节点,其处理能力会更强,处理能力弱的节点,其处理能力更弱),反之减小.实验结果如图 9 所示.

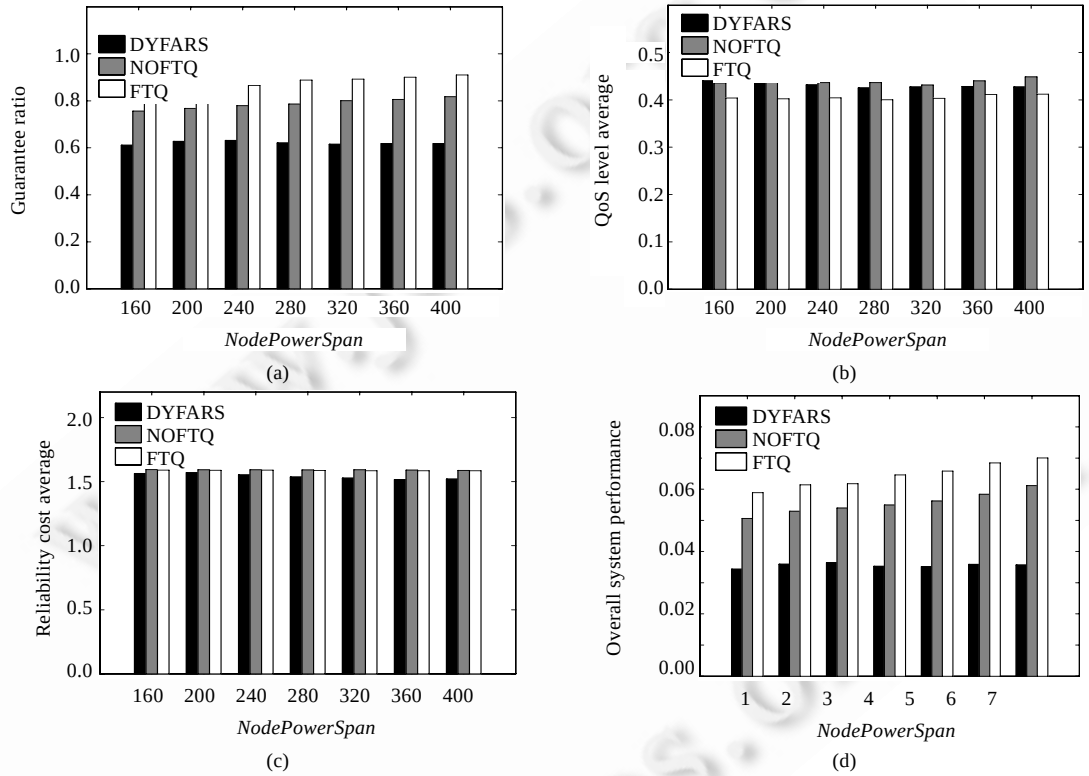

Fig.9 Performance impact of node heterogeneity

图 9 节点异构性对性能的影响

图 9(a)显示了在参数 *NodePowerSpan* 从 160 变化到 400 的过程中,FTQ 的调度成功率始终高于 DYFARS 和 NOFTQ 的调度成功率.这表明,系统中的节点无论是在同构还是异构情况下,FTQ 都具有最好的性能.FTQ 的调度成功率高于 DYFARS 和 NOFTQ 的调度成功率.其原因在于,FTQ 采用了副版本重叠技术,提高了系统资源的利用率,因此具有较高的调度成功率.而 NOFTQ 的调度成功率大于 DYFARS 调度成功率,是因为 NOFTQ 可以通过降低任务的 QoS 级别来提高任务的调度成功率.从图 9(a)中还可以看出,FTQ 和 NOFTQ 的调度成功率略有增加.这是因为任务被优先分配到处理能力较强的节点上(因为在本实验中,处理能力强的节点有较高的可靠性),当节点异构性变大时,处理能力较强节点的处理能力会变得更强,所以调度成功率增加;而处理能力较弱

节点的处理能力虽然变弱,但是 FTQ 和 NOFTQ 的 QoS 级别的自适应性抵消了它的负面影响.而对于 DYFARS 算法,强节点处理能力变大虽然也有利于它的调度成功率的提高,但是由于它不具有自适应性,弱节点处理能力变小抵消了强节点处理能力变大带来的正面效果,所以整体上无明显的变化趋势.

从图 9(b)中可以看出,FTQ 的 QoS 级别低于 DYFARS 和 NOFTQ 的 QoS 级别.因为在系统资源不够充分的情况下,FTQ 为了提高任务的调度成功率而自适应地降低了任务的 QoS 级别.

图 9(c)显示,随着节点异构性的变化,DYFARS,NOFTQ 和 FTQ 的可靠性开销没有因为节点异构性的变化而产生较大的影响.这是因为 DYFARS,NOFTQ 和 FTQ 都充分考虑了节点的异构性问题,并且将系统的可靠性开销融入到了调度算法中.

图 9(d)给出了在节点发生异构性变化时,DYFARS,NOFTQ 和 FTQ 的总体性能.从图中可以看出,无论节点异构性如何变化,FTQ 始终具有最好的性能,平均高于 DYFARS 和 NOFTQ,分别为 81%和 16.2%.

4.4 任务截止期对性能的影响

本节通过一组实验来观察任务截止期对 DYFARS,NOFTQ 和 FTQ 的影响.

在本组实验中,调整参数 *TimeBaseDeadline* 从 360~1 440,步长为 180.实验结果如图 10 所示.

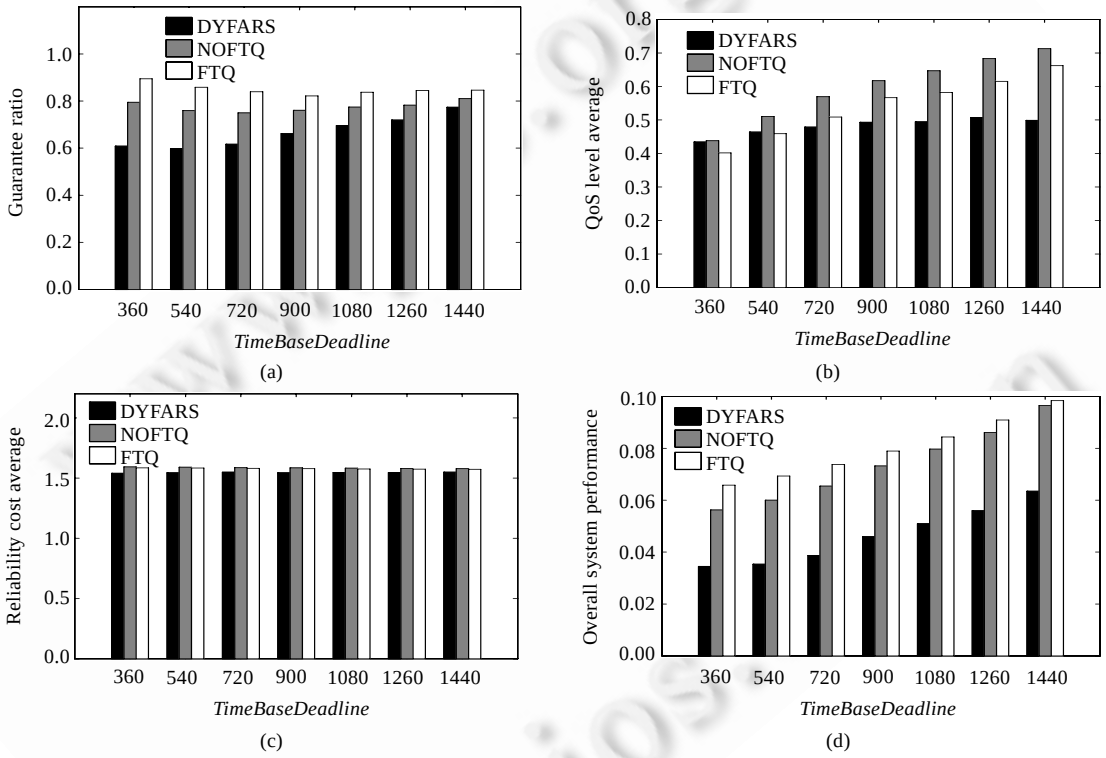

Fig.10 Performance impact of task deadline

图 10 任务截止期对性能的影响

图 10(a)表明,随着任务截止期的延长,DYFARS,NOFTQ 和 FTQ 调度成功率的变化趋势.DYFARS 基本处于一个上升的趋势,这是因为任务的基本截至期延长了,可以使系统有比较充裕的时间接受更多的任务,从而提高了任务的调度成功率.而 NOFTQ 和 FTQ 的变化先是稍微下滑,然后又缓慢上升.从图 10(a)可以看出,NOFTQ 在 *TimeBaseDeadline* 为 720 时的调度成功率最低,FTQ 在 *TimeBaseDeadline* 为 900 时的调度成功率最低.这是因为,当任务的基本截至期延长后,由于 NOFTQ 和 FTQ 采用了自适应的 QoS 级别调整方法,优先提升了任务的 QoS 级别,使得任务的执行时间较长,导致后续任务无法执行,从而使调度成功率稍有降低.但当任务截至期足够

大时,即使任务具有较高的 QoS 级别,也能够被成功接受,因此,NOFTQ 和 FTQ 的调度成功率开始回升.

图 10(b)表明,随着任务截止期的延长,DYFARS,NOFTQ 和 FTQ 调度成功率的变化趋势.DYFARS 一直保持上升状态,当 QoS 级别在 0.5 附近时保持稳定状态.这是因为,DYFARS 的 QoS 级别是随机取值,整体任务的 QoS 级别最大就在 0.5 附近.而 NOFTQ 和 FTQ 的调度成功率虽然不是单调变化趋势(如图 10(a)所示),但是它们的 QoS 级别一直处于上升趋势.这是因为,任务截止期的延长,使得 NOFTQ 和 FTQ 可以进一步提高任务的 QoS 级别.

图 10(c)表明,随着任务截止期的延长,DYFARS,NOFTQ 和 FTQ 的可靠性开销并没有明显变化.这是因为,DYFARS,NOFTQ 和 FTQ 都充分考虑了系统的可靠性开销并将其融入到调度算法中.

图 10(d)给出随着任务截止期的延长,DYFARS,NOFTQ 和 FTQ 的总体性能.从图中可以看出,FTQ 总体性能明显高于其他算法,平均高于 DYFARS 和 NOFTQ,分别为 76%和 9.5%.

4.5 节点出错率对性能的影响

本节我们将通过一组实验观察节点出错率变化对 DYFARS,NOFTQ 和 FTQ 的影响.

实验中,用参数 *NodeErrorRateSpan* 代表节点出错率的异构性.*NodeErrorRateSpan* 增大,则节点出错率的异构性增大,反之减小.实验结果如图 11 所示.

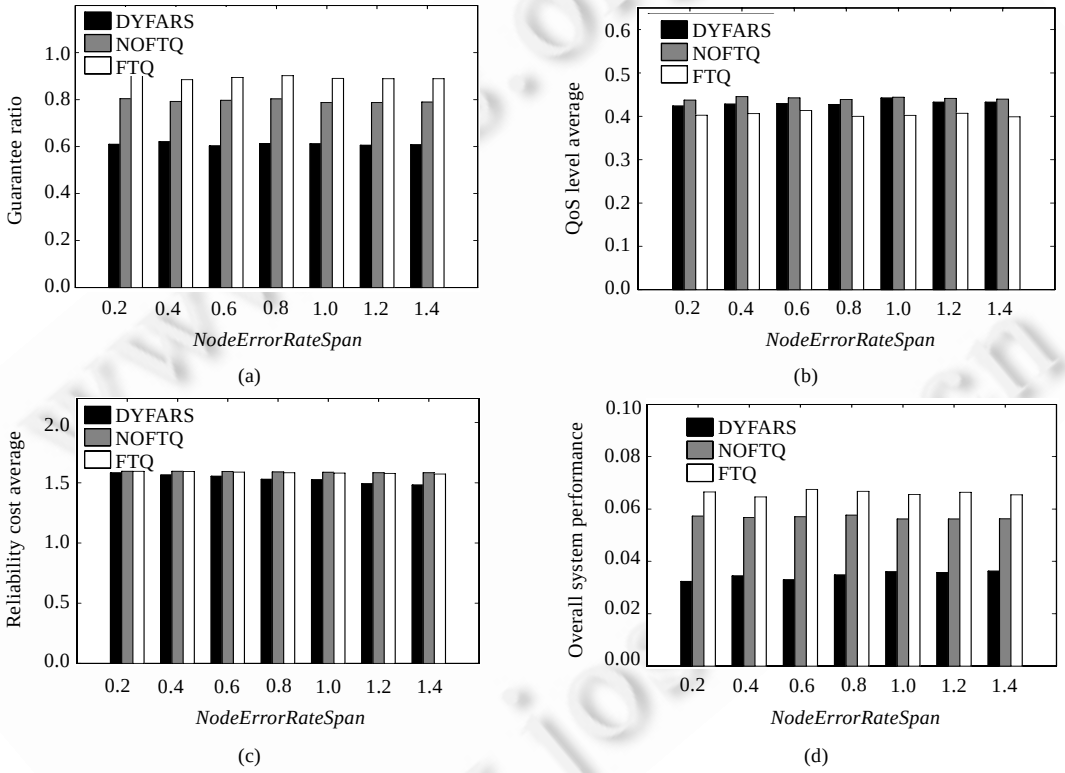

Fig.11 Performance impact of node error rate

图 11 节点出错率对性能的影响

图 11(a)表明,节点出错率范围的变化对 DYFARS,NOFTQ 和 FTQ 的调度成功率变化趋势并没有产生明显的影响.这是因为,DYFARS,NOFTQ 和 FTQ 都充分考虑了系统节点的出错率并将其融入到调度算法中.

图 11(b)表明,节点出错率范围的变化对 DYFARS,NOFTQ 和 FTQ 的 QoS 级别变化趋势并没有产生明显的影响.这是因为,DYFARS,NOFTQ 和 FTQ 都充分考虑了系统节点的出错率并将其融入到调度算法中.

图 11(c)表明了节点出错率范围的变化对 DYFARS,NOFTQ 和 FTQ 可靠性开销变化趋势的影响.其中,

DYFARS 的可靠性开销逐渐降低,这是因为,出错率范围的扩大使得原本可靠性高的节点具有更高的可靠性,原本可靠性低的节点具有更低的可靠性.而当 DYFARS 的任务调度成功率处于比较低的水平时,它的大部分任务都是在可靠性高的节点上完成的,所以 DYFARS 的可靠性开销会有逐渐变小的趋势.NOFTQ 和 FTQ 的可靠性开销也有细微的变小趋势,但是很不明显.这是因为 NOFTQ 和 FTQ 对资源利用率都比较高,对可靠性高的节点的利用虽然多于对可靠性较低节点的利用,但是相差不是很大.所以,当节点的出错率范围变大时,它们的可靠性开销变动较小.

图 11(d)给出了随着节点出错率范围的增大,DYFARS,NOFTQ 和 FTQ 的总体性能.从图中可以看出,FTQ 总体性能明显高于其他算法,平均高于 DYFARS 和 NOFTQ,分别为 91%和 16.4%.

5 结论及下一步工作

容错调度算法是提高系统可靠性的有效手段之一,目前已有很多实时任务的容错调度算法,但是还没有关于具有 QoS 需求的实时任务容错调度算法.本文将传统的实时容错调度问题拓展到具有 QoS 需求的实时容错调度问题,提出了一种动态的实时容错调度算法 FTQ,用于调度异构集群系统中独立、非周期、有 QoS 需求的实时任务.首先提出了一种新的适合具有 QoS 需求的实时任务容错调度模型,该模型在传统的主版本/副版本容错模型的基础上引入了系统的可靠性分析,将系统的可靠性开销加入该模型中.同时,采用自适应方式尽量为任务提供较高的 QoS 级别.在调度模型的基础上提出了一种新的容错调度算法 FTQ,该算法综合考虑了任务的时间限制、QoS 需求、系统的可靠性,同时采用了副版本重叠技术,提高了系统资源的利用率.此外,该调度算法尽量让副版本以被动方式执行,进一步提高了系统的资源利用率和系统的可调度性.通过模拟实验比较了 DYFARS,NOFTQ 和 FTQ 这 3 种算法的性能,实验结果表明,FTQ 算法优于其他方法,具有很强的灵活性,适合于异构集群环境中具有 QoS 需求的实时任务动态容错调度.

下一步的研究工作主要包括以下两个方面:一是进一步完善调度模型,在调度中考虑通信开销,将网络传输延迟考虑到任务的运行时间中去;二是研究主-副版本重叠技术,从而进一步提高系统的资源利用率.

References:

[1] Yuan YG. The Principal of Fault-Tolerance Computing. Harbin: Harbing Engineering University Press, 2006 (in Chinese).

[2] Krishna CM, Shin KG. Real-Time Systems. New York: McGraw-Hill, 1997.

[3] Abdelzaher TF, Atkins EM, Shin KG. QoS negotiation in real-time systems and its application to automated flight control. IEEE Trans. on Computers, 2000,49(11):1170−1183. [doi: 10.1109/12.895935]

[4] Zhu XM, Lu PZ. Scheduling of real-time signal processing in cluster-based software radio systems. Journal of Software, 2009,20(3): 766−778 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3313.htm [doi: 10.3724/SP.J.1001.2009.03313]

[5] Pyndiah RM. Near-Optimum decoding of product codes: Block turbo codes. IEEE Trans. on Communications, 1998,46(8): 1003−1010.

[6] Chi P, Song LL, Parhi KK. A study on the performance, complexity tradeoffs of block turbo decoder design. In: Proc. of the IEEE Int'l Symp. on Circuits and Systems, Vol.4. Sydney: IEEE Press, 2001. 65−68. http://www.informatik.uni-trier.de/~ley/db/conf/iscas/iscas2001-4.html [doi: 10.1109/ISCAS.2001.922170]

[7] Yu NY, Kim Y, Lee PJ. Iterative decoding of product codes composed of extended hamming codes. In: Tohmé S, Ulema M, eds. Proc. of the 5th IEEE Int'l Symp. on Computers and Communications. Antibes-Juan Les Pins: IEEE Press, 2000. 732−737. [doi: 10.1109/ISCC.2000.860728]

[8] Kopetz H, Grunsteidl G. TTP—A protocol for fault-tolerant real-time systems. IEEE Computer, 1994,27(1):14−23. [doi: 10.1109/2.248873]

[9] Qin X, Han ZF, Pang LP. Real-Time scheduling with fault-tolerance in heterogeneous distributed systems. Chinese Journal of Computers, 2002,25(1):49−56 (in Chinese with English abstract).

[10] Manimaran G, Murthy CSR. A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis. IEEE Trans. on Parallel and Distributed Systems, 1998,9(11):1137−1152. [doi: 10.1109/71.735960]

[11] Garey MR, Johnson DS. "Strong" NP-completeness results: Motivation, examples, and implications. Journal of the Association for Computing Machinery, 1978,25(3):499−508. [doi: 10.1145/322077.322090]

[12] Qin X, Jiang H. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. Journal of Parallel Computing, 2006,32(5-6):331−356. [doi: 10.1016/j.parco.2006.06.006]

[13] Ghosh S, Melhem R, Mossé D. Fault-Tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. IEEE Trans. on Parallel and Distributed Systems, 1997,8(3):272−284. [doi: 10.1109/71.584093]

[14] Al-Omari R, Somani AK, Manimaran G. Efficient overloading technique for primary-backup scheduling in real-time systems. Journal of Parallel and Distributed Computing, 2004,64(5):629−648. [doi: 10.1016/j.jpdc.2004.03.015]

[15] Tsuchiya T, Kakuda Y, Kikuno T. A new fault-tolerant scheduling technique for real-time multiprocessor systems. In: Proc. of the 2nd Int'l Workshop on Real-Time Computing Systems and Applications. Tokyo: IEEE Press, 1995. 197−202. http://www.informatik.uni-trier.de/~ley/db/conf/rtcsa/rtcsa1995.html [doi: 10.1109/RTCSA.1995.528772]

[16] Yang CH, Deconinec G, Gui WH. Fault-Tolerant scheduling for real-time embedded control systems. Journal of Computer Science and Technology, 2004,9(2):191−202. [doi: 10.1007/BF02944797]

[17] Al-Omari R, Somani AK, Manimaran G. An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems. Journal of Parallel and Distributed Computing, 2005,65(5):595−608. [doi: 10.1016/j.jpdc.2004.09.021]

[18] Pang LP, Lu WA, Han ZF. Task scheduling in DRT-UNIX system. Journal of Software, 1999,10(9):1003−1008 (in Chinese with English abstract). http://www.jos.org.cn/ch/reader/view_abstract.aspx?flag=1&file_no=19990917&journal_id=jos

[19] Luo W, Li JL, Yang FM, Tu G, Pang LP, Shu L. DYFARS: Boosting reliability in fault-tolerant heterogeneous distributed systems through dynamic scheduling. In: Feng W, Gao F, eds. Proc. of the 8th ACIS Int'l Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing. Qingdao: IEEE Press, 2007. 640−645. [doi: 10.1109/SNPD.2007.430]

[20] Zhu XM, Lu PZ. Multi-Dimensional scheduling for real-time tasks on heterogeneous clusters. Journal of Computer Science and Technology, 2009,24(3):434−446. [doi: 10.1007/s11390-009-9235-2]

[21] Srinivasan S, Jha NK. Safety and reliability driven task allocation in distributed systems. IEEE Trans. on Parallel and Distributed Systems, 1999,10(3):238−251. [doi: 10.1109/71.755824]

[22] Xie T, Qin X. Scheduling security-critical real-time applications on clusters. IEEE Trans. on Computers, 2006,55(7):864−879. [doi: 10.1109/TC.2006.110]

[23] Zhu XM, Lu PZ. Scheduling for security-critical real-time applications on heterogeneous clusters. Chinese Journal of Computers, 2010,33(12):2364−2377 (in Chinese with English abstract).

附中文参考文献:

[1] 袁由光.容错计算原理.哈尔滨:哈尔滨工程大学出版社,2006.

[4] 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. http://www.jos.org.cn/1000-9825/3313.htm [doi: 10.3724/SP.J.1001.2009.03313]

[9] 秦啸,韩宗芬,庞丽萍.基于异构分布式系统的实时容错调度算法.计算机学报,2002,25(1):49−56.

[18] 庞丽萍,吕文安,韩宗芬.DRT-UNIX 系统的任务调度.软件学报,1999,10(9):1003−1008. http://www.jos.org.cn/ch/reader/view_abstract.aspx?flag=1&file_no=19990917&journal_id=jos

[23] 朱晓敏,陆佩忠.异构集群系统中安全关键实时应用调度研究.计算机学报,2010,33(12):2364−2377.

朱晓敏(1979－),男,辽宁盘锦人,博士,讲师,CCF 会员,主要研究领域为并行与分布式计算,系统优化与调度.

马满好(1974－),男,博士,副教授,主要研究领域为作战模型与模拟.

祝江汉(1972－),男,博士,副教授,CCF 会员,主要研究领域为任务规划,军事运筹.