异构分布式系统中基于负载均衡的容错调度算法

郭 辉[1)] 王智广[1)] 周敬利[2)]

[1)](中国石油大学(北京)计算机科学技术系 北京 100220)

[2)](华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 提出了基于主/从版本的具有容错功能的进程调度算法HDALF和HDLDF,且分别给出两种算法的时间复杂度并对算法的负载均衡性和节点资源利用率作了讨论.与以往容错调度算法不同的是,此算法是在被动进程复制模式下、适合于异构分布式系统的容错调度算法.而以往的研究都是建立在主从版本进程有相等的负载或执行时间相同的模型基础上,或者仅适合于同构分布式系统.实验结果表明,HDALF算法和HDLDF算法的性能比基于同构分布式模型下的两阶段算法更加优越.并且得出了这样的结果:当系统发生故障前后的负载均衡性权值相等时,在负载均衡和处理机资源利用率方面,HDLDF算法都要优于HDALF算法.

关键词 容错系统;异构分布式系统;负载均衡;容错调度算法;进程分配

中图法分类号 TP302

Load Balancing Based Process Scheduling with Fault-Tolerance in Heterogeneous Distributed System

GUO Hui[1)] WANG Zhi-Guang[1)] ZHOU Jing-Li[2)]

[1)](*Department of Computer Science and Technology, Petroleum University of China, Beijing* 100220)

[2)](*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074)

Abstract The primary-backup process model for process allocation is an important research area in the study of fault-tolerant distributed system. In this paper, two heuristic approximation algorithms are proposed and analyzed, which are named as HDALF(heterogeneous distributed-system actual load first) and HDLDF(heterogeneous distributed-system load difference first). And the algorithms' complexity is presented. Different from the previous work, the new fault-tolerant process model is based on the passive process replica model. Previous research work usually provided that the primary and backup processes have the same overhead all the time, or are based on homogeneous distributed system. The proposed algorithms are compared with the two-stage allocation algorithm, which is for homogeneous distributed system. The experimental results show that the proposed algorithms have significantly better performance than the two-stage algorithm. Additionally, in the case that the relative important weights are equal between before and after fault occurrence, the performance of HDLDF algorithm is better than HDALF from the view of load-balancing and the processor utilizing rate.

Keywords fault-tolerant system; heterogeneous distributed system; load balancing; fault-tolerant scheduling algorithm; process allocation

---

收稿日期:2003-09-04;修改稿收到日期:2005-08-04.本课题得到国家"八六三"高技术研究发展计划项目基金(2001AA111011)资助.
郭 辉,男,1978年生,博士,研究方向为分布式计算机系统、网络容错技术.E-mail:hguo@wtwh.com.cn.王智广,男,1964年生,副教授,研究方向为计算机体系结构、人工智能.周敬利,女,1945年生,教授,博士生导师,研究领域为多媒体计算与网络通信.

1 引 言

在对分布式容错系统的研究中硬件冗余[1,2]是一种解决系统中永久错误的有效途径.然而,它需要额外的硬件代价来实现系统的可靠性.在对系统的软件容错的研究中活动/备用技术[3]是一种重要的软件容错模型.在活动/备用计算模型中,每一个进程都有主、从两个版本,从版本是主版本进程的一个副本.同一时间只有主版本的进程(primary process)运行.当主版本运行出现故障时,从版本进程(backup process)接替主版本进程的工作继续执行.为了保证系统的容错特性,主进程与从进程不能分配到同一个节点机上.除了基于活动/备用技术的容错模型外,还有其它各种软件容错技术如分布式投票技术[4]、回卷恢复技术[5]等实现系统的容错功能.

许多学者对分布式系统中具有主/从多个版本的进程调度问题做了大量的研究[6~9].文献[6]研究了如何将不含备份进程的非容错调度算法转换为含有备份进程、具有容错特性的调度算法.文献[7]提出了在分布式实时系统中同时调度具有容错需求与无容错需求进程的混合调度算法.文献[8]研究了将进程的负载均匀地分布到各个节点上的负载均衡容错调度算法.文献[9]引入可靠性代价概念对异构系统中的可靠性进行了评估并提出了最大化系统可靠性的调度算法.然而,对于以上研究,其调度算法是建立在备份进程与主进程具有相同的负载或执行时间相同的假设的基础上.基于这种假设的进程调度模型称为主动进程复制(active process replica)[10]模型.

事实上,在主/从版本进程模式中,同一时间内只有主版本的进程处于激活状态,即运行状态.而备份进程处于非激活状态,即睡眠状态.为了保证节点故障发生时备份进程有足够的信息用以接管主进程的工作,主进程定期地产生进程检查点(checkpoint)并发送给备份进程.备份进程此时处于两种可能的状态:或者等待接收主进程的检查点信息或者正在保存已收到的主进程检查点.因此,备份进程的负载应该远小于主进程的负载.在本文后面所做的分析中假设备份进程为主进程负载的5%～10%.当该主进程所在的处理机节点发生故障时,备份进程找到最近的一次检查点并激活,从检查点位置开始执行,接管主进程的工作.此时接管的备份进程与原主进程具有相同的负载.这种容错特性的进程调度模型称为被动进程复制(passive process replica)[11].被动进程复制模型更符合实际的系统.

目前对被动进程复制模型下进程分配算法的研究并不多.在Paralex[12]系统中通过"延迟绑定"技术实现了被动进程复制模型下进程动态分配算法.Kim[13]所提出的两阶段算法是一个经典的被动进程复制模型下基于负载均衡的静态分配算法.该算法解决了在故障发生前后同时保证处理机的负载均衡问题.接着,Lee[14]在此基础上将算法扩展到允许多个备份进程失效的情况.然而,以上所研究的多机系统或分布式系统都属于同构模型,即系统中各个处理机的性能完全相同.随着高速网络和高性能PC/工作站的发展,异构分布式系统越来越广泛地应用到各种计算密集型或数据密集型的应用中.因此,针对异构分布式环境下高效的容错调度算法问题由此提出.本文所做的研究是异构分布式系统中基于被动进程复制模型并考虑处理器负载均衡的静态容错调度算法.

本文首先分析了异构分布式系统中的容错模型,通过给出处理机性能参数τ对两阶段算法进行了改进[13].由于容错进程调度问题是一个NP难问题[15],我们提出了针对异构分布式系统的两个启发式算法HDALF和HDLDF.之后对算法的负载均衡性作出了讨论,并通过对所需最小处理机个数的研究,比较了两阶段算法、HDALF和HDLDF三种算法对处理机资源的利用率.结果表明,HDLDF算法具有更好的资源利用率.而两阶段算法由于未考虑异构分布式系统中处理机性能差异的特点,对处理机资源的利用率较低.

2 容错调度模型与前提假设

2.1 前提假设

本文所研究的容错调度模型的前提与假设是:有m个主进程运行在含有n个节点机的分布式系统中,并且系统中的一个节点的失效可立即被检测出.由于每个主进程仅含有一个相应的副本,所以我们所研究的分布式容错系统仅允许单点失效.若使用更多的进程副本,该模型可以扩展到允许多点失效[14].同时我们考虑的是静态容错调度算法,即m个主进程及其副进程的CPU负载预先已知道.这种假设可应用到实际的在线事务处理或实时系统中.因为这些应用在运行期间周期性地调用相同的进程,队列中的进程信息预先已知道.图1显示了m

个具有容错功能的主/从版本进程运行在有 n 个节点机的分布式系统上.

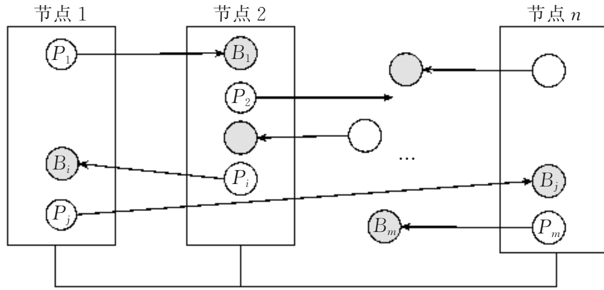

图 1　进程运行示意图

2.2　异构分布式容错系统模型

定义 1.　$\psi=\{P_1,P_2,\cdots,P_m\}$表示为一组具有容错功能的进程集合. 其中 $P\in\psi$ 可以描述为一个二元组：$P=(P_{pm},P_{bk})$. P_{pm}和 P_{bk}分别表示该进程的主版本和副版本. P_{pm}和 P_{bk}也由一个二元组(nps,ld)来描述. 其中 nps 表示该版本的进程所调度到的节点机，ld 表示该版本的进程的负载.

定义 2.　设分布式系统中处理机节点个数为 n，该系统中处理机集合定义为 $\phi=\{N_1,N_2,\cdots,N_n\}$，$N_i$表示第 i 个处理机节点. 每个处理机节点可以由一个五元组$(\tau,\lambda,\eta_m,\eta_s,\rho)$来表示. 其中 τ 为处理机的性能参数，λ 为调度到该处理机上的进程的负载总和，η_m为调度到该处理机上的主版本进程的集合，即 $\eta_m=\{P_1.P_{pm},P_2.P_{pm},\cdots,P_j.P_{pm},\varnothing\}$. η_s表示为调度到该处理机上副版本进程的集合，即 $\eta_s=\{P_1.P_{bk},P_2.P_{bk},\cdots,P_j.P_{bk},\varnothing\}$. ρ 为一组节点标号的集合$\{\theta_1,\theta_2,\cdots,\theta_x\}$. $\theta\in\rho$ 表示为含有分配在该处理机节点上的备份进程，其相应主进程被调度在节点 N_θ上.

由定义 2 看出，我们在定义节点机时引入了处理机性能参数 τ，根据该性能参数对异构分布式系统中各节点机不同的处理能力来调度进程. 而不是像同构分布式系统那样把所有处理机的处理能力等同.

定理 1.　集合 ψ 中的进程在一个处理机失效时仍然可以继续运行，当且仅当该进程的主版本和副版本被调度到不同的处理机上. 该充要条件的形式化描述为

$$\text{对于}\ \forall P\in\psi,\text{满足条件}:P.P_{pm}.nps\neq P.P_{bk}.nps \tag{1}$$

定理 2.　设节点机满负荷工作时所能承受的最大负载为 100，则一次进程调度算法执行成功的充要条件是任意节点上的负载在单点故障发生前后始终小于或等于 100.

在故障发生前，算法执行成功的充要条件形式化描述为

$$\text{对于}\ \forall N\in\phi,\text{满足条件}:N.\lambda\le 100 \tag{2}$$

当一个节点发生故障后，由于分配在该节点上的主进程的相应副版本进程会被激活而接管主进程的工作，此时这些分配在其它节点上的副版本进程与原主进程有相同的负载. 也就是说，在其它节点上的负载都会产生一个增值. 在计算当前每个节点的负载时应把这个增值考虑进来. 为简化问题又不失一般性，我们假设节点 N_i已失效，并考虑在节点 N_j上负载的变化.

定义 ω_{ij}为主版本分配到节点机 N_i上且副版本分配到节点 N_j上的进程的集合. 即$\omega_{ij}=\{P\in\psi\mid P.P_{pm}.nps=i\wedge P.P_{bk}.nps=j\}(i,j<m\wedge i\neq j)$. 则当节点$N_i$失效时在节点$N_j$上负载的增值为$\sum\limits_{P\in\omega_{ij}}(P.P_{pm}.ld-P.P_{bk}.ld)$. 那么此时节点机 N_j上的实际负载为故障发生前的负载加上这一个增值，即$N_j.\lambda+\sum\limits_{P\in\omega_{ij}}(P.P_{pm}.ld-P.P_{bk}.ld)$. 若考虑$\forall N_i\in\phi(N_i\neq N_j)$发生故障时，在节点 N_j上负载所发生变化的最大值为$\max\limits_{N_i\in\phi}\left[\sum\limits_{P\in\omega_{ij}}(P.P_{pm}.ld-P.P_{bk}.ld)\right]$. 由此得出在任意单点故障发生后其余的任意一节点负载始终小于等于 100 的形式化描述为

$$\forall N_j\in\phi,N_j.\lambda+\max_{N_i\in\phi}\Big(\sum_{P\in\omega_{ij}}(P.P_{pm}.ld-P.P_{bk}.ld)\Big)\le 100 \tag{3}$$

因为在故障发生后每一节点的负载都会产生一个增值，其负载必定大于故障发生前的负载，即式(3)已蕴含了式(2)的条件. 因此定理 2 可重新描述为：设节点机满负荷工作时所能承受的最大负载为 100，则一次进程调度算法执行成功的充要条件是任意节点上的负载在发生任意单点故障后始终小于或等于 100. 在后面我们所提出的两个进程调度算法中在判断一次进程调度是否成功时用到了该定理.

3　启发式进程调度算法

3.1　两阶段分配算法

两阶段分配算法是 Kim 和 Lee 提出的一个经典的同构分布式环境下的静态负载均衡调度算法[13]. 该算法保证了在同构分布式模型下故障发生

前与故障发生后同时具有较好的负载均衡性. 在描述该算法之前我们首先给出两个定义:

(1)进程负载增值. 它指该进程的主版本负载与其对应的备份进程的负载之间的差值.

(2)进程实际负载. 它指该进程的主版本负载或其对应的备份进程的实际负载值.

该算法大致描述如下:第一阶段,使用标准的负载均衡算法分配主进程. 首先将所有进程按其主版本负载从大至小的非增序列排序. 通过使用启发式贪婪算法(greedy method)依次将负载最大的进程分配到负载最轻的节点上,使每一节点得到近似相等的负载. 第二阶段,根据在发生故障后每一节点可能产生的负载增值分配备份进程. 假设目前分配算法进行到节点 i 上,将节点 i 上的主进程对应的备份进程分配成$(n-1)$组(n 为系统中节点机的个数),且这$(n-1)$组备份进程在节点 i 失效时有近似相等的负载增值. 将这$(n-1)$组备份进程分配到除节点 i 以外的$(n-1)$个节点上,目的是为了保证在节点 i 失效后仍保持其余节点的负载均衡. 下面给出备份进程组的定义.

定义 3. 设分布式系统中节点个数为 n,则$(n-1)$组备份进程定义为集合 $\varepsilon=\{G_1,G_2,\cdots,G_{n-1}\}$. ε 中的元素 $G\in\varepsilon$ 表示为一个二元组,$G=(\xi,\pi)$. ξ 为调度到该组的备份进程的负载增值总和. π 为该组中备份进程的集合,定义为 $\pi=\{P_1.P_{\mathrm{bk}},P_2.P_{\mathrm{bk}},\cdots,P_j.P_{\mathrm{bk}},\varnothing\}$.

在分配备份进程组时同样依据贪婪算法,将总计为 $n\times(n-1)$组备份进程根据其实际负载值依次将负载最大的备份进程组分配到负载最小的节点机上,同时应满足两个条件:

(1)主进程分配在某一节点上的备份进程组不能又调度到该节点上;即 $\forall P\in\psi,P.P_{\mathrm{pm}}.nps\neq P.P_{\mathrm{bk}}.nps$;

(2)同一节点上不能有两个来自相同节点的备份进程组. 即$\exists P_i.P_{\mathrm{bk}}\in G_m.\pi$, $\exists P_j.P_{\mathrm{bk}}\in G_n.\pi$,$(P_i.P_{\mathrm{pm}}.nps=P_j.P_{\mathrm{pm}}.nps)\wedge(G_m\neq G_n)\rightarrow P_i.P_{\mathrm{pm}}.nps\neq P_j.P_{\mathrm{bk}}.nps$.

算法 1. 两阶段分配算法.

输入:进程集合 Ψ,n

输出:result,节点机集合 Φ

1. for $i=1$ to n do $N_i.\eta_m=\varnothing$; $N_i.\eta_s=\varnothing$; $N_i.\rho=\varnothing$; $N_i.\lambda=0$;

2. 按主版本负载非增的要求对 Φ 中的进程重新排序,并重新取名为 $P_1,P_2,\cdots,P_m$;即满足 $1\leq i<j\leq m\rightarrow P_i,P_j\in\psi$, $P_i.P_{\mathrm{pm}}.ld\geq P_j.P_{\mathrm{pm}}.ld$;

3. for $i=1$ to m do /* 调度主版本进程 */

3.1. 选择处理机 N_j 使之满足:$\forall N\in\phi(N_j.\lambda\leq N.\lambda)$;

3.2. $P_i.P_{\mathrm{pm}}.nps=j$; $N_j.\eta_m=N_j.\eta_m+\{P_i\}$;

3.3. $N_j.\lambda=N_j.\lambda+P_i.P_{\mathrm{pm}}.ld$;

4. if $\exists N\in\phi(N.\lambda>100)$ then $ret_val=Failed$; exit;

5. for $i=1$ to n do

5.1. 计算分配在节点 N_i 上的主进程与其对应的备份进程负载的差值;

5.2. 根据这些备份进程的负载差值按非增序列排序;

5.3. 将这些备份进程分成$(n-1)$组,使得每组有近似相等的负载增值;

6. 计算这 $n\times(n-1)$组备份进程的实际负载,并按非增序列对 $n\times(n-1)$组备份进程排序,取名为 $G_1,G_2,\cdots,G_{n\times(n-1)}$;

7. for $i=1$ to $n\times(n-1)$ do /* 调度备份进程 */

7.1. 选择处理机 N_k 使之满足:

$\forall N\in(\phi-\{N_{G_i.\pi.P_0.P_{\mathrm{pm}}.nps}\}),(k\neq G_i.\pi.P_0.P_{\mathrm{pm}}.nps\wedge N_k.\lambda\leq N.\lambda)\wedge(\forall\theta\in N_k.\rho,\theta\neq G_i.\pi.P_0.P_{\mathrm{pm}}.nps)$

/* 找出负载最小的节点 K,且满足两个条件:(1)来自节点 K 的备份进程组 G_i 不能分配在节点 K 上;(2)在任意节点中不能有来自同一节点的备份进程组,即来自某一节点的$(n-1)$组备份进程分配在$(n-1)$个不同的节点上 */

7.2. for $j=0$ to $G_i.\pi.P_j$!= $\varnothing$ do

7.2.1. $G_i.\pi.P_j.P_{\mathrm{bk}}.nps=k$; $N_k.\eta_s=N_k.\eta_s+\{G_i.\pi.P_j\}$;

7.2.2. $N_k.\lambda=N_k.\lambda+G_i.\pi.P_j.P_{\mathrm{bk}}.ld$;

7.2.3. $j++$;

7.3. $N_k.\rho=N_k.\rho+\{G_i.\pi.P_0.P_{\mathrm{pm}}.nps\}$;

8. if $\exists N_j\in\phi,N_j.\lambda+\max\limits_{N_i\in\{\phi-N_j\}}\Big(\sum\limits_{P\in\omega_{ij}}(P.P_{\mathrm{pm}}.ld-P.P_{\mathrm{bk}}.ld)\Big)>100$ then $ret_val=Failed$; exit;

9. $ret_val=Success$;

在两阶段算法中,将备份进程分成$(n-1)$组的目的是为了使每组备份进程有近似相等的负载增值并将这$(n-1)$组备份进程分配给$(n-1)$个不同的节点,保证了在单点故障发生后节点间的负载均衡性. 而计算每组备份进程的实际负载并根据这 $n\times(n-1)$组备份进程的实际负载分配进程保证了在故障发生前的节点间负载均衡性. 然而从两阶段算法看出,在计算每一节点机的当前负载时,并未把处理机的性能参数 τ 考虑进来,而是看作所有处理机有同样的处理能力,因此该算法只适合同构分布式模型下的容错调度算法.

3.2 异构分布式模型下的容错调度算法

在实际的应用中经常会遇到异构分布式的环境,即每个节点有不同的处理能力. 此时进程的负载并不是一个固定的值,而要随所分配的节点的性能而变化. 同一个进程分配到性能较高的节点上对该节点的负载相对较轻,若分配到性能较差的节点上

相对有较高的负载. 因此我们在同构分布式模型的基础上引入了处理机性能参数 τ，该值以 1 为中心上下浮动，τ 值越小节点的性能就越高. 每个进程都有一个主版本与从版本的标准负载值 $P.P_{\mathrm{pm}}.ld$，$P.P_{\mathrm{bk}}.ld$，当某一进程 P_i 主版本分配到节点 N_j 上，从版本分配到节点 N_k 上时，那么该进程在两节点上的实际负载值分别为 $P_i.P_{\mathrm{pm}}.ld\times N_j.\tau$ 和 $P_i.P_{\mathrm{bk}}.ld\times N_k.\tau$. 由于这种负载以及负载增值随节点变化而变化的不确定性因素，在考虑异构分布式环境下的进程调度算法时，不可能像同构分布式模型那样在故障发生前后同时保证具有最优的负载均衡性. 因此我们根据对备份进程不同的调度方案，提出了进程实际负载优先考虑算法（HDALF）和进程负载增值优先考虑算法（HDLDF）. HDALF 算法根据进程的实际负载分配进程，优先考虑故障发生前的负载均衡性. HDLDF 算法根据进程主从版本的负载差值分配备份进程，优先考虑故障发生后的负载均衡性.

3.2.1　实际负载优先考虑算法

实际负载优先考虑算法，简称为 HDALF（Heterogeneous Distributed-system Actual Load First）算法. 该容错算法与两阶段算法类似，只是在计算进程的负载时要将该进程所分配的节点机的性能参数为系数再乘以该节点的标准负载. 首先将集合 ψ 中的进程按其主版本负载非增排序，然后采用启发式贪婪算法对这些进程的主版本进行调度. 根据分配在每一节点上的主版本进程，将该节点上的主版本进程对应的备份进程分配成（$n-1$）组，且这（$n-1$）组备份进程在该节点失效时有近似相等的负载增值. 注意此时所说的负载增值是指标准负载增值，具体分配到不同的节点上进程的负载增值会有相应的变化. N 个节点总共有 $n\times(n-1)$ 组备份进程，同样依据贪婪算法根据每组进程的实际负载进行调度，将实际负载值较大的组分配到负载相对较小的处理机上. 我们在最后一步考虑的是备份进程的实际负载进行的调度，这保证了在故障发生前的负载均衡性. 然而在故障发生后，由于具体分配到每个节点的备份进程组的负载增值会发生变化，这样就不能保证故障发生后有最优的节点负载均衡. 这样的分配方案优先考虑了备份进程的实际负载，确保在未发生单点故障时节点有最优的负载均衡性.

算法 2.　HDALF 算法.

输入：进程集合 Ψ，处理机个数 n

输出：result，节点机集合 Φ

1. for $i=1$ to n do $N_i.\eta_m=\varnothing$； $N_i.\eta_s=\varnothing$； $N_i.\rho=\varnothing$； $N_i.\lambda=0$；

2. 按主版本负载非增的要求对 Φ 中的进程重新排序，并重新取名为 $P_1,P_2,\cdots,P_m$； 即满足 $1\le i<j\le m\rightarrow P_i,P_j\in\psi, P_i.P_{\mathrm{pm}}.ld\ge P_j.P_{\mathrm{pm}}.ld$；

3. for $i=1$ to m do　/* 调度主版本进程 */

3.1. 选择处理机 N_j 使之满足：$\forall N\in\phi(N_j.\lambda\le N.\lambda)$

3.2. $P_i.P_{\mathrm{pm}}.nps=j$； $N_j.\eta_m=N_j.\eta_m+\{P_i\}$；

3.3. $N_j.\lambda=N_j.\lambda+P_i.P_{\mathrm{pm}}.ld\times N_j.\tau$；

4. if $\exists N\in\phi, N.\lambda>100$ then $ret_val=Failed$； exit；

5. for $i=1$ to n do

5.1. 计算分配在节点 N_i 上的主进程与其对应的备份进程负载的差值；

5.2. 根据这些备份进程的负载差值按非增序列排序；

5.3. 将这些备份进程分成（$n-1$）组，使得每组有近似相等的负载增值；

6. 计算这 $n\times(n-1)$ 组备份进程的实际负载，并按非增序列对 $n\times(n-1)$ 组备份进程排序，取名为 $G_1,G_2,\cdots,G_{n\times(n-1)}$（$G_i=\{P_1.P_{\mathrm{bk}},P_2.P_{\mathrm{bk}},\cdots,P_j.P_{\mathrm{bk}},\varnothing\}$）；

7. for $i=1$ to $n\times(n-1)$ do　/* 调度备份进程 */

7.1. 选择处理机 N_k 使之满足：

$\forall N\in(\phi-\{N_{G_i.\pi.P_0.P_{\mathrm{pm}}.nps}\}),(k\ne G_i.\pi.P_0.P_{\mathrm{pm}}.nps)\wedge(N_k.\lambda\le N.\lambda)\wedge(\forall\theta\in N_k.\rho,\theta\ne G_i.\pi.P_0.P_{\mathrm{pm}}.nps)$

/* 找出负载最小的节点 K，且满足两个条件：(1) 来自节点 K 的备份进程组 G_i 不能分配在节点 K 上；(2) 在任意节点中不能有来自同一节点的备份进程组，即来自某一节点的（$n-1$）组备份进程分配在（$n-1$）个不同的节点上 */

7.2. for $j=0$ to $G_i.\pi.P_j$!= $\varnothing$ do

7.2.1. $G_i.\pi.P_j.P_{\mathrm{bk}}.nps=k$；$N_k.\eta_s=N_k.\eta_s+\{G_i.\pi.P_j\}$；

7.2.2. $N_k.\lambda=N_k.\lambda+G_i.P_j.P_{\mathrm{bk}}.ld\times N_k.\tau$；

7.2.3. $j++$；

7.3. $N_k.\rho=N_k.\rho+\{G_i.P_0.P_{\mathrm{pm}}.nps\}$；

8. if $\exists N_j\in\phi, N_j.\lambda+\max\limits_{N_i\in\{\phi-N_j\}}\left(\sum\limits_{P\in\omega_{ij}}(P.P_{\mathrm{pm}}.ld-P.P_{\mathrm{bk}}.ld)\right)>100$ then $ret_val=Failed$； exit；

9. $ret_val=Success$；

HDALF 算法的时间复杂度分析如下.

主进程分配阶段：对 m 个主进程按负载非增排序的时间复杂度为 $O(m\log m)$；m 个进程分配给 n 个节点所需时间 $O(m\log n)$. 备份进程分配阶段（在此考虑最坏情况下的时间复杂度，即 m 个进程分配到一个节点的情况）：对 m 个主进程相对应的备份进程排序的时间复杂度为 $O(m\log m)$；将 m 个备份进程分成（$n-1$）组具有近似相等的负载增值并计算每组的实际负载所需时间 $O(m\log n+m)$；以上过程重复 n 次，所需时间 $O(n(m\log m+m\log n+m))$；将 $n\times(n-1)$ 组备份进程排序使用的时间是 $O(n^2\log n)$；

再对 n 个节点排序并找出适当的节点放置某一备份进程组所需时间为 $O(n\log n+n)$；因此，分配 $n\times(n-1)$ 组备份进程到各节点需要 $O(n(n-1)(n\log n+n))$ 时间. 备份进程分配阶段的时间复杂度为

$$O(n(m\log m+m\log n+m)+n^2\log n+n(n-1)(n\log n+n))=O(nm\log mn+n^3\log n),$$

最后，HDALF 算法的时间复杂度总计为

$$O(m\log m+m\log n+nm\log mn+n^3\log n)=O(nm\log mn+n^3\log n).$$

3.2.2 负载增值优先考虑算法

负载增值优先考虑算法，简称为 HDLDF(Heterogeneous Distributed-system Load Difference Fist)算法. 它与 HDALF 算法不同的是，在分配备份进程时优先考虑的是备份进程的负载差值，而不是根据其实际负载值分配. 这样保证了在发生故障后有较好的负载均衡性. 然而，由于没有按照备份进程的实际负载分配到各节点，因此在故障发生前节点的负载均衡性不如 HDALF 算法. HDLDF 算法所采用的方案是当分配到某节的主版本进程其对应的备份进程分成 $(n-1)$ 组时，规定所分的第 k 组 $(1<k<=n-1)$ 备份进程即分配到除该节点以外的第 k 个节点. 则该组备份进程的实际负载增值可通过所分配的节点 k 知道. $P_m.loadDiff=(P_m.P_{pm}.ld-P_m.P_{bk}.ld)\times N_k.\tau$.

算法 3. HDLDF 算法.

输入：进程集合 Ψ，处理机个数 n

输出：result，节点机集合 Φ

1. for $i=1$ to n do $N_i.\eta_m=\varnothing$；$N_i.\eta_s=\varnothing$；$N_i.\rho=\varnothing$；$N_i.\lambda=0$；

2. 按主版本负载非增的要求对 Φ 中的进程重新排序，并重新取名为 $P_1,P_2,\cdots,P_m$；即满足 $1\le i<j\le m\rightarrow P_i,P_j\in\psi(P_i.P_{pm}.ld\ge P_j.P_{pm}.ld)$；

3. for $i=1$ to m do /* 调度主版本进程 */

3.1. 选择处理机 N_j 使之满足：$\forall N\in\phi(N_j.\lambda\le N.\lambda)$；

3.2. $P_i.P_{pm}.nps=j$；$N_j.\eta_m=N_j.\eta_m+\{P_i\}$；

3.3. $N_j.\lambda=N_j.\lambda+P_i.P_{pm}.ld\times N_j.\tau$；

4. if $\exists N\in\phi, N.\lambda>100$ then $ret_val=Failed$；exit；

5. for $i=1$ to n do

5.1. 计算分配在节点 N_i 上的主进程与其对应的备份进程负载的差值；

5.2. 根据这些备份进程的负载差值按非增序列排序；

5.3. for $j=1$ to $n-1$ do

5.3.1. $G_j.\xi=0$；

5.3.2. $G_j.\pi=\varnothing$；

5.4. for $j=1$ to $Ni.\eta_m.P_j$!= $\varnothing$ do /* 将这些备份进程分成 $(n-1)$ 组，使得每组有近似相等的负载增值 */

5.4.1. 选择组 G_k 使之满足：$\forall G\in\varepsilon(G_k.\xi\le G.\xi)$

5.4.2. if $k>=i$ then $N_i.\eta_m.P_j.P_{bk}.nps=k+1$；
else $N_i.\eta_m.P_j.P_{bk}.nps=k$；

5.4.3. $G_k.\pi=G_k.\pi+\{N_i.\eta_m.P_j.P_{bk}\}$；
$G_k.\xi=G_k.\xi+(N_i.\eta_m.P_j.P_{pm}.ld-N_i.\eta_m.P_j.P_{bk}.ld)\times N_i.\tau$；

5.3.4. $j++$；

6. if $\exists N_j\in\phi, N_j.\lambda+\max\limits_{N_i\in\{\phi-N_j\}}\left(\sum\limits_{P\in\omega_{ij}}(P.P_{pm}.ld-P.P_{bk}.ld)\right)>100$ then $ret_val=Failed$；exit；

7. $ret_val=Success$；

HDLDF 算法在将每一节点的备份进程分成 $(n-1)$ 组之后对整个进程的分配就已完毕，因此对该算法的时间复杂度分析要比 HDALF 算法简便. 该算法的复杂度分析如下.

对 m 个主进程按负载非增排序所需时间 $O(m\log m)$，m 个进程分配给 n 个节点需要时间 $O(m\log n)$；备份进程分配阶段仍然考虑最坏情况下的时间复杂度，即 m 个进程分配到一个节点的情况. 对 m 个主进程相对应的备份进程排序所需时间 $O(m\log m)$；将 m 个备份进程分成 $(n-1)$ 组的时间复杂度为 $O(m\log n)$；以上过程重复 n 次，所需时间 $O(n(m\log m+m\log n))$；整个 HDLDF 算法所需时间复杂度为

$$O(m\log m+m\log n+n(m\log m+m\log n))=O(nm\log mn).$$

假设队列中需要调度进程数远大于分布式系统中的节点数，即 $m>n^2$，由 HDALF 算法所得结果 $O(nm\log mn+n^3\log n)=O(nm\log mn)$. 此时 HDALF 算法与 HDLDF 算法有相同的时间复杂度.

4 性能分析

在这一节中，我们对 HDALF 算法与 HDLDF 算法的负载均衡性作出了对比与分析，并对两阶段算法、HDALF 算法和 HDLDF 算法在异构分布式环境下的资源利用效率作出了比较.

4.1 节点间的负载均衡性

节点的负载均衡问题可以通过最大负载节点与最小负载节点之间的差值来衡量. 定义 α 为节点中所包含的最大负载与最小负载之间的差值，表示为

$$\alpha(N)=\max(N.\lambda)-\min(N.\lambda),$$

那么在故障发生前这个差值表示为 $\alpha(N_{non})$，在故障发生后表示为 $\alpha(N_{fault})$. 基于负载均衡的容错调度算法问题就是为每个进程找出相应的节点，使得在

所有进程分配完毕后 $\alpha(N_{non})$ 与 $\alpha(N_{fault})$ 的值尽量达到最小,同时满足式(1)与式(3)蕴含的条件. 然而要使 $\alpha(N_{non})$ 与 $\alpha(N_{fault})$ 的值同时达到最小往往会出现矛盾,我们可以通过给它们建立一个权值,这个权值决定两个不同的衡量标准所占的重要程度. 定义为

$$\beta = w_1 \times \alpha(N_{non}) + w_2 \times \alpha(N_{fault}),$$

那么本文所讨论的进程调度问题就是如何使得在所有进程分配完毕后 β 值最小.

我们可以通过仿真的方法得到 HDALF 算法与 HDLDF 算法的负载均衡性. 在仿真中使用的环境变量如下:首先要保证每一节点的负载不能超过其极限值 100,即满足式(3). 进程主版本的负载等概率分布在 0.5～9 之间,因为这个范围是绝大部分进程负载所属的范围. 从版本的进程负载以等概率分布在其主版本负载的 5%～10%之间. 处理机的性能参数 τ 在 0.5～1.5 之间变化,即性能最高的节点机是性能最差节点机处理能力的三倍. 在实验中首先随机生成主从版本的进程集合和节点机集合,关键是生成参数 $P.P_{pm}.ld$,$P.P_{bk}.ld$ 和 $N.\tau$,其概率密度函数服从均匀分布. 之后分别使用 HDALF 算法和 HDLDF 算法进行容错调度,输出相应的节点机集合 ϕ,并计算出 $\alpha(N_{non})$,$\alpha(N_{fault})$ 及 β 值. 该步骤重复 100 次,求其平均值.

图 2 显示了 HDALF 算法与 HDLDF 算法在故障发生前后的 $\alpha(N_{non})$ 与 $\alpha(N_{fault})$ 值,其中使用的处理机个数为 25 个,进程个数在 100～400 之间变化. 图中上面的两条曲线代表发生单点故障后两个算法的 $\alpha(N_{fault})$ 值,下面的两条曲线表示故障发生前两个算法的 $\alpha(N_{non})$ 值. 从图中看出,在故障发生前,HDALF 算法有良好的负载均衡性,HDLDF 算法相对较差. 但在故障发生后,HDALF 算法的性能有明显下降,并随着进程数的增加,负载均衡性有继续

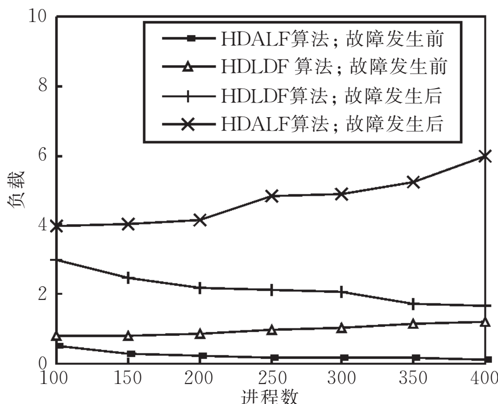

图 2　HDALF 与 HDLDF 算法的 $\alpha(N_{non})$ 与 $\alpha(N_{fault})$ 值

下降的趋势. 而此时 HDLDF 算法的负载均衡性要好于 HDALF 算法. 这样的结果是由于 HDALF 算法优先考虑的是故障发生前的实际负载,HDLDF 算法优先考虑故障发生后的负载增值.

为了比较 HDALF 算法与 HDLDF 算法的总体负载均衡性,图 3 显示了随着进程个数变化 HDALF 算法与 HDLDF 算法的 β 值. 图中使用的处理机个数仍然为 25 个,进程个数在 100～400 之间变化. 从图中看出,若假设系统在发生故障前后有相等的权值,即 $w_1 = w_2 = 1$,此时 HDLDF 算法的 β 值小于 HDALF 算法. 当 $w_1 = 1.5$,$w_2 = 0.5$ 时,HDLDF 算法与 HDALF 算法的 β 值近似相等. 这也说明当 w_1 值增大而 w_2 值减小时,将有利于 HDALF 算法的总体负载均衡性,反之则有利于 HDLDF 算法. 因此对于两种算法的选用可取决于系统发生故障前和发生故障后的负载均衡性所占的权值. 若 $w_1 \geq 3w_2$,可选用 HDALF 算法;否则可选用 HDLDF 算法进行容错调度.

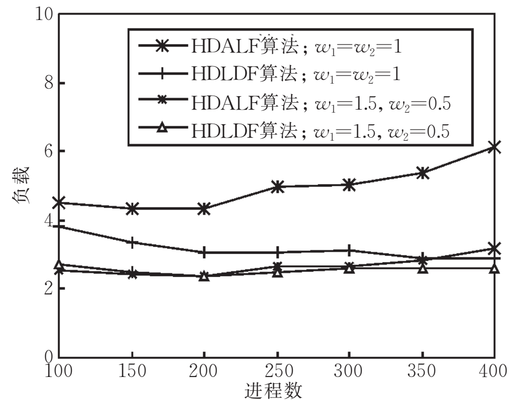

图 3　HDALF 与 HDLDF 算法的 β 值

4.2　算法资源利用率

HDALF 和 HDLDF 算法都是启发式的异构分布式模型下基于负载均衡的容错调度算法,HDALF 和 HDLDF 算法的共同特点是输入带有主从版本的进程集合 ψ 和处理机个数 n,算法可判断出 n 个处理机能否保证所有进程调度成功,即在故障发生前或故障发生后每个处理机上的负载都不会超过其所能承受的最大极限值. 在系统发生故障前和发生故障后负载均衡的重要性相等的情况下,可通过对算法资源利用率的分析对各种算法的选取作出选择. 这里使用文献[7]提出的求解最小处理机个数算法(FMNP),通过对所需最小处理机个数的讨论,得出两阶段算法、HDALF 算法和 HDLDF 算法的资源利用率以及相关参数对系统资源利用率的影响. 我们使用 $Func_FT$ 参数代表 FMNP 算法使用三种不

同的容错调度算法进行容错调度.

算法 4. FMNP 算法.

输入:进程集合 Ψ,调用参数 $Func_FT$

输出:$LeastNode$,节点集合 Φ

1. $Lower=1$; $Upper=m$;

2. $LeastNode=[(Lower+Upper)/2]$;

if ($Lower=LeastNode$) then

$LeastNode=LeastNode+1$; exit;

3. $Func_FT(\Psi, LeastNode; Result, \Phi)$; /* 调用容错调度算法(Two-stage Allocation or HDALF or HDLDF) */

4. if ($Result=Success$) then $Upper=LeastNode$;

else $Lower=LeastNode$; goto 2;

在实验中与 4.1 节的实验步骤相似,首先随机产生主从版本对的进程集合及节点机集合,所取参数值与 4.1 节相同.输入主从版本进程集合 ψ 和调用参数 $Func_FT$,FMNP 算法通过 $Func_FT$ 参数分别调用两阶段算法、HDALF 算法和 HDLDF 算法并分别输出三种算法所需最小处理机个数 $LeastNode$.最后将 FMNP 算法调用 100 次,求其 $LeastNode$ 的平均值.通过对系统所需最小处理机个数的讨论,可以对三种算法在异构分布式环境下的资源利用率作出比较.

图 4 为随着调度进程个数的改变,三种算法所需最小处理机个数的变化曲线.实验中节点性能参数 τ 取值在 0.5~1.5 之间,进程个数从 50~400 之间变化.图中结果表明,随着所调度进程个数的增加,所需的最小处理机个数呈线性增加趋势.这是因为调度的进程数越多,就需要更多节点机保证每一节点的负载不会超过其所能承受的最大值.同时还可以比较出,在异构分布式模式下,两阶段算法所需的最小处理机个数最多.这是因为两阶段算法是在同构模型下提出的容错调度算法,这样就不能充分利用处理机性能差异的特点.HDALF 算法与 HDLDF 算法利用了不同处理机的不同处理能力,调度的结果将性能较高的节点机分配了较多的进程,而性能较差的节点机所分配的进程较少.这样性能好的节点机任务比较繁重,充分利用了性能较高的处理机资源,使调度的结果更加紧凑,所需的最小节点机个数也会相对较少.同时,HDLDF 算法比 HDALF 算法所需最小节点机个数略少一些,这是由于在 w_1 与 w_2 权值相等的情况下,HDLDF 算法的 β 值比 HDALF 算法要小的缘故,即此时 HDLDF 算法有更好的负载均衡性,所需的最小节点机个数也会相对较少.

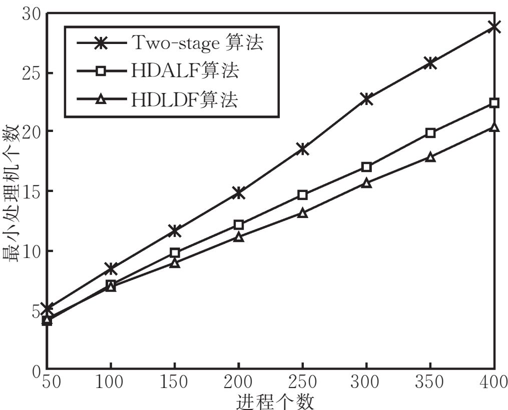

图 4 三种不同算法性能的比较

图 5 显示了处理机参数 τ 与最小处理机个数的关系.处理机性能参数 τ 是反映分布式系统中处理机性能差异的标准,参数 τ 的值变化区间越大,说明系统中处理机之间性能的差异越大.该实验中我们取 τ 值分别在 0.8~1.2,0.5~1.5,0.2~1.8 三个区间上变化,调用 HDALF 算法进行分析,得出三种不同的最小处理机个数曲线值.从结果看出,τ 值的变化区间越大,异构分布式系统所需的节点机个数越少.其原因是节点间性能差异越大,就有更多的进程调度到性能更高的节点上,性能好的节点资源利用率相对更多,所需的节点数相对就较少.假设参数 τ 恒等于 1 时,HDALF 算法与同构分布式模型下的两阶段算法会有相同的值.同样对 HDLDF 算法的作用具有相同的特征,我们不再列出.

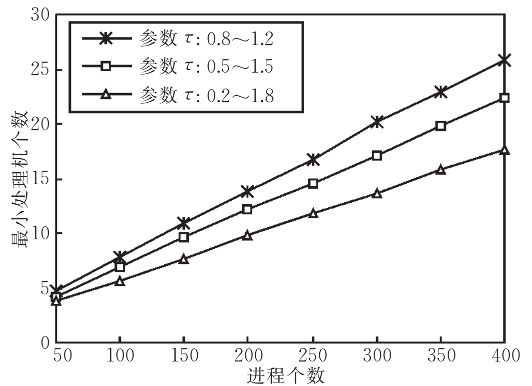

图 5 不同的 τ 值范围对 HDALF 算法性能的影响

图 6 显示了三种算法的最小处理机个数随参数 τ 变化的关系.图中定义了一个变量 σ,表示 $\tau\in(1-\sigma, 1+\sigma)$.我们取 σ 值在 0.1~0.9 之间变化,调度进程个数为 250 个.从图中看出,随着 σ 值的变化,两阶段算法不受任何影响.这是因为两阶段算法未考虑处理机之间的性能差异,始终保持同样的水平.而 HDALF 算法与 HDLDF 算法曲线随着 σ 值的增加逐渐下降,并随着 σ 值的不断增加,有下降更加明显

的趋势.说明了 HDALF 算法与 HDLDF 算法能很好地适应异构分布式系统，系统中处理机之间性能差异越大，处理机的资源利用率就越充分.同时还可以得出，HDLDF 算法的曲线值略小于 HDALF 算法，因此从资源利用率的角度看，当 $w_1=w_2$ 时，也应该考虑使用 HDLDF 算法.

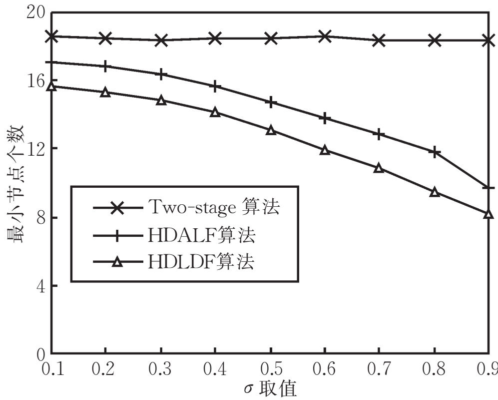

图 6　处理机个数与参数的关系曲线

5　总　结

本文所做的主要工作有以下几点：首先提出了异构分布式系统下的容错调度模型，根据此模型提出基于被动进程复制模式的负载均衡容错调度算法 HDALF 和 HDLDF，并分别给出了两种算法的时间复杂度.这两个容错调度算法是根据同构分布式模型下的两阶段算法转换得来.HDALF 算法优先考虑进程的实际负载，保证在故障发生前的负载均衡性.HDLDF 算法优先考虑进程主从版本的负载差值，保证了在故障发生后有更好的负载均衡性.之后，我们给故障发生前后的负载均衡性设定相应的权值，对算法的总体负载均衡性作出了讨论.并通过对所需最小处理机个数的讨论，对两阶段算法、HDALF 算法和 HDLDF 算法的资源利用率作出了比较.实验结果表明，无论从算法的负载均衡性还是从处理机资源利用率的角度来看，HDLDF 算法都优于 HDALF 算法.两阶段算法由于仅仅考虑各处理机相同情况的同构分布式环境，因此在异构分布式环境下的性能表现最差.

基于主从版本的容错调度算法是计算机容错系统研究的重要分支之一，其进程调度的优化问题属于 NP 难度问题.本文所研究的进程调度属于静态进程调度，即进程分配的开始阶段一次将所有进程全部分配完毕.静态分配算法对于进程的信息预先已经知道的情况下比较有效.例如在线事务处理、实时系统的计算环境.这些计算环境大部分进程都有连续性和周期性，对进程的负载及到达时间容易掌握.而对于进程的各项信息预先并不知道的计算环境，就需要根据当时各节点机的状况动态地插入新进程或移走已运行完毕的进程.这种异构分布式模型下的动态进程调度算法是我们尚在研究的内容.

参　考　文　献

1　Hermann K., Gunter G.. TTP——A protocol for fault-tolerant real-time systems. IEEE Computer, 1994, 27(1): 14～23

2　Birman K.P.. The process group approach to reliable. Distributed computing. Communications of the ACM, 1993, 36(12): 37～53

3　Siewiorek D.P., Swartz R.S.. Reliable System Design: The Theory and Practice. New York: Digital Press, 1992

4　Xu L.H., Bruck J.. Deterministic voting in distributed systems using error-correcting codes. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(8): 813～824

5　Lin Tein-Hsiang, Shin K.G.. Damage assessment for optimal rollback recovery. IEEE Transactions on Computers, 1998, 47(5): 603～613

6　Nieuwenhuis L.J.M.. Static allocation of process replicas in fault-tolerant computing systems. In: Proceedings of the 20th International Symposium on Fault-Tolerant Computing, Newcastle, UK, 1990, 298～306

7　Qin Xiao, Pang Li-Ping, Han Zong-Fen. Algorithms of fault-tolerant scheduling in distributed real-time systems. Chinese Journal of Computers, 2000, 23(10): 1056～1063(in Chinese)
(秦　啸，庞丽萍，韩宗芬.分布式实时系统的容错调度算法.计算机学报，2000, 23(10): 1056～1063)

8　Bannister J.A., Trivedi K.S.. Task allocation in fault-tolerant distributed systems. Acta Informatica, 1983, 20(3): 261～281

9　Shatz S.M., Wang J.P., Goto M.. Task allocation for maximizing reliability of distributed computer systems. IEEE Transactions on Computers, 1992, 41(9): 1156～1168

10　Chereque M., Powell D. *et al*.. Active replication in Delta-4. In: Proceedings of the 22nd International Symposium on Fault-Tolerant Computing, Boston, MA, 1992, 28～37

11　Speirs N., Barrett P.. Using passive replicates in Delta-4 to provide dependable distributed computing. In: Proceedings of the 19th International Symposium on Fault-Tolerant Computing, Chicago, Illinois, 1989, 184～190

12　Davoli R., Giachini L.-A., Babaoglu Ö. *et al*.. Parallel computing in networks of workstations with Paralex. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(4): 371～384

13　Kim Jong, Lee Heejo, Lee Sunggu. Process allocation for load

distribution in fault-tolerant multicomputers. In: Proceedings of the 25th International Symposium on Fault-Tolerant Computing, Pasadena, CA, 1995, 124～129

14 Lee H., Kim J., Hong S. *et al.*. Fault-tolerant process allocation with load balancing. In: Proceedings of the 1995 Pacific Rim International Symposium on Fault-Tolerant Systems, Nowport Beach, CA, 1995, 124～129

15 Garey R., Johnson D. S.. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W. H. Freeman Company, 1979

GUO Hui, born in 1978, Ph.D.. His current interests include distributed systems, network storage, streaming media and multimedia communication.

WANG Zhi-Guang, born in 1964, associate professor. His research interests include distributed computing, artificial intelligence and pattern recognition.

ZHOU Jing-Li, born in 1945, professor, Ph.D. supervisor. Her research interests include multimedia computing, computer network and network storage.

Background

This work was supported by the National High Technology Research and Development Program of China (863 Program) under grant No. 2001AA111011.

Process allocation is a pivotal issue in distributed systems. In this area, primary-backup process model is an important scheme for system fault-tolerance. The traditional algorithms (such as two-stage algorithm) based on primary-backup process mode only adapt to homogeneous distributed system. As the heterogeneous multi-processor system is more prevalent now, authors' research work focused on process allocation issue in heterogeneous multi-processor circumstance, and the allocation results should guarantee load balance for each processor.

In this paper, two heuristic approximation algorithms are proposed and analyzed. The experimental results show that the proposed algorithms have significantly better performance especially for heterogeneous distributed system environment.

第三届全国 Web 信息系统及其应用学术会议(WISA2006)
征文通知

Web 在人类信息的存储和交换过程中发挥着日益重要的作用,适用于网络平台、动态特性的 Web 技术层出不穷,提高办公效率,节约资源消耗和扩大信息共享的 Web 应用和服务蓬勃发展.但随着 Web 规模的不断膨胀,Web 上数据资源以爆炸性的趋势飞速增长,而且信息的结构和内容越来越复杂,使得管理、查询和使用 Web 信息变得愈加困难.

全国 Web 信息系统及其应用会议(WISA)是中国计算机学会电子政务与办公自动化专委会主办的系列会议.首届会议 WISA2004 于 2004 年 10 月在武汉圆满召开,会议共收到应征论文 368 篇,其中 54 篇论文在《武汉大学学报(英文)》(EI 源刊)作为正刊专辑发表(已经全部被 EI 收录).第二届会议 WISA2005 于 2005 年 9 月在沈阳召开,会议共收到应征论文 606 篇,录用 239 篇,其中 62 篇论文在《武汉大学学报(英文)》(EI 源刊)作为正刊专辑发表,118 篇在《计算机科学》上发表,59 篇在由清华大学出版社出版的会议论文集《Web 信息系统与技术》上发表.

WISA2006 将于 2006 年 10 月在南京召开.会议将继续这一良好的传统,在 Web 技术、信息系统、电子政务与办公自动化等方面进行深入广泛的学术交流.会议论文集仍将分两部分出版,录用论文中将选择出 60 篇左右高水平论文,以英文方式继续由《武汉大学学报(英文)》(EI 源刊)正刊专辑出版,中文论文集将由著名计算机核心期刊《计算机科学》专刊和中央级出版社出版.会议期间除进行会议论文交流外,还将邀请著名学者作特邀报告.本次会议仍将评选大会优秀学生论文.

一、征文范围(包括但不限于)

Web 信息挖掘与检索
Web 测试与 Web 应用的质量保证
Web 与数据库技术
组件与中间件技术
Web 应用框架和体系结构
决策支持与分析技术
Web 系统度量与分析技术
语义 Web 与智能 Web
Web 与网格计算
工作流模型
Web 信息系统环境与基础
自动文本索引与分类技术
Web 信息系统开发工具
电子政务与办公自动化发展现状与趋势
Web 站点逆向工程与维护技术
多媒体数据管理
XML 与半结构化数据管理
代理技术及信息管理
Web 与信息系统安全性
电子政务与电子商务框架及应用

二、来稿要求

1. 本次会议只接受 E-mail 投稿.
2. 中英文稿均可,一般不超过 6000 字.为了便于出版论文集,来稿必须附中英文摘要、关键词、资助基金与主要参考文献,注明作者及主要联系人姓名、工作单位、详细通信地址(包括 E-mail 地址)与作者简介.稿件要求采用 WORD 或 PDF 格式.

三、联系信息

1. 投稿地址:东北大学信息科学与工程学院 王国仁(wanggr@mail.neu.edu.cn).
2. 会务情况:东南大学计算机科学与工程系 徐宝文 许蕾(xlei@seu.edu.cn).
3. 大会网站:http://www.neu.edu.cn/wisa2006/.

四、重要日期

1. 征文截止日期:2006 年 3 月 25 日.
2. 录用通知发出日期:2006 年 4 月 15 日.
3. 正式论文提交日期:2006 年 4 月 30 日.