doi:10.11959/j.issn.1000-436x.2015203

基于遗传算法的异构系统多副本容错调度算法

何忠政[1]，门朝光[1]，陈拥军[2]，李香[1]
(1. 哈尔滨工程大学 计算机科学与技术学院，黑龙江 哈尔滨 150001；2. 中国新兴建设开发总公司，北京 100143)

摘 要：针对异构系统中基于多副本机制的容错调度方法忽略调度 makespan、任务间依赖与系统链路失效及严格调度方式调度 makespan 较长问题，首先提出通用调度方式下同时考虑节点和链路失效的可靠性计算方法；然后给出该通用调度问题的 0-1 整数规划模型；接着提出可靠性意识多副本任务通用调度(RAMD_TGS, reliability-aware multi-duplication task general scheduling)算法，通过遗传算法种群进化来搜索副本映射节点和开始执行时间。实验表明该算法不仅满足可靠性要求，而且与严格调度方式相比能进一步减小调度 makespan，该算法资源占用开销也是可接受的。

关键词：容错调度；异构分布式系统；任务副本；遗传算法
中图分类号：TP302 **文献标识码**：A

Multi-duplication fault tolerant scheduling algorithm based on genetic algorithm in heterogeneous systems

HE Zhong-zheng[1], MEN Chao-guang[1], CHEN Yong-jun[2], LI Xiang[1]
(1. Department of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China;
2. China Xinxing Construction & Development General Corporation, Beijing 100143, China)

Abstract: The fault-tolerant task scheduling mechanisms based on multi-duplication didn't consider the scheduling makespan, the dependencies between tasks, the failures of the links and the longer scheduling makespan caused by the strict scheduling method in the heterogeneous distributed system. So the reliability calculation method that can involve the processor failures and the link failures was proposed firstly. Then the 0-1 integer linear program was proposed for the general scheduling problem. At last, the RAMD_TGS(reliability-aware multi-duplication task general scheduling) algorithm was proposed to solve the 0-1 integer linear program. The algorithm searched the mapped processor and the start execution time on the mapped processor for the task duplication by the evolution of the genetic algorithm. The experiments show that the RAMD_TGS algorithm can meet the reliability requirements and outperforms the existing scheduling algorithms based on the strict scheduling method in terms of scheduling makespan. The resource usages of the algorithm are also acceptable.

Key words: fault tolerant scheduling; heterogeneous distributed system; task duplication; genetic algorithm

1 引言

随着高速网络的出现，将分布式、低成本而且极有可能是异构的资源连接作为计算环境是可行的，因此分布式系统（如云计算、网格计算、分布式移动计算）中计算机的异构性将会逐步增强，这提供了一种称为异构分布式计算(HDC, heterogeneous distributed computing)系统的平台[1]。HDC 系统已成为流行的高性能计算和信息处理的计算设备，且已逐步被关键系统所使用，如飞行控制、工业处理控制等[1]。HDC 系统比同构系统和中心控制系统更复杂，额外的复杂性可能造成更多的系统失

收稿日期：2014-03-07；**修回日期**：2015-03-17
基金项目：国家自然科学基金资助项目（60873138, 61100004）；中央高校基本科研业务费专项基金资助项目（HEUCF100607）
Foundation Items: The National Natural Science Foundation of China (60873138, 61100004); The Fundamental Research Funds for the Central Universities (HEUCF100607)

效。在 HDC 系统中，安全关键应用程序不仅要在故障发生的情况下能够容错，还要满足时间约束条件。

HDC 系统中有效的任务调度算法在满足用户或系统要求及实现高性能方面扮演着关键角色。任务调度旨在于确定任务在调度节点的开始执行时间，在满足任务间依赖关系条件下，最大化可靠性和最小化调度 makespan。当前任务调度问题大多针对独立任务，忽视了任务间数据关联与依赖约束关系，同时最小化调度 makespan 和最大化可靠性常常是相互冲突的[1]，因此设计必须同时考虑调度 makespan 和可靠性的调度算法。文献[2]中的可靠性计算方法是基于严格调度方式，在通用调度方式下可靠性计算方法并不如此。该文在时间测试时并未考虑任务可靠性，任务可靠性与其开始执行时间紧密关联，因为任务开始执行时间决定任务能够接收到的先行任务消息数量。通用调度方式可进一步减小当前调度任务副本开始执行时间，因此通用调度 makespan 很有可能比严格调度小。本文采用基于通用调度方式的多副本容错调度机制，在满足可靠性要求前提下，利用遗传算法优化调度 makespan。

2 相关工作

容错调度算法已进行了大量研究，但许多容错调度算法没有采用多副本容错方法，所以其可靠性受单个节点可靠性的限制。针对这些算法在可靠性提升方面的不足，容错调度可采用被动复制(PB, primary/backup)机制和主动复制机制来提高可靠性。

PB 机制将任务的 2 个版本分配给不同节点，在主版本节点失效时，任务会在副版本节点执行。为进一步减小调度 makespan，文献[3]基于 PB 机制的容错调度算法允许任务副版本和其后续任务的主版本重叠。文献[4]基于 PB 机制采用副本调度策略来最小化任务响应时间和复制开销，但该算法没有考虑链路故障。文献[5]在网格计算环境下采用副版本重叠机制以较小复制成本来最小化任务调度 makespan，该机制仅在同时只有一个节点失效时才有效。文献[6]基于 PB 机制设计了云平台下实时任务容错调度算法，但其针对的是独立任务。PB 机制在发生故障时任务执行时间较长，极有可能不满足实时任务时间要求。

主动复制机制将多个任务副本在不同节点并行执行来实现容错。文献[7]基于主动复制机制提出双目标调度算法 BSH。该算法引入通信同步节点，其调度 makespan 被延长。该算法调度时需考虑节点集合所有子集，在系统节点较多时开销较大。大多主动复制容错调度机制盲目地将任务复制一定次数以容忍指定数量节点失效。如 FTBAR 算法[8]和 FTSA 算法[9]分别将任务调度至最小调度压力和最小完成时间的前 $\varepsilon+1$ 个节点来容忍 ε 个节点失效。这 2 个算法都未给出可靠性分析。文献[10]基于检查点卷回恢复和主动复制机制来容忍固定次数失效，并未给出可靠性分析。文献[11]的容错调度算法只能容忍一定数量节点失效。该算法通过限制与任务通信的先行任务副本数量来减少通信开销，其可靠性提升有限。文献[12]基于主动复制机制的容错调度算法选取可靠性最高节点执行任务副本，因此任务可能被调度至高可靠性但执行时间却较长的节点。文献[1]的基于插入机制容错调度算法中只有存在空闲时间时才能插入副本。文献[13]提出能同时优化可靠性和调度 makespan 的算法。但该算法使用严格调度方式，且该算法未考虑链路故障。文献[14]基于主动复制机制设计的任务容错调度算法考虑了节点和链路失效，但并未采用通用调度方式。文献[15]建立了异构分布式系统中的任务容错调度的可靠性模型，并设计相应调度算法。但该算法并没有考虑链路故障，且其采用的是严格调度方式。

随机搜索算法在任务调度算法中已被广泛采用。遗传算法(GA)是一种利用自然选择和进化思想在高维空间中寻优的方法，它采用有指导信息的非遍历随机搜索机制，可快速收敛到全局近优解。文献[16]基于 GA 进行异构移动网格任务副本调度来提高可靠性，但其针对的是独立任务。双目标遗传算法(BGA)[17]能同时优化调度 makespan 和可靠性，但在进化时 BGA 有可能违背任务间依赖。文献[18]采用 GA 在满足任务依赖关系的同时，采用两阶段策略来优化调度 makespan 和可靠性，但该算法没有采用多副本机制，因此其可靠性提升是有限的。

文献[19]证明异构计算系统中能优化可靠性的副本调度问题为 NP 完全问题。因此本文采用替代方法来进行任务调度。算法只要满足任务可靠性要求即可，并不需要最大化任务可靠性。Benoit 等[20]证明无论是瞬时故障还是永久故障，评估通用调度可靠性都是#P'完全问题，该问题至少是与 NP 完全问题同等难度。即使获得调度方案，还是不能在多项式时间内计算任务集可靠性。因此本文根据文献

[12]的任务集可靠性下限分析，在满足任务可靠性要求和任务间依赖约束关系前提下，进一步优化调度 makespan。

3　任务调度模型

3.1　异构计算系统模型

本文系统模型为无向图：$G_S=\langle P,L\rangle$。其中 $P=\{p_1,p_2,\cdots,p_M\}$ 是 M 个处理器节点集合，顶点 p_i 代表节点 i，$M=|P|$。L 是 $|L|$ 个无向边的集合。无向边 $l_{i,j}$ 代表节点 p_i 和 p_j 间的双向通信链路，假定系统中每 2 个节点间都存在连接链路。$w(p_i)$ 表示节点 p_i 单位时间内可执行的计算量，$w(l_{i,j})$ 表示链路 $l_{i,j}$ 单位时间内可以传输的数据量。λp_i 代表处理器节点 p_i 的失效概率，$\lambda l_{i,j}$ 代表链路 $l_{i,j}$ 的失效概率。

3.2　任务模型

应用程序 G 由有向无环图(DAG)表示，即 $G=\langle V,E\rangle$。任务集合 V 的任务数量 $N=|V|$。任务 v_i 直接先行任务集合为 $pred(v_i)$，任务 v_i 直接后续任务集合为 $succ(v_i)$。$w(v_i)$ 为任务 v_i 负载大小。E 为 V 中任务间有向通信权重边集合，$e_{i,j}$ 为任务 v_i 和 v_j 间传输消息，$w(e_{i,j})$ 为消息 $e_{i,j}$ 通信开销大小。本文有一个 entry 任务和一个 exit 任务的任务集，其他结构任务集可转换成该结构任务集。$ST(v_i,p_n)$ 为任务 v_i 在节点 p_n 开始执行时间，$ET(v_i,p_n)$ 为任务 v_i 在节点 p_n 执行时间，$ET(v_i,p_n)=w(v_i)/w(p_n)$，$FT(v_i,p_n)$ 为任务 v_i 在节点 p_n 完成执行时间，$FT(v_i,p_n)=ST(v_i,p_n)+ ET(v_i,p_n)$。在异构计算系统中相同任务在不同节点执行时间不同，相同消息在不同链路传输时间不同。节点 p_k 完成时间为

$$PFT(p_k)=\max_{v_i\in V,v_i\in ptsk(p_k)}\{FT(v_i,p_k)\} \quad (1)$$

其中，$ptsk(p_k)$ 为节点 p_k 所映射任务副本集合。

多副本任务容错调度算法必须满足任务依赖约束，即如果任务 v_i 和 v_j 存在依赖($e_{i,j}\in E$)，那么调度须确保任务 v_j 开始执行时，任务 v_i 执行完成且消息 $e_{i,j}$ 成功传输至任务 v_j 映射节点。对于消息 $e_{i,j}$ 和节点对 (p_k,p_n)，如果任务副本 v_i^k ($p_k\in proc(v_i)$，$proc(v_i)$ 为任务 v_i 所映射节点集合)完成执行时间 $FT(v_i,p_k)$ 和消息 $e_{i,j}$ 在节点 p_k 与 p_n 间通信链路通信时间之和不大于任务副本 v_j^n 开始执行时间 $ST(v_j,p_n)$，则副本对 (v_i^k,v_j^n) 有效，否则为无效。

基于多副本方式的任务调度有：严格调度和通用调度[20]。严格调度是指任务只有在其所有的直接先行任务的所有副本都执行完成且依赖消息到达当前调度任务映射节点时，才能开始执行。通用调度是指只要任务的每个直接先行任务有一个副本执行完成且该副本的消息成功传输至当前调度任务映射节点，当前调度任务就能开始执行。严格调度下当前调度副本开始执行时间为所有先行任务副本和其各自的通信时间之和中的最大值。而通用调度下当前调度副本开始执行时间可为部分先行任务副本和其各自的通信时间之和中的最大值。通用调度方式只要将所需部分任务副本间依赖消息发送即可。严格调度和通用调度方式可靠性计算方式不同，严格调度中任务副本的可靠性需考虑该任务所有先行任务的所有副本，而通用调度中任务副本的可靠性只需考虑所有先行任务副本中完成执行时间和消息通信时间之和小于当前调度任务副本开始执行时间的所有副本。

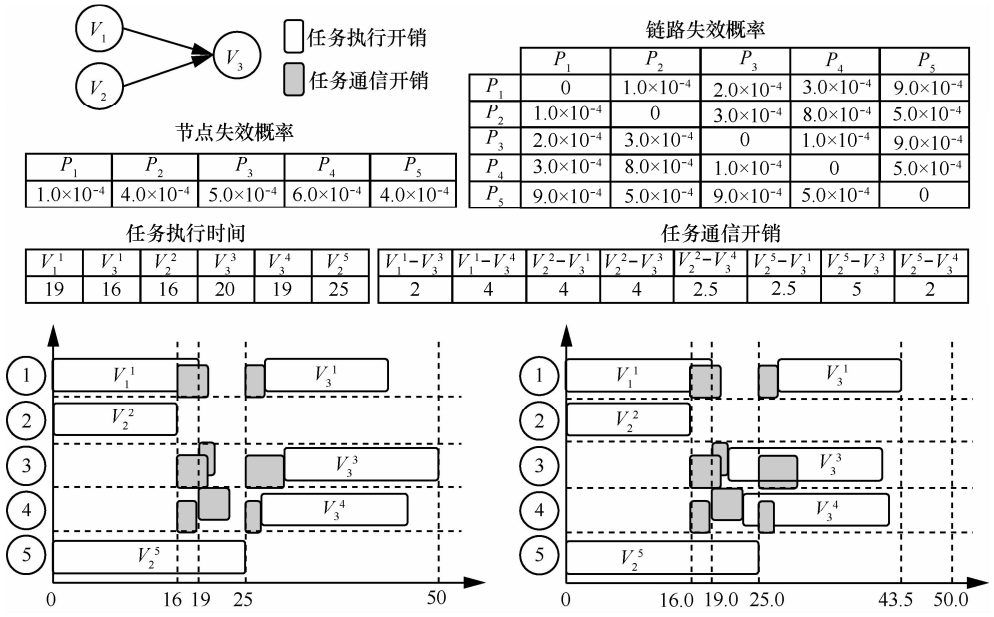
如图 1 所示，将任务集 $V=\{v_1,v_2,v_3\}$ 调度至节点集合 $P=\{p_1,p_2,p_3,p_4,p_5\}$ 时通用调度和严格调度的对比。图 1 左图严格调度方式中，v_3 完成时间为 50，可靠性为 0.999 999 547 0。图 1 右图通用调度方式中，v_3 完成时间为 43.5，可靠性为 0.999 999 415 8。图 1 中通用调度在满足任务可靠性前提下，将任务副本 v_3^1 和 v_3^2 开始执行时间往前推移，这较严格调度进一步减小任务副本完成执行时间。相对于严格调度，通用调度时任务 v_3 完成执行时间减少 13%，但可靠性减少却为 $1.312\,001\times10^{-5}$%，可靠性相差甚少。

4　基于多任务副本的通用调度可靠性模型

4.1　任务副本开始执行时间

任务调度算法需计算副本的最早开始执行时间(EST, earliest start time)和最晚开始执行时间(LST, latest start time)。消息到达时间 $ave(v_i^k,p_n)$ 为任务副本 v_i^k 的消息从节点 p_k 到达节点 p_n 的时间。该时间为任务副本 v_i^k 完成时间和链路 $l_{k,n}$ 的 ready 时间 $rdy(l_{k,n})$ 两者的最大值和通信开销 $LCT_{v_i^k,v_j^n}$ 之和。即

$$ave(v_i^k,p_n)=\max\{FT(v_i,p_k),rdy(l_{k,n})\}+LCT_{v_i^k,v_j^n} \quad (2)$$



P_1	P_2	P_3	P_4	P_5
1.0×10^{-4}	4.0×10^{-4}	5.0×10^{-4}	6.0×10^{-4}	4.0×10^{-4}

	P_1	P_2	P_3	P_4	P_5
P_1	0	1.0×10^{-4}	2.0×10^{-4}	3.0×10^{-4}	9.0×10^{-4}
P_2	1.0×10^{-4}	0	3.0×10^{-4}	8.0×10^{-4}	5.0×10^{-4}
P_3	2.0×10^{-4}	3.0×10^{-4}	0	1.0×10^{-4}	9.0×10^{-4}
P_4	3.0×10^{-4}	8.0×10^{-4}	1.0×10^{-4}	0	5.0×10^{-4}
P_5	9.0×10^{-4}	5.0×10^{-4}	9.0×10^{-4}	5.0×10^{-4}	0

V_1^1	V_3^1	V_2^2	V_3^3	V_3^4	V_2^5
19	16	16	20	19	25

$V_1^1-V_3^3$	$V_1^1-V_3^4$	$V_2^2-V_3^1$	$V_2^2-V_3^3$	$V_2^2-V_3^4$	$V_2^5-V_3^1$	$V_2^5-V_3^3$	$V_2^5-V_3^4$
2	4	4	4	2.5	2.5	5	2

图 1 严格调度和通用调度方式对比

其中，$LCT_{v_i^k,v_j^n}$ 为副本 v_i^k 映射节点 p_k 通过链路 $l_{k,n}$ 传输消息 $e_{i,j}$ 给副本 v_j^n 映射节点 p_n 的通信时间，$LCT_{v_i^k,v_j^n}=\dfrac{w(e_{i,j})}{w(l_{k,n})}$。如果映射节点相同，即 $p_k=p_n$，那么 $rdy(l_{k,n})$ 为 0，通信开销 $LCT_{v_i^k,v_j^n}$ 为 0，即 $ave(v_i^k,p_n)=FT(v_i,p_k)$。

任务 v_j 在节点 p_n 最早开始执行时间 $EST(v_j,p_n)$ 为满足执行条件最少消息到达时间和节点准备执行时间 $rdy(p_n)$ 的最大值，时间 $rdy(p_n)$ 为当前调度情况下节点 p_n 完成时间 $PFT(p_n)$。$EST(v_j,p_n)$ 可计算为

$$EST(v_j,p_n)=\max\left\{\max_{v_i\in pred(v_j)}\left\{\min_{v_i^k\in rep(v_i)}\left\{ave(v_i^k,p_n)\right\}\right\},rdy(p_n)\right\} \quad (3)$$

其中，$rep(v_i)$ 为任务 v_i 的副本集合。

任务 v_j 在节点 p_n 最晚开始执行时间 $LST(v_j,p_n)$ 为

$$LST(v_j,p_n)=\max\left\{\max_{v_i\in pred(v_j)}\left\{\max_{v_i^k\in rep(v_i)}\left\{ave(v_i^k,p_n)\right\}\right\},rdy(p_n)\right\} \quad (4)$$

4.2 可靠性模型

异构计算环境下不同节点和链路的失效概率不同，因此将任务分配至不同节点其可靠性不同。通用调度方式中任务副本的开始执行时间不同，有效先行任务副本数目不同，其可靠性与完成执行时间不同。假设系统中节点或链路发生的故障符合泊松分布且相互独立，该故障模型已被广泛采用。如果分配给节点 p_{dst} 的任务 v_j 要成功执行，那么节点 p_{dst} 必须是无故障的，且该任务输入数据必须成功传输给节点 p_{dst}。

调度 S 的可靠性 $rel(S)$ 为所有任务能够成功执行的概率。对于永久性故障，任务副本 v_i^j 能够成功执行，当且仅当：任务副本 v_i^j 映射节点 p_j 自开始执行第一个任务至副本 v_i^j 执行完成过程中没有失效。对于任务 v_i 的每个直接先行任务 v_l，至少有一个有效任务对 (v_l^k,v_i^j)，使任务 v_i^j 能够执行完成。且链路 $l_{k,j}$ 从开始传输第一个消息到消息 $e_{l,i}$ 传输完成过程中没有失效。因此需定义节点累积可靠性和链路累积可靠性。

定义 1 节点累积可靠性。节点从开始执行第一个调度任务到当前调度任务的完成时间，节点没有失效的概率。即该可靠性为任务与在任务 v_j 前已经调度至节点 p_n 的所有任务副本的可靠性乘积

$$PAR[E_{v_j^n}]=\prod_{v_i^n\in ptsk(p_n)\cap tv_i^n\leqslant tv_j^n}\left(R[E_{v_i^n}]\right) \quad (5)$$

其中，$R[E_{v_i^n}]$ 是 v_i 在节点 p_n 单独执行的可靠性，$\leqslant$ 表示时间顺序，即当前调度任务副本之前执行任务副本。

定义 2 链路累积可靠性。任务副本 v_j^n 的先行任务副本 v_i^k 通过链路 $l_{k,n}$ 传输消息到该任务副本时，直到任务 v_j^n 的输入完成传输时通信链路 $l_{k,n}$ 没有失效的概率。该概率为

$$LAR_{v_i^k,v_j^n}=\prod_{et_{p,q}\in ON(l_{k,n})\cap et_{p,q}\leqslant et_{i,j}}\left(P[E_{l_{k,n}}^{et_{p,q}}]\right) \quad (6)$$

其中，$P[E_{l_{k,n}}^{et_{p,q}}]$ 为消息 $e_{p,q}$ 在链路 $l_{k,n}$ 单独传输时的可靠性，$ON(l_{k,n})$ 代表在链路 $l_{k,n}$ 发生的所有通信，$et_{p,q}\leqslant et_{i,j}(v_p,v_q\in V)$ 为链路 $l_{k,n}$ 上消息 $e_{p,q}$ 的开始通信时间小于或等于消息 $e_{i,j}$ 的开始通信时间。如果任务副本 v_i^k 和 v_j^n 的映射节点相同，那么其链路通信时间为 0，该消息的可靠性为 1。因此从先行任务 v_i 到 v_j 的链路可靠性为 1。

对于 entry 任务 v_{entry}，$pred(v_{entry})=\varnothing$。因此，该任务的可靠性仅受限于其副本所在节点。对于其他任务副本 v_j^n，其可靠性受限于执行该任务节点和该任务的有效先行任务副本输入消息传输链路。基于 $PAR[E_{v_j^n}]$ 和 $LAR_{v_i^k,v_j^n}$ 的定义，通用调度时，任务 v_j 提交至节点 p_n 执行时可靠性计算式如下

$$AR_{v_j}^{p_n}=\begin{cases}PAR[E_{v_j^n}]=R[E_{v_j^n}],\ pred(j)=\varnothing\\ PAR[E_{v_j^n}]\prod\limits_{v_i\in pred(v_j)}\left(1-\prod\limits_{\substack{p_k\in proc(v_i),\\ ave(v_i^k,p_n)\leqslant ST(v_j,p_n)}}\left(1-LAR_{v_i^k,v_j^n}\right)\right),\text{其他}\end{cases} \quad (7)$$

当任务 v_j 在节点 p_n 单独执行时，可靠性为

$$R[E_{v_j^n}]=\exp\{-\lambda p_n ET(v_j,p_n)\} \quad (8)$$

对于任务副本 v_j^n 和其先行任务副本 v_i^k，消息 $e_{i,j}$ 在链路 $l_{k,n}$ 单独传输时可靠性 $P[E_{l_{k,n}}^{et_{i,j}}]$ 为

$$P[E_{l_{k,n}}^{et_{i,j}}]=\exp\left\{-\lambda l_{k,n}\frac{w(e_{i,j})}{w(l_{k,n})}\right\} \quad (9)$$

因此任务 v_j 的可靠性为

$$P[E_{v_j}]=1-\prod_{p_n\in proc(v_j)}\left(1-AR_{v_j}^{p_n}\right) \quad (10)$$

调度 S 的可靠性 $P[S]$ 为任务集 V 中的所有任务的可靠性乘积，可表示为

$$P(S)=\prod_{v_i\in V}P[E_{v_i}] \quad (11)$$

调度问题即为最大化可靠性且最小化调度 makespan。在 $\forall p_n\in P$，$EST(v_{entry},p_n)=0$ 时，调度 makespan 为 exit 任务的所有副本中最大完成执行时间。因此调度问题可形式化为

$$\max\ \ P[S] \quad (12)$$

$$\min\ \ Time(S)=\max_{p_n\in proc(v_{exit})}(FT(v_{exit},p_n)) \quad (13)$$

5　可靠性意识多副本任务通用调度算法

5.1　任务优先级

为确保任务执行顺序满足任务的依赖约束，任务优先级通常采用底端优先级(BL, the bottom level)方法和顶端优先级(TL, the top level)方法来决定。MaxRe[2]和 HEFT[21]算法使用 BL 方法来确定任务优先级，而 FTSA[9]、CAFT[11]和 CPOP[21]算法使用(TL+BL)方法来确定任务优先级。文献[14]中实验表明与基于(TL+BL)的任务排序方法相比，基于 BL 的排序方法调度时能够获得较小的执行时间。因此本文采用 BL 值计算任务优先级。任务 v_i 的底端优先级 $bl(v_i)$ 计算式为

$$\forall v_i\in V,\text{if } succ(v_i)=\varnothing \text{ then } bl(v_i)=\overline{ET(v_i,P)},$$
$$\text{else } bl(v_i)=\overline{ET(v_i,P)}+\max_{v_j\in succ(v_i)}(\overline{C(e_{i,j},L)}+bl(v_j)) \quad (14)$$

其中，$\overline{ET(v_i,P)}$ 为任务 v_i 在所有节点的平均计算成本，$\overline{C(e_{i,j},L)}$ 为消息 $e_{i,j}$ 在所有链路的平均传输开销。

5.2　任务集调度算法

任务调度算法针对输入的任务集、节点集合及可靠性要求计算任务集调度方案。基于文献[12]中的任务可靠性定义，设置任务 v_j 的可靠性要求 r_x (x 为该任务在优先级队列中的位置)为

$$r_x=\sqrt[n-x+1]{R\Big/\prod_{i=0}^{x-1}r_i'} \quad (15)$$

其中，$1\leqslant x\leqslant n$，且符合任务的优先级排序；r_i' 代表优先级队列中位置为 i 的任务实际所能达到的可靠性，$r_0'=1$；R 为任务集可靠性要求。如果该任务为优先级最高任务(entry 任务)，那么其可靠性要求 $r_1=\sqrt[n]{R}$。

可靠性意识多副本任务通用调度(RAMD_TGS, reliability- aware multi-duplication task general scheduling)算法如图 2 所示。RAMD_TGS 算法中调度队列 *Sch_Que* 的有效任务即为先行任务已经调度完成或者不存在先行任务的任务。最终 RAMD_TGS 算法将调用第 6 节中的 TMD_GS 算法来计算该任务的副本调度节点和开始执行时间。

```
Input: V={v1,v2,…,vN}, P={p1,p2,…,pM}, R
Output: the task scheduling result
1)calculate ET(vi,pj) of any task vi for any node pj
2)calculate the ET(vi,P) for any task vi and the C(ei,j,L) for any message ei,j
3)calculate the priority for any task according to formula (14)
4)add valid tasks to Sch_Que according to non-increasing order of task priorities
5)while the Sch_Que != NULL do
6)   schedule highest priority task vj and calculte its rx according to formula (15)
7)   if(rx≥1) then exit
8)   call the TMD_GS algorithm
9)   delete the scheduled task and add new valid task to the Sch_Que
10)end while
```

图 2　RAMD_TGS 算法

定理 1 假设每个任务的副本数量没有超过系统中处理器节点数量 M，异构分布式计算系统 RAMD_TGS 算法的可靠性为 $\Re$，可有 $\Re \geqslant R$。

证明 该证明与文献[12]中的证明相似，除了在计算每个任务副本的可靠性时，本文通用调度考虑的是有效先行任务副本，并不是文献[12]中严格调度算法的所有先行任务副本。

6 任务副本调度算法

6.1 任务多副本通用调度算法

任务多副本通用调度(TMD_GS, task multi-duplication general scheduling)算法为调度队列中最高优先级任务 v_j 选取映射节点和开始执行时间。该算法过程如图 3 所示。如果直到空闲节点集合为空，任务 v_j 可靠性仍不能满足要求，在后续任务的副本调度时可以通过可靠性计算公式来弥补该任务可靠性损失，以确保满足任务集合可靠性要求。

```
Input: v_j, P = {p_1, p_2, ..., p_M}, r_x
Output: the task scheduling result of task v_j
1) initialize the scheduled nodes of task v_j is NULL
2) initialize the spare node set is P
3) if task v_j is the entry task then
4)   while (r_x is not satisfied)&&(the spare node set != NULL) do
5)     schedule task v_j to node p_n which can complete task v_j first
6)   end while
7) else
8)   call the TMDGS_GA algorithm
```

图 3 TMD_GS 算法

6.2 基于遗传算法的任务多副本通用调度算法

如果任务 v_j 存在先行任务，那么在任务副本调度时，需要考虑任务间通信消息，此时在每个节点存在多个任务副本开始执行时间位置。在多副本调度算法中，首先需得到任务 v_j 的所有先行任务的每个副本在系统中每个节点的所有开始执行时间位置。

定义 3 对于任务 v_j 的先行任务副本 v_i^k，任务副本 v_j^n 在节点 p_n 对应的开始执行时间位置为 $V_{v_i\ p_n}^{p_k}$。

节点调度时的映射位置需确保当前调度任务能够接收到其先行任务所发送的消息，因此需定义任务的有效开始执行时间(ESTE, the effective start time of the execution)位置。该位置要确保当前调度任务能够接收到该任务的每个先行任务的所有副本中至少一个依赖消息。任务 v_j 在节点 p_n 的最小 ESTE 位置为 $EST(v_j, p_n)$，任务 v_j 在节点 p_n 的最大 ESTE 位置为 $LST(v_j, p_n)$。

定理 2 将 N 个任务构成的任务集调度至拥有 M 个处理器节点的系统中，最小化当前调度任务副本完成时间的映射位置选取问题搜索空间为 $O((M(N-1)+1)^M)$。

证明 ESTE 位置数目最大为该任务所有的先行任务的所有副本所对应的开始执行时间位置之和。每个先行任务最多有 M 个副本，当前调度任务最多有 $(N-1)$ 个先行任务，那么每个节点上当前调度任务的开始执行位置最多为 $M(N-1)$，还需考虑不映射任务副本的情况，所以共有 $M(N-1)+1$ 种可能的取值。因此，在拥有 M 个节点的系统中，最多有 $(M(N-1)+1)^M$ 种位置取值组合，因此该问题的搜索空间为 $O((M(N-1)+1)^M)$。

任务 v_j 在由 M 个节点组成系统中的调度问题可以形式化为如下的双目标 0-1 整数规划问题

$$\min\ Z_1 = \max_{\substack{1\leqslant n\leqslant M,\\ |s_0|+|s_1|+|s_2|\cdots+|s_{n-1}|+1\leqslant i\leqslant |s_0|+|s_1|+|s_2|\cdots+|s_n|}} \left\{x_i FT\left(v_j, p_n\right)\right\} \tag{16}$$

$$\max\ Z_2 = 1-\prod_{n=1}^{M}\left(1-AR_{v_j}^{p_n}\sum_{i=|s_0|+|s_1|+|s_2|\cdots+|s_{n-1}|+1}^{|s_0|+|s_1|+|s_2|\cdots+|s_n|} x_i\right) \tag{17}$$

$$\text{s.t.}\sum_{i=|s_0|+|s_1|+|s_2|\cdots+|s_{n-1}|+1}^{|s_0|+|s_1|+|s_2|\cdots+|s_n|} x_i \leqslant 1 \tag{18}$$

$$x_i \in \{0,1\},\ i=1,2,\cdots,|s_1|+|s_2|+\cdots+|s_M| \tag{19}$$

式(18)确保在每个节点的所有 ESTE 位置中只有一个位置值为 1，即在每个节点不能重复映射同一任务的副本。$|s_i|$ 为节点 p_i 中 ESTE 位置数目，$|s_0|=0$。由于该问题的指数级搜索空间，系统中节点数量达到一定规模时，寻找其最优解是非常耗时的。因此为了提高搜索效率，本文采用 GA 求解该问题的近优解。

6.2.1 *编码*

遗传算法的个体基因中只能包含每个节点在最小 ESTE 位置和最大 ESTE 位置间的位置。为了防止将任务多次映射至相同节点，每个节点对应基因中最多只能有一个位置值为 1，其他位置值为 0，即 Condition 1。编码方案还要包含每个节点在个体编码中的开始位置，该位置由数组 s 表示。图 4 为将任务集 $V=\{v_1,v_2,v_3\}$ 中任务 v_3 调度至节点集合 $P=\{p_1,p_2,p_3,p_4,p_5,p_6\}$ 时的 ESTE 位置。图 5 为图 4 中调度的一种编码方案。如图 5 所示，个体 g_i 中任务 v_3 映射至节点 p_1、p_2、p_3、p_4 的位置分别为 $V_{2\,1}^4$、$V_{2\,2}^5$、$V_{1\,3}^2$ 和 $V_{1\,4}^2$。

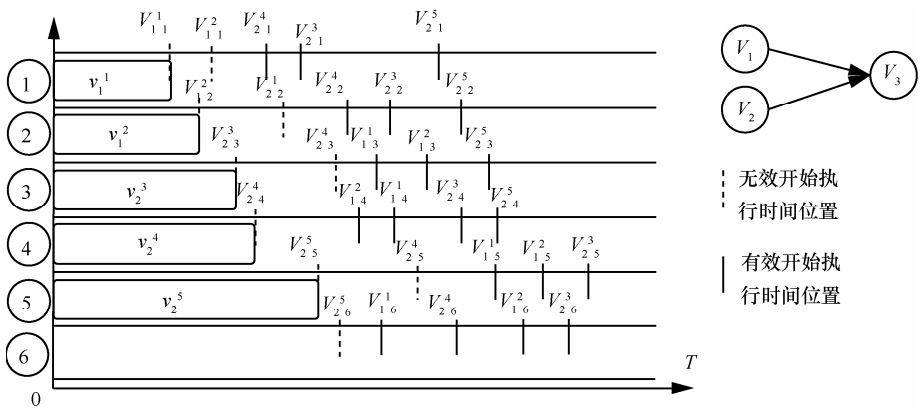


图 4　任务有效开始执行时间位置

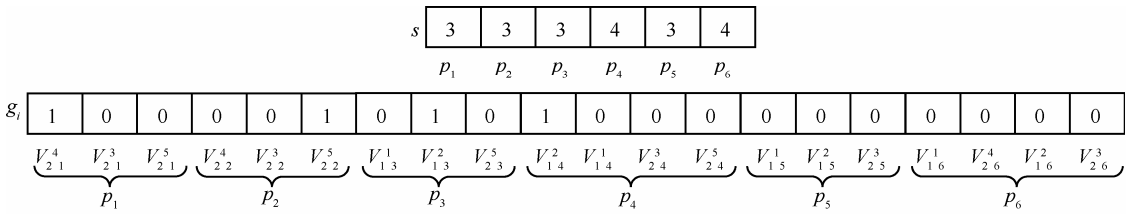


图 5　编码方案

如果任务 v_j 分配给节点 p_n 中第 k 个 ESTE 位置，那么个体 g_j 中的第 l 个基因 $g_{j,l}=1$，$l=k+\sum_{i=0}^{n-1}|s_i|$，$|s_i|$ 为数组 s 中 s_i 所代表节点 p_i 的 ESTE 位置个数。编码个体的长度为 $\sum_{i=1}^{M}|s_i|$，数组元素 s_i 在个体 g_j 对应的基因集合为 $\left\{g_{j,l}\left|\sum_{k=0}^{i-1}|s_k|<l<1+\sum_{q=0}^{i}|s_q|\right.\right\}$。

条件 1　映射至相同节点的相同任务的副本最多只能有一个，即染色体中映射节点相同的编码基因和小于等于 1，如下式所示。

$$\sum_{p_i\in P,\sum_{k=0}^{i-1}|s_k|<l<1+\sum_{q=0}^{i}|s_q|} g_{j,l}\leqslant 1,\ g_{j,l}\in\{0,1\} \quad (20)$$

6.2.2　交叉

交叉操作需要更改 2 个父代个体中映射节点相同的编码基因值，而且 2 个个体中该映射节点对应的编码基因值应该不同，因为更改 2 个值为 0 或者是非零且相同基因值没有意义。因此交叉操作需要满足 2 个个体对应映射节点的编码基因组成数值的异或值不为 0，即 Condition 2。交叉操作可同时交换一个节点或几个节点的任务副本映射方案。如图 6 所示为交叉操作交换个体 g_i 和 g_j 中节点 p_2 映射任务副本。

定义 4　$\bigcup_{g_j,(s_i)}$ 操作为个体 g_j 中数组元素 s_i 对应的基因按位连接操作组成的相应二进制数。

条件 2　种群中节点 p_i 对应的 2 个个体的基因能够进行交叉操作，当且仅当基因按位连接操作后获得数值的异或操作值不为 0。即

$$\bigcup_{g_i,(s_i)}\oplus\bigcup_{g_j,(s_i)}!=0 \quad (21)$$

6.2.3　变异

变异操作可将个体中的任务副本映射方案进行更改，以保持种群多样性，同时能够跳出局部最优，避免算法陷入早熟困境。变异操作即将个体中基因值进行更改，将基因值 1 改为 0 以取消任务映射，将基因值 0 改为 1 以增加任务映射。变异操作包括更改任务副本在映射节点的开始执行时间位置和任务副本映射节点 2 种。为提高搜索效率，算法采用有指导的变异方式。变异的 4 种方式如下。

方式 1　将个体中节点 p_i 映射任务的开始执行时间往后推迟来增大任务可靠性。该方式需满足条件

$$\left(\bigcup_{g_{Mj},(s_i)}>\bigcup_{g_j,(s_i)}\right)\wedge$$
$$\left(\sum_{p_i\in P,\sum_{q=0}^{i-1}|s_q|<k<1+\sum_{l=0}^{i}|s_l|} g_{Mj,k}\leqslant 1, g_{Mj,k}\in\{0,1\}\right) \quad (22)$$

其中，g_{Mj} 为与 g_j 对应的变异后个体。

方式 2　将个体中节点 p_i 映射任务的开始执行时间往前推移，这可减小任务可靠性，但只要满足相应的可靠性条件即可。该方式需满足条件

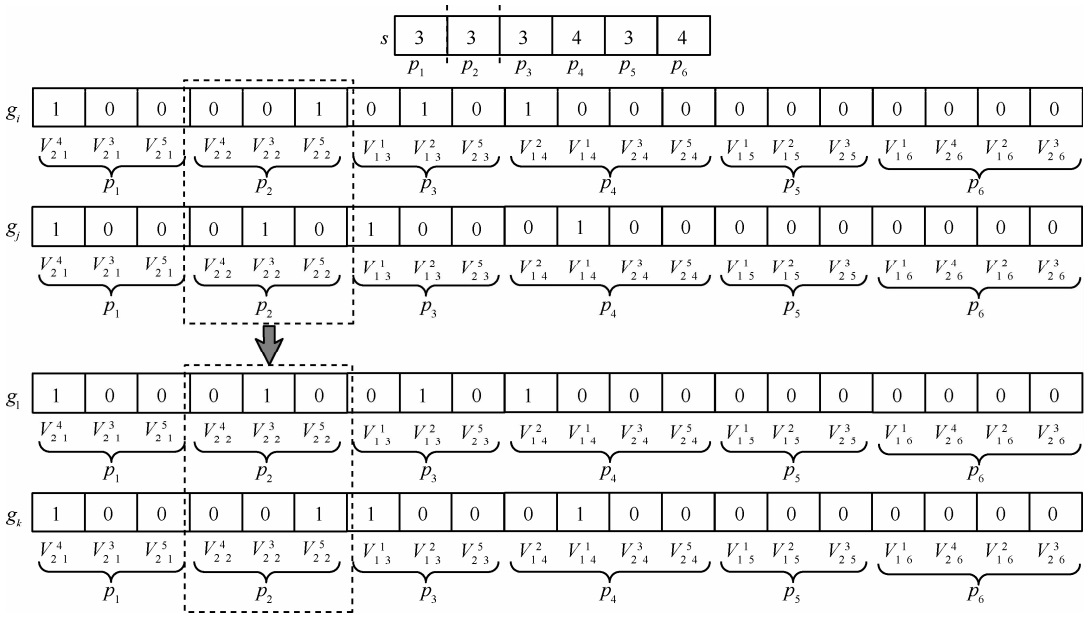


图 6　交叉操作

$$\left(\bigcup_{g_{Mj},(s_i)}<\bigcup_{g_j,(s_i)}\right)\wedge$$
$$\left(\sum_{p_i\in P,\sum_{q=0}^{i-1}|s_q|<k<1+\sum_{l=0}^{i}|s_l|} g_{Mj,k}\leqslant 1, g_{Mj,k}\in\{0,1\}\right) \quad (23)$$

方式 3　在个体中没有映射任务副本的节点 p_i 映射任务副本来提高可靠性。该方式需满足条件

$$\left(\bigcup_{g_{Mj},(s_i)}>\bigcup_{g_j,(s_i)}\right)\wedge\left(\bigcup_{g_{Mj},(s_i)}\neq 0\right)\wedge\left(\bigcup_{g_j,(s_i)}=0\right)\wedge$$
$$\left(\sum_{p_i\in P,\sum_{q=0}^{i-1}|s_q|<k<1+\sum_{l=0}^{i}|s_l|} g_{Mj,k}\leq 1,\ g_{Mj,k}\in\{0,1\}\right) \quad (24)$$

方式 4　将个体中已映射任务副本节点 p_i 的副本取消，这样会减小可靠性，但只要确保满足任务可靠性要求即可。该方式需满足条件

$$\left(\bigcup_{g_{Mj},(s_i)}<\bigcup_{g_j,(s_i)}\right)\wedge\left(\bigcup_{g_{Mj},(s_i)}=0\right)\wedge\left(\bigcup_{g_j,(s_i)}\neq 0\right)\wedge$$
$$\left(\sum_{p_i\in P,\sum_{q=0}^{i-1}|s_q|<k<1+\sum_{l=0}^{i}|s_l|} g_{Mj,k}\leqslant 1,\ g_{Mj,k}\in\{0,1\}\right) \quad (25)$$

如图 7 所示为个体 g_l 中更改任务副本在节点 p_4 的开始执行时间位置。如图 8 所示为个体 g_l 在节点 p_5 的 ESTE 位置 $V_{1\,5}^{1}$ 增加新的任务副本。

6.2.4　选择

本算法采用完成时间评估函数 F_{Tim} 和可靠性评估函数 F_{Rel} 2 个异构适应值函数来评估解的质量，因此本算法采用 RR(round-robin)机制来选择个体。根据 F_{Tim} 和 F_{Rel} 函数值由大到小排列得到 2 个排序个体队列。然后以 RR 机制从 2 个排序队列中选择个体，直到满足种群数量要求。在最终种群能够满足可靠性要求的个体中，执行时间最小个体即为最终调度方案。

函数 F_{Tim} 评估每个个体的完成时间，可定义为

$$F_{Tim}(g_i)=1-\max_{1\leqslant k\leqslant M}\left\{FT(v_j,p_k)\left|\sum_{\sum_{q=0}^{k-1}|s_q|<p<1+\sum_{l=0}^{k}|s_l|} g_{i,p}=1\right.\right\} \quad (26)$$

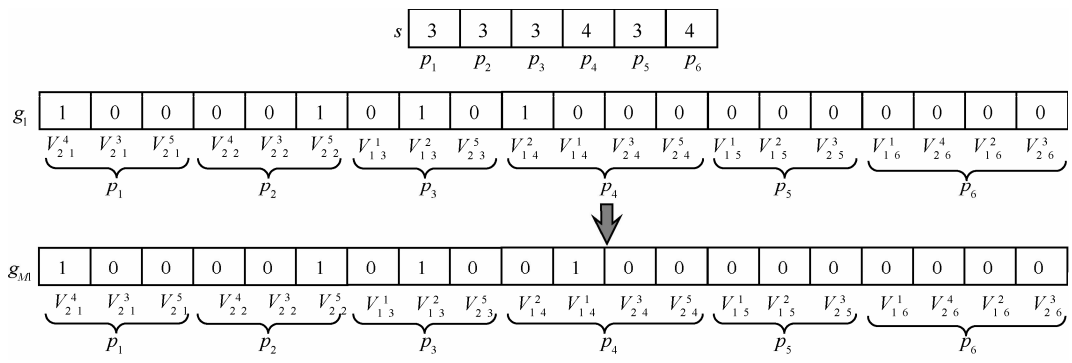


图 7　变异操作方式 1

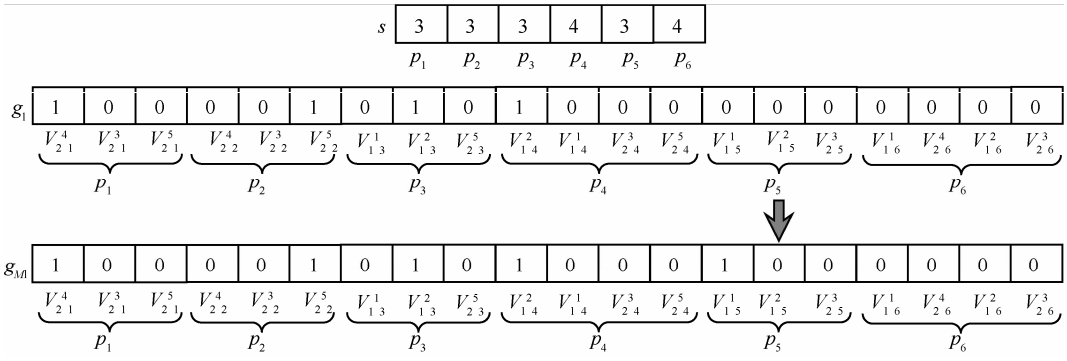


图 8　变异操作方式 3

函数 F_{Rel} 评估每个个体的可靠性，可定义为

$$F_{Rel}(g_i)=P[E_{v_j}]=1-\prod_{\substack{1\leqslant k\leqslant M,\\ \sum_{q=0}^{k-1}|v_q|<p<1+\sum_{t=0}^{k}|v_t|}^{g_{i,p}=1}\left(1-AR_{v_j}^{p_k}\right) \quad (27)$$

6.2.5　算法步骤

基于遗传算法的任务多副本通用调度算法(TMDGS_GA, task multi-duplication general scheduling based on genetic algorithm)步骤如图 9 所示。

```
Input: v_j, P={p_1, p_2, ..., p_M}, r_x
Output: the scheduling result of task v_j
1)initialize the crossover probabilty pc, the mutation probabilty pm
2)initialize population quantity PN, evolution number EN, crossover number CN
3)for (ParNum=0; ParNum<PN; ParNum++) do
4)   while (the r_x is not satisfied)&&(the spare node set != NULL) do
5)     schedule task v_j to the randomly selected node p_n
6)     randomly select the ESTE position of task v_j
7)   end while
8)   generate a particle according to the scheduling scheme
9)end for
10)for (EvNum=0; EvNum<EN; EvNum++) do
11)   for any two particles of the CN randomly selected particles do
12)     if(the random number < pc) then
13)       randomly select one or more elements which satisfy condition 2 from array s
14)       change the selected elements of the two particles
15)   end for
16)   for any particle g_j of the population do
17)     if(the random number < pm) then
18)       randomly select one element s_i from the array s
19)       if(the reliability of the particle ⩽ r_x) then
20)         if(∪_{g_j,(s_i)} != 0) then mutate by the way of No.1
21)         if(∪_{g_j,(s_i)} == 0) then mutate by the way of No.3
22)       if(the reliability of the particle > r_x) then
23)         if(exist a ESTE position whose start time is smaller) then
24)           mutate by the way of No.2
25)         if(the start time of current position is smallest) then
26)           mutate by the way of No.4
27)   end for
28)   calculate the fitness of every particle according to formula (26) and (27)
29)   sort the particles to two queues according to the completion time and reliability
30)   select the new particles from the two queues based on the RR mechanism
31)end for
```

图 9　TMDGS_GA 算法

7　仿真实验与结果分析

7.1　仿真环境

7.1.1　任务集合

本文实现了一个任务集 DAG 随机生成器，任务集参数与文献[1,12,21]中参数一致。任务集的任务数量 N 从 20 到 100，增长步长为 10。与文献[21]的随机初始化每个任务在每个节点的执行时间不同，任务执行时间由任务负载开销和处理器执行速度决定。$\bar{w}$ 为 [1 000, 2 000]上均匀分布随机值，任务负载开销符合$[0, 2\bar{w}]$上的均匀分布。依赖任务间通信开销使用(CCR, communication to computation ratio)来初始化，该值是任务间通信开销平均值与任务负载开销平均值比率。CCR 取值为 $\{0.1,0.4,0.6,0.8,1,2,4,5\}$。依赖任务通信开销符合均值由 CCR 和任务负载开销平均值决定的均匀分布。任务出度取值集合为 $\{1,2,3,4,5\}$。任务集 DAG 高度取值符合均值为 $\frac{\sqrt{N}}{\alpha}$ 的均匀分布，每个层级宽度取值符合均值为 $\sqrt{N}\alpha$ 的均匀分布，$\alpha=\{0.3,0.4,0.5,0.6,0.7,0.8\}$。

7.1.2　系统环境

实验中节点数量 M 为[20, 1 000]的随机值，节点处理速度为[5,30]上均匀分布随机值。因此每个任务的执行时间可通过任务负载除以节点处理速度获得。网络结构为全连接网络，网络中任意 2 个节点间均存在通信链路，但链路通信速率不同。为体现系统中通信链路异构性，每个链路单位时间内传输消息大小为[0.5,10]上的随机值。节点和链路失效概率符合泊松分布，泊松分布参数 λ 为 $\{1,2,3,4,5,6,7,8,9\}\times10^{-4}$ 中随机值。实验中 FTSA_BL 算法[9]能够容忍的故障次数 ε 取值为 $\{1,2,3,4\}$。

7.1.3　仿真过程

本文实验环境为：Intel(R) i7-4710MQ CPU，8 GB 内存，500 GB 硬盘，采用 C 语言编写相应的仿真程序。实验将 RAMD_TGS 算法与基于 BL 任务排序方法的 FTSA_BL 算法[9]和 RR 算法[12]进行对比。因为：1) FTSA_BL 算法基于 HEFT 算法，HEFT 算法是常见的异构计算环境任务调度算法，FTSA_BL 算法使用主动复制机制来提高可靠性；

2) RR 算法可靠性计算方法与 RAMD_TGS 算法相似，只是 RR 算法是基于严格调度方式，将 RAMD_TGS 算法的通用任务调度方式与 RR 算法的严格调度方式进行比较。

实验在不同条件下进行：任务数量、CCR 及需容忍故障次数。实验将 3 个算法的可靠性、调度 makespan 及调度资源开销进行了对比。实验可靠性要求以 FTSA_BL 算法在一定故障次数下所能够达到的可靠性为下限。实验默认配置为：任务数量 N 为 40，节点数量 M 为 50，CCR 为 0.4，FTSA_BL 算法需容忍故障次数 ε 为 2。后续实验中除非特别声明，否则实验参数为默认配置参数。为了减小遗传算法随机性对实验结果的影响，本实验取相同配置条件下的 10 次运行结果的平均值。遗传算法的交叉概率取 0.5，变异概率取 0.25，种群规模为 20，进化次数为 10。

7.2 可靠性对比

FTSA_BL、RR 和 RAMD_TGS 算法在相同系统配置条件下，随任务数量变化三者可靠性对比关系如表 1 所示。由图可知，虽然 RAMD_TGS 算法可靠性低于 RR 算法，但仍大于 FTSA_BL 算法。表明 RAMD_TGS 算法能够满足任务可靠性要求。任务调度时动态的任务副本数量同样能够获取较高可靠性。

表 1 同任务数量任务集 3 种算法的可靠性对比

任务数量	FTSA_BL	RAMD_TGS	RR
20	90.836%	94.286%	96.356%
30	90.365%	93.457%	95.684%
40	90.015%	92.846%	95.185%
50	89.564%	92.259%	95.067%
60	89.356%	91.948%	94.646%
70	89.249%	91.649%	93.452%
80	88.934%	91.146%	92.126%
90	88.948%	90.264%	91.568%
100	86.684%	90.576%	91.249%

7.3 调度 makespan 对比

FTSA_BL、RR 和 RAMD_TGS 算法在相同系统配置条件下，随任务数量变化三者调度 makespan 对比关系如图 10 所示。在相同系统配置条件下不同 CCR 值时，3 种算法调度 makespan 对比关系如图 11 所示。

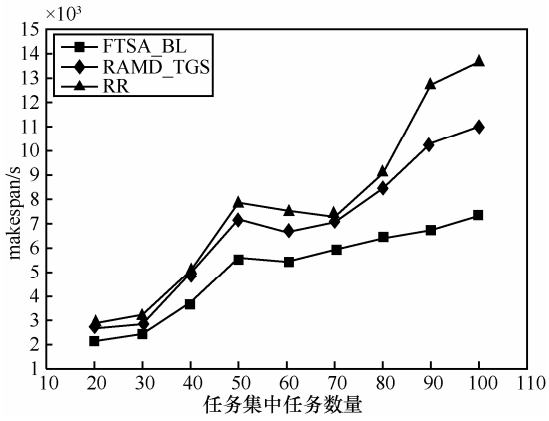


图 10 不同任务数量任务集 3 种算法的调度 makespan 对比

由图 10 知，相同配置条件下 RAMD_TGS 算法调度 makespan 优于 RR 算法，但是次于 FTSA_BL 算法。因为 FTSA_BL 算法在调度时每次都选择最早完成执行时间节点。而 RR 算法在调度时仅考虑调度节点可靠性，所以其调度 makespan 最大。RAMD_TGS 算法综合考虑可靠性和 makespan 进行全局搜索，所以能够获得较优 makespan。大多数情况下 RAMD_ TGS 算法调度 makespan 明显优于 RR 算法，有时两者差不多。这是因为如果 RR 算法中可靠性高节点的任务执行时间也较短，那么 2 种算法的 makespan 相差不大。而如果可靠性高节点任务执行时间较长，那么 RAMD_TGS 算法的 makespan 将优于 RR 算法。

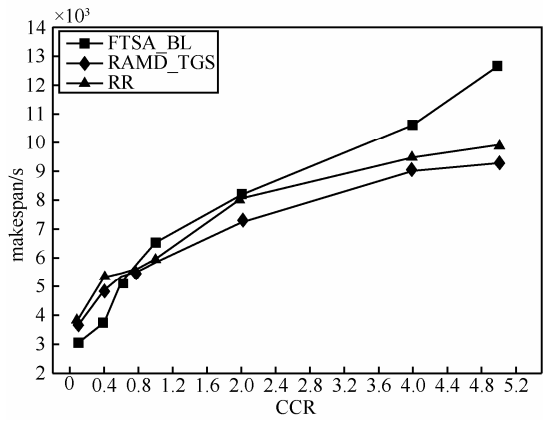


图 11 不同 CCR 任务集 3 种算法的调度 makespan 对比

由图 11 可知，随 CCR 增长，RAMD_TGS 算法调度 makespan 增长速率比 FTSA_BL 算法小。因为 RAMD_TGS 算法不需等待所有消息通信完成，只需等待部分先行任务副本消息。当 CCR 值增大到某一值时，FTSA_BL 算法调度 makespan 将大于 RAMD_ TGS 算法和 RR 算法的 makespan。因为随

CCR 的增长，任务集的任务间通信开销增长，尤其在异构计算环境链路传输速度差异性较大时，导致当前调度任务在映射节点开始执行时间差异较大。采用严格调度方式的 FTSA_BL 算法需要等待所有先行任务副本的依赖消息传输完成，且该算法只关注任务副本的完成执行时间，因此其任务副本开始执行时间增长幅度较大。RR 算法选取可靠性较高节点时倾向于选取与先行任务副本映射节点相同的节点，因此能够在一定程度上减少依赖消息通信开销造成的影响，但是 RAMD_TGS 算法的 makespan 仍优于 RR 算法。

7.4　调度资源开销对比

在基于任务复制的可靠性调度方法中，资源的占用开销是一个不得不面临和考虑的问题。在调度任务 v_j 时，资源占用开销包括计算资源占用开销和通信资源占用开销。计算资源占用开销 $RU_{comp}(v_j)$ 为任务 v_j 的所有任务副本的执行时间之和，可定义为

$$RU_{comp}(v_j)=\sum_{p_n\in proc(v_j)} ET(v_j,p_n) \tag{28}$$

通信资源占用开销 $RU_{comm}(v_j)$ 为该任务的所有先行任务的副本与该任务的所有副本的通信时间之和。通信开销仅包含有效通信过程所占用的时间开销。$RU_{comm}(v_j)$ 可定义为

$$RU_{comm}(v_j)=\sum_{v_i\in pred(v_j)}\sum_{\substack{p_k\in proc(v_i),p_n\in proc(v_j),\\ ave(v_i^k,p_n)\leqslant ST(v_j,p_n)}} LCT_{v_i^k,v_j^n} \tag{29}$$

任务集中 entry 任务的通信资源占用开销为 0。

3 种算法的计算资源占用开销随任务集中任务数量变化的对比情况如图 12 所示。

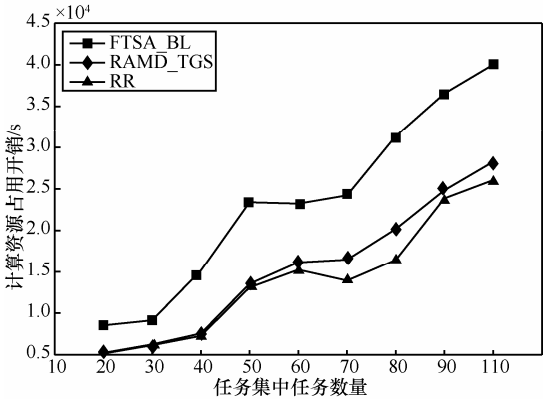


图 12　不同任务数量任务集 3 种算法的计算资源占用对比

由图 12 可知，当 DAG 任务规模增大时，计算资源占用开销也增大。这是由于系统中含有的任务数量规模增加，会导致系统中任务副本数量增大，需要更多的计算资源。从图中可以看出，无论 DAG 规模多大，RAMD_TGS 算法的计算资源占用开销较 RR 算法大，但是有时两者相差较小，相差不到 2%；但两者计算资源占用开销较 FTSA_BL 算法有很大改善。

3 种算法的通信资源占用开销随任务集中任务数量变化对比情况如图 13 所示。当 DAG 任务规模增大时，通信资源占用开销也增大。这是由于系统中任务数量规模和任务副本数量增加，需更多的通信资源来进行通信。FTSA_BL 算法通信资源占用开销随任务数量增长变化较大，而 RAMD_TGS 算法和 RR 算法通信资源占用开销变化并不是太大。这是因为 RAMD_TGS 和 RR 算法倾向于将依赖任务调度至相同节点以提高任务可靠性，因此能减少通信开销。但无论 DAG 规模多大，RAMD_TGS 算法通信资源占用开销较 RR 算法大。但有时两者相差较小，甚至 RAMD_TGS 算法要优于 RR 算法。这是由于 RR 算法需要等待所有依赖任务副本消息传输完成，在异构计算环境下由于通信链路异构性，可能存在通信速率较低链路，导致通信消息在该链路传输开销较大，因此 RR 算法可能会存在较大通信资源占用开销；而通用调度却不需等待所有任务副本消息到达，因此可减少一定的通信资源占用开销。RAMD_TGS 算法任务副本数量可能较 RR 算法多，但通用调度方式可使该算法不需等待通信开销较大通信过程完成，所以其通信开销与 RR 算法相差不大。FTSA_BL 算法任务副本数量较多，且为严格调度，所以其通信开销最大。

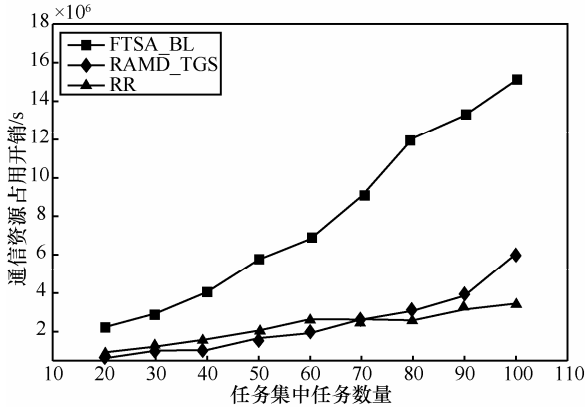


图 13　不同任务数量任务集 3 种算法的通信资源占用对比

3 种算法计算资源占用开销随 CCR 变化的对比情况如图 14 所示。随着 CCR 的增长，RAMD_TGS

和 RR 算法的计算资源占用开销降低，虽然 RAMD_TGS 算法的计算资源占用开销较 RR 算法大，但是相差不大，而 FTSA_BL 算法的计算资源占用开销却变化不大。因为 RAMD_TGS 和 RR 算法为了提高可靠性倾向于将依赖任务调度至相同节点来避免大量通信开销，尤其当 CCR 较大时。链路可靠性提高可减少任务副本数量。RAMD_TGS 算法能够进一步减小长时间的依赖任务间通信开销，因此其链路可靠性较 RR 算法高，但是 RAMD_TGS 算法的通用调度方式导致其可能较 RR 算法有较多的任务副本数量，所以其计算资源占用开销略微大于 RR 算法。

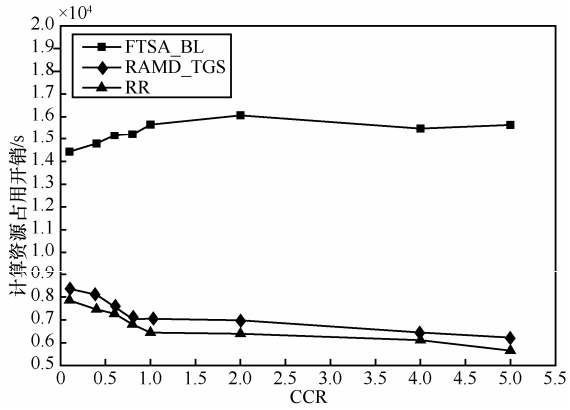


图 14 不同 CCR 任务集 3 种算法的计算资源占用对比

3 种算法的通信资源占用开销随 CCR 变化的对比情况如图 15 所示。与图 14 中原因相似，RAMD_TGS 算法和 RR 算法倾向于将依赖任务调度至相同节点来避免大量通信开销，尤其是当 CCR 值较大时。

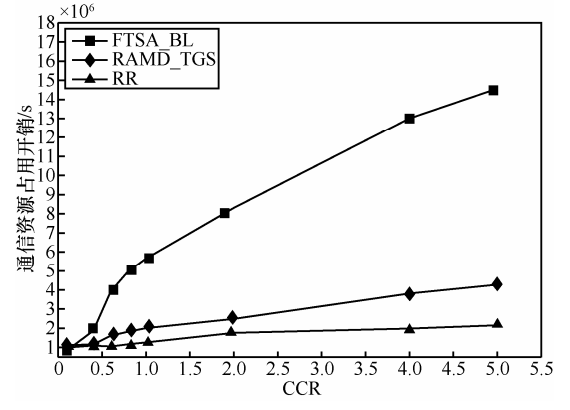


图 15 不同 CCR 任务集 3 种算法的通信资源占用对比

3 种算法计算资源占用开销随故障次数 ε 变化情况如表 2 所示。与 FTSA_BL 算法相比 RAMD_TGS 和 RR 算法能显著降低计算资源占用开销。RAMD_TGS 算法计算资源占用开销与 RR 算法相比，增长并不是太大。随着 ε 增长，FTSA_BL 算法计算资源占用开销几乎是线性增长，而 RAMD_ TGS 算法和 RR 算法计算资源占用开销增长趋势相对较小。这表明动态数量的任务副本能够达到相同可靠性，并不需盲目地进行任务复制。

表 2 不同故障次数时 3 种算法的计算资源占用对比

故障次数	FTSA_BL	RAMD_TGS	RR
1	0.599×10^4	0.589×10^4	0.588×10^4
2	1.499×10^4	0.797×10^4	0.748×10^4
3	2.128×10^4	1.249×10^4	0.985×10^4
4	2.569×10^4	1.465×10^4	1.168×10^4

3 种算法通信资源占用开销随故障次数 ε 的变化情况如表 3 所示。随着 ε 的增长，RAMD_TGS 和 RR 算法的通信开销增长趋势并不是太大，相对于 FTSA_BL 算法能够节省更多通信资源。RAMD_TGS 算法和 RR 算法相对于 FTSA_BL 算法能够节约超过 100%的通信资源占用开销。而 RAMD_TGS 算法相对于 RR 算法通信资源占用开销增加最大不超过 50%。

表 3 不同故障次数时 3 种算法的通信资源占用对比

故障次数	FTSA_BL	RAMD_TGS	RR
1	1.984×10^5	1.623×10^5	1.094×10^5
2	4.127×10^5	1.649×10^5	1.154×10^5
3	6.858×10^5	3.639×10^5	2.662×10^5
4	10.256×10^5	4.562×10^5	3.469×10^5

8 结束语

针对异构分布式计算环境下的可靠性调度问题，本文首先提出能够同时考虑系统处理器节点和链路失效的通用调度方式任务可靠性计算方法及该通用调度问题的 0-1 整数规划模型；然后基于遗传算法，提出采用通用调度方式的 RAMD_TGS 算法。该算法采用遗传算法的进化操作能够决定动态的任务副本数量，避免了任务副本的盲目复制，同时该算法能够确保任务间的依赖关系。该算法在满足可靠性要求的基础上，能够进一步优化任务的调度 makespan。仿真实验表明 RAMD_TGS 算法的计算资源占用开销和通信资源占用开销与 FTSA_BL

算法的盲目任务复制机制有大幅降低，同时 RAMD_TGS 算法的任务调度 makespan 与 RR 算法相比有进一步的减小，且能够满足可靠性要求。针对异构分布式计算系统存在大量节点和任务的问题，在后续研究中将通过 GPU 加速技术来提高算法的执行效率，以有效进行大规模异构分布式计算系统任务多副本容错调度算法的研究。

参考文献：

[1] XIAO Y T, KENLI L, RENFA L, *et al.* Reliability-aware scheduling strategy for heterogeneous distributed computing systems[J]. Journal of Parallel and Distributed Computing, 2010, 70(9): 941-952.

[2] ZHAO L, REN Y, XIANG Y, *et al*. Fault tolerant scheduling with dynamic number of replicas in heterogeneous system[A]. IEEE International Conference on High Performance Computing and Communications[C]. Melbourne, 2010.434-441.

[3] QIN X, JIANG H. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems[J]. Parallel Computing, 2006, 32(5): 331-356.

[4] QIN Z, BHARADWAJ V. On the design of communication-aware fault-tolerant scheduling algorithms for precedence constrained tasks in grid computing systems with dedicated communication devices[J]. Journal of Parallel and Distributed Computing, 2009, 69(3): 282-294.

[5] QIN Z, BHARADWAJ V, CHEN K. On the design of fault-tolerant scheduling strategies using primary-backup approach for computational grids with low replication costs[J]. IEEE Transactions on Computers, 2009, 58(3):380-393.

[6] 王吉, 包卫东, 朱晓敏. 虚拟化云平台中实时任务容错调度算法研究[J]. 通信学报, 2014, 35(10): 171-180.
WANG J, BAOW D, ZHU X M. Fault-tolerant scheduling algorithm forreal-time tasks in virtualized cloud[J]. Journal on Communications, 2014, 35(10): 171-180.

[7] GIRAULT A, KALLA H. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate[J]. IEEE Transactions on Dependable and Secure Computing, 2009, 6(4): 241-254.

[8] GIRAULT A, KALLA H, SIGHIREANU M, *et al*. An algorithm for automatically obtaining distributed and fault-tolerant static schedules[A]. International Conference on Dependable Systems and Networks[C]. 2003.159-168.

[9] ANNE B, MOURAD H, YVES R. Fault tolerant scheduling of precedence task graphs on heterogeneous platforms[A]. Proceedings of the 22nd International Conference on Parallel and Distributed Processing[C]. 2008.1-8.

[10] PAUL P, VIACHESLAV I, PETRU E, *et al*. Design optimization of time and cost-constrained fault-tolerant embedded systems with check pointing and replication[J]. IEEE Transactions on Very Large Scale Integration Systems, 2009, 17(3): 389-402.

[11] ANNE B, MOURAD H, YVES R. Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems[J]. Parallel Computing, 2009, 35(2): 83-108.

[12] ZHAO L, REN Y, SAKURAI K. Reliable workflow scheduling with less resource redundancy[J]. Parallel Computing, 2013, 39(10): 567-585.

[13] ALAIN G, ÉRIK S, DENIS T. Reliability versus performance for critical applications[J]. Journal of Parallel and Distributed Computing, 2009, 69(3): 326-336.

[14] ZHAO L, REN Y Z, SAKURAI K. A resource minimizing scheduling algorithm with ensuring the deadline and reliability in heterogeneous systems[A]. International Conference on Advanced Information Networking and Applications[C]. 2011.275-282.

[15] 谢国琪, 李仁发, 刘琳等. 异构分布式系统 DAG 可靠性模型与容错算法[J]. 计算机学报, 2013, 36(10): 2019-2032.
XIEG Q, LIR F, LIUL, *et al*.DAG reliability model and fault-tolerant algorithm for heterogeneous distributed systems[J]. ChineseJournal of Computers, 2013, 36(10): 2019-2032.

[16] CHIN S H, SUH T, YU H C. Genetic algorithm based scheduling method for efficiency and reliability in mobile grid[A]. International Conference on Ubiquitous Information Technologies & Applications[C]. 2009.1-6.

[17] ATAKAN D, FÜSUN Ö. Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems[J]. The Computer Journal, 2005, 48(3): 300-314.

[18] WANG X F, YEO C S, BUYYA R, *et al*. Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm[J]. Future Generation Computer Systems, 2011, 27(8):1124-1134.

[19] BENOIT A, DUFOSSÉ F, GIRAULT A, *et al.* Reliability and performance optimization of pipelined real-time systems[J]. Journal of Parallel and Distributed Computing, 2013, 73(6): 851-865.

[20] ANNE BENOIT, CANON L C, JEANNOT E, *et al*. Reliability of task graph schedules with transient and fail-stop failures: complexity and algorithms[J]. Journal of Scheduling, 2012, 15(5):615-627.

[21] TOPCUOGLU H, HARIRI S, WU M. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274.

作者简介：

何忠政（1986-），男，山东潍坊人，哈尔滨工程大学博士生，主要研究方向为容错计算、移动计算、容错调度。

门朝光[通信作者]（1963-），男，黑龙江哈尔滨人，哈尔滨工程大学教授，主要研究方向为容错计算、移动计算、容错调度。E-mail:menchaoguang@hrbeu.edu.cn。

陈拥军（1971-），男，北京人，中国新兴建设开发总公司教授级高工，主要研究方向为容错计算、移动计算、容错调度、土木工程。

李香（1975-），女，黑龙江哈尔滨人，哈尔滨工程大学副教授，主要研究方向为容错计算、移动计算、容错调度。