实时分布系统中 Out-Tree 任务的调度与检查点策略*)

方 明[1,2] 袁由光[2]

(哈尔滨工程大学计算机学院 哈尔滨 150001)[1] (武汉数字工程研究所产品研发部 武汉 430074)[2]

摘 要 针对实时分布系统中的 Out-Tree 任务，提出了一种启发式的调度算法(HSA_OT)，并开发了一种多处理机上的最优检查点策略。该调度算法能够保证任务的调度长度最小，所需处理器数目尽量少，没有处理机间通信开销。该检查点策略没有检查点全局一致性开销，可保证各处理机的失效率最低。

关键词 检查点，任务调度，Out-Tre 任务图，实时分布系统

Scheduling and Checkpointing Strategies for Out-Tree Tasks of Distributed Real-Time System

FANG Ming[1,2] YUAN You-Guang[2]

(Institute of Computer Science, Harbin Engineering University, Harbin 150001)[1] (Wuhan Digital Engineering Institute, Wuhan 430074)[2]

Abstract A heuristic scheduling algorithm (HSA_OT)is proposed for Out-Tree tasks of Distributed Real-Time System, and an optimal checkpointing scheme is derived also. The proposed scheduling algorithm can generate a schedule with the least Scheduled Length and a minimal number of Processors, and without any communication between Processors. The proposed checkpointing scheme has no overhead of global consistency, and can ensure the least failure probability of each processor.

Keywords Checkpoint, Task scheduling, Out-Tree task graph, Distributed real-time system

1 引言

任务的相关性一般采用 DAG 图来表示。Out-Tree 任务图则是一类特殊的 DAG 图，其特点是图中除了根结点之外，每个结点都只有一个父亲结点。Out-Tree 任务图代表了并行计算中分治(divide-and-conquer)算法一大类问题。

对于实时分布系统中的 Out-Tree 任务，最大的挑战是在系统发生故障时仍然要保证所有任务都在其时限内完成执行。因此，不仅要采用实时调度技术来保证系统的实时响应能力，还要同时采用容错技术来保障系统的可靠性。

对于分布实时系统中常见的瞬时故障，目前主要有两类容错处理方法：一种是基于任务复制的容错调度方法[1~4]，一种是基于检查点技术的卷回恢复方法[5~8]。一般而言，前者对系统的依赖较少，实现方便，后者则可大幅降低容错所需的重执行时间，资源利用率较高。对于独立任务，采用上述两种容错处理方法都具备可行性且各有利弊；而对于相关任务，若采用基于任务复制的容错调度，则每个处理机上运行的任务队列至少要在另一台处理机上保留一个热备份，导致资源利用率总是低于 50%。因此，Out-Tree 任务宜采取基于检查点的卷回恢复方法。

在现有的文献中，已有一些对单实时任务的检查点策略进行了研究[9~11]，少数文献研究了不相关多实时任务在单机上的检查点策略[12]，尚未见到相关多实时任务在多机上检查点策略的研究。从本文后面的阐述中可以发现，这是由问题的高复杂性所决定的，其复杂性体现在任务的相关性与调度策略，系统的故障模型，以及检查点的全局状态一致性。

本文选择结构规则的 Out-Tree 任务图，提出一种启发式的 Out-Tree 任务图调度算法，并通过符合实际地简化其故障模型，开发了一种无需考虑全局状态一致性的最优检查点策略。本文后面按如下组织：第 2 节研究 Out-Tree 任务图的启发式调度；第 3 节研究在启发式调度基础上的最优检查点策略；最后总结全文并指出进一步研究方向。

2 Out-Tree 任务图的分布实时调度

2.1 调度模型

Out-Tree 任务图可用一个四元组(V, E, τ, C)来表示，其中 $V=\{n_i \mid n_i$ 是有序任务，$i=1,2,\cdots,|V|\}$，其中 $|V|$ 表示任务的数目$\}$；$E=\{e_{ij} \mid e_{ij}$ 表示任务 n_i 到 n_j 的通信$\}$；$\tau=\{\tau_i \mid \tau_i$ 表示任务 n_i 的计算时间，$i=1,2,\cdots,|V|\}$；$C=\{c_{ij} \mid c_{ij}$ 表示任务 n_i 到 n_j 的通信时间$\}$。记任务 n_i 的前驱 $pred(n_i)=\{n_j \mid e_{ji} \in E\}$，任务 n_i 的后继 $succ(n_i)=\{n_j \mid e_{ij} \in E\}$。

对于 Out-Tree 任务图，存在唯一的称为根结点的任务 n_0，其前驱的数目 $|pred(n_0)|=0$，其后继的数目 $|succ(n_i)| \geqslant 2$。对于任何不为根结点的任务 n_i，其前驱的数目 $|pred(n_i)|=1$，其后继的数目 $|succ(n_i)| \geqslant 0$；若 $|succ(n_i)|=0$，则称任务 n_i 为叶子结点。

图 1 为一个 Out-Tree 任务图的实例。其中的圆圈表示任务结点，圆圈旁的数字表示其计算时间，带箭头的线段表示任务间的通信，线段上的数字表示其通信时间。

与调度 DAG 图一样，调度 Out-Tree 任务图的结果是一组映射到不同处理器的有序任务簇(Clusters)，簇内的各任务在一个处理器上非抢占地顺序执行。记一组有序任务簇为 C_k，$k=0,1,2,\cdots,M-1$，M 表示有序任务簇的数目，即执行 Out-Tree 任务所需处理器的数目。若以 $L(C_k)$ 表示一个任务簇全部任务的执行时间，则 $L(C_k)$ 最大的任务簇称为关键任务簇。定义 $SL=\max(L(C_k))$ 为该 Out-Tree 任务图的调度长

*)本文受十五国防重点预先研究项目(413160201)资助。方 明 博士研究生，主要研究方向是分布实时计算及计算机容错技术；袁由光 研究员，博士生导师，主要研究方向是计算机可靠性理论、容错及分布计算技术。

度。若将 Out-Tree 任务图的完成时限记(deadline)为 D，则必须满足 $SL \leqslant D$，否则调度失败。

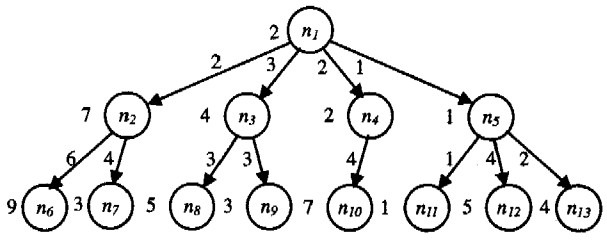

图 1　一个 Out-Tree 任务图的实例

2.2　HSA_OT 调度算法

对于 Out-Tree 任务图的调度，在处理器同构的系统环境下，现有的 DAG 图调度算法和专用 Out-Tree 任务图调度算法存在以下几方面的不足：1)算法时间复杂度过高，如 OSA 算法[13]、ESA 算法[14]；2)算法调度长度达不到最优，如 OPTA 算法[15]，或达到最优存在限制条件，如 LWB 算法[16]、TDS 算法[17]；3)减少处理器数目不够，且未考虑处理器数目有限时的调度，如 OSA，ESA，TDS，OPTA，TSA_OT[18]等算法；4)未考虑减少调度后处理机间的通信开销，如 OSA，ESA，TDS，OPTA 等算法。

```
Algorithm HSA_OT(Out-Tree)
{
//input: Out-Tree 任务图.
//output: 一组有序任务簇 Clusters, 使 SL 最小, 处理机间通信为 0,
        M 尽可能小.
Creat_Clusters(Out-Tree); //求一组 Clusters, 确定 C_key, 计算 SL.
Merge(Clusters); //合并非关键任务簇, 使 M 减少.
} //end of HSA_OT
```

图 2　HSA_OT 算法的伪代码

因此，求解 Out-Tree 任务图的任务调度问题，需要寻找一个算法，在满意的算法复杂度下，使调度长度最小，处理机间通信开销为 0，同时尽量减少所需处理器数目。为此本文提出了一个基于任务复制的启发式的调度算法，简称为 HSA_OT 算法(即 Heuristic Scheduling Algorithm for Out-Tree)。其伪代码如图 2。

函数 Creat_Clusters()是易于实现的，其实现步骤是：

步骤 1　采用任务复制的思想，将 Out-Tree 上的根结点到每个叶子结点的路径作为一个任务簇。

步骤 2　找到具有最大执行时间的关键任务簇，其执行时间为调度长度。

函数 Merge()采用启发式方法使非关键任务簇在不超过调度长度前提下充分合并，从而减少所需处理机数目。能否有效地减少处理机数目，关键在于是找到合理的启发函数。

若非关键任务簇 C_i，C_j($i \neq j$)可以合并，其合并实际上是将两者不重合的任务序列直接连接到其中一个任务簇上(如 C_i)。此时，我们可以说 C_i 上存在合并的空间，而 C_j 上存在可被合并的任务序列。

定义 1　合并空间

对于任何非关键任务簇 C_i，定义 $R_i = SL - L(C_i)$ 为 C_i 的合并空间。

定义 2　合并序列

若非关键任务簇 C_i 的合并空间 $R_i > 0$，则对于任何其它非关键任务簇 C_j，$j \neq i$，C_j 上总存在一个由与 C_i 不重合的任务所构成的任务序列，定义该任务序列为合并空间 R_i 在 C_j 上的合并序列，其计算时间之和称为合并长度 $L_{i,j}$。

显然　当 $R_i \geqslant L_{i,j}$ 时，合并才能发生。

定义 3　合并成本

若非关键任务簇 C_i 与 C_j 之间发生一次有效的合并(即后新任务簇的执行时间不超过 SL)，则所需处理器数目减少 1 个，C_i 的合并空间 R_i 减少了 R'_i，则定义一次合并的合并成本为 R'_i 与 R_i 的比率。

HSA_OT 算法采用一种启发式策略，即最小合并成本策略。其启发函数为：

$$\min(R'_i/R_i)=\begin{cases}1 & \text{(若不允许发生合并，则将 } R_i \text{ 被置 0，即 } R'_i=R_i\text{)} \quad (1)\\ 1 & \text{(若只允许发生 1 次合并，则在合并后将 } R_i \text{ 被置 0，即 } R'_i=R_i\text{)} \quad (2)\\ \min(L_{i,j})/R_i & \text{(若允许发生多次合并，则选 } L \text{ 值最小的任务簇合并，} R'_i=\min(L_{i,j})\text{，并置 } R_i=R_i-R'_i\text{)} \quad (3)\end{cases}$$

情形(1)将 R_i 置 0 表示任务簇 C_i 已没有合并空间。情形(2)无论选哪个满足 $R_i \geqslant L_{i,j}$ 的任务簇合并，合并后都不允许在发生合并，因此也将 R_i 被置 0。此时为充分利用合并空间，应选择合并长度最大的任务簇进行合并。情形(3)则直接选择具有最小合并成本的任务簇合并，合并后通过 $R_i = R_i - R'_i$ 计算新的合并空间。

函数 Merge()的实现步骤为：

1)选择 R 值最大的非关键任务簇，计算其它非关键任务簇上对应的 L 值。

2)判断选择：1)若不能发生合并(全部 L 值都大于该 R 值)，置该 R 值为 0；2)若只能发生一次合并(不存在多个 L 值之和小于等于 R 值)，则选择其中 L 值最大的任务簇参与合并，合并后置该 R 值为 0，并删除被合并任务簇；3)若可能发生多次合并(存在多个 L 值之和小于等于该 R 值)，则选择其中 L 值最小的任务簇参与合并，合并后重新计算其 R 值，并删除被合并任务簇。

3)返回步骤 1。当所有任务簇的 R 值都为 0 时，结束循环，输出全部任务簇并映射至处理器。

定理 1　HSA-OT 算法可以使 Out-Tree 任务图的调度长度最小，且处理器机间通信开销为 0。

证明：HSA-OT 算法的 Creat_Clusters()函数采用任务复制的方法，使 Out-Tree 任务图中任何结点(除根结点外)的前驱结点都被复制到同一任务簇，并映射到同一处理器。因此，任何结点(除根结点外)只依赖在同一处理机上执行的前驱结点的通信，而不需要其它处理机上结点的通信。由于同一处理机上任务间的通信开销可忽略不计，因此 HSA-OT 算法使 Out-Tree 任务图的处理机间通信开销为 0。

由于每个处理机都是从根结点到某个叶子结点无等待地执行，因此可保证各任务簇在最短时间执行完成，所以将其中最大的执行时间取为调度长度必为最小。得证。

2.3　实验

与专门的 Out-Tree 任务图调度算法 TSA_OT 相比，HSA_OT 克服了 TSA_OT 按照结点顺序随机选择非关键任务簇合并的缺点，采用启发式方法确定合并顺序，使非关键任务簇合并更充分，使所需处理机数目更少。为了验证 HSA-OT 算法减少处理器数目的效果，表 1 比较了 HSA_OT 和 TSA_OT 算法对于图 1 中 Out-Tree 任务图的调度结果，可以发现 HSA_OT 算法在调度长度上与 TSA_OT 算法持平，二者都不存在处理机间通信，但 HSA_OT 算法使所需处理机数目更少。图中 HSA_OT 的调度结果需要 4 个

处理机，而 TSA_OT 的调度结果需要 5 个处理机。

HSA_OT 算法对于图 1 中 Out-Tree 任务图的调度过程如下：

1)将根结点 n_1 到叶子结点 n_6，n_7，…，n_{13} 的路径分别各作为一个任务簇，记为 $C_0=\{n_1, n_2, n_6\}$，$C_1=\{n_1, n_2, n_7\}$，…，$C_7=\{n_1, n_5, n_{13}\}$。

表 1　HSA_OT 和 TSA_OT 算法对图 1 任务图的调度结果

调度所需要的处理机		P_0	P_1	P_2	P_3	P_4
HSA_OT 算法	任务簇	$\{n_1, n_2, n_6\}$	$\{n_1, n_3, n_9, n_4, n_{10}\}$	$\{n_1, n_2, n_{11}, n_{13}, n_7\}$	$\{n_1, n_5, n_{12}, n_3, n_8\}$	
	完成时间	18	18	18	17	
TSA_OT 算法	任务簇	$\{n_1, n_2, n_6\}$	$\{n_1, n_2, n_7, n_5, n_{11}\}$	$\{n_1, n_3, n_8, n_9\}$	$\{n_1, n_4, n_{10}, n_5, n_{12}\}$	$\{n_1, n_5, n_{13}\}$
	完成时间	18	14	16	17	8

2)由 $\max(Length(C_i))=Length(C_0)=18$，则 C_0 为关键任务簇，$SL=18$；则非关键任务簇的集合 $A=\{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$。

3)在非关键任务簇中计算最大的合并空间，有 $\max(R_i)=R_5=18-(\tau_1+\tau_5+\tau_{11})=14$，其它非关键任务簇对应于 C_5 的合并序列的合并长度 $L_{5,1}$，$L_{5,2}$，$L_{5,3}$，$L_{5,4}$，$L_{5,6}$，$L_{5,7}$ 分别为 10，9，7，9，5，4，因此可能发生多次合并。按照最小合并成本策略，由于 $L_{5,7}=4$ 最小，合并 C_7 到 C_5，得 $C_5=\{n_1, n_2, n_{11}, n_{13}\}$，$Length(C_5)=8$；并从集合 A 中删除 C_7。

4)重复步骤 3，有 $\max(R_i)=R_5=10$，对应合并序列的合并长度 $L_{5,1}$，$L_{5,2}$，$L_{5,3}$，$L_{5,4}$，$L_{5,6}$ 分别为 10，9，7，9，6，因此只能发生一次合并。按照最小合并成本策略，由于 $L_{5,1}=10$ 最大，合并 C_1 到 C_5，得 $C_5=\{n_1, n_2, n_{11}, n_{13}, n_2, n_7\}$，$Length(C_6)=18$；并从集合 A 中删除 C_1。

5)重复步骤 3，有 $\max(R_i)=R_6=10$，对应合并序列的合并长度 $L_{6,2}$，$L_{6,3}$，$L_{6,4}$ 分别为 9，7，9。只能发生一次合并，则合并 C_2 到 C_6，得 $C_6=\{n_1, n_5, n_{12}, n_3, n_8\}$，$Length(C_6)=17$；并从集合 A 中删除 C_2。

6)重复步骤 3，有 $\max(R_i)=R_3=9$，对应合并序列的合并长度 $L_{3,4}$ 为 9。只能发生一次合并，则合并 C_4 到 C_3，得 $C_3=\{n_1, n_3, n_9, n_4, n_{10}\}$，$Length(C_6)=18$；并从集合 A 中删除 C_4。

7)重复步骤 3，有 $\max(R_i)=R_6=1$，不存在合并序列。

8)重复步骤 3，不存在合并空间。结束循环，将关键任务簇 C_0 和集合 A 中剩余的非关键任务簇 C_0，C_3，C_5，C_6 映射到 4 个处理机 P_0，P_1，P_2，P_3。

3　最优检查点策略

对于单任务而言，最优检查点间隔决定于任务的执行时间 τ，用于卷回恢复的松弛时间 s，检查点的负载 cp，以及故障发生率和恢复率(λ, μ)[12]。因此，最优检查点间隔将决定于松弛时间 s。这一结论同样适用于单机上的多任务，即若给定 τ_i、t_{cp}、λ 和 μ，每个任务的最优检查点间隔决定于其相应的松弛时间 s。这说明检查点间隔是分别确定的。然而，每个任务的松弛时间 s 决定于多任务的调度策略。

现在我们考虑多机多任务。对于一组以 Out-Tree 任务图表示的有序实时任务 n_i，通过调度后分配到一组处理机上不强占地执行，该组处理机的检查点负载为 cp_j，故障发生率和恢复率为(λ_j, μ_j)，若每个实时任务的松弛时间可以确定，则每个任务的检查点间隔也是分别确定的。

至此，问题转化为两点：(1)确定每个任务的松弛时间 s_i；(2)当每个任务的松弛时间已确定时，确定每个任务检查点间隔，使执行有序实时任务 n_i 的失效率 p 最小。

3.1　实时任务的松弛时间的确定

在 Out-Tree 任务图经 HSA_OT 算法调度所得到的甘特图上，对于 $\forall n_j \in$ 任务簇 C_i，我们引入 4 个值来刻画其执行时间：

$estc_i(n_j)$ 和 $ectc_i(n_j)$ 分别表示 C_i 中任务 n_j 的最早开始时间和最早结束时间。对于根结点，$estc_i(n_i)=0$，$ectc_j(n_i)=\tau_i$；对于其他结点 n_j，$estc_i(n_j)$ 在完成调度时得到，$ectc_i(n_j)=estc_i(n_j)+\tau_j$；对于叶子结点，$estc_j(n_i)=SL-\tau_j$，$ectc_j(n_i)=SL$。

$lstc_i(n_j)$ 和 $lctc_i(n_j)$ 分别表示 C_i 中任务 n_j 的最迟开始时间和最迟结束时间。对于每个叶子结点，其 $lctc_i(n_i)$ 值等于 Out-Tree 任务图的完成时限 D，即 $lctc_i(n_i)=D$，则 $lstc_i(n_i)=D-\tau_i$；对于其他结点 n_j，其 $lstc_i(n_j)$ 和 $lctc_i(n_j)$ 值可从叶子结点反向求得。

图 3 显示了任务 n_k 设置检查点后的时间构成，其中 Δ_k 表示检查点间隔，r_k 表示检查点的数目。则

$$\Delta_k=\tau_k/r_k+t_{cp} \tag{4}$$

因此，

$$s_k=lctc_i(n_i)-(estc_i(n_k)+r_k*\Delta_k)=lctc_i(n_i)-estc_i(n_k)-\tau_k-r_k*t_{cp} \tag{5}$$

等式(5)显示检查点的数目确定时，任何任务的松弛时间都是确定的。

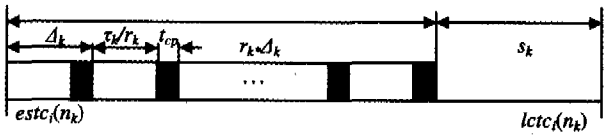

图 3　任务 n_k 时间构成

3.2　最优检查点间隔的确定

在一般故障模型下，当一个处理机上执行的任务数超过 3 时，失效率的估计是非常困难的。为了简化问题，本文做如下假定：

*A*1. Out-Tree 任务图的周期等于其时限 D，且调度长度 $SL \leqslant D$。

*A*2. 对于不同处理机上的任何任务，瞬时故障都服从故障发生率为 λ_i，恢复率为 μ_i 的 Poisson 分布。

*A*3. 任何瞬时故障发生总是导致错误发生和检查点恢复。

*A*4. 当任务没有足够的松弛时间进行恢复时，运行该任务的处理机失效。

A5. 对于任何处理机 P_i，假定其故障发生率 λ_i 非常小，以至在一个周期内最多只发生一次故障。

假定 A1 已在前面讨论。假定 A2 是通常的瞬时故障模型[10,19]。假定 A3 和 A4 在文[12]中已有论及。当故障发生率 λ_i 足够小时，假定 A5 也是合理的。此时，一个处理机发生瞬时故障的状态由{0,1,2,…}简化为{0,1}。这里 0，1 分别表示无故障和发生故障。由此我们可以在合理的计算复杂度下推导出每个处理机失效率的近似值。

设任务簇 C_i 包含 m 个任务。对于任务簇 C_i 上的任务 n_k，以 $V_{i,x}$ 表示检查点间隔 $x=[0,\Delta]$ 内无故障的转移概率矩阵，以 $U_{i,x}$ 表示检查点间隔 $x=[0,\Delta]$ 内发生故障的转移概率矩阵，则

$$V_{i,x}=\begin{vmatrix}\phi_{00}(x) & 0\\ 0 & 0\end{vmatrix},U_{i,x}=\begin{vmatrix}\phi_{00}(x)-\phi_0(x) & \phi_{10}(x)\\ \phi_{01}(x) & \phi_{11}(x)\end{vmatrix} \tag{6}$$

这里，$\phi_{pq}(x)(p,q\in\{0,1\})$ 表示在 $[0,\Delta]$ 内的开始状态为 p $(t=0)$，终止状态为 q $(t=\Delta)$ 时故障发生概率，$\phi_0(x)$ 表示在 $[0,\Delta]$ 内无故障发生的概率。

记 $Q_{i,k}$ 为任务 n_k 在一个周期内发生一次故障，且通过恢复成功完成执行的转移概率矩阵。根据假定 A4，处理机 P_i 在 m 个任务中任一任务未能成功完成执行时失效。假定 $R_{\max}$ 为系统允许的一次故障可恢复的最大次数，而松弛时间允许恢复发生的最大次数为 $\lfloor S_k/\Delta_k \rfloor$，则 $R=\min(R_{\max},\lfloor S_k/\Delta_k \rfloor)$ 为一次故障可恢复的实际次数。对于任务 n_k，图 3 已描述了其设置检查点后的时间构成。故

$$Q_{i,k}=\sum_{l=0}^{r_k-1}\sum_{m=1}^{R}(V_{i,\Delta k})^l(U_{i,\Delta k})^m(V_{i,\Delta k})^{r_k-1}$$

若任务簇 C_i 上的任务 n_1，n_2，…，n_m 中分别设置 r_1，r_2，…，r_m 个检查点，记 $q_{i,k}(r_1,r_2,\cdots,r_m)$ 为在一个周期内只有任务 n_k 发生一次瞬时故障，且 C_i 上全部任务都在各自时限内成功完成执行的概率，则

$$q_{i,1}(r_1,r_2,\cdots,r_m)=EQ_{i,1}(V_{i,\Delta_2})^{r_2}(V_{i,\Delta_3})^{r_3}\cdots(V_{i,\Delta_m})^{r_m}[1\ 0]^T \tag{8}$$

$$q_{i,k}(r_1,r_2,\cdots,r_m)=E(V_{i,\Delta_1})^{r_1}(V_{i,\Delta_2})^{r_2}\cdots(V_{i,\Delta_{k-1}})^{r_{k-1}}Q_{i,k}(V_{i,\Delta_{k+1}})^{r_{k+1}}\cdots(V_{i,\Delta_m})^{r_m}[1\ 0]^T \tag{9}$$

$$q_{i,m}(r_1,r_2,\cdots,r_m)=E(V_{i,\Delta_1})^{r_1}(V_{i,\Delta_2})^{r_2}\cdots(V_{i,\Delta_{m-1}})^{r_{m-1}}Q_{i,m}[1\ 0]^T \tag{10}$$

这里 $E=[\mu/(\mu+\lambda)\quad\lambda/(\mu+\lambda)]$，为任务簇 C_i 开始一个周期执行时的初始概率。

进一步定义 $q_i(r_1,r_2,\cdots,r_m)$ 为一个周期内任务簇 C_i 上任意一个任务发生一次瞬时故障，且 C_i 上全部任务都在各自时限内成功完成执行的概率，则

$$q_i(r_1,r_2,\cdots,r_m)=\sum_{n=1}^{r}q_{i,n}(r_1,r_2,\cdots,r_m)+E(V_{1,\Delta_1})^{r_1}(V_{1,\Delta_2})^{r_2}\cdots(V_{1,\Delta_m})^{r_m}[1\ 0]^T \tag{11}$$

其中，等式(11)右边第二项为一个周期内没有任务发生瞬时故障的概率。

记 $p_i(r_1,r_2,\cdots,r_m)$ 为任务簇 C_i 在处理机 P_i 上一个周期内的失效率，则

$$p_i(r_1,r_2,\cdots,r_m)=1-q_i(r_1,r_2,\cdots,r_m) \tag{12}$$

Out-Tree 任务图中全部任务在一个周期上的失效率 $p(r_1,r_2,\cdots,r_m)$ 可通过对各任务簇的失效率求积求得，即：

$$p=\prod_i^m p_i(\cdots)=\prod_i^m(1-q_i(\cdots))$$

由等式(12)可知，当失效率 $p_i(r_1,r_2,\cdots,r_m)$ 取最小值时，r_1，r_2，…，r_m 的取值为任务簇 C_i 上各任务的最优检查点个数。最优检查点个数可由等式(4)计算。

注意，由于不存在处理机间的通信，本文提出的检查点策略不需要考虑检查点的全局一致性的方案，其开销为 0，

3.3 实验

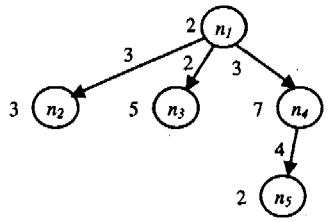

图 4 一个简单的 Out-Tree 任务图

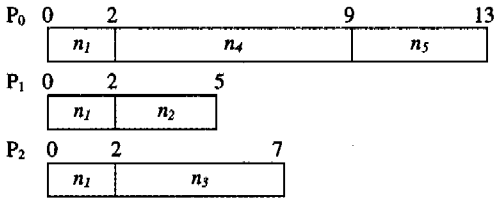

图 5 HSA-OT 算法调度图 4 任务图的结果

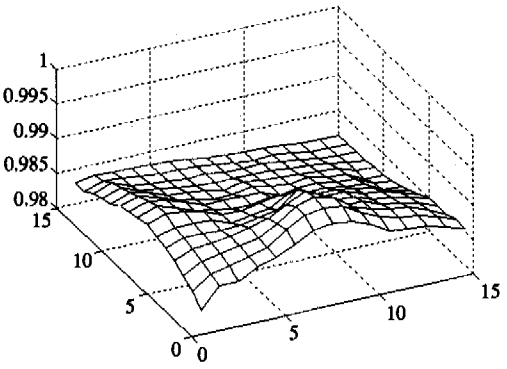

图 6 概率 $q_1(r_1,r_2)$ 的曲面图
($[\tau_1,\tau_2]=[2,3]$，$[\lambda_1,\mu_1]=[0.0012,8]$，and $t_{cp}=0.01$)

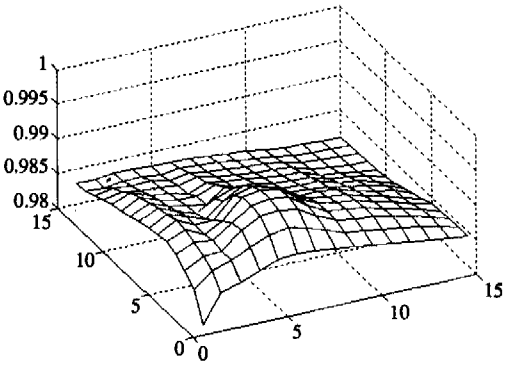

图 7 概率 $q_2(r_1,r_3)$ 的曲面图
($[\tau_1,\tau_3]=[2,5]$，$[\lambda_1,\mu_1]=[0.0015,8]$，and $t_{cp}=0.01$)

以下以一个简单的 Out-Tree 任务图(如图 4)为例进行说明。其中 $[\tau_1,\tau_2,\tau_3,\tau_4,\tau_5]=[2,3,5,7,4]$，$[e_{12},e_{13},e_{14},e_{45}]=[3,2,3,4]$，deadline D=16。该任务图经过 HSA-OT 算法调度的调度结果为 3 个任务簇 $C_0=\{n_1,n_4,n_5\}$，$C_1=\{n_1,n_2\}$，$C_2=\{n_1,n_3\}$(如图 5 所示)，并分别映射到处理机 P_0、

P_1、P_2 上执行。假定 P_0、P_1、P_2 的故障发生率和恢复率分别为$[\lambda_0, \mu_0]=[0.001, 10]$,$[\lambda_1, \mu_1]=[0.0012, 8]$,$[\lambda_2, \mu_2]=[0.0015, 8]$。

按照上述推导经编程计算，容易求解其结果：对于处理机 P_0，$p_0(r_1, r_4, r_5)$在$(r_1, r_4, r_5)=(4,8,5)$时取得最小值，这意味着当任务 n_1，n_4，n_5 分别设置 4，8，5 个检查点时，处理机 P_0 的失效率最低；同样，对于处理机 P_1，$p_1(r_1, r_2)$在$(r_1, r_2)=(5, 6)$时取得最小值；对于处理机 P_2，$p_2(r_1, r_3)$在$(r_1, r_3)=(3, 8)$时取得最小值。

由于处理机 P_1、P_2 的失效率 $p_1(r_1, r_2)$和 $p_2(r_1, r_3)$分别随(r_1, r_2)和(r_1, r_3)的变化是三维的，图 6 和图 7 给出了其曲面图。为便于观察，图中采用的是概率 $q_1(r_1, r_2)$和 $q_2(r_1, r_3)$的曲面图$(1\leqslant r_1, r_2, r_3\leqslant 15)$。由等式(12)可知，当概率 $q_1(r_1, r_2)$和 $q_2(r_1, r_3)$分别取最大值时，失效率 $p_1(r_1, r_2)$和 $p_2(r_1, r_3)$为最低。

结论 本文针对实时分布系统中的 Out-Tree 任务图，提出了一种启发式的 Out-Tree 任务图调度算法，并通过符合实际地简化其故障模型，开发了一种多处理机上相关多任务的最优检查点策略。该调度算法能够保证任务的调度长度最小，所需处理器数目尽量少，没有处理机间通信开销；该检查点策略没有检查点全局一致性开销，可保证各处理机的失效率最低。

参 考 文 献

1 Ghosh S, Melhem R, Mosse D. Fault-tolerant rate-monotonic scheduling. Journal of Real-Time System, 1998, 15(2): 149～181
2 Ghosh S, Melhem R, Mosse D. Enhancing real-time schedules to tolerate transient faults. In: Proceedings of the 16th IEEE Real-Time Systems Symposium, 1995. 120～129
3 Pedro M A, Mosse D. A responsiveness approach for scheduling fault recovery in real-time systems. In: Fifth IEEE Real-Time Technology and Applications Symposium, 1998. 4～13
4 Pedro M A, Aydin H, Mosse D, et al. Scheduling optional computations in fault-tolerant real-time systems. In: Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications, 2000. 323～331
5 Seong W K. Reliability analysis and design of real-time fault tolerant control systems under transient faults: [Ph. D thesis]. Korea Advanced Institute of Science and Technology, 2000
6 Krishna C M, Singh A D. Optimal configuration of redundant real-time systems in the face of correlated failure. IEEE Trans on Reliability, 1995, 44(4): 587～594
7 Krishina C M, Shin K G. Real-Time Systems. New York: McGraw Hill, 1997
8 Daniel P S, Robert S S. Reliable Computer Systems: design and evaluation. 2nd ed. Burlington, Digital Press, 1992
9 Geist R, Reynolds R, Westall J. Selection of a checkpoint interval in a critical-task environment. IEEE Trans on Reliability, 1988, 37(4): 395～400
10 Shin K G, Lin T H, Lee Y H. Optimal checkpointing of real-time tasks. IEEE Trans on Computers, 1987, 36(11): 1328～1341
11 Krishna C M, Singh A D. Reliability of checkpointed real-time systems using time redundancy. IEEE Trans on Reliability, 1993, 42(3): 427～435
12 Seong W K, Byung J C, Byung K K. Checkpointing strategy for multiple real-time tasks. In: Proceedings of the Seventh International Conference on Real-time Systems and Applications, 2000. 517～522
13 Park CI, Choe T Y. An optimal scheduling algorithm based on task duplication: [Technical Report]. Dept of CSE, POSTECH, August 2000
14 Ruan Y, Liu G, Li Q, et al. An efficient scheduling algorithm for dependent tasks. In: Proceedings of the Fourth International Conference on Computer and Information Technology, 2004
15 Zhang H, Hu M, Fang B X, et al. A sub-optimal algorithm on allocating a single task cluster on NOWs. Journal of Computer Research and Development, 1999, 36(9): 1076～1079
16 Colin J Y, Chritienne P. C. p. m. scheduling with small communication delays and task duplication. Operations Research, 1991, 39(4): 680～684
17 Darbha S, Agrawal D P. Optimal scheduling algorithm for distributed- memory machines. IEEE Trans on Parallel and Distributed systems, 1998, 9(1): 87～95
18 Liu Z Y, Fang B X, Zhang Y, et al. A scheduling algorithm for an Out-Tree DAG. In: Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in Asia-Pacific Region, 2000
19 Ziv A, Bruck J. An on-line algorithm for checkpoint placement. IEEE Trans. on Computers, 1997, 46(9): 976～985

(上接第 279 页)

块引用就可以了，而依赖类则在 ImportDeclPart 语句中将这些操作引入。这与依赖关系处理的区别在于 ExportDeclPart 语句中只列出一些操作而不会列出任何属性。

5 相关工作比较

目前在对 UML 类图的形式化描述方面，国内外已做出了相关工作[9～11]，这些工作大多是基于 Z 的静态描述语言如 B、COOZ 等进行形式化定义的，但它们都存在一些不足之处：一方面 Z 语言对大型系统的模块化能力不足，另一方面尽管这些静态语言可以描述系统的组成等结构性特征，但对于系统的动态行为特征描述显得力不从心。因此，若对 UML 类图采用这些静态语言来描述，则会在实用性上受到限制。而时序逻辑语言 XYZ/E 则强调静态语义与动态语义并重，它是一种广谱语言，这种语言同时具有多种级别语言的特点，既可以用来描述规格说明，也可以用来编写可运行的程序代码，而且支持混合语言。因此我们选择 XYZ/E 来对 UML2.0 中的类图进行形式化描述，相对来说具有较好的实用性。

结束语 本文在对 UML2.0 类图进行形式化描述的过程中，采用了 XYZ/E 来表示类图元素的形式化语义，并且给出了一套其形式化描述的转换规则。有了这样的一组规则，就可以在对一个 UML2.0 类图结构进行描述时，根据我们所给定的转换规则，很容易得到其精确的形式化描述。面向方面编程(AOP)是对 OOP 的继承和发展，AOP 将系统分解成核心关注点和横切关注点，对于核心关注点使用传统的面向对象机制，对于横切关注点则使用 Aspect 机制。这就要求 UML 不仅能表示出系统中的类，也要能表示其中的 aspects。我们下一步的工作是，扩展 UML2.0 以表示出系统中的横切关注点，同时为了能够得到精确的形式化描述，也需要进行 XYZ/E 对面向方面的扩充工作。

参 考 文 献

1 The precise UML group. http://www.cs.york.ac.uk/puml/uml2-0
2 朱雪阳. 软件体系结构形式描述研究:[博士学位论文]. 北京：中国科学院软件研究所，2005.2
3 黄正宝，张广泉. UML2.0 顺序图的 XYZ/E 时序逻辑语义研究. 计算机科学，2006，33(8)：249～251
4 陈琳琳，戎玫，张广泉. 体系结构描述语言 XYZ/ADL 到 UML 的映射. 计算机应用，2006，26(2)：468～471
5 唐稚松，等. 时序逻辑程序设计与软件工程. 北京：科学出版社，2002
6 Object Management Group. UML 2.0 Superstructure Specification: Final Adopted Specification. http://www.omg.org/docs/ptc/04-10-02.pdf
7 Hans-Erik Eriksson 等. UML 2 工具箱. 北京：电子工业出版社，2004
8 郭亮，唐稚松. XYZ/E 面向对象程序语义概述. 软件学报，2003，14(3)：356～361
9 Kim S K, Carrington D. A Formal Mapping between UML Models and Object-Z Specifications. ZB 2000, LNCS 1878, Berlin Heidelberg: Springer-Verlag, 2000. 2～21
10 Ramalho F, Robin J. Mapping UML Class Diagrams to Object-Oriented Logic Programs for Formal Model-Driven Development. http://www.metamodel.com/wisme-2004/ accept/2.pdf
11 周瑾，马应龙，李巍，吴志林. UML 的形式化及其应用. 计算机科学，2005，32(3)：136～140