·移动互联与通信技术· 文章编号：1000—3428(2013)12—0102—05 文献标识码：A 中图分类号：TP393

基于裂痕故障块的自适应容错路由表算法

林 沛[1]，杨 裔[2]，陈宜漂[2]，邓毓博[2]

(1. 兰州文理学院网络中心，兰州 730000；2. 兰州大学信息科学与工程学院，兰州 730000)

摘 要：基于裂痕故障块的二维网格自适应容错路由算法是一种有效的容错算法，不仅能够解决活锁问题，而且克服了传统故障块模型中状态良好的节点不能参与路由的缺陷，但同时具有明显的缺点：每次路由到以故障块边界节点为根节点的内部树时，都需要遍历此内部树，因此算法的路由长度并不是最短的。针对上述问题，提出基于裂痕故障块的自适应容错路由表算法，其中路由表由裂痕故障块内部树上的节点创建，通过路由表上保留的有用消息决定是否遍历内部树。实验结果证明，随着网格规模的扩大，该算法最大可减少 70%的平均路由长度，并且其实现简单，可以有效地延长网络寿命。

关键词：自适应路由；裂痕故障块；虚拟网络；容错；路由表；二维网格

Adaptive Fault-tolerant Routing-table Algorithm Based on Cracky Fault Block

LIN Pei[1], YANG Yi[2], CHEN Yi-piao[2], DENG Yu-bo[2]

(1. Network Center, Lanzhou University of Arts and Science, Lanzhou 730000, China;
2. School of Information Science and Engineering, Lanzhou University, Lanzhou 730000, China)

【Abstract】The 2D mesh adaptive fault-tolerant routing algorithm based on cracky fault block is an effective fault-tolerant algorithm. It not only can solve the problem of live lock, but also can overcome the drawback that good nodes in the faulty block can not be used to take part in normal routing. However, this algorithm still has explicit disadvantages that its routing path is not the shortest due to its need to traverse every interior spanning tree while passing the cracky fault block. This paper proposes an adaptive fault-tolerant routing-table algorithm aiming at the problem of the shortest routing length. The table is created in the tree nodes memory and preserves the useful information to decide whether to traverse the interior tree or not. Experimental results prove that the proposed algorithm can reduce the average length of routing path by 70% with the expansion of mesh grid, and the algorithm is easy to implement and can significantly prolong networks lifetime.

【Key words】adaptive routing; cracky fault block; virtual network; fault-tolerance; routing-table; 2D mesh

DOI: 10.3969/j.issn.1000-3428.2013.12.022

1 概述

消息路由就是将组成消息的数据包在网络中通过一定的中转节点次序从源节点发送到目标节点的过程，因此，路由策略的好坏直接关系到网络系统的性能。从适应性的角度分类，网络中的路由方式可分为确定性路由和适应性路由。在确定性路由中，路径的选择完全是由源节点和目标节点来决定的，而不用考虑其他节点的状况。在适应性路由中路径的选择根据网络的一些实际情况随时进行调整，比如出现网络拥塞情况和坏节点、坏连接的情况。

确定性路由的优点是简单、容易实现，而自适应路由比确定性路由灵活，并且自适应路由在减少网络延迟、提高网络吞吐量以及容错性能等方面[1]要比确定性路由更有优势。本文给出了基于裂痕故障块模型的二维网格自适应容错路由表算法。该算法的基本思想是，在网络好的情况下采用贪婪算法实现最短路由，而遇到出错节点时，采用贪婪算法会引起死锁问题，因此，构造故障块来解决死锁问题，并通过构造有缝隙的故障块，实现内部节点的访问。在本文提出的容错路由表算法中，处于块边界及内部的点都有特殊的路由，并且证明只要节点是连通的，节点间的消息就一定能被传送到，从而克服潜在的活锁。传统的故障块[2-4]由于将正常的节点包含进去，导致大量正常节点无

基金项目：甘肃省自然科学基金资助项目(1107RJZA188)

作者简介：林 沛(1983－)，男，讲师、硕士、CCF 会员，主研方向：网络安全，网络规划与优化；杨 裔，副教授、博士；陈宜漂，硕士；邓毓博，博士

收稿日期：2013-07-24 **修回日期：**2013-09-28 **E-mail：**chenkunkiss@163.com

法参与并行计算机的其他工作，而在裂痕故障块模型下，故障块内部的节点也能被访问到，只要故障块内部的点能够与故障块外面的网络连通，就能像正常节点那样参与路由，并且当网络中小部分节点出错时，不会导致大量节点不能使用，从而提高了算法的节点利用率以及并行计算机网络的性能。

2 二维网格中的自适应容错路由

网状结构是多处理器计算机系统互联常用的拓扑结构，它已被广泛应用在商业产品中。网状结构中的低维网格由于其使用简单、处理起来容易，因此比高维网格更流行。比如二维网格[5]，在 Intel Touchstone DELTA[6]、Intel paragon 中都被作为拓扑结构得到应用，而 MIT J-Machine 采用三维网格。《通信网络和无线网络传感器网络中的应用》[7]中提到的算法就是以网格作为并行计算机的拓扑结构，并在其上进行路由规则和容错规则的研究。

一个好的路由规则必须很容易实现，不需要全局信息，并且支持容错。文献[8]通过使用 4 个虚拟通道(virtual channel)，实现了在 n 维网格的无死锁(deadlock-free)路由。该算法不仅可以对网格内部的错误区域进行容错路由，并且可以对网格的边界错误实现自适应容错路由。而文献[9]提出了错误环(fault ring)的方法实现网络的自适应容错路由。有故障的节点或链路包含在错误环内，路由时对错误环上的点采用 misrouting 路由。错误环的优点是它的构建只由局部信息完成，没有全局消息的参与，当错误环发生互相交叠的情况时，算法将使用 4 个虚拟通道的方法实现无活锁路由和无死锁路由。

文献[10]通过将网络划分成相互链接的 2 个虚拟子网络(virtual network)(VIN1, VIN2)实现自适应无死锁路由。其中，确定性无死锁路由在 VIN1 虚拟网络上进行，而在 VIN2 虚拟子网络上进行完全自适应路由。网络中的所有节点被分成安全(safe)、不安全(unsafe)和错误(faulty)3 种状态。当 VIN1 或者 VIN2 处于可用状态时，它们都被用来进行正常的消息路由。当有故障发生时，算法随即将出错的节点或者链接标记成不安全或者错误状态，当节点或者链接被标记为这种状态时，将在与之相邻的 VIN2 中进行容错路由，最终消息将通过 VIN2 绕过故障块，实现自适应容错路由。

文献[11]通过用错误块标记网络中发生的故障，将发生故障的节点和链接包含在一个凸(convex)形的区域中，将有错的节点或链接与无错的部分区分开来，实现容错路由。但采用了凸形形状后，一部分正常节点和链接被包含在这个凸区域中，从而降低了节点的利用率，而且当故障发生在网络边界时，无法解决边界的容错路由。文献[12]仅使用 2 条虚拟通道实现了在 3 维网格网络上的无死锁路由，而且无死锁情况可以拓展到 n 维网格上。文献[13]提出了使用错误立方体的方法寻找 3 维网格网络中最短容错路由路径的方法。错误立方体(faulty cube)把出错的节点包含进去，通过收集一些信息来寻找一条从源节点到目的节点的最短路径。文献[14]在基于奇偶转向模型的基础上，提出了一种基于哈密顿路径的自适应路由算法，该算法不仅解决了网络的死锁问题，而且具有较低的延迟和网络功耗，最大化了网络路由路径的选择。

二维网格记为 $M(m_1,m_2)$，其中，m_1、m_2 都是整数并且 $m_1\geqslant 2$、$m_2\geqslant 2$；网格中节点 N 用二维向量 $\boldsymbol{N}(n_1, n_2)$表示；$0\leqslant n_1\leqslant m_1-1$，$0\leqslant n_2\leqslant m_2-1$，$n_1$、$n_2$ 分别表示二维网络的一维和二维，即相当于横坐标和纵坐标。2 个不同节点 $X(x_1, x_2)$、$Y(y_1, y_2)$满足 $x_1=y_1$ 且 $x_2=y_2\pm 1$ 或者 $x_1=y_1\pm 1$、$x_2=y_2$ 表示这 2 个点是邻居。对任意的节点 v，令 S 为 $M(m_1, m_2)$中与 v 相邻的点构成的集合(简称邻集)。对任意的非边界点 $X=(x_1, x_2)$，X 有 4 个邻点，分别用 $W(X)$、$E(X)$、$N(X)$和 $S(X)$表示，其中，$W(X)=(x_1-1,x_2)$; $E(X)=(x_1+1,x_2)$; $N(X)=(x_1,x_2+1)$; $S(X)=(x_1,x_2-1)$。若 $X=(x_1,x_2)$是边界节点，则 X 的邻居是 $W(X)$、$E(X)$、$N(X)$和 $S(X)$中的 2 个~3 个。两节点 $X=(x_1,x_2)$和 $Y=(y_1, y_2)$之间的距离用 $d(X,Y)$表示，所谓距离是从 X 到 Y 所经过的最少链路数。X 和 Y 的距离 $d(X,Y)$包括横向距离和纵向距离，分别定义为 $d_1(X,Y)$和 $d_2(X,Y)$，其中，$d_1(X,Y)=\|x_1-y_1\|$；$d_2(X,Y)=\|x_2-y_2\|$，于是有 $d(X,Y)=d_1(X,Y)+d_2(X,Y)$。

3 二维网格中裂痕故障块的形成

3.1 二维网格故障块的形成

在含有故障的网格中形成一些极小的不交块，使得处于块的边界和块的外部的链路都无故障，而所有的故障链路或节点都位于块的内部，称这样的块为故障块。

考虑网络中故障的基本元素为链路和节点。当节点 X 连接到其他节点的链路只要有一条发生故障时，则这个点被标记为故障点；为构造故障块，本文采用发送系统消息的办法，通过发送系统消息，通知邻居节点自己被标记的状态，从而使得接收该系统消息的节点做出相应的标记，并将改变后的状态继续告知周围的邻居节点，直到最终状态稳定后不再发送系统消息。该系统消息用 M 表示，$M(Xp)$表示节点 Xp 发送的系统消息，其中，Xp 表示发送系统消息给节点 X 的邻点。

定义节点 X 的故障数为网格中与 X 关联的故障链路的数目，记为 $fd(X)$。易知，$0<fd(X)<4$。记 $State(X)$为节点 X 的状态，X 的状态有 $State(X)=\{Ø,1,S,W,N,E\}$，令 $A(X)$为网格中与 X 相邻且连接的链路没有故障的所有节点 Y 构成的集合。当节点 X 收到来自某个邻点 Xp 的系统消息 $M(Xp)$时，做交集运算 $M(Xp)\cap State(X)$。若 $State(X)\cap M(Xp)\cap State(X)$，则对 $State(X)$进行新的赋值 $State(X)=M(Xp)\cap State(X)$，再根据新的状态 $State(X)$，按以上规则发送新的系统消息 $M(X)$给 $A(X)$中所有的点或部分点。否则，不更新 X 的状态，也不再发送新的系统消息。

3.2 裂痕故障块的形成

如果 X 的状态为空，即 $State(X)=Ø$，但 X 能通过一条

由无故障链路组成的路连接到故障块的边界点，则称 X 点是连通的。将故障块内部的连通点经无故障链路悬挂在以边界点为根的树上，由故障块的边界和“内部树”所形成的子图称为裂痕故障块。下面介绍裂痕故障块的形成过程：设布尔状态集为 $L=\{h,f\}$，待节点的状态稳定后，若 X 的状态不为空集，即 $State(X)\neq\varnothing$，则将 X 点挂起来，即令 $l(X)=h$；反之，则 X 点是自由的，即令 $l(X)=f$。若 $l(X)=f$，存在 $Y_i\in A(X)$使得 $l(Y_i)=h$，则称 Y_i 是 X 的前任，记为 $pred(X)$，且将 X 的布尔状态改为 h。直到故障块内所有的连通节点的布尔状态都为 h，此时网格中的裂痕故障块便形成。完整裂痕故障块和不完整裂痕故障块，如图 1 所示。

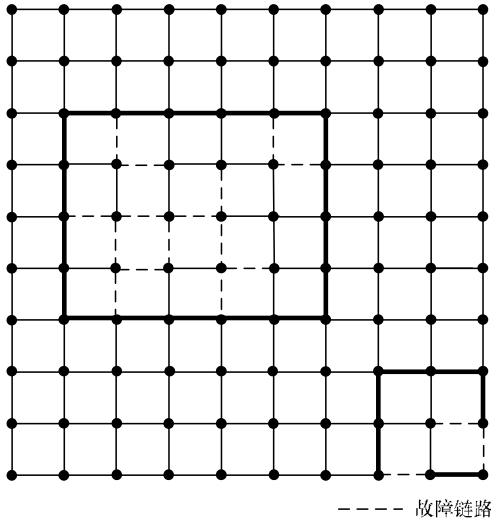

图 1 完整裂痕故障块和不完整裂痕故障块

4 自适应容错路由表算法

基于裂痕故障块的自适应容错算法无论目标节点有没有在故障块内，都要遍历故障块的内部树，导致浪费很多不必要的节点，增大了时间开销，进而影响并行计算的整体性能。而且当故障块很大时，遍历内部树花费的无用时间远远大于有效的路由时间，基于此，提出一种改进的算法，该算法就是对内部树的每个节点存储相关的后继节点，当消息路由在内部树根节点或者内部的树节点时，通过判断要到达的目标节点是否在当前节点的后继集中来确定是否遍历内部树。

4.1 路由表构造

首先定义节点 X 的路由表 $list(X)$为：

(1)如果 X 的后继集不为空，则将 X 后继集添加到 X 的路由表 $list(X)$中。

(2)对于新添加进来的每个元素如果也有后继，则这些后继继续添加到 $list(X)$中。

(3)重复步骤(2)，直到新添加进来的元素不再有后继。

算法 1 故障块内的点添加路由表

```
procedure initial_list (node X)
if (Succ(X)!=Ø)
then list(X)←Succ(X)
end if
end procedure
procedure Algorithm list_ create (node X)
for every x_i∈list(X)
if (list(x_i)!=Ø)) then
do
list(X)←Succ(x_i)
end if
end procedure
```

添加了路由表后，对应的自适应容错路由算法如下：

算法 2 自适应容错路由表算法

```
procedure Table-routing(node X)
 if X is in good status
then
      route by greedy routing;
end if
else if X's Table is not empty and the destination node is in the table;
then
      traverse it;
end if;
else if X's status is Ø;
do
      X=Pred(X);
      Check the table of X, visit the other branches of X, if the
      destination node is in its table;
while the X is not the root node or the destination node;
end if;
else route along the edge of cracky block clockwise(anticlock-wise);
end else
```

4.2 自适应容错路由表算法分析

现在假设有一消息要从节点 s 发送到节点 d，如图 2 所示。按照普通路由的方法则路由经过的点为 s-a-b-c-g-c-b-a-h-e-h-i-j-k-o-k-l-m-n-d，路由路径长度为 19，其中无效的路由长度为 10，有效的长度为 9，无效路径长度是有效路径长度的 1.1 倍，可见，这种自适应容错算法的路由效率不高。

若给故障块的每个节点增加路由表，以图 2 所示的故障块为例对故障块内的每个节点增加路由表进行具体说明，节点 a 的后继有 b、c，根据定义，节点 a 的路由表为 $\{b,c,g\}$，依次类推有节点 b 的路由表$\{c,g\}$。这时只需判断目标节点是否在该路由表中，若是，则进入故障块内进行路由，否则继续沿着故障块的边界路由，从而彻底消除无效的路由。按照这种方法，当消息传到节点 a 时，遍历节点 a 的路由表，发现节点 d 不在故障块内，直接向上路由到 h，同样发现节点 d 也不在 h 的路由表中，因而继续往上走，直到到达 d 点，此时路由经过的点分别为 $s\rightarrow a\rightarrow h\rightarrow$

$i \rightarrow j \rightarrow k \rightarrow l \rightarrow m \rightarrow n \rightarrow d$，路由路径长度为 9，改进后的算法路由长度是原来路由方法的路由路径长度的 47.37%。

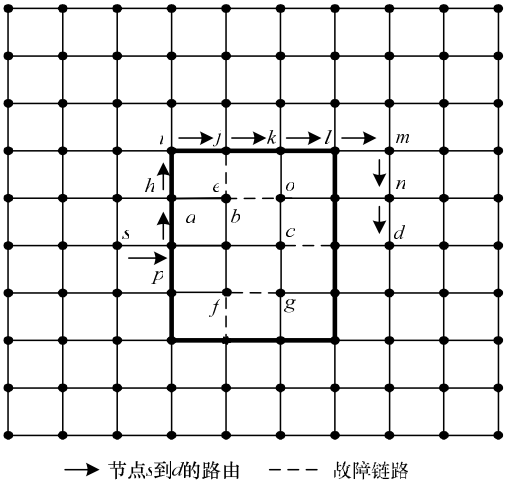

图 2　有路由表的裂痕故障块及相应路由

5　实验设计

本文使计算机随机产生 1 000 对不同的发送点和接收点，且随机产生不同规模的故障块，在不同网格规模下(25×25、50×50、100×100、200×200)按照故障块占比由小到大进行实验。实验选择故障块占比按 10%的梯度递增，生成的故障块也是由随机产生的故障节点构造的。不同网格规模和故障块占比下 2 种算法的平均路由长度如表 1 所示。数据结果显示，2 种算法随着节点规模的增大，在故障块占比为 0 的情况下，路由路径长度按一定比例增大，基本上满足线性关系，即当节点规模扩大 4 倍时，路由路径长度大概增加 1 倍。对于不同的网格规模，当故障块占比小于 20%时，自适应容错路由表算法路径长度基本上与故障块占比为 0 的情况相同，可是自适应容错路由算法的路由长度却已经放大了好几倍，并且随着节点规模的增大，自适应容错路由算法的路由长度放大的倍数会更大，例如 2 500 节点规模，故障块占比 20%时，放大 2.58 倍(85/33)，到 4 万节点规模时，已经放大到 3.73 倍(496/133)，同比自适应容错路由表算法的路由长度，在 4 万节点规模下，在故障块占比为 20%时，路由路径长度为故障块占比为 0 时的 2.44 倍(325/133)。

表 1　不同网格规模和故障块占比下算法的平均路由长度

故障块占比/(%)	25×25 网格规格			50×50 网格规格			100×100 网格规格			200×200 网格规格		
	自适应容错算法	自适应容错路由表算法	平均路由长度减少比例/(%)	自适应容错算法	自适应容错路由表算法	平均路由长度减少比例/(%)	自适应容错算法	自适应容错路由表算法	平均路由长度减少比例/(%)	自适应容错算法	自适应容错路由表算法	平均路由长度减少比例/(%)
0	16	16	0.00	33	33	0.00	67	67	0.00	133	133	0.00
10	22	17	22.73	44	34	22.59	110	78	29.09	209	155	25.86
20	29	19	34.48	85	54	36.47	252	159	36.75	496	325	34.51
30	39	21	48.72	150	78	48.00	453	237	47.62	970	550	43.30
40	72	29	59.72	236	109	53.81	819	352	57.02	1 992	900	54.80
50	109	43	60.55	401	159	60.35	1 630	616	62.22	4 331	1 576	63.61
60	205	61	70.24	764	265	70.31	3 687	1 071	70.94	9 080	2 790	69.27
70	293	120	59.04	1 141	494	56.70	5 583	2 432	56.44	21 602	9 319	56.86
80	439	275	37.36	1 624	1 002	38.30	6 683	4 403	34.11	27 219	16 530	39.27

表 1 的数据还有一个明显的规律，自适应容错算法当故障块规模达到 70%以上时，路由路径长度基本不再发生很大变化，原因是当规模达到 70%以上时，故障块几乎遍布整个网格，近似每次实验产生相同的故障块大小，因此，路由路径也应该变化不大。而且从表中数据可看到，4 种节点规模的情况下,2 种算法在故障块占比从 50%提高到 60%时，自适应容错算法路由路径长度显著增大。对于自适应容错路由表算法，当网格规格为 100×100 和 200×200 时，路径长度增大了 2 倍~3 倍左右，其他情况下都小于 2 倍。

当故障块占比超过 60%时，发现自适应容错算法的路由路径增长速度变缓，这是因为故障块占比超过 60%时，每次产生的随机发送点和接收点落在故障块内的概率很大，自适应容错路由算法几乎要把故障块内的每个连通的路由节点跑一遍，因此，故障块占比继续提高对自适应容错算法的路由长度影响没有开始时那么大。

自适应容错路由表算法的路由长度一直都小于自适应容错路由算法的路由长度。随着故障块占比的不断增大，路由长度也很快地增大，当故障块占比为 60%时，4 种网格规格下，自适应容错路由表算法的路由长度比自适应容错路由算法的路由长度减少比例均达到了最大值。

为了评估增加后继节点后对于路由表空间的开销，固定故障块占比为 10%，随机产生 100 对不同的路由，对于不同规模的网格进行实验，结果如表 2 所示。经过多次仿真求平均的结果，虽然每次的实验结果都不一样，但围绕着表中的数据上下波动，因此表 2 中的数据具有代表性和可信性。

表 2 不同网格规格下路由表后继节点数所占比例 (%)

路由表后继节点数	网格规格			
	25×25	50×50	100×100	200×200
2 个后继节点	16.39	21.30	31.58	39.18
3 个后继节点	7.72	9.39	13.09	29.07
4 个后继节点	3.33	5.02	7.61	12.85
多于 4 个后继节点	1.09	2.15	5.08	8.68

表 2 中的数据是稳定后的结果，由于被测试数据是随机产生的，故障块也是随机产生的，因此每次程序运行结果、数据都不一样，但是程序运行产生的结果都是围绕表格中的数据上下波动，因此，能够在趋势上较准确地反映实际结果。

从表 2 中可以总结以下特点：

(1)路由表中存在相当数量的后继节点路由。200×200 的网格中后继节点所占比例最大，接近 40%。

(2)绝大多数节点的后继节点数目都不大于 4。具有 4 个以上后继节点的路由所占比例都是很小的，其中，25×25 和 50×50 的网格中所占比例最小。

通过表 1 和表 2 分析计算，随着故障块占比由 10%增大到 80%，在网格规模为 25×25 时，本文算法路由长度平均减少约 49.11%，最大超过了 70%；网格规模为 50×50 时，平均减少约 48.31%；网格规模为 100×100 时，平均减少约 49.27%，最大减少为 70.94%；网格规模为 200×200 时，自适应容错路由表算法得到的路由长度比自适应容错路由算法平均减少约 48.43%。因此，在二维网格中，自适应容错路由表算法的平均路由长度比自适应容错路由算法的平均路由长度减少了大约 50%，最大的减少比例超过了 70%，而且添加路由表后，对于路由表的空间开销几乎可以忽略不计，因此，添加路由表的算法具有重要的现实意义。

6 结束语

本文基于裂痕故障块的二维网格自适应容错路由算法，提出一种自适应容错路由表算法。使用路由表存储相关节点的后继节点，从而可以获得最短的路由路径，有效地减少了平均路由长度。实验结果表明，本文算法在路由表空间方面的开销很小，今后将把基于裂痕故障块的自适应容错路由表算法应用到无线传感器网络中，重点研究其负载平衡问题。

参考文献

[1] Chou W, Bragg A W, Nilsson A A. The Need for Adaptive Routing in the Chaotic and Unbalanced Environment[J]. IEEE Transactions on Communications, 1981, 29(4): 481-490.

[2] 陈庆强, 罗兴国, 张 帆, 等. 基于故障节点再利用的细粒度 NoC 容错路由算法[J]. 计算机应用研究, 2012, 29(7): 2586-2617.

[3] 胥大成, 樊建席, 张书奎. 基于 2D-Mesh 的容错路由算法[J]. 计算机科学, 2012, 39(3): 113-137.

[4] 段新明, 武继刚, 张大坤. Torus 网络自适应容错路由算法[J]. 计算机科学, 2012, 39(2): 115-117.

[5] 陈美润. 强定向图的强距离及网格的容错自适应路由[D]. 厦门: 厦门大学, 2009.

[6] Lillevik S L. The Touchstone 30 Gigaflop Delta Prototype[C]// Proceedings of the 6th Distributed Memory Computing Conference. [S. l.]: IEEE Press, 1991.

[7] 杨 裔. 图论在计算机和无线传感器网络中的应用[D]. 兰州: 兰州大学, 2009.

[8] Shamaei A, Sarbazi-Azad H. An Adaptive and Fault-tolerant Routing Algorithm for Meshes[C]//Proceedings of International Conference on Computational Science and Its Applications. Berlin, Germany: [s. n.], 2008.

[9] Chalasani S, Boppana R V. Fault-tolerant Wormhole Routing Algorithms for Mesh Networks[J]. IEEE Transactions on Computers, 1995, 44(7): 848-864.

[10] Su Chien-Chun, Shin G K. Adaptive Fault-tolerant Deadlock-free Routing in Meshes and Hypercubes[J]. IEEE Transactions on Computers, 1996, 45(6): 666-683.

[11] Chien A W, Kim J H. Planar-adaptive Routing: Low-cost Adaptive Networks for Multiprocessors[J]. Journal of the ACM, 1995, 42(1): 92-123.

[12] Xiang Dong, Zhang Yueli, Pan Yi. Practical Deadlock-free Fault-tolerant Routing in Meshes Based on the Planar Network Fault Model[J]. IEEE Transactions on Computers, 2008, 58(5): 620-633.

[13] Wu Jie. A Fault-tolerant Adaptive and Minimal Routing Approach in 3-D Meshes[C]//Proceedings of the 7th International Conference on Parallel and Distributed Systems. Washington D. C., USA: IEEE Press, 2000.

[14] Daneshtalab M, Ebrahimi M, Xu T C, et al. A Generic Adaptive Path-based Routing Method for MPSoCs[J]. Journal of Systems Architecture, 2011, 57(1): 109-120.

编辑 陆燕菲