Chapter 4

Fault-Tolerant Node-to-Set Disjoint-Path Routing

In this chapter we present methodology based on the use of disjoint paths to provide fault tolerance in the presence of components failures. The objective of the proposed method is to provide an alternative disjoint path which is non-faulty, when failure occurs in the path.

Data transmission is the critical issue when dealing with parallel processing systems. As the number of processors increases day by day inside the supercomputers, faults are very common to occur. Therefore, generating node-disjoint paths is the guarantee to sending message from source process to destination process successfully, since single faulty process can affect maximum one path. Beyond selecting right and efficient interconnection network, routing is also a major issue to propagate the data in a efficient way. This chapter presents a fault tolerant routing algorithm for faulty hypercube networks which finds n disjoint paths from source process s to n destination processes in n-dimensional hypercube in $O(n^2)$ time with optimal path lengths at most $n + f + 1$, where n is the number of destination node and f is the number of faulty nodes. The simulation results showed that the proposed algorithm reduce the average path length by about 20% in comparison of Bossard's algorithm in 8-dimensional hypercube (H_8).

Some general properties of hypercube and related works is described in section 4.1. Section 4.2 describes the importance of subcubes for reliable computation. In section 4.3, the node-to-set routing algorithm is described. The correctness and the complexity of the algorithm are studied and a formal proof is defined in section 4.4. Section 4.5 provide the evaluation model of the contribution. Performance analysis of the algorithm is showed in section 4.6. Finally, the proposal is summarized and discussed in section 4.7.

4.1 Overview

Parallel processing is a very attractive research topic because today's computer systems equipped with various CPU cores. Personal computers consists few number of cores (currently eight), but supercomputers consists hundreds of thousands of cores. Recently, Tianhe-2 (MilkyWay-2) supercomputer developed by China's National University of Defense Technology (NUDT) connects 3,120,000 cores with a performance of 33.86 petaflops [1]. Hence, retaining high performance computing (HPC) in the presence of faults when routing the message from source node s to d destination nodes is a critical issue.

Hypercube topology is one of the most popular interconnection network of parallel systems due to its elegant properties like simplicity, regularity, high symmetric, maximal fault-tolerance, and strong hierarchical structure. An n-dimensional hypercube H_n, also called n-cube or binary n-cube, has 2^n nodes and $n.2^{n-1}$ links and each nodes represented by n-bit unique binary address. Two nodes are connected by a link in a hypercube if their Hamming distance is equal to one. Hence, the degree and diameter of hypercube is equal to n [107]. For any dimension $(0 \leq i \leq n-1)$, an n-dimensional H_n consists of two $(n-1)$-dimensional subcubes H^0_{n-1} and H^1_{n-1}, whose most significant bit (MSB) position or i^{th} bit is set to 0 and 1, respectively. Figure 2.9 shows a 4-dimensional hypercube H_4 and its subcubes H^0_3 and H^1_3, whose MSB position or 3^{rd} bit is set to 0 and 1, respectively.

In a hypercube, there may be faulty nodes and/or faulty links. In this case the hypercube is called faulty hypercube. If non-faulty nodes and links exist in n-cube, it is called a complete cube. A cube is called a subcube of complete n-cube if at least one dependent (0 or 1) coordinate exists in the cube with dimension n. A cube H_j is the subcube of cube H_i if $H_i \otimes H_j = H_j$. A subcube H_i is called a maximal subcube, if subcube H_i is not a subcube of any other subcubes. Otherwise the subcube H_i is called nonmaximal. A subcube H_f is called complete faulty subcube if H_f is a subcube including only faulty nodes and/or links. In complete n-cube $W(r), r$-subcubes exist. Equation 4.1 determined the number of all subcubes in complete n-cube:

$$W(r) = (0 \leq r \leq n) = \sum_{r=0}^{n} 2^{n-r} \begin{bmatrix} n \\ r \end{bmatrix} \tag{4.1}$$

Naturally, a set of the maximal subcubes, nonevidently includes nonmaximal subcubes. Thus, in order to define the desired subcube, it is necessary and sufficient to have a set of maximal subcubes.

The interesting properties of hypercube topology are very much useful for parallel computing and therefore, motivated to researchers to solve classical problems e.g., image and signal processing [23], traveling salesman problem (TSP) [17], finding node-disjoint shortest paths [72], node-to-set disjoint path routing [15]. In a parallel systems, when the size of the network grows, the probability of node or link faults occur more frequently. So, maintaining reliability of hypercube based parallel systems is highly desirable. Several fault-tolerant routing strategies for any interconnection network has been proposed to resolve the faults when transmitting the data. There are many variants of the hypercube topology like crossed cubes [20], twisted cubes [21], folded cubes [45] and hierarchical cubes [86]. Hence, all these variants of hypercube also increase the importance of hypercube routing.

Creating node-disjoint paths has advantages in fault tolerance routing. The node-to-set disjoint path routing problem for any interconnection network is as follows: given a source node s and a set of n destination nodes, to find the n node disjoint paths from source node to each distinct destination nodes. The two paths are disjoint if they do not have common node except the source node. The node-to-set node disjoint path routing has the ability to tolerant the faults for multicasting communication. There are many algorithms for node-to-set routing problem [16, 55, 79]. These algorithms are very much useful for fault-tolerant routing.

In 1998, Gu and Peng [55] described a node-to-set fault-tolerant routing algorithm for hypercube in $O(|F|n)$ time, where F is the set of faulty nodes, with path length at most $n+2$. In 2010, Bossard et al. [16] proposed an extended fault-tolerant node-to-set disjoint-path routing algorithm, that finds paths of length at most $n+k+4$ in $O(n^2)$ time in hypercubes, where k is the number of destination nodes. In 2012, Lai [72] proposed an optimal all shortest node-disjoint paths in hypercube in $O(mn^{1.5}+m^3n)$

time with minimized length in the worst case, where m is the number of destination nodes. Recently, in 2014, Bossard and Kaneko [15] improves the time complexity by $O(kn)$, where $(k \leq n)$ with optimal path length (at most $n+1$) for node-to-set disjoint paths routing in hypercubes. However, no distributed algorithm for data routing over node-disjoint paths in a hypercube topology exists. In this chapter, we propose a fault-tolerant routing algorithm in hypercube in $O(n^2)$ time with maximum $n+f+1$ path length.

4.2 Subcube Reliability Computation

According to the reliability concept with respect to subcubes, it can be computed in terms of the number of disjoint subcubes in hypercube H_n. Subcubes can be embedded in an n-cube in the presence of node and/or link failures [108]. In the presence of any type of failures, $(n-1)$-dimensional subcube embedded in an n-dimensional cube. Considering only node failures in the system, then one can embedded a fault-free $(n-1)$-subcube in an n-cube. In the case of one faulty node in an n-cube, we can split it into a fault-free $(n-1)$-subcube. But, two faulty nodes could damage all $(n-1)$-subcubes of an n-cube.

Example 4.2.1. In an n-dimensional hypercube, if node $0 = 00\ldots00$ and node $N-1 = 11\ldots11$ are faulty, then there is no mode of embedding a $(n-1)$-subcube. If all failures occur in H_n such a way that they can be enclosed in k-subcube with condition $k < n$, then a fault-free $(n-1)$-subcube exists. In a 4-dimensional cube, Figure 4.1 and Figure 4.2 illustrates the case when embedding is not possible since no fault-free 3-cube exists and Figure 4.3 and Figure 4.4 illustrates the case when embedding is possible since fault-free 3-cube exists.

4.3 Algorithm

In this section, we propose an algorithm NoSeRo (Node to Set Routing) which finds disjoint paths from one source node s to n destination nodes inside an n-dimensional hypercube. Suppose a set of destination nodes be $D = \{d_1, d_2, \ldots, d_n\}$ and F is a set of faulty nodes. Due to symmetric structure of hypercubes, we can always take the source node s is $00\ldots0$ without loss of generality.

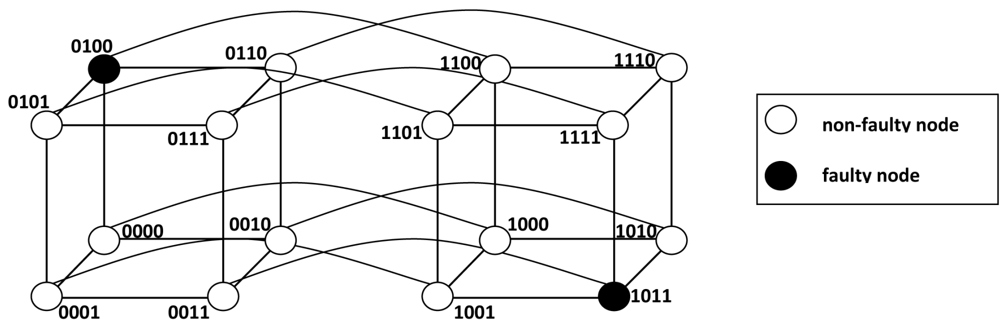


Figure 4.1: Embedding is not possible in a faulty 4-cube since no fault-free 3-cube exist.

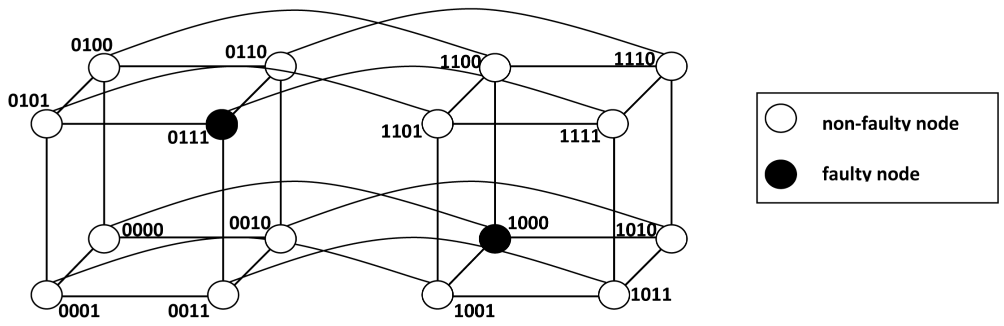


Figure 4.2: Embedding is not possible in a faulty 4-cube since no fault-free 3-cube exist.

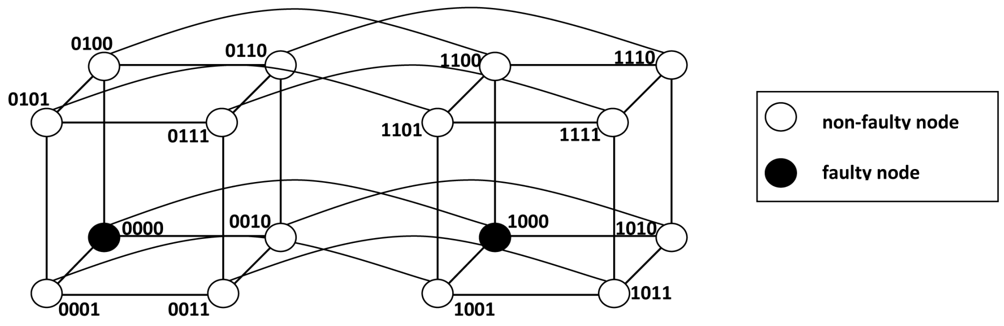


Figure 4.3: Embedding is possible in a faulty 4-cube since fault-free 3-cube exists.

The first subsection 4.3.1 describes the fault-tolerant node-to-node routing in hypercube from s to d of length at most $n+2$ in $O(n)$ time and second subsection 4.3.2 describes the fault-tolerant node-to-set routing in hypercube from s to n destination nodes of length at most $n+f+1$ in $O(n^2)$ time.

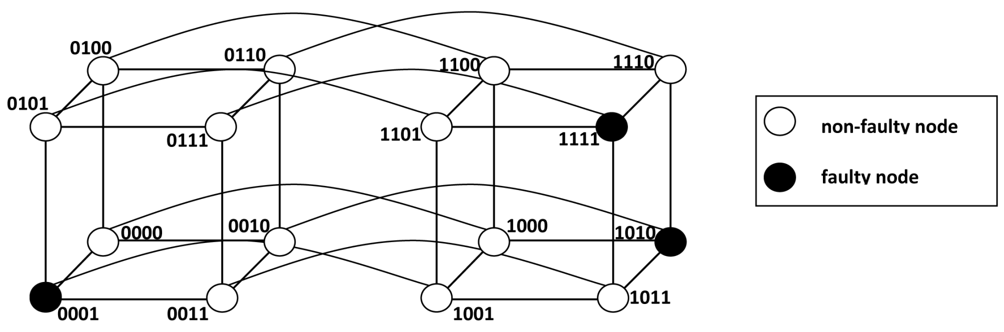


Figure 4.4: Embedding is possible in a faulty 4-cube since fault-free 3-cube exists.

4.3.1 Fault-Tolerant Node-to-Node Routing in Hypercube

Given a source node s, a set of destination nodes D and a set of faulty nodes F, such that $|F| \leq n-1$ and $|D| \leq 2^n - |F|$. We will explain the fault-tolerant node-to-node routing in hypercube with the help of three cases. Before applying the cases, we have to divide the hypercube H_n into two subcube of smaller dimension $n-1$ until each subcube is fault-free.

Case 1- $|F| = 0$

If there is no faulty nodes in the hypercube topology, then apply a shortest-path routing algorithm to find out the path from s to D with nearest destination node.

Case 2- $|D| = 1$

If there is only single destination node in the hypercube topology, then apply fault-tolerant node-to-node routing algorithm to find the path from single source node s to single destination node D.

Case 3- Otherwise

If there are several faulty nodes in the hypercube, then reduce H_n along a dimension into two $(n-1)$-dimensional subcubes H^0_{n-1} and H^1_{n-1} such that

$$F \cap H^1_{n-1} \neq \phi$$

Case 3-1- $H^1_{n-1} \supset D$

Case 3-1-1- $H^0_{n-1} \cap F = \phi$

If there is no faulty node in H^0_{n-1}, then by using one link map one $d \in H^1_{n-1}$ onto a $d' \in H^0_{n-1}$ and if $d' \neq s$, then connect them by applying a shortest-path routing algorithm inside H^0_{n-1}.

Case 3-1-2- Otherwise

If there is faulty node in H^0_{n-1}, then map $s \in H^0_{n-1}$ onto a node $s' \in H^1_{n-1}$ with a fault-free path of length at most 2. If $s' \in D$ we accomplished, otherwise recursively apply this algorithm on H^1_{n-1}.

Case 3-2- $H^0_{n-1} \cap D \neq \phi$

Recursively apply this algorithm on H^0_{n-1}

In this approach, we can always map $s \in H^0_{n-1}$ onto a node $s' \in H^1_{n-1}$ with a fault-free path of length at most 2. Since $|F| < n$, and each time reduction of H_n sets at least one faulty node into H^1_{n-1}. Hence, during a reduction of H_n which is always done in parallel, all time decreasing the dimension by one puts one faulty node into H^1_{n-1}. So, there is at least one fault-free path of length at most 2 to map s onto s' in H^1_{n-1}.

Finding a suitable dimension which fulfils all the above condition can be exercised in $O(n)$ time with path length at most $n+2$. Hence, the time complexity of fault-tolerant node-to-node algorithm is $O(n)$ with the path length at most $n + \frac{|F|}{2} + 2$.

4.3.2 The Proposed Node-to-Set Routing Algorithm

In this section, we propose an algorithm NoSeRo that finds disjoint paths connecting with one common source node s and n distinct destination nodes in an n-dimensional hypercubes H_n. Let source node be $s \in H_n$, set of destination nodes be $D = \{d_1, d_2, \ldots, d_n\}$, and set of faulty nodes $F = \{f_1, f_2, \ldots, f_{n-1}\}$.

This algorithm uses the symmetric and recursive properties of the hypercube H_n to perform node-to-set fault-tolerant node-disjoint path routing. By using recursive property, the H_n can be split in arbitrary dimension i $(0 \leq i \leq n-1)$ into two subcubes of lower dimension, called H^0_{n-1} and H^1_{n-1}. Subcube H^0_{n-1} contains the source node s and s' is the set of neighbour nodes of s. For Algorithm 3, the inputs are single common source node, multiple destination nodes, number of faulty nodes and output is node-disjoint paths from source to destination with optimal path lengths. NoSeRo algorithm have several distinguished cases.

Case 1: If $D = 1$ and $F = \phi$, propagate the message by flipping the corresponding bit position according to $HD(s, d)$, which produce optimized path.

Case 2: If $F \geq 1$, split H_n into H^0_{n-1} and H^1_{n-1} along an arbitrary dimension i $(0 \leq i \leq n-1)$.

Case 3: If H^1_{n-1} has at least one faulty node, then two cases exists

Algorithm 3 NoSeRo $(H_n, D = \{d_1, d_2, \ldots, d_n\}, F = \{f_1, f_2, \ldots, f_{n-1}\})$

```
1: if $D = 1$ and $F = \phi$ then $i = (i + 1)\ mod\ n$
2:     Propagate the message to next processor through $HD(s, d)$
3: end if
4: Split $H_n$ to $H^0_{n-1}$ and $H^1_{n-1} | H^0_{n-1} \cap F = \phi$
5: if $\exists\ s'$ then
6:     Route the message from $s$ to $D$ via $s'$
7: else
8:     for all $d_i | D \cap H^1_{n-1}$ do
9:         Route the message from the destination node $H^1_{n-1}$ to another destination
           node in $H^0_{n-1}$
10:    end for
11: end if
```

Case 3.1: If s' in H^1_{n-1} is a non-faulty node, route the message from s to s' then one destination node of H^1_{n-1} and finally map back into H^0_{n-1} all the destination nodes of H^1_{n-1} remaining.

Case 3.2: If s' in H^1_{n-1} is a faulty node, then perform a back-map to route all the destination nodes of H^1_{n-1} back into H^0_{n-1} with maximum 1 link.

After finishing these cases, recursively call NoSeRo algorithm in subcube H^0_{n-1} for routing all the destination nodes.

Example 4.3.1. Let Figure 4.5 for 5-dimensional hypercube. For $n = 5$, this example perform the routing according to the NoSeRo algorithm in a hypercube H_5. Let the source s, the set of destination nodes and the set of faulty nodes are as follow:

$s = 00000$

$D = \{d_1 = 01011, d_2 = 10100, d_3 = 10111\}$

$F = \{f_1 = 10010, f_2 = 10101\}$

First of all, H_5 is partitioned by dimension 4 into two subcubes H^0 and H^1. H^0 has $\{d_1\}$ and does not contain any faulty nodes; i.e., H^0 is fault free hypercube. Therefore, we can directly apply unicast algorithm on H^0 and find out the optimal path from s to $\{d_1\}$. One can find out optimal path by evaluating the Hamming distance between source node to destination node; i.e., $HD(0000, 01011) = 01011$. By altering the bit-position of $0, 1$ and 3 in the source node, the optimal path is given below–

$00000 \xrightarrow{0} 00001 \xrightarrow{1} 00011 \xrightarrow{3} 01011$

In this way, we find out the path from s to d_1.

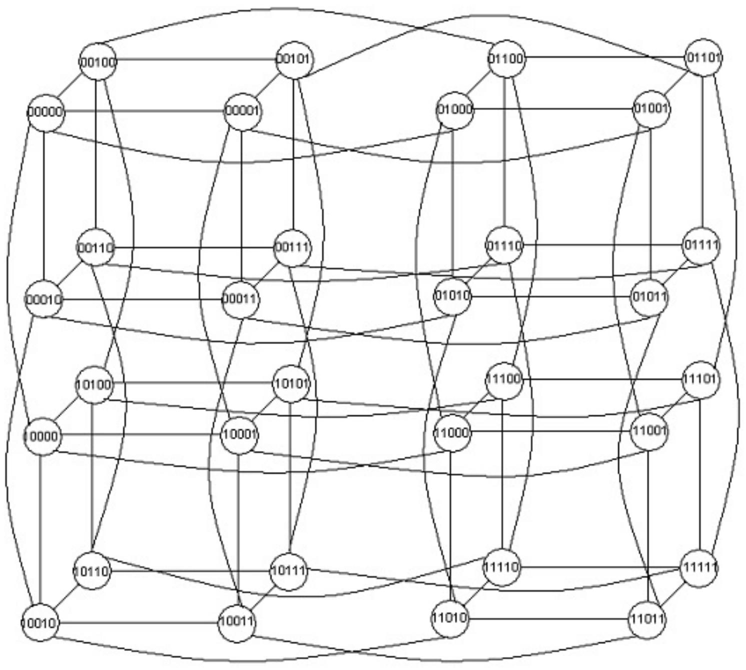


Figure 4.5: A 5-dimensional hypercube.

H^1 contains $\{d_2, d_3\}$ and has $\{f_1, f_2\}$. Since s' is a non-faulty node in H^1, a routing path is constructed from s via s' to d_2. The path is given below–

$00000 \xrightarrow{4} 10000 \xrightarrow{2} 10100$

In this way, we find out the path from s to d_2 and d_3 is mapped to $d_3{}'(00110)$ in H^0 via 10110. The map back path is given below–

$10111 \xrightarrow{0} 10110 \xrightarrow{4} 00110$

Secondly, H_4 is partitioned by dimension 3 into two subcubes H^0 and H^1. H^0 contains $\{d_2, d_3\}$ and $\{f_1, f_2\}$. This time H^1 does not contain any destination and/or faulty nodes; i.e., H^1 is fault free hypercube. Since s' is a non-faulty node in H^0, a routing path is constructed from s via s' to d_3. The path is given below–

$00000 \xrightarrow{0} 00001 \xrightarrow{4} 10111$

4.4 Complexities Analysis

As we know that a hypercube H_n reduction uses just one bit of the n bits in node addresses. For each reduction performed, the described NoSeRo algorithm always set at least one faulty node inside reduced hypercube before recursive call. In this way, the proposed NoSeRo algorithm decreases the number of faulty nodes for the next reductions by at least one. Therefore, the NoSeRo algorithm will perform maximum F hypercube reductions.

For one reduction, $|F|$ decreases by at least one. Therefore, $|D|$ may also decreases. For the initial condition $|D| + |F| \leq n$ and $|F| \leq n$, we assure that the n bits of node addresses sufficient to build the $|D|$ disjoint paths. In other words, n reductions is sufficient to build the $|D|$ disjoint paths.

Now we analyse the time complexity and maximal path length of proposed NoSeRo algorithm.

Let $T(n, n)$ denotes the time complexity of NoSeRo algorithm in H_n with $n = D$. Suppose $n_0 = D \cap H^0_{n-1}$ and $n_1 = D \cap H^1_{n-1}$. From step 1 to 3 in NoSeRo algorithm (when $D = 1$), then algorithm generates single shortest path routing. In this case the time complexity be $T(1, n) = O(n_1)$. From step 5 to 11 of the Algorithm 3 requires $T(n_1, n-1)$ time for H^1_{n-1} subcube and another subcube H^0_{n-1} also requires $T(n_0, n-1)$. Therefore, total time complexity induced by all the cases have:

$$\begin{aligned} T(1, n) &= O(n) \\ T(n, n) &= T(n_0, n-1) + T(n_1, n-1) + O(n_1) \\ &= O(n^2) \end{aligned} \tag{4.2}$$

Where $1 \leq n_0, n_1 \leq n - 1$ and $n_0 + n_1 = n$. So, the total time complexity of NoSeRo algorithm is $O(n^2)$.

We can also understand directly from the NoSeRo algorithm that each subcube required $O(n)$ time complexity to pass the message from single source node to all destinations nodes, so both subcubes or an n-dimensional hypercube required $O(n^2)$ time complexity.

An n-dimensional hypercube H_n can tolerate maximum $n - 1$ faulty nodes $f(f \leq n - 1)$. Hence, there exists $m(m \leq n - f)$ non-faulty nodes in H^1_{n-1} to H^0_{n-1} for node-disjoint paths of length at most 2. So, the destination nodes can be explored back with at most $2 * f$ times. Suppose subcube H_m of H_n has single destination node, then the path length will be at most $m(m = n - f)$ for generating unicast path. If subcube H_m of H_n has multiple destination nodes, then the path length will be at most

$n-f+1$. Therefore, in H_n with f faulty nodes and multiple destinations, the maximal path length

$$\begin{aligned} P_{max} &= max\{2f+(n-f), 2f+(n-f+1)\} \\ &= max\{n+f, n+f+1\} \\ &= n+f+1 \end{aligned} \tag{4.3}$$

From this discussion, we state the main results of this chapter in Theorem 4.4.1.

Theorem 4.4.1. Given a source node s, a set of n distinct destination nodes D and a set of faulty nodes F, one can find n node-disjoint paths from s to D of optimal path lengths at most $n+f+1$ in $O(n^2)$ time complexity in an n-dimensional hypercube H_n.

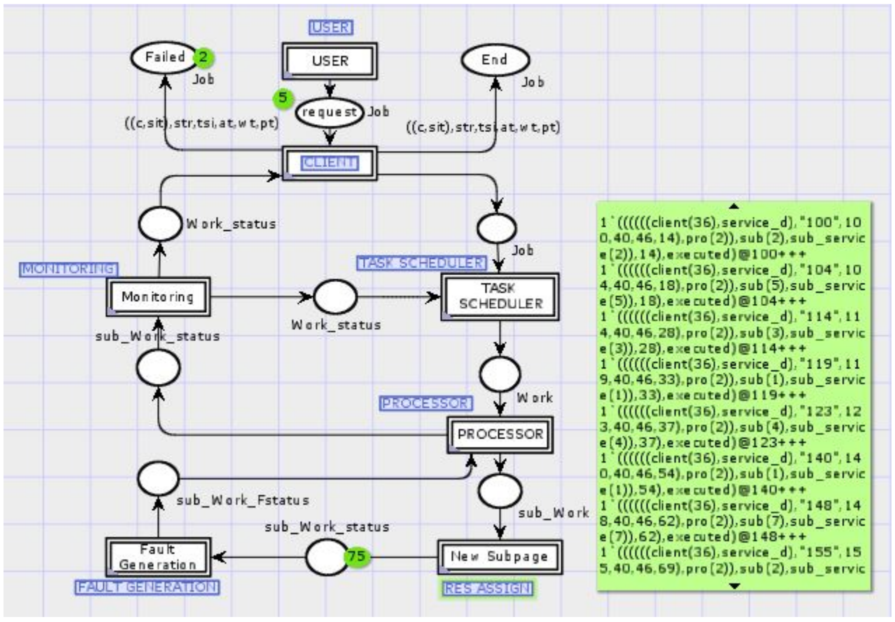
4.5 Evaluation Model

We have simulated our work on CPN (Colored Petri nets) tool to validate the performance of the improved algorithm in hypercube topology. Figure 4.6 shows the simulator's configuration. Firstly, service requests are sent to the client from the users, then the service requests are transferred to the scheduler. The scheduler receives the service request and decides its sequence according to the schedule rules and dispatches these results to processor and further it decides which one processor to execute the service request. The main purpose of our algorithm is to realize fault-tolerance. Our simulations are based on exponential distribution of node failures, i.e., every node has an continuous and independent failure probability.

Figure 4.10 represents the monitoring of service where the complete service having no fault is sent to client and then to user. If there is any fault in the execution of service, then service is sent to task scheduler from where it is either rescheduled or is aborted.

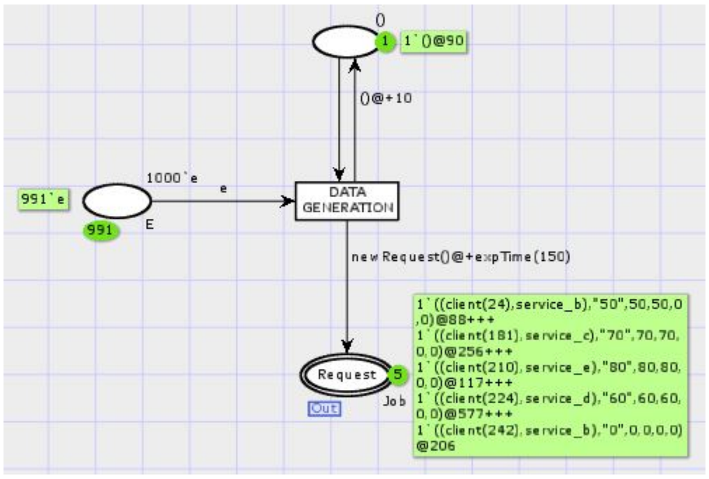
4.6 Performance Evaluation

The algorithm has been tried statically for hypercubes of various dimensions. We have implemented this algorithm using the CPN tools running over 32-bit Windows operating system with core i7 and 4 GB RAM for various hypercube dimensions and injected faults at various nodes and showed that the algorithm provides correct



```
1`((((((client(36),service_d),"100",10
0,40,46,14),pro(2)),sub(2),sub_servic
e(2)),14),executed)@100+++
1`((((((client(36),service_d),"104",10
4,40,46,18),pro(2)),sub(5),sub_servic
e(5)),18),executed)@104+++
1`((((((client(36),service_d),"114",11
4,40,46,28),pro(2)),sub(3),sub_servic
e(3)),28),executed)@114+++
1`((((((client(36),service_d),"119",11
9,40,46,33),pro(2)),sub(1),sub_servic
e(1)),33),executed)@119+++
1`((((((client(36),service_d),"123",12
3,40,46,37),pro(2)),sub(4),sub_servic
e(4)),37),executed)@123+++
1`((((((client(36),service_d),"140",14
0,40,46,54),pro(2)),sub(1),sub_servic
e(1)),54),executed)@140+++
1`((((((client(36),service_d),"148",14
8,40,46,62),pro(2)),sub(7),sub_servic
e(7)),62),executed)@148+++
1`((((((client(36),service_d),"155",15
5,40,46,69),pro(2)),sub(2),sub_servic
```

Figure 4.6: CPN model for computing.



```
1`((client(24),service_b),"50",50,50,0
,0)@88+++
1`((client(181),service_c),"70",70,70,
0,0)@256+++
1`((client(210),service_e),"80",80,80,
0,0)@117+++
1`((client(224),service_d),"60",60,60,
0,0)@577+++
1`((client(242),service_b),"0",0,0,0,0)
@206
```

Figure 4.7: CPN model for date generation.

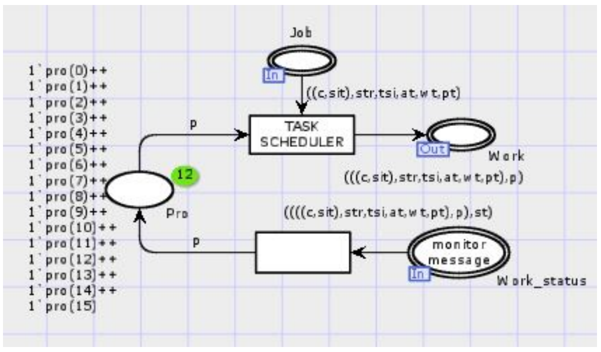


Figure 4.8: CPN model for task scheduler.

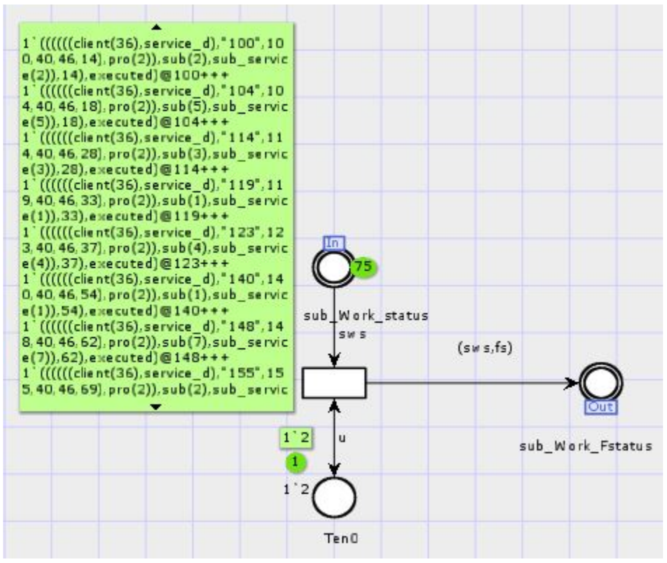

```
1`((((((client(36),service_d),"100",10
0,40,46,14],pro(2)),sub(2),sub_servic
e(2)),14),executed]@100+++
1`((((((client(36),service_d),"104",10
4,40,46,18],pro(2)),sub(5),sub_servic
e(5)),18),executed]@104+++
1`((((((client(36),service_d),"114",11
4,40,46,28],pro(2)),sub(3),sub_servic
e(3)),28),executed]@114+++
1`((((((client(36),service_d),"119",11
9,40,46,33],pro(2)),sub(1),sub_servic
e(1)),33),executed]@119+++
1`((((((client(36),service_d),"123",12
3,40,46,37],pro(2)),sub(4),sub_servic
e(4)),37),executed]@123+++
1`((((((client(36),service_d),"140",14
0,40,46,54],pro(2)),sub(1),sub_servic
e(1)),54),executed]@140+++
1`((((((client(36),service_d),"148",14
8,40,46,62],pro(2)),sub(7),sub_servic
e(7)),62),executed]@148+++
1`((((((client(36),service_d),"155",15
5,40,46,69],pro(2)),sub(2),sub_servic
```

Figure 4.9: CPN model for computing.

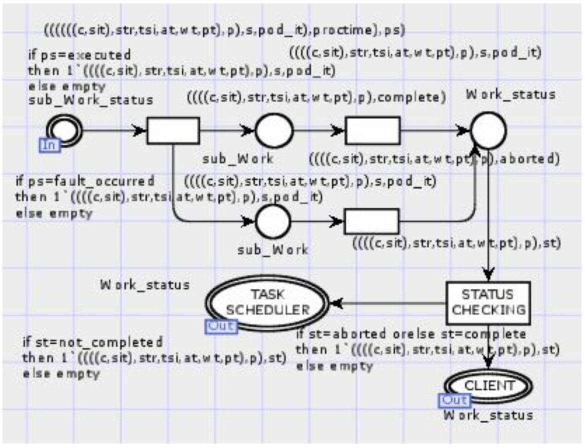

((((((c,sit),str,tsi,at,wt,pt),p),s,pod_it),proctime),ps)
if ps=executed
then 1`((((c,sit),str,tsi,at,wt,pt),p),s,pod_it)
else empty
((((c,sit),str,tsi,at,wt,pt),p),s,pod_it)
sub_Work_status
(((c,sit),str,tsi,at,wt,pt),p),complete)
Work_status
In
sub_Work
((((c,sit),str,tsi,at,wt,pt),p),aborted)
if ps=fault_occurred
then 1`((((c,sit),str,tsi,at,wt,pt),p),s,pod_it)
else empty
((((c,sit),str,tsi,at,wt,pt),p),s,pod_it)
sub_Work
((((c,sit),str,tsi,at,wt,pt),p),st)
Work_status
TASK SCHEDULER
Out
STATUS CHECKING
if st=not_completed
then 1`((((c,sit),str,tsi,at,wt,pt),p),st)
else empty
if st=aborted orelse st=complete
then 1`((((c,sit),str,tsi,at,wt,pt),p),st)
else empty
CLIENT
Out
Work_status

Figure 4.10: CPN model for computing.

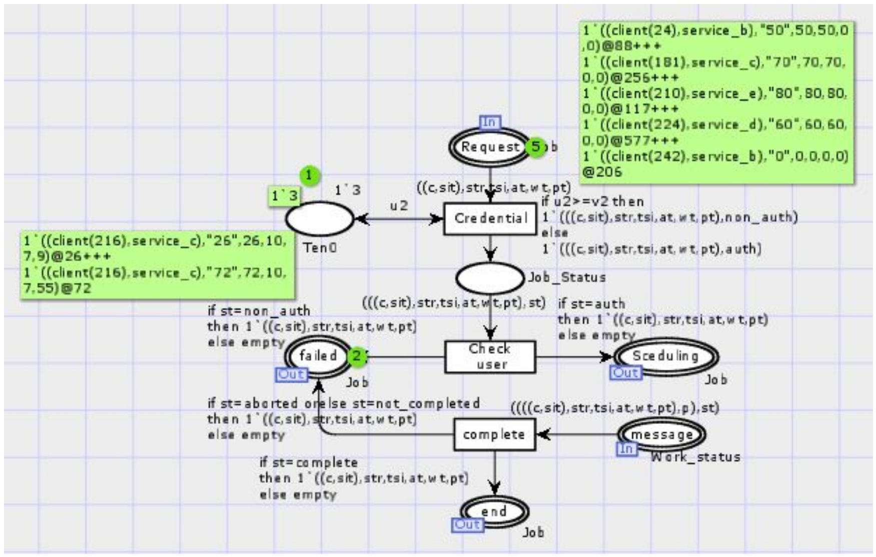

1`((client(24),service_b),"50",50,50,0,0)@88+++
1`((client(181),service_c),"70",70,70,0,0)@256+++
1`((client(210),service_e),"80",80,80,0,0)@117+++
1`((client(224),service_d),"60",60,60,0,0)@577+++
1`((client(242),service_b),"0",0,0,0,0)@206
In
Request
5
Job
1
1`3
1`3
u2
((c,sit),str,tsi,at,wt,pt)
Credential
if u2>=v2 then
1`(((c,sit),str,tsi,at,wt,pt),non_auth)
else
1`(((c,sit),str,tsi,at,wt,pt),auth)
Ten0
1`((client(216),service_c),"26",26,10,7,9)@26+++
1`((client(216),service_c),"72",72,10,7,55)@72
Job_Status
(((c,sit),str,tsi,at,wt,pt), st)
if st=non_auth
then 1`((c,sit),str,tsi,at,wt,pt)
else empty
if st=auth
then 1`((c,sit),str,tsi,at,wt,pt)
else empty
failed
2
Out
Job
Check user
Sceduling
Out
Job
if st=aborted orelse st=not_completed
then 1`((c,sit),str,tsi,at,wt,pt)
else empty
((((c,sit),str,tsi,at,wt,pt),p),st)
complete
message
In
Work_status
if st=complete
then 1`((c,sit),str,tsi,at,wt,pt)
else empty
end
Out
Job

Figure 4.11: CPN model for computing.

routes in the event of failed nodes. We gave performance comparison among Lai's method [72], Bossard's algorithm [16] and the NoSeRo Algorithm 3 on H_8. For practical behaviour, we set $s = 0$, destination nodes $D = (d_1, d_2, \ldots, d_7)$ and faulty nodes $F = (f_1, f_2, \ldots, f_{8-d})$. Then execute the NoSeRo algorithm to get the paths and analyse the collected data for measuring the average paths length on 10,000 times of the hypercube node-to-set disjoint paths routing problem.

Figure 4.12 and Figure 4.13 showed the node-to-set disjoint paths routing results. One can analysed from Figure 4.12 that the path length of NoSeRo algorithm has about 12% better than Bossard's algorithm and about 8% worse than Lai's method in the case of without any faulty nodes (set $F = \phi$). Figure 4.13 showed that the path length of NoSeRo algorithm has about 20% better than Bossard's algorithm while Lai's method can not work in the case of faulty nodes ($F = 8 - d$). So, NoSeRo algorithm effectively generates the n paths which are mutually node-disjoint in both cases in the system.

The node-disjoint property follows from the fact that, we are computing the path from the source node to the n destination nodes and at any stage the hypercube has two subcubes: the 0-subcube and the 1-subcube. Since n-dimensional hypercubes have n links connecting n nodes, so in the presence of failures they choose the alternative paths. Thus, the paths computed at any stage of recursion cannot intersect with each other. The advantageous of node-disjoint paths is if a task tries to move to a faulty

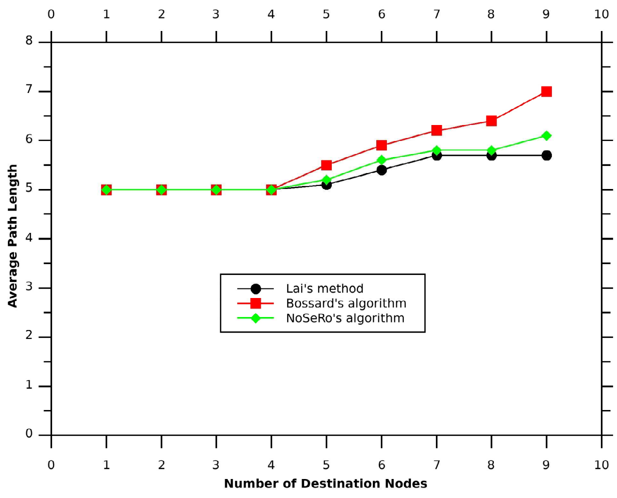


Figure 4.12: Comparison in terms of path length with failures for node-to-set disjoint path routing in H_8

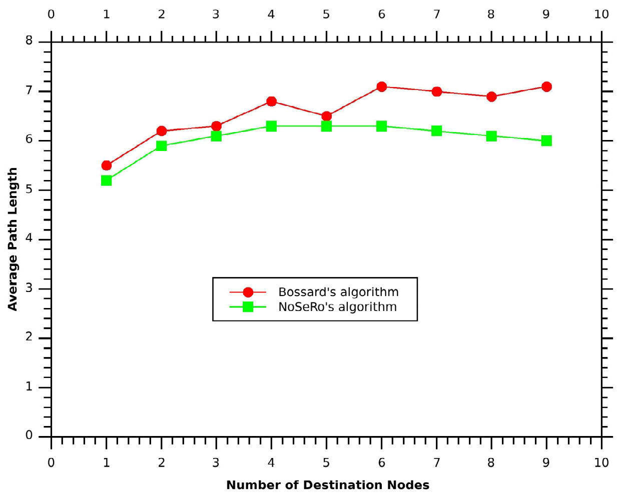


Figure 4.13: Comparison in terms of path length without failures for node-to-set disjoint path routing in H_8

node, it will try an alternative route.

4.7 Conclusion

In this chapter we have described an optimal node-to-set fault-tolerant routing algorithm in hypercubes, which finds disjoint paths from single source node to multiple destinations nodes in $O(n^2)$ time with optimal path length at most $n + f + 1$. Since $N = 2^n$ which can rewrite in this way $n = log_2N$. Thus we can say that this algorithm runs in Logarithmic Time which is log_2^2N. The proposed NoSeRo algorithm can tolerate maximum $n - 1$ faulty nodes. Experimental results showed that applying NoSeRo algorithm approach reduce path length about 20% in H_8.

The Table 4.1 showed that our algorithm is better in terms of average path length and time complexity.

Authors	Year	Path length	Time complexity
A. Bossard, K. Kaneko and S. Peng	2010	$n + k + 4$	$O(n^2)$
A. Bossard and K. Kaneko	2012	$n + 1$	$O(kn)$
C. N. Lai	2012	does not work	$-$
E. Wang	2014	$n + f + 2$	$O(nm)$
Our	2015	$n + f + 1$	$O(n^2)$

Table 4.1: Comparison in terms of path length and time complexity

The proposed method does not need the use of virtual channels hence avoiding scalability problems. The main advantage of the proposed algorithm is that they avoid the faults by accessing resources as per need and independent from the other faults. The NoSeRo algorithm provides the feasible solution to the problem which is useful for current HPC systems. This approach may be applied to the other interconnection networks like Recursive Dual-Net (RDN) for node-to-set disjoint-path routing, since RDN is a newly proposed interconnection network for massive parallel computers.