Generalized Star Crossed Cubeにおける平均パケットレイテンシと耐故障経路探索アルゴリズム

著者	佐藤 智文
出版者	法政大学大学院情報科学研究科
雑誌名	法政大学大学院紀要．情報科学研究科編
巻	14
ページ	1-6
発行年	2019-03-31
URL	http://doi.org/10.15002/00021940

Generalized Star Crossed Cube における平均パケットレイテンシと耐故障経路探索アルゴリズム
The Average Packet Latency and Fault Tolerant Routing Algorithm for Generalized-Star Crossed Cube

佐藤 智文 *
Tomofumi Sato
法政大学大学院　情報科学研究科　情報科学専攻
Email: 17t0012@cis.k.hosei.ac.jp

***Abstract*—In the previous research, we proposed a Generalized-Star Crossed Cube (GSCC) interconnection network, which focuses on the cost reduction and flexibility in network size. In this research, we discuss the topological properties of GSCC, examine the average packet latency, and propose a fault tolerant routing algorithm. Average packet latency is the time a packet travels from the source node to the destination node. Multiple nodes send packets simultaneously, and there are conflicts on the paths. The fault tolerant routing algorithm tries to find a routing path in the system where some nodes and links may be faulty. As a result, the average packet latency for GSCC is better than hypercube and (n, k)-Star Graph when traffic load is low, and the proposed fault tolerant routing algorithm achieves 30 percent better performance than the shortest path routing algorithm.**

1. 導入

近年，世界中で大規模な計算を並列分散で処理できるスーパーコンピュータが AI 分野や Deep Learning など多岐にわたり利用されている． その性能を向上させるために，コンピュータ同士を相互接続する規模は年々増大し続けている．特にコンピュータの性能をランキング形式にした Top500 では，2018 年 11 月現在第 1 位で海外のスーパーコンピュータである Summit は 230 万個を超えるコア数で構成されている．日本でも 2018 年に稼働開始した AI 向けクラウド向け計算システムのスパコンである ABCI (AI Bridging Cloud Infrastructure - AI 橋渡しクラウド) が第 7 位へとランクインして，そのコンピュータの規模は 391,000 個ものコアで構成されている．これらのことから，コンピュータの更なる計算速度向上のためには，コンピュータの数を増加することが必要であるといえる．ここで，並列分散処理をするコンピュータの速度を向上させる方法の 1 つとして，コンピュータ同士の接続方法があげられる．処理速度を高速にするつなぎ方の例として，全てのコンピュータ同士を単純に全接続することである．しかしながら，次数の増大によるリンク数も膨大となり，費用が非常に高くなる問題点がある．一方で，コンピュータ同士をリング状に接続すると費用を抑えることができるが，データ送信する際に経由するコンピュータの数が多くなり処理に時間がかかる問題点がある．これらのことから，これら 2 つの問題点を解消できるグラフ (トポロジ）を設計する必要がある．

以前の研究で新たなトポロジである Generalized-Star Crossed Cube (GSCC(n, k, m)) [1] を提案し，膨大なネットワークサイズでも低次数，小さい直径でかつネットワークサイズに柔軟性があることを示した．今回は，実際にノードの一部である計算ノードからパケットを生成して宛先ノードに送信して実行時間を計測する Average Packet Latency [2] を実装する．そして，既存の研究である Hypercube [3] と Crossed Cube [4]，(n, k)-Star Graph [5]，Generalized-Star Cube [6] と比較し，本トポロジの有意性を検証する．さらに，本研究では，ノードやリンクが故障し使用不可の場合でも迂回して宛先ノードまで経路探索を行える耐故障経路探索アルゴリズムを実装する．正しく宛先ノードへデータを送信できるアルゴリズムを作成し，かつ実行時間をできるだけ抑えることができるアルゴリズムを提案することを目標とする．

* Supervisor: Prof. Yamin Li

2. Generalized-Star Crossed Cube

この節では，Generalized-Star Crossed Cube の性質と，そのトポロジの最短経路探索アルゴリズムについて述べる．

2.1. Generalized-Star Crossed Cube の性質

Generalized-Star Crossed Cube (GSCC(n, k, m)) は，Crossed Cube と (n, k)-Star Graph の積グラフのことであり，Generalized-Star Cube (GSC(n, k, m)) が持つ Hypercube 部分を Crossed Cube に置き換えたものである．積グラフとは，トポロジのノード部分に他のトポロジを埋め込むことで設計することができる．これによって，ノード数は 2 つのトポロジの積で表し，次数と直径は 2 つのトポロジが持つ次数と直径の和で表現することができる．つまり，パラメータを Crossed Cube が持つパラメータ m と (n, k)-Star Graph が持つ 2 つのパラメータ n, k の 3 つを用いることができるため，ネットワークサイズの柔軟性が非常に高く，かつ Hypercube が持つ大きめな直径を削減することができるため，コストを軽減することができる．ここで，パラメータが $n = k$ の場合，Star-crossed cube [7] と同形になる．GSCC のアドレス番号 GSCC$(n, k, m) = (s, u)$ は CQ(m) が持つ m 桁の 2 進数アドレス ($s = \{s_{m-1}s_{m-2}...s_1s_0\}, s_i \in \{0, 1\}$) と，$(n, k)$-Star Graph が持つ k 桁の 1 から n までの 10 進数アドレス ($u = \{u_0u_1...u_{k-1}\}, u_j \in \{1, 2, ..., n\}$) で表すことができる．このグラフの特徴として，任意のネットワークのサイズが設計でき柔軟性が高い点や，直径が小さいことによるコストパフォーマンスがよくなる点，最短経路探索アルゴリズムが比較的容易な点などが挙げられる．表 1 では，Crossed Cube と GSC，GSCC の位相幾何学的性質を示したものである．GSC と比較すると，GSCC は Hypercube が持つ直径 m が Crossed Cube が持つ直径 $\lceil (m + 1)/2 \rceil$ とおよそ半分になることによりその分だけ直径が小さくなる．

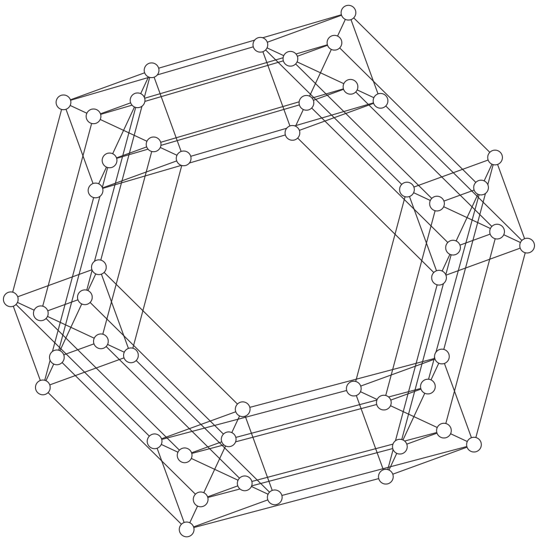

図 1. CQ(3) x NK-Star(3,2)

表 1. 位相幾何学的性質

トポロジ	CQ(m)	GSC(n,k,m)	GSCC(n,k,m)
ノード数	2^m	$2^m \times \frac{n!}{(n-k)!}$	$2^m \times \frac{n!}{(n-k)!}$
次数	m	$m+n-1$	$m+n-1$
直径	$\lceil \frac{m+1}{2} \rceil$	$m+2k-1$ (if $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$) $m+k+\lfloor \frac{n-1}{2} \rfloor$ (if $\lfloor \frac{n}{2} \rfloor < k \leq n-1$)	$\lceil \frac{m+1}{2} \rceil + 2k - 1$ (if $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$) $\lceil \frac{m+1}{2} \rceil + k + \lfloor \frac{n-1}{2} \rfloor$ (if $\lfloor \frac{n}{2} \rfloor < k \leq n-1$)

2.2. 最短経路探索アルゴリズム

この小節では，パケット送信などに使われる経路探索アルゴリズムについて述べる．最短経路アルゴリズムは次の Algorithm 1 ～ 4 からなる．このアルゴリズムは Crossed Cube 部分を探索したあと，(n,k)-Star Graph 部分を探索するものになる．Algorithm 1 は Crossed Cube 部分の最短経路探索アルゴリズム [8] である．アルゴリズムについて説明する前に Crossed Cube の隣接ノードのアドレス条件について紹介する．例として，8 次元 Crossed Cube のアドレス CQ(8) = 01101100 で最上位ビットを変化させるとき，最下位ビットから変化させるビットまでの 2 ビットずつ値を習得する．その 2 ビットが次の R ($R = (00,00),(10,10),(01,11),(11,01)$) によってアドレスが決まる．これを Crossed Cube のペア関係といい，00 または 10 であった場合変化されず，01 または 11 であった場合上位ビットも反転させる．つまり先程与えられた CQ(8) の最上位ビットを変化させたときの隣接ノードのアドレスは次のようになる．

$$(0(1)(10)(11)(00)) \rightarrow (1(1)(10)(\underline{0}1)(00))$$

ここで，最短経路の条件として ρ を定義する．ρ はペア関係の選択される回数を表し, 数値分回選択される．ρ を定義する前に i^* を定義する．まず，l を s と t で異なる最大ビットとする．次に $i^* = \lfloor l/2 \rfloor$ と定義する．これは，Crossed Cube の隣接ノードによるビットの変化が 2 ビットずつで決定するために必要になる．$j \geq i^*$ のとき，ρ と s,t の関係は式 (1)，式 (2) のようになる．i^* より大きい場合，ビット反転する必要がないため 0 となる．また，$j = i^*$ のときはペア関係でないため両方のビットが違う場合 2 となり，片方のみのビットが異なる場合 1 となる．

$$\rho_{i^*}(s,t) = 0 \quad (j \geq i^*+1) \tag{1}$$

$$\rho_j(s,t) = \begin{cases} 2 & (s_{i^*+1}s_{i^*} = \bar{t}_{i^*+1}\bar{t}_{i^*}) \\ 1 & (Otherwise) \end{cases} \tag{2}$$

Algorithm 1 CQ_Part(s,t)

Require: CQ(m) of Source node s
Require: CQ(m) of Destination node t
Ensure: Routing order of S
  define $\rho_i(s,t)$ for all $0 \leq i \leq \lfloor \frac{m}{2} \rfloor$,
  $Q \leftarrow \{j | \rho_j(s,t) \neq 0\}$,
  $T \leftarrow \{\rho_j(s,t) \neq 0, j < i^*$ and
    $\bar{s}_{2j+1}s_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}$ or $s_{2j+1}\bar{s}_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}\}$
  if $T \neq \phi$ **then**   /***** Step1 *****/
    find a $j \in T$ and call *ONE_STEP_ROUTE*(i,Q)
  end if
  while $Q \neq \phi$ **do**   /***** Step2 *****/
    if either $\bar{s}_{2i+1}s_{2i} \overset{\text{d.-p.}}{\sim} t_{2i+1}t_{2i}$ or $s_{2i+1}\bar{s}_{2i} \overset{\text{d.-p.}}{\sim} t_{2i+1}t_{2i}$ holds for some $i \in Q$ **then**
      choose such smallest i
    else
      $i \leftarrow max\{j | j \in Q\}$
    end if
  end while
  return S

Algorithm 2 ONE_STEP_ROUTE(j,Q)

  if $\rho_j(s,t) = 2$ **then**
    route to s', i.e., the $2j$th or $(2j+1)$th neighbor of s
    $\rho_j(s,t) \leftarrow \rho_j(s,t) - 1$;
  else
    if $\bar{s}_{2j+1}s_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}$ **then**
      route to s' the $(2j+1)$th of s
    else if $s_{2j+1}\bar{s}_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}$ **then**
      route to s' the $2j$th of s
    end if
    $Q \leftarrow Q - \{j\}$
    $\rho_j \leftarrow \rho_j(s,t) - 1$
  end if
  $S \leftarrow S + \{s'\}$
  $s \leftarrow s'$

次に，$j < i^*$ のときを考える．Crossed Cube は上記の通り上位ビットを変化すると下位ビットも条件により変化する．そのために，下の 3 つ条件のいずれかを満たす s と t を距離維持ペア関係 (distance-preserve pair related) が成り立つとして，その式を $s_{2j+1}s_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}$ のように表す．これにより距離維持ペア関係が成り立つ部分のビット部分を変化させないようにする．また，$j < i^*$ のときの ρ と s,t の関係は式 (3) のようになる．

1) $(s_{2j+1}s_{2j}, t_{2j+1}t_{2j}) \in \{(01,01),(11,11)\}$ かつ $\sum_{i=j+1}^{\lfloor \frac{m-1}{2} \rfloor} \rho_i(s,t)$ が偶数
2) $(s_{2j+1}s_{2j}, t_{2j+1}t_{2j}) \in \{(01,11),(11,01)\}$ かつ $\sum_{i=j+1}^{\lfloor \frac{m-1}{2} \rfloor} \rho_i(s,t)$ が奇数
3) $(s_{2j+1}s_{2j}, t_{2j+1}t_{2j}) \in \{(00,00),(10,10)\}$

Algorithm 3 NKStar_Part(u, v)

Require: (n,k)-Star of Source node u
Require: (n,k)-Star of Destination node v
  while $u \neq v$ **do**
    if $u_0 = v_0$ **then**
      find minimum j that satisfies $u_j \neq v_j$
      swap u_0 for u_j
    end if
    if $u_0 = v_j$ **then**
      swap u_0 for u_j
    else
      find min. value min of v that satisfies $u \not\subset v$
      $u_0 \leftarrow min$
    end if
    $S \leftarrow S + \{u\}$
  end while
  return S

Algorithm 4 Shortest Path Routing Algorithm

Require: CQ(m), (n,k)-Star of Source node s，u
Require: CQ(m), (n,k)-Star of Destination node t，v
Ensure: Routing order of S
  $C \leftarrow$ CQ_Part(s, t)
  $N \leftarrow$ nkStar_Part(u, v)
  $S \leftarrow C + N$

$$\rho_j(s,t) = \begin{cases} 0 & (s_{2j+1}s_{2j} \overset{\text{d.-p.}}{\sim} t_{2j+1}t_{2j}) \\ 1 & (Otherwise) \end{cases} \tag{3}$$

また，(n,k)-Star Graph 部分の隣接ノードは，先頭のアドレスだけが異なる，あるいは先頭と先頭を除いたアドレスの長さの 1 つにあるものが交換されたものである．経路探索の流れとして，先頭のアドレスが異なる場合，それが宛先ノードのアドレスにある場合それと交換する．宛先ノードのアドレスに無い場合は，宛先ノードに存在して送信元ノードにない最小の値を入れる．またアドレスの先頭が同じであるが他のアドレス番号が異なる場合，先頭とその異なる部分の中から最も左にあるものと入れ替える．これらのことををまとめた (n,k)-Star Graph 部分の最短経路アルゴリズムは次の Algorithm 3 に示す．また，2 つの経路を合わせた Generalized-Star Crossed Cube の探索アルゴリズムを Algorithm 4 に示す．

3. Average Packet Latency

この節では，Average Packet Latency について説明し，それの実験方法について述べる．

3.1. Average Packet Latency について

各ルータに図 2 のような送信されてきたパケットを管理して隣接ノードにパケットを送信するためのシステムを導入する．他ノードの計算ノードから作られたパケットが複数の隣接ノードから送信され，入力として pi から Packet FIFO に入れられる．Packet FIFO はリンクの個数分と 1 つの計算ノードからパケットを入れるものがあり，1 つのリンクごとに 1 つの FIFO で管理される．一定時間ごとにすべての FIFO から 1 つ宛先ノードなどをの情報を Switch Controller で処理させる．ここで，FIFO の中身が空ならば何も処理しないようにする．Switch controller は FIFO の先頭のパケットの宛先ノードを確認して，送信先には経路探索アルゴリズム

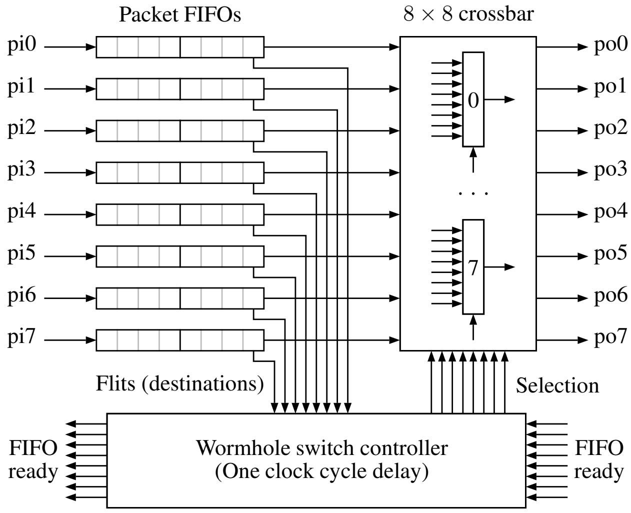

図 2. Router Block Diagram

を用いて次に送信する隣接ノードを決定する．送信したいパケットが同じ隣接ノードに複数存在した場合最も長く FIFO に残っているものを選択して (LRU)，ここで選択されないものはそのまま FIFO 内に残す．そこで決定した FIFO の番号を Crossbar へ送信する．ここで隣接ノードの FIFO が満杯などによる入力が不可能な場合，FIFO ready 信号の Switch Controller で Crossbar 内でパケットを送信できないように処理する．Crossbar は Switch controller から送られてきた信号からパケットをどの隣接ノードに送信するか 1 つずつ選択する．候補がない場合または隣接ノードの FIFO が満杯の場合は何も送信しない．またパケットの生成方法として，一部のノードに計算ノードが 1 つ付属してあることを想定し，その部分から一定時間ごとに 1 つずつ Packet FIFO に入力される．そのパケットを Crossbar で他のノードに送信することでパケットの送受信ができるものになる．ここでも FIFO が満杯の場合はパケットを生成しないようにする．

3.2. Average Packet Latency の実験方法

はじめに，全ノードからパケット送信する計算ノードの割合であるトラフィック負荷 ($0.0 < \lambda \leq 1.0$) を設定する．このトラフィック負荷 λ を 0.05 刻みで 1 まで増大させて，実行時間の変化を検証する．計算ノードは，宛先ノードを自身のアドレス以外を指定し，パケットを生成すると同時にパケットを隣接ノードへ送信する際に実行時間を増加させるとする．ここで，パケットが宛先ノードに到着した際に，送信元ノードの到着カウンタを 1 つ増やす．これらの操作を繰り返し，パケットを送信している送信元ノードの到着カウンタが一定数を超えたときの実行時間を測定する．今回はすべての到着カウンタが 200 を達成したときの実行時間を計測した．この実験を 100 回繰り返し，その実行時間の平均値を結果とした．今回は 5 種類のトポロジを用いて実験を行った．トポロジの種類やパラメータなど次の表 2 からなる．

また，今回使用するアルゴリズムとして，最短経路アルゴリズム (Algorithm 1 から 4) を用いる．λ を増大させることによる実行時間の変化を調べる．また，Packet FIFO の容量を 2 と 8 に設定して，実行時間にどのような差が生じるか確認する．

表 2. トポロジのパラメータ設定

トポロジ	HQ(11)	CQ(11)	(8,4)-Star	GSC(5,3,5)	GSCC(5,3,5)
ノード数	2048	2048	1680	1920	1920
次数	11	11	7	10	10
直径	11	6	7	10	8

3.3. Average Packet Latency の結果と考察

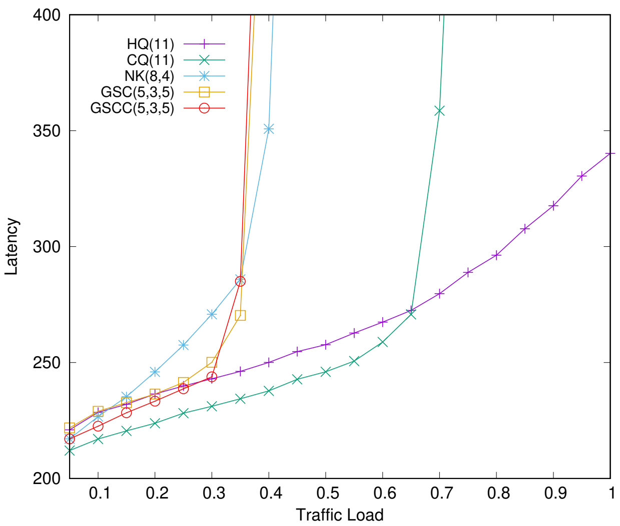

図 3. FIFO 容量が 2 のときの Average Packet Latency

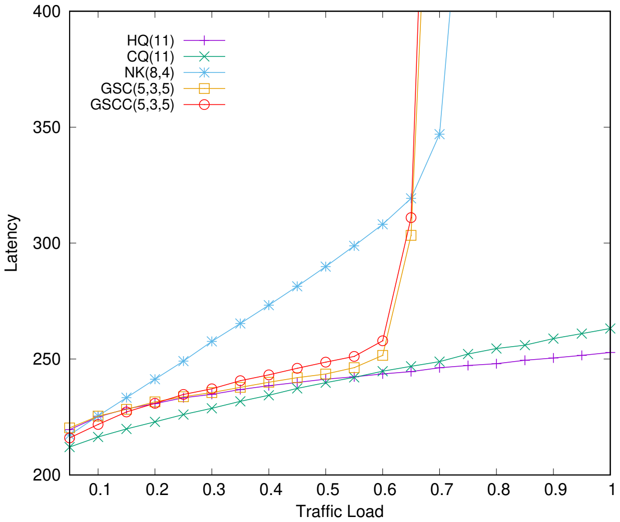

図 4. FIFO 容量が 8 のときの Average Packet Latency

この小節では比較を行う．ここでは先行研究とグラフ単体でのトポロジ 5 つで比較する．この実験では，同じノード数にすることは不可能なため，ノード数をなるべく近づけて実験した．図 3 と図 4 では，表 2 からおよそノード数 2,000 個のときのトラフィック負荷 λ の増加による実行時間の変化を示している．図 3 の FIFO 容量が 2 のときの Hypercube と Crossed Cube を比較すると，λ が小さいときは直径が小さいことでホップ数が少なくなるため，Crossed Cube が実行時間が最大で 12 小さくなっている．しかしながら，λ が 0.65 を超えたあたりから Crossed Cube の傾きが非常に大きくなり，Hypercube のほうが実行時間が小さくなる．これは最短経路を探索するため，パケットが同じ隣接ノードを参照してパケット同士が詰まるためであると考えられる．(n,k)-Star Graph と Generalized-Star Cube，Generalized-Star Crossed Cube を比較すると，(n,k)-Star Graph は λ が小さいときでも傾きが大きいが，λ が 0.4 より大きくなると GSC と GSCC よりも実行時間が小さくなる．GSCC は λ が 0.25 までは Hypercube，GSC よりも小さくなるが，0.3 で Hypercube より大きくなり，0.35 以上になると GSC より大きくなることが確認できる．これらのことからトラフィック負荷が小さいとき実行時間が他トポロジより小さくなるため GSCC が優れていることと判断できる．また，GSC と GSC が (n,k)-Star Graph 単体と比較して傾きが急上昇する λ が小さく，かつ傾きがより急である理由として，(n,k)-Star Graph 側の実行時間が Hypercube と Crossed Cube よりも大きく，(n,k)-Star Graph 部分を探索するパケットが多くなった結果パケット同士が詰まってしまい，Hypercube と Crossed Cube 側にも影響を及ぼしているためであると考えられる．

次に図 4 で比較を行う．これは FIFO 容量が 8 のときのそれぞれの Average Packet Latency の実験を行ったものである．Hypercube と Crossed Cube で比較を行うと，どちらも傾きが一定であることが確認できる．しかしながら Crossed Cube がより傾きが大きいため，λ が 0.6 以上になると Hypercube より実行時間が長くなることが確認できる．ここで，他の 3 つのグラフで比較すると，(n,k)-Star Graph は λ が 0.65 までは実行時間が最も大きくなり，その後は GSC と GSCC が大きくなる．GSCC は λ が 0.2 までは GSC と Hypercube よりも実行時間が小さくなり，その後は 0.55 までは Hypercube と同様に傾きが一定である．しかしながら，λ が 0.6 を超えると傾きが急激に大きくなり，実行時間が指数関数的に大きくなることが確認できる．これも FIFO の容量が 2 の時と同様に (n,k)-Star Graph 側の実行時間が Hypercube と Crossed Cube よりも実行時間がかかるためであると考えられる．

これら 2 つのことから，トラフィック負荷が小さいときは GSCC が Hypercube や (n,k)-Star Graph，GSC よりも優れているが，負荷が大きくなると実行時間が指数関数的に大きくなり，最終的には 5 つのトポロジの中で実行時間が最大となるためあまり適していないことが判明した．また，FIFO 容量を大きくした場合でも実行時間が指数関数的に増加するトラフィック負荷が 0.6 からと遅くなるのみで，それ以上のときの増加量はトポロジによって実行時間の順番に変化が起こるものではないため容量に影響するものではないと判明した．

4. 耐故障経路探索アルゴリズム

この節では，耐故障経路探索アルゴリズムとして新たに提案した 3 つの手法と実験内容，結果と考察について述べる．

4.1. Reversible Algorithm

Reversible Algorithm とは Crossed Cube 部分の最短経路探索時に次探索のノードまたはリンク部が故障していた場合，一度 (n,k)-Star Graph 部分を探索するアルゴリズムのことである．(n,k)-Star Graph 部分を一度探索した後，再び Crossed Cube 部分を探索して宛先ノードへ近づける．Crossed Cube 部分の探索と (n,k)-Star Graph 部分の次の経路が両方とも故障していた場合，または

Crossed Cube 部分が探索済みで (n,k)-Star Graph の次の経路が故障していた場合探索失敗とする．このアルゴリズムの利点として，最短経路と同じホップ数で宛先ノードに到達することができる点と 2 つの最短経路探索アルゴリズムをそのまま利用できる点が挙げられる．また，この経路探索アルゴリズムの実行時間も最短経路探索アルゴリズムと同様に $O(n+k)$ で表すことができる．

4.2. Pair-Related and Put-Head Algorithm

Pair-Related and Put-Head Algorithm (PRPH) は，Crossed Cube と (n,k)-Star Graph の両方に対して耐故障経路を設定したものである．Crossed Cube では最短経路探索時に故障が発生した際に反転したいビットより下のビットを確認する．ペア関係 $(R' = (10,00),(00,10))$ のどちらかが成り立つ場合，上位ビットからこれを探索するようにする．これにより経路探索数をできるだけ小さく探索することができるため優先して探索させる．ペア関係がない場合，アドレスの上位ビットから異なる部分を探索させる．これによる Crossed Cube 部分の探索が最長でも m 回となり，経由回数を抑えることができかつ到達率を向上させることができる．

また，(n,k)-Star Graph 部分の探索では宛先ノードにあり送信元ノードにないアドレスを小さい順に 1 つずつ代入している．それらの隣接ノードが全て故障により経路探索できない場合，失敗となる．このアルゴリズムの利点として，最短経路アルゴリズムと同様に Crossed Cube 部分と (n,k)-Star Graph 部分の 2 つのアルゴリズムを分けて設計することができる点が挙げられる．このアルゴリズムの実行時間は 1 回経由するために m 回または $k/2$ 回かかるため，最短経路より大きい $O(m^2+k^2)$ となる．また，Crossed Cube が持つパラメータ m が (n,k)-Star Graph が持つパラメータ k に対して十分大きい場合，実行時間は $O(m^2)$ となり，反対に十分小さくなる場合 $O(k^2)$ となる．

4.3. Mixture Algorithm

Mixture Algorithm とは 4.1 節，4.2 節で紹介した耐故障経路探索アルゴリズム 2 つを合わせたものである．経路探索の順番として，はじめに Crossed Cube の最短経路を探索し，その経路部分が故障していた場合耐 Crossed Cube 用アルゴリズムを実行する．Crossed Cube 部分の探索ができない場合，(n,k)-Star Graph 部分の最短経路アルゴリズム，耐 (n,k)-Star Graph 用のアルゴリズムの順番に実行する．これら全てが探索できない場合，経路探索失敗となる．このアルゴリズムの利点として，4 つのアルゴリズムのいずれかで経路探索成功すれば経路探索し続けることができるため耐故障性に優れている点，このアルゴリズムの実行時間は 1 回経由するために $m+k/2$ 回経由するため，PRPH アルゴリズムよりも少し大きい $O((m+k)^2)$ となる．また，PRPH と同様に Crossed Cube が持つパラメータ m が (n,k)-Star Graph が持つパラメータ k に対して十分大きい場合，実行時間は $O(m^2)$ となり，反対に十分小さくなる場合 $O(k^2)$ となる．これらは，PRPH の実行時間と全く同じになる．

4.4. 最短経路探索アルゴリズムの実験内容

実際のスーパーコンピュータ内のパソコンまたは接続部分が故障してその部分が利用できないことを想定し，トポロジのノードまたはリンクの一部が故障していて経路探索するときその部分を経由することを不可能とする．そのとき，別の経路を探索できない場合，探索失敗と判定する．ここではノード故障の場合は送信元ノードと宛先ノードは故障しないものとする．また，リンク故障は送信元ノードと宛先ノードが孤立している場合も考える．ここで，ノード・リンクの故障率を設定し，5%刻みで 5%から 95%まで増加させる．これらを最短経路探索アルゴリズムと今回提案した 3 つのアルゴリズムとで 10000 回繰り返し，これらを比較して到達率の変化を調べる．また，到達成功時の平均ホップ数を調べ，最短経路探索アルゴリズムとの差を調べる．今回使用する Generalized-Star Crossed Cube のパラメータ n,k,m をそれぞれ $5,3,7$ と設定した．それによるノード数，次数，直径はそれぞれ 7680，12，9 となる．

4.5. ノード故障の場合の結果と考察

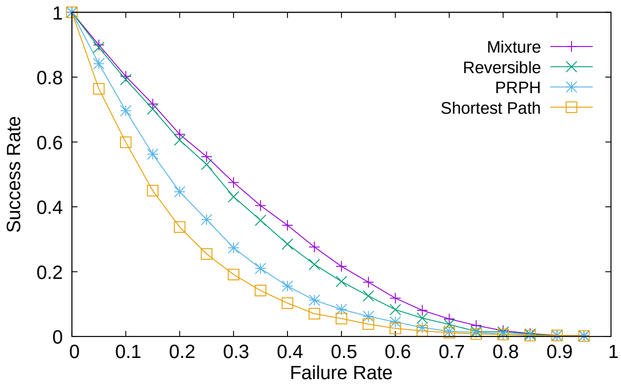

図 5. ノード故障のときの到達成功率

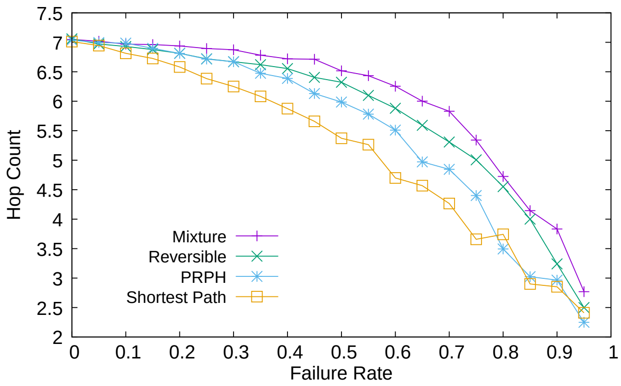

図 6. ノード故障のときのの平均ホップ数

図 5 と図 6 はそれぞれノード故障のときの故障率による到達率と平均ホップ数を示したものである．ノード故障の場合，最短経路探索アルゴリズムでは故障率が増加すると同時に急激に到達率が減少している．PRPH も最短経路と同様に故障率が増加すると急激に到達率が減少しているが故障率 20%のときおよそ 10%の差を生じている．Reversible は故障率が 50%まではほぼ直線的に到達率が減少していて，特に故障率が 30%のとき最短経路との差が最も大きくなる．Mixture は Reversive と比較すると故障率が 40%のときが最も差が大きくなり，その差は 5%である．最短経路と比較すると，故障率が

25%では最短経路の到達率は25%であるがMixtureの到達率は55%と30到達率が向上していることが判明した．

また，図6では平均ホップ数のグラフを示している．故障率0%からの平均ホップ数はおよそ7でありホップ数が1小さくなるまでに最短経路は故障率40%であるが，PRPHは故障率50%，Reversibleは故障率60%，Mixtureは故障率70%で1小さくなっていることが確認できる．故障率75%のとき最短経路とMixtureの平均ホップ数の差が最も大きくなり，その差は2.7となる．このことから送信元ノードから宛先ノードまでの距離が大きくても到達できる回数が増え，少し遠回りしても到達することができると判断できる．

4.6. リンク故障の場合の結果と考察

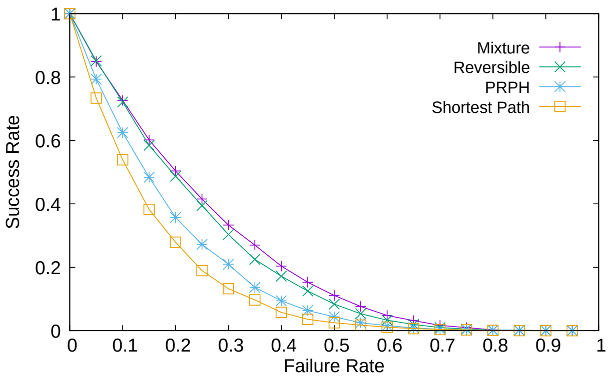

図 7. リンク故障のときの到達成功率

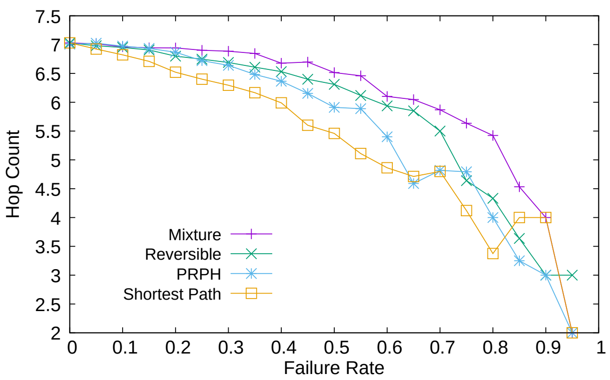

図 8. リンク故障のときの平均ホップ数

図7と図8はそれぞれリンク故障のときの故障率による到達率と平均ホップ数を示したものである．ノード故障と比較すると全体的に到達率が低下している．これは実験節で述べた送信元ノードまたは宛先ノードの全てのリンクが故障していた状態のとき，到達不能になるためであると考えられる．最短経路探索アルゴリズムでは，故障率が20%のとき到達率は28%であるが，PRPHは36%で8%到達率が向上し，ReversibleとMixtureでは到達率が50%と20%到達率が向上していることが確認できる．ReversibleとMixtureを比較すると故障率35%のとき最も差が大きくなり，その差は5%である．

また，図8の平均ホップ数では故障率0%のときの平均ホップ数はおよそ7であり，平均ホップ数が1減少するのに最短経路では故障率40%，PRPHは故障率50%，Reversibleは故障率60%，Mixtureは故障率70%である．これはノード故障のときと同じである．しかしながら，故障率60%を超えたあたりからMixtureを除いた3つのアルゴリズムの平均ホップ数が大きく異なっている．これは図8から最短経路の到達率が10000回中100回未満と非常に小さい結果を示していたためであると考えられる．それに対してMixtureは図6と同様にどの故障率でも安定した平均ホップ数を記録している．また他のアルゴリズムと比較しても最大で1以上の差を付けていることから，到達率が優れていると考えられる．

5. まとめと今後の課題

本論文ではAverage Packet Latency によるGeneralized-Star Crossed Cube の性質，および本トポロジの耐故障経路探索アルゴリズムによる到達率を向上させる3つのアルゴリズムを提案した．Average Packet Latencyを実行したところ，トラフィック負荷が小さい場合，直径が小さいことで経由するノードが少なくなるため他トポロジよりも実行時間に有用性があることを示した．また，耐故障経路探索アルゴリズムではCrossed Cube部分と(n,k)-Star Graph両方の耐故障経路探索アルゴリズムを提案し，さらに積グラフの性質を活かし2つの耐故障経路探索アルゴリズムを混ぜて探索することで特にノード故障のときの到達率を最大で30%向上することができた．しかしながら，パケット送信する際にトラフィック負荷が大きくなると，パケット同士の衝突によるデッドロックが頻繁に発生して処理に時間がかかることやパケット送信ができなくなるなどの問題点が挙げられる．それを回避するためのデッドロックを考慮したものあるいはデッドロックフリーのアルゴリズムを提案することが重要になる．また，耐故障経路のCrossed Cube部分と(n,k)-Star Graphのアルゴリズムはどちらも数ステップ戻って探索し直す操作を実装していないため，さらに到達率を向上させるためにはこのような操作も重要である．今後の課題として，それぞれの問題点を解決し，更に性能向上できるアルゴリズムを提案することが必要である．

参考文献

[1] T. Sato and Y. Li, "Generalized-star crossed cube - a flexible interconnection network with high-performance at low-cost," in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, Nov 2017, pp. 153–158.

[2] Y. Li and W. Chu, "Mikant: A mirrored k-ary n-tree for reducing hardware cost and packet latency of fat-tree and clos networks," 10 2018, pp. 1643–1650.

[3] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Transactions on Computers*, vol. 37, no. 7, pp. 867–872, Jul 1988.

[4] K. Efe, "The crossed cube architecture for parallel computation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 5, pp. 513–524, Sep. 1992. [Online]. Available: http://dx.doi.org/10.1109/71.159036

[5] W.-K. CHIANG and R.-J. CHEN, "Topological properties of the (n,k)-star graph," *International Journal of Foundations of Computer Science*, vol. 09, no. 02, pp. 235–248, 1998. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0129054198000167

[6] D. Arai and Y. Li, "Generalized-star cube: A new class of interconnection topology for massively parallel systems," in *2015 Third International Symposium on Computing and Networking (CANDAR)*, Dec 2015, pp. 68–74.

[7] N. Adhikari and C. Ranjan Tripathy, "Star-crossed cube: An alternative to star graph," *TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES*, vol. 22, pp. 719–734, 01 2014.

[8] C.-P. Chang, T.-Y. Sung, and L.-H. Hsu, "Edge congestion and topological properties of crossed cubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 1, pp. 64–80, Jan 2000.