Vol.2021-DPS-186 No.56
Vol.2021-CSEC-92 No.56
2021/3/16

複数の時間間隔に基づくオーバレイネットワークにおけるルーティング効率化手法の検討

久保 達也[1,a)]　川上 朋也[1]

概要：センサデバイスの低廉化が普及の背景にある IoT では，位置や時間と密接に関連付いた情報を大量に扱うという特徴がある．大量のセンサデータを高スケーラビリティで扱うための手法のうち，特に時間の結びつきに着目してクエリを効率良く扱う手法が提案された．この手法は一定時間間隔でデータを要求されるケースにおいて効率的なデータ探索を実現するが，ノードやネットワークへの負荷が増大する問題が残っていた．本稿ではノードの仮想化及びルーティングのアルゴリズムを変更することにより効率化を図る手法を提案する．特定ノードへの負荷集中を回避するために仮想化を行ったネットワークについて，従来は考慮されなかった仮想化される前の物理的なレイヤーへもルーティングの処理対象を広げ，冗長なやり取りを削減した．そして，従来よりもネットワークに対する負担や各ノードの処理回数を軽減する効果があることを確認した．

1. はじめに

近年の急激なデバイス数の増加そして回線品質向上に伴い急速な成長を続けるインターネットは，その中でもとりわけセンサデバイスの低廉化や社会からのニーズが高まりを受けた Internet of Things（IoT）分野の発展が著しい．Cisco 社が行った予測 [1] によると，世界中のデバイスと接続数は世界の総人口やインターネット接続人口よりも速い年 10%のペースで増加を続けるとされている．特に IoT の根幹である Machine to Machine（M2M，ユーザーが介在しない機械同士の通信）は最も著しい増加を見せる分野になり，接続数は 2018 年の 61 億に比べて 2024 年には 2.4 倍の 147 億になると予想されている．

上記のように大量のセンサデバイス及び測定データからなるネットワークやサービスを提供するには，従来のクライアントサーバモデルでは負荷の上昇に耐えられなくなることが考えられる．負荷が高くなりすぎるとやがてサーバはクエリを処理しきれなくなり障害が発生し，サービスの可用性を低下させることとなる．対策としてはサーバの設備増強が行われるが，しかし年に十数%と予想されるクライアント側（IoT 機器）の増加に追いつくことは技術的にも経済的にも限界がある．

そこで，この問題の解決策として考えられるのが Peer to Peer（P2P）方式によるネットワークである．これはネットワーク上で対等なノード（端末）が協調して処理を行うモデルであり，中央集権的なサーバを持たずにサービスを構築することができる．P2P のネットワークは大きく分けて構造化オーバレイネットワークと非構造化オーバレイネットワークの二種類に分かれる．本研究で扱う構造化オーバレイネットワークは，データの分散処理を確実に実現するために分散ハッシュテーブル（Distributed Hash Table, DHT）の仕組みを用いているものが多い．DHT はネットワークに入力されたデータを確実に分散させる能力に優れている．しかし，センサデータは高頻度であり位置・時間の情報に深く結びついているという特徴があり，DHT ではうまく扱えずに冗長な通信が発生したり負荷が集中する可能性がある．そのためセンサデータを効率よく扱うための DHT によらない構造化オーバレイネットワークが提案されている．既存研究は位置情報に基づいてネットワークを構築するもの [2] や，データの効率良い範囲探索を行うためのもの [3] が主であったが，本研究ではクエリの時間的間隔に着目した先行研究 [4] のルーティングを効率化する手法を提案し，冗長な通信の削減とネットワークの負荷低減を目指す．

[1] 福井大学
University of Fukui
[a)] hb170408@u-fukui.ac.jp

Vol.2021-DPS-186 No.56
Vol.2021-CSEC-92 No.56
2021/3/16

2. 関連研究

2.1 オーバレイネットワーク

インターネットは Internet Protocol（IP）によって経路が制御されるネットワークを ISP がクライアントに提供し，さらに ISP が ISP 同士あるいはより上位の ISP に接続することによって実現されている．一方で，サーバを持たない P2P では自立した各ノードが既存 IP ネットワーク上に論理的な異なる構造のネットワークを形成する．このようにアプリケーション層で作られるネットワークはそれより下層の IP ネットワークに覆いかぶさるように仮想化していることからオーバレイネットワーク（Overlay Network）と呼ばれる．

オーバレイネットワーク上で行われた通信を実際に送信する際にはノードの IP アドレスに基づいてアンダーレイネットワーク上で通常と同じルーティングを行う．そのため，オーバレイネットワークネットワーク上でのノード間の隣接関係がアンダーレイネットワークと大きく異なることが起こりうる．そのような場合，オーバレイネットワーク上では最短経路の効率良い通信が行えていても，実際にはネットワーク上で遠く離れた場所への無駄なトラヒックを発生させてしまっている可能性がある．このようにオーバレイネットワークにはアンダーレイネットワークとの齟齬が少なからず存在している [5].

2.2 Chord

Chord [6] は，分散ハッシュテーブルを利用した構造化オーバレイネットワークのアルゴリズムである．構造化オーバレイネットワークは，ネットワークを生成する際に環状や木構造といった数学的な規則に基づいたトポロジに従ってネットワークを構築するオーバレイネットワークである．また，DHT は構造化オーバレイネットワークを実装する方法の一つである．

DHT ではネットワーク上のノードと入力されるデータはすべて同じ空間上のキーを持つ．各ノードに SHA-1 等のハッシュ関数によってユニークなキー（識別子,ID）を割り当て，そのキーの値によってネットワークのどの位置に配置されどのノードと隣接関係を結ぶかが定められる．さらにネットワークに入力されるデータにも同じハッシュ関数によってキーを算出し，そのキーとの比較によってどのノードに格納されるかが決定される．データを検索する際にはクエリからハッシュ値を算出し，そこから問い合わせるべきノードを特定する．基本的にハッシュ関数である SHA-1 を利用し，2^{160} の大きさのキー空間を分割する形で各ノードがネットワークを構成するが，それ以外の大きさのキー空間でも同様のアルゴリズムが利用可能である．

Chord によるネットワークの例を図 1 に示す．各ノードは Successor・Predecessor・Finger Table の三種類の経路情報を保持する．ネットワーク上のノードのうち，自身より ID が大きくかつ最も近い ID を持つノードを Successor とし，反対に ID が自身より小さい最初のノードを Predecessor とする．これに加えて離れた場所にあるノードの情報を Finger Table として保持する．この Finger Table には自身から $2^k (0 < k < N, N$ はキー空間のビット数) 離れたキーを担当するノードへのリンクを保持する．このとき，ID 空間の終端に達した際には最初に戻って続行する．データを取り扱う際には，データのキーより ID が小さくかつ最も近いノードが担当となる．すなわちノードの ID と Predecessor の ID の間のキーを持つデータを格納する．

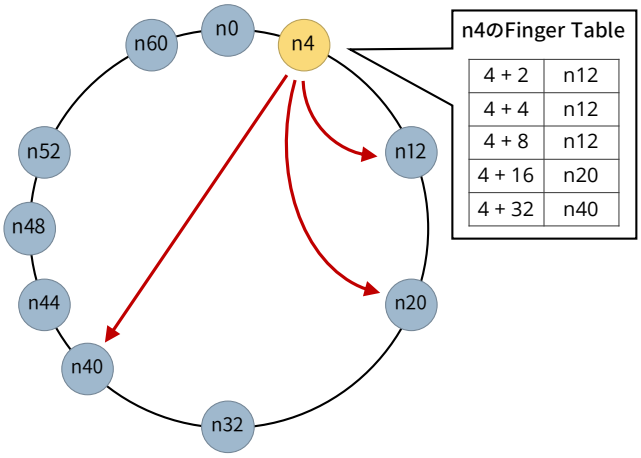

図 1 2^6 の ID 空間における Chord の模式図

Chord のネットワーク上でデータやクエリを転送するときには必ず Successor への一方通行，すなわち ID の昇順の時計回りでのみで転送する．これに加えて Finger Table も通信に利用される．単純に Successor への転送を繰り返した場合に比べ，Finger Table を利用することでネットワーク上のどこに目的のノードがあっても常に経路の長さを半分程度短縮することができる．これにより，N ノードを持つネットワークでのデータの検索に必要な平均ホップ数は $O(\log N)$ となる．このように，Finger Table は Chord の効率化に大きく寄与するものである．

Chord は環状のネットワークであるため常に一方向にデータを送信すれば目的のノードに到達することができ，末端部などでの特別な処理も不要で非常に簡素なアルゴリズムで実装できる．そのため，DHT の最も代表的なアルゴリズムとして知られており関連研究も多い．

Chord$^{\#}$ [3] は関連研究の一つで，DHT の機能をなくしたものである．この手法ではデータの持つそのままの値をキーとしている．似たようなデータは似た場所に格納されることになるため，範囲検索を行う際にはノードを順番に辿っていけばすべてのデータが得られ，最小限のトラヒックに抑えることができる．

しかし負荷分散の仕組みであったハッシュ値を取り除くと入力データの偏りがそのままネットワークに反映され

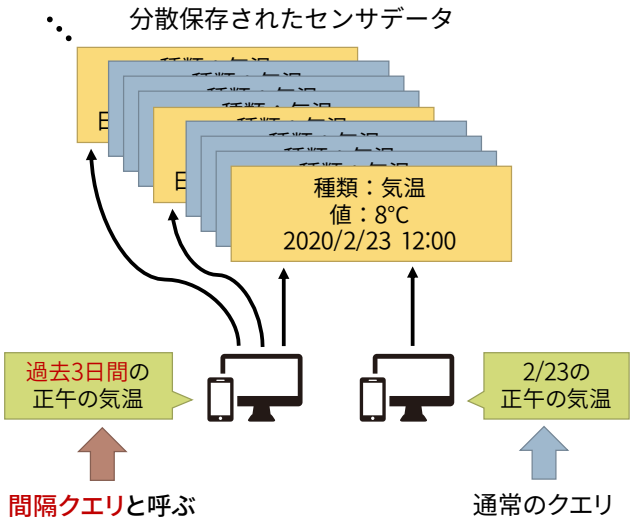

図 2 間隔クエリの概念図

ることとなる．特定のキーのデータが多い場合，その担当ノードに負荷が集中することとなる．その対策として，過負荷になっているノードの隣に動的に負荷が低いノードが入ってデータを分割する手法 [7] を利用している．

2.3 複数の時間間隔に基づくネットワーク

IoT において中心的存在となるセンサデータは高頻度で更新され位置・時間の情報に深く結びついているという特徴がある．よって，通常のオーバレイネットワークとは異なる特性を持たせる必要がある．

付加すべき機能には様々なものが考えられるが，ここでは本研究の基となるデータと時間との結びつきについて着目した研究 [4] について詳細に述べる．すなわち，センサデータの時間的な結びつきによりネットワークの利用者（またはシステム）はデータをある特定の時間間隔という条件で複数要求することが考えられる．例としては「去年までの 7 月の気温」や「毎週日曜日の電力消費量」といったものである．このような複数のクエリで構成されるクエリをまとめて間隔クエリと呼び，概要を図 2 に示す．間隔クエリは過去のデータによる統計情報を得て，今後の予測を立てるといったビッグデータ等の用法に有用であると考えられる．

本研究では Chord と同じ環状のネットワークを想定し，データの記録された時間をキーとしてネットワークを構築する．ここでデータの記録された時間は UNIX 時間といった特定の基準日時からの相対的なものによる単純な一次元の値とする．Chord$^{\#}$ と同じく，連続したデータを効率よく扱うためにハッシュ値は使用せずに直接時間情報をキーとする．一方で間隔クエリは様々な単位で要求が行われ，ネットワーク上での単位時間より離れた間隔となるため単純な範囲クエリとは異なったものである．そこで，この手法では Chord の Finger Table に相当するものを間隔クエリを想定した複数の時間間隔に基づくものに差し替える．

この手法も DHT ではないため，Chord$^{\#}$ と同じ問題が起こる．すなわち，センサがデータを取得した時刻が偏っているとその時刻の担当ノードが多くのデータを保管しなければならず負荷が集中することになる．そのため，この研究では仮想化を利用して負荷分散を行った [8]．

3. 提案内容

3.1 既存手法の問題点

2.3 節で述べた既存研究には，仮想化がない場合に比べ物理ノードの負荷は軽減されるがネットワーク上のトラヒックが増大するという解決されていない問題点がある．従来は p 個の物理ノードが単一のノードとしてネットワーク上に参加していたものを各物理ノードが v 個の仮想ノードを持った場合，ネットワーク上に存在するノードの数は p 個から pv 個へと増加することになる．そのため，何も対策をしなかった場合 Chord で示された法則に従い平均ホップ数はネットワーク上のノードの数に対して対数で増加していくこととなる．クエリを処理するのに必要なホップ数が増加することはネットワーク上でやり取りされるメッセージの総数も増加させる．メッセージ数が増えるとネットワークの帯域を過剰に消費するだけではなく各ノードがメッセージの転送や応答を行う回数も増加し，負荷分散の性能を低下させることになる．間隔クエリは必ず複数のクエリで構成されるためにメッセージ数増大の影響は単一クエリよりも大きなものとなり，特に対策が必要である．

3.2 冗長なトラヒックを削減する手法

3.2.1 問題点の原因

前節の問題は，オーバレイネットワークで仮想化を行う場合に一般に起こりうる問題である．構造化オーバレイのひとつである Skip Graph [9] に対して仮想化を適用した小西らの研究 [10] にも同様の問題が発生していた．

Skip Graph は Chord と同じ平均 $O(\log N)$ ホップでの検索が行え，また範囲検索を効率良く行うことができる．しかし欠点として，構造上ひとつのノードにはただ一つのデータしか格納することができない．ノード仮想化を導入することでこの問題を解決したが，前節の問題が発生した．

小西らはこのメッセージ数増大の原因を物理ノード間での冗長なやり取りとした．そして，ネットワークのトポロジは異なるが同様の問題がこの複数の時間間隔に基づくネットワークでも発生していると考え，これへの対策が有効な手段であると考えた．

すなわち，同じ物理ノードが同一のクエリをルーティング中に何度も受け取ってしまうという点が原因のひとつであると仮定した．仮想化されたネットワークにおいてはネットワーク上に仮想ノードのみが存在することとなる．

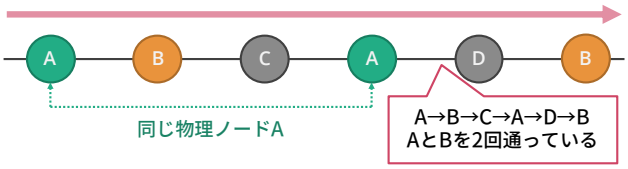

図 3 物理ノード間での冗長な通信の概要

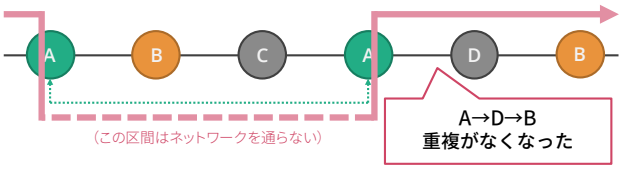

図 4 物理ノードを考慮したルーティングの提案手法

よって通常の Chord と同じルーティングを行った場合には仮想ノード単位でのルーティングであり，物理ノードの存在は考慮されない．そのため，ネットワーク上では最短経路での最適な通信が行われているように見えても，実際には同じ物理ノードの間での通信が繰り返し行われていたり同じ場所を往復していたりといった冗長な通信が発生していることが考えられる．これは，2.1 節で述べたオーバレイネットワークの齟齬と似たような問題である．

図 3 にこの問題点を示した．この図はネットワークの一部を切り取って簡略化したものである．丸が仮想ノードを表し，書かれているアルファベットはどの物理ノードに関連付けられているかを示す．いま，この図の左端のノードから右端のノードまでクエリが転送されていくとする．説明のため，ショートカットテーブルは省略し常に Successor に対してのみ通信を行うとする．仮想ノード単位で見ると順番に重複せずネットワークを流れているように見えるが，例えば物理ノード A に着目すると二度同じクエリを受信して転送していることがわかる．このように，図中の全体では物理ノード A と B に対して冗長な通信が発生している．

3.2.2 物理ノードを考慮したルーティング手法

この問題点の対処として，クエリ転送時に選択する転送先の候補を仮想ノードだけではなく物理ノード単位に拡大したルーティング手法を提案する．従来は仮想ノードはクエリを受け取ったら，そのままその仮想ノードが持つショートカットテーブルもしくは Successor から転送先を決定していた．それに対し提案手法は，以下の手順によってルーティングを行う．ただし，物理ノード P は仮想ノード $v_0, v_1, v_2, ..., v_n$ を持っているものとする．また仮想ノード v のキーを $K(v)$，クエリの宛先のキーを q，キー空間のビット数を m とする．

手順 1　仮想ノード $v_k (0 \leq k \leq n)$ に対して送られてきたクエリを，一旦物理ノード P が受け取る

手順 2　P は配下の各仮想ノード v_k についてショートカットテーブルや Successor でルーティングを行い，v_k からの次の転送先 v'_k をそれぞれ収集する．

手順 3　収集された次ホップ候補 v' の中から，最も $(K(v'_k) - q) \bmod 2^m$ が小さい，つまり最もクエリの宛先に近い v_k から次の仮想ノードへ転送する

すなわち，より高レベルである物理ノードからの視点もルーティングに加味する手法である．手順 2 では仮想ノードのルーティングテーブルを参照するのみであり，外部との通信は発生しない．ルーティングテーブルを参照する処理は増加するが，$O(n)$ の処理でありまたルーティングテーブルの要素数も決まっているため大きな負担とはならない．また，手順 1 でクエリを受信した仮想ノードと手順 3 で送信した仮想ノードが異なっていたときには，単に物理ノードのプログラム内で転送元を付け替えるだけで処理することができるため，完全にネットワークを迂回して経路を短縮できる．

図 4 に提案手法によるルーティングを示した．図 3 と同じ状況で図の左端のノードから右端のノードまでクエリが転送されていく模式図であるが，ここでは最初にクエリを受信した物理ノードの A がネットワークの先にあるもう一つの仮想ノードのほうがより目標に近いためショートカットが機能している．その結果，図 3 では物理ノード A・B が 2 回づつクエリを受信していたが，重複がなくなり，経路長も 5 ホップから 2 ホップへ軽減された．なお，ここで仮にクエリの担当ノードが左から 2・3 番目の仮想ノードだった場合はこの動作は行われない．

提案手法のネットワークは木構造のように途中で経路の枝分かれはなく必ずキー順に一直線である．よって，一度物理ノード内で最もクエリの宛先に近づけるように転送を行えば残りの仮想ノードはクエリの後方となり，通信が一方通行であることから同じクエリをまた受信することはない．よって，この手法により 1 回のクエリ転送で特定の物理ノードを経由するのは 1 回のみとなり，物理ノード間での冗長なやり取りを解消することができる．

4. 評価

提案手法の有用性を示すため，仮想化を行った複数の時間間隔に基づくネットワークのシミュレーションを行った．シミュレータは Java を用いて実装し，本章ではシミュレーション環境とその結果について述べる．

4.1 シミュレーション環境

本シミュレーションではキーの最小間隔を 1 時間に設定し，キー空間のサイズは 16 ビットとした．よってノードのストレージ容量が十分であれば最大で $2^{16} = 65536$ 時間 $\approx$ 7.48 年 のデータを格納することができる．すべての仮想ノードは表 1 に示すショートカットを持ち，キーは空間内からランダムに選択される．全ての物理ノードは予め決め

Vol.2021-DPS-186 No.56
Vol.2021-CSEC-92 No.56
2021/3/16

表 1 構築するショートカットの種類と数

種類	間隔	数
年	2^0 年，2^1 年，2^2 年...$2^n - 1$ 年	n
月	6 ヶ月，3 ヶ月，1 ヶ月	3
日	15 日，7 日，3 日，1 日	4
時	12 時間，6 時間，3 時間，1 時間	4

表 2 共通の実験条件

パラメータ	値
キーの単位時間	1 時間
キー空間の大きさ	16 ビット (65536 時間)
各仮想ノードのキー	一様なランダム選択
仮想ノードと物理ノードの関係	一様なランダム選択
間隔クエリのパターン	168 時間間隔で 52 個
間隔クエリの起点	一様分布・正規分布

られた数の仮想ノードを持ち，どの物理ノードがどの仮想ノードを担当するかはランダムに決定される．

ネットワークに入力される間隔クエリは，186 時間の間隔を開けた 52 個の連続した昇順のクエリで構成される．これは，1 週間間隔で 1 年分のデータを要求することと等しい．間隔クエリの起点となるキーは，一様分布・正規分布に従う．正規分布は 平均 $\mu = 2^{15} = 32768$，分散 $\sigma = \frac{2^{15}}{3} \approx 10922.7$ とした．$\pm 3\sigma$ の範囲に ID 空間が収まり，正規分布の性質より 99.73%のクエリ起点がキー空間の中に収まる．

間隔クエリはネットワーク上を再帰的に転送されていく．各クエリの担当ノードは，受信した間隔クエリの中から自身が担当となるクエリを取り除き次のクエリの担当ノードへ向けて間隔クエリを転送する．52 個目である最後のクエリの担当ノードは単一のクエリを受け取ることとなり，その処理を持って間隔クエリ全体の処理が終了する．

比較手法として，従来の仮想ノード単位でのみのルーティングを同条件で行う．また，簡略化のために通信途中でのノードの参加や離脱は起こらないものとした．ここまでに述べた共通の実験条件を表 2 に示す．

4.2 ネットワーク負荷に対する評価

この節では，間隔クエリを全ての担当ノードに転送するのにどれだけのメッセージ数が必要になるかを指標にネットワーク全体への負荷がどう変化したかを評価する．

4.2.1 この実験での追加条件

ネットワーク上の各物理ノードが 1・2・4・8・16・32 個の仮想ノードを取ったとき，物理ノード数を 250・500・750・1000 個と変化させるとどのようにメッセージ数が変化していくかを確認した．なお，1 個の仮想ノードを持つときにはネットワーク上で仮想化は行われないことになる．間隔クエリを 20 回送信し，転送にかかったメッセージ数の平均を評価指標とする．

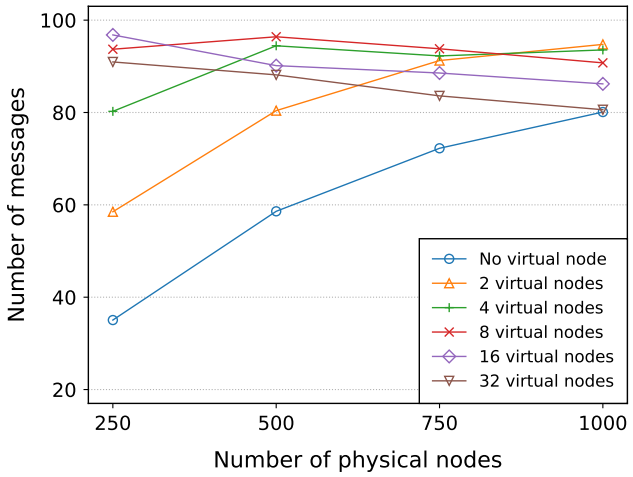

図 5 間隔クエリ起点が一様分布でのメッセージ数（既存手法）

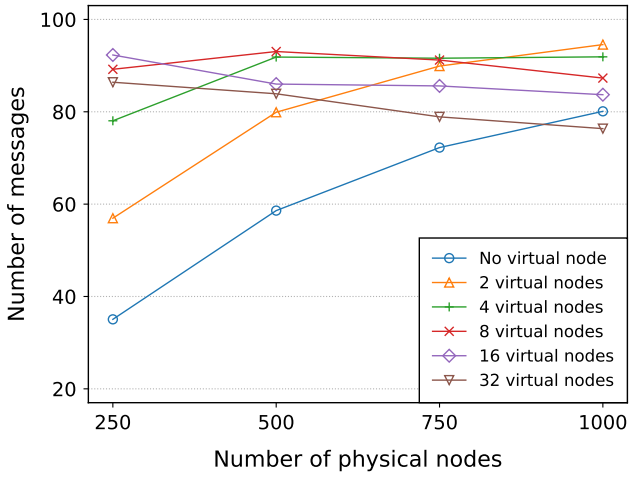

図 6 間隔クエリ起点が一様分布でのメッセージ数（提案手法）

4.2.2 結果

クエリ起点を一様分布とした際の既存手法・提案手法それぞれの結果を図 5 と図 6 に示す．どちらの手法でも仮想ノードがない場合はルーティング方法は同じため結果は全く同じになっている．

まず，どちらの手法でも仮想ノードがあるときはないときよりもメッセージ数は多くなり，メッセージ数は増大するという全体的な傾向は変わっていない．しかし既存手法（図 5）と比較すると提案手法（図 6）は仮想ノードを持つときの結果が全て減少しており，提案手法は一定のメッセージ数の削減効果があることが示された．表 3 に各条件について提案手法によって既存手法から削減されたメッセージ数のパーセンテージを示す．全体として仮想ノードの数が多いときほど提案手法の効果が大きくなり，多くの場合同じ物理ノード数のときには単純に仮想ノードが多いほうが削減量が大きくなっていることが分かる．また，同じ仮想ノードでは物理ノード数が少ないほうがわずかに削減率が上昇する傾向がある．これらの結果は仮想ノードが多くなるほど同一物理ノードの内でのショートカットが動

表 3　提案手法によるメッセージ数の削減率（一様分布）

物理ノード数	仮想ノード数				
	2	4	8	16	32
250	2.65	2.74	4.80	4.65	5.00
500	0.62	2.75	3.48	4.60	4.82
750	1.48	0.70	2.77	3.33	5.62
1000	0.21	1.76	3.80	2.90	5.27

（単位：%）

表 4　提案手法によるメッセージ数の削減率（正規分布）

物理ノード数	仮想ノード数				
	2	4	8	16	32
250	1.39	2.59	3.90	5.09	5.05
500	0.63	2.42	3.51	4.37	4.82
750	0.38	1.41	4.17	3.32	4.23
1000	1.19	1.22	3.40	3.80	4.40

（単位：%）

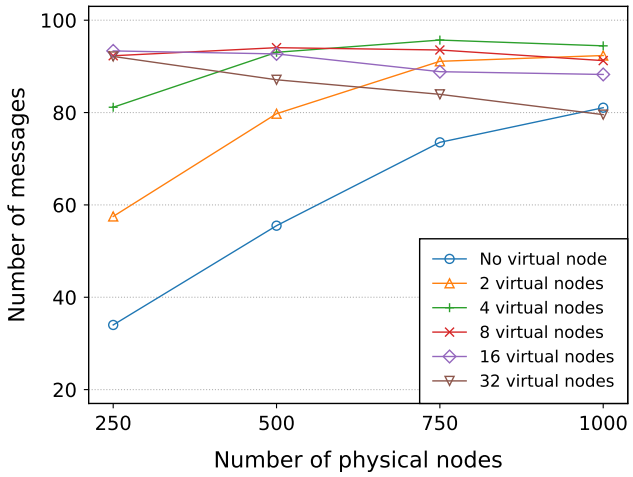

図 7　間隔クエリ起点が正規分布でのメッセージ数（既存手法）

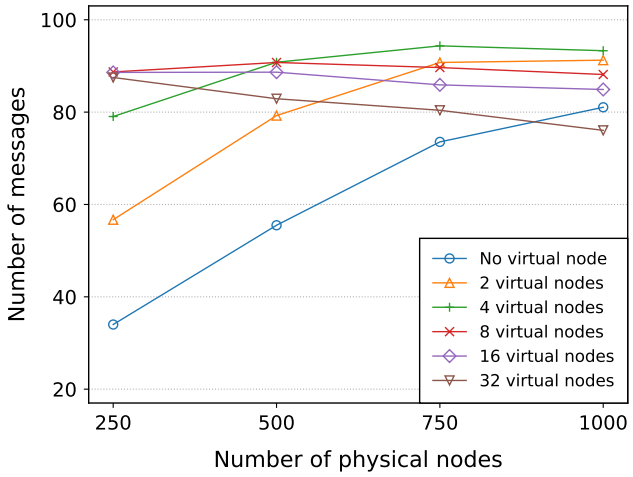

図 8　間隔クエリ起点が正規分布でのメッセージ数（提案手法）

作する機会が多くなり，また物理ノードの少ないほどそのショートカット効果を維持したままネットワークの規模が小さくなったためだと考えられる．

次に，クエリ起点を正規分布とした場合のそれぞれのメッセージ数を図 7, 8 に，既存手法と提案手法の差を表 4 に示す．既存・提案とも平均値は一様分布のときとほとんど変わらずグラフの全体的な傾向も変わらない．しかし仮想ノードが 8・16 個では一様分布のときに比べ特に物理ノードが多いときに削減効果が大きくなっているのに対し，2・4 個ではほとんど変わらず若干効果が小さくなっているところも存在する．このように，一様分布のときと比べて仮想ノードの数による削減効果の差が若干大きくなっていることが分かる．また，同一仮想ノード数のとき物理ノードの数が少ないほど削減率が向上する傾向が一様分布のときに比べて強くなっている．このことからより現実的でランダム性が高いリクエストが行われた際には，より大きな変化をもたらす可能性があると考えられる．

4.3　物理ノードの処理負荷に対する評価

この節ではメッセージ数の変化が各物理ノードの負荷にどのような影響をもたらすかを明らかにするため，間隔クエリの処理中に物理ノードが何回アクセスされるかを計測した．

4.3.1　この実験での追加条件

この実験では物理ノードのアクセス数の分布を集計しどのような傾向があるかを示す．ここでアクセス数とは各ノードがクエリを送信・転送・受信した回数のことであり，クエリの転送経路上にあるノード全てに 1 回づつカウントアップされる．すなわち，物理ノードがクエリを処理した回数の総計である．ノードがデータを蓄えるストレージに対する負荷とは別に，ネットワークとの通信を処理する負荷を集計することを意図している．実際にはクエリに対する処理によって処理負荷は異なると考えられるが，ここでは純粋なカウントのみとし重み付けは行わない．

基本的な条件は前節の実験の変わらないが，いくつか変更した点がある．まず，前節の実験により物理ノードが少なく仮想ノード多いときに特に提案手法の効果があることが分かったため，物理ノード数を 300・仮想ノード数を 32 とした．またこの実験ではネットワーク全体の物理ノードについて計測を行い，そこから分布を示すため試行回数が少ないとネットワーク全体にクエリが行き渡らずうまく結果を得られない．そのためこの実験では仮想クエリを 180 回送信し，その累計を結果とした．

4.3.2　結果

まず，クエリ起点を一様分布とした際の結果を図 9 に示す．最小値・最大値・四分位数がすべて提案手法は下がっていることが分かる．一方で四分位範囲の大きさはほとんど変化していない．よって，提案手法によって物理ノードへのアクセス数は偏りなく均等に削減され，どの物理ノードもルーティング処理の負荷が下がるといえることが示された．

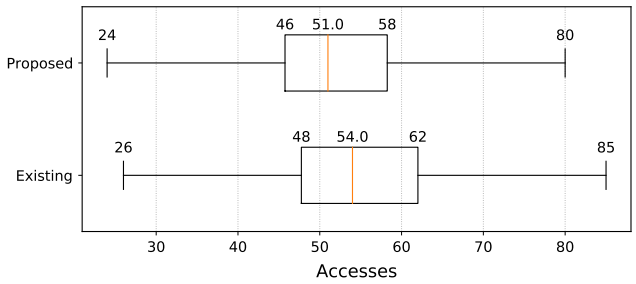

図 9 間隔クエリ起点が一様分布でのアクセス数分布

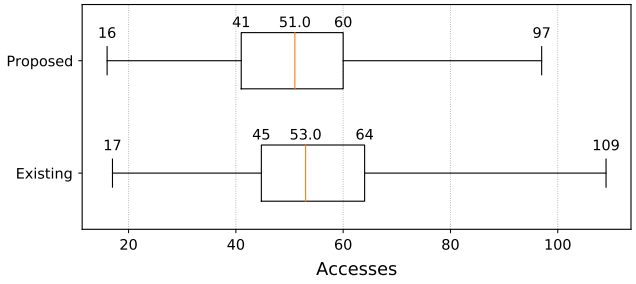

図 10 間隔クエリ起点が正規分布でのアクセス数分布

次に，クエリ起点を正規分布とした際の結果を図 10 に示す．

図 9 に比べ四分位範囲の大きさが倍程度になっている．また，第 3 四分位数から最大値，すなわち右のひげの長さが伸びている．これはクエリの宛先が偏って分散が大きくなったことに加え，一部のノードにアクセスが集中していることを示している．前項と同じく四分位数はすべて提案手法により下がっていることに加え，提案手法は既存手法に比べて右側ひげの長さが 2 割ほど短くなり最大値が大きく減少している．よって，提案手法は負荷が集中しているノードを減らすことにも効果があるといえる．

4.4 考察

4.4.1 提案手法の効果について

表 3 および表 4 は提案手法によって既存手法からどれだけメッセージ数を削減できたかの割合を示す．この表によれば，提案手法は常に既存手法よりもメッセージ数が減少し増加することはなかった．この点より提案手法は当初の目的を満たしたが，一方で削減量は最大でも 5%程度にとどまっている．

ルーティングの経路上で各物理ノードがどの仮想ノードを選択したかを詳細に確認した結果，多くの場合クエリを受け取った仮想ノードがそのまま次の転送に使用するノードに選ばれていることがわかった．すなわち，多くの場合既存手法と変わらないルーティングが行われているということである．

このことを詳細に調べるため，物理ノード数 250 で仮想ノード数 16 のネットワークに間隔クエリを 20 回送信したときに，何ホップ目でどれだけの割合で物理ノード単位

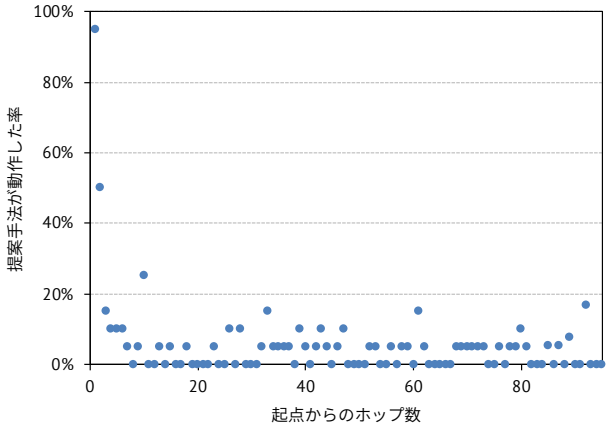

図 11 ホップ毎に提案手法によるショートカットが動作した割合

でのショートカットが作動したかを集計した．その結果を図 11 に示す．始まりの方，特に最初の 1 ホップでは高い確率で提案手法が働いているものの，その後は時折 10%程度の確率で作動するにとどまっている．平均ホップ数は 90.15（図 6 とほぼ同じ結果である）だったが，このうち物理ノードでのショートカットが作動したのは平均 4.8 ホップ，5.3%であった．

提案手法が働いた回数が程度の割合に留まった理由として，前節でも触れた間隔クエリが比較的狭い間隔であることが影響している可能性がある．最初の 1 ホップは担当ノードまで遠く離れている可能性があるが，それから後は一定間隔となるため長距離の転送がほとんどなくなると考えられる．図 11 でこの後の区間でショートカットがときおり機能しているのは，担当区域の狭い仮想ノードが間に挟まっていた等が理由として考えられる．

4.4.2 各ノードでの処理負荷の低減について

4.3 節では，提案手法は各物理ノードでのアクセス数を軽減させることが示された．外れ値を出すことなく全体で均等な負荷軽減が行えたうえ，リクエストに偏りがあるときは外れ値的に負荷の高いノードを抑えることができた．

一方で，提案手法ではクエリ処理時に物理ノードで行うべき処理が既存手法よりも多い．既存手法ではなかった同じ物理ノードの他の仮想ノードのルーティングテーブルをすべて調べるという動作が入る．仮想ノードでのルーティングは，現状ルーティングテーブルにある候補ノードのキーを for 文で線形探索することで実装している．ルーティングテーブルの要素数を T とすると，表 1 より間隔クエリを効率良く扱うためのルーティングテーブルにおいては $T = 11 + \log_2 N$（N はキー空間の大きさ）である．よって仮想ノード数を V とすると探索のコストは既存手法の $O(\log_2 N)$ に比べて提案手法は $O(V \log_2 N)$ となる．仮想ノード数によっては負荷が増す可能性があるが，常に全ての仮想ノードを探索しているわけではないこと，ルーティングテーブルは高速な HashMap に格納されていることや

仮想ノード数が数百になることは考えにくいことから，重大な影響を及ぼす可能性は低いと言える．ただし，今後の状況変化によってはアルゴリズムの改善が必要となることも考えられる．

5. おわりに

5.1 まとめ

近年広まる IoT は，時間や位置に深く結びついたデータを高頻度かつ大量に扱うという今までのネットワークとは異なる特徴を持つ．このため，従来のクライアントサーバモデルに代わる分散ネットワークの仕組みが必要である．既存手法である複数の時間間隔に基づくネットワークは，時間の結びつきに着目して複数のデータの検索効率を向上させた．この手法ではノードを仮想化することでノードへの負荷の偏りを軽減していたが，それによりネットワークへの負荷が高くなる問題が残されていた．

本論文では，この問題を解決するために新たなルーティングの手法を提案した．仮想ノードだけではなく，物理ノードレベルへルーティングの対象を広げることで仮想ノード同士のネットワークでは見えない同じ物理ノード間での冗長な通信をなくすことができる．この手法をシミュレーションで実験した結果，既存手法よりメッセージを削減できることを確認した．仮想ノードが多いときに効果が大きく，またクエリの傾向によっては物理ノード数との関係も見られた．メッセージ数の減少に伴い各物理ノードへのアクセス回数も均等に減少し，特に外れ値的にアクセスの多いノードを減らすことができた．これにより，提案手法によって各物理ノードの負荷を軽減する効果も期待できる．

5.2 今後の課題

今後の課題として最も大きいものが，メッセージ数削減効果のさらなる向上である．本論文では最大でも 5%程度にとどまり，最も理想的な結果であると考えられる仮想ノードを使用しないときの結果とは隔たりが大きい箇所があった．この問題の解決を目指し，以下のことが具体的に挙げられる．

まず，本論文の提案手法はあくまで単一のクエリの転送経路内での重複を防ぐことを目指したものであった．しかし間隔クエリは複数の連続したクエリで構成されるものである．よって，複数のクエリにまたがって内容を先読みするといった間隔クエリ全体で重複を防ぐ仕組みが必要となる．

また，Chord の平均メッセージ数は $O(\log N)$ であり，対数規模ではあるがネットワーク上のノード数が直接メッセージ数に影響する．よって，ネットワーク上にノードを増やす仮想化だけによってメッセージ数を削減することには限界がある可能性も考えられる．ハッシュ値以外での仮想化以外のノードの負荷を軽減するための方法には，動的なノード再割当てがある．これには Chord$^{\#}$ で使用されている負荷の高いノードに割り込む形のもの [7] や，仮想化されたネットワークで平時はデータを持たずに負荷が増加した際には受け皿となる仮想ノードを用意する手法 [11] などが提案されている．こういった他の手法を複数の時間間隔に基づくネットワークに適用してメッセージ数の削減等に繋げられるかの検討についても今後の課題としたい．

謝辞 本研究の一部は福井大学研究育成経費の助成による成果である．

参考文献

[1] Cisco: Cisco Annual Internet Report（2018 〜 2023 年），White paper, Cicso VNI (2020).
[2] 北條真史，長尾洋也，宮尾武裕，首藤一幸：柔軟な経路表に基づく二次元平面上の構造化オーバレイ，情報処理学会論文誌，Vol. 56, No. 2, pp. 439–447 (2015).
[3] Schütt, T., Schintke, F. and Reinefeld, A.: Range queries on structured overlay networks, *Computer Communications*, Vol. 31, No. 2, pp. 280–291 (2008). Special Issue: Foundation of Peer-to-Peer Computing.
[4] Kawakami, T.: A Structured Overlay Network Scheme Based on Multiple Different Time Intervals, *Journal of Information Processing Systems*, Vol. 16, No. 6, pp. 1447–1458 (2020).
[5] 藤樫淳平：オーバレイネットワーク構築のためのネットワークトポロジ提供機構に関する研究，修士論文，奈良先端科学技術大学院大学 (2009).
[6] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F. and Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications, *IEEE/ACM Transactions on Networking*, Vol. 11, No. 1, pp. 17–32 (2003).
[7] Karger, D. R. and Ruhl, M.: Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems, *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 36–43 (2004).
[8] 久保達也，川上朋也：複数の異なる時間間隔に基づく構造化オーバレイネットワークでのノード仮想化の検討，2020 年度情報処理学会関西支部支部大会, G-42, 6 pages (2020).
[9] Aspnes, J. and Shah, G.: Skip Graphs, *ACM Transactions on Algorithms*, Vol. 3, No. 4, pp. 1–37 (2007).
[10] 小西佑治，吉田 幹，竹内 亨，寺西裕一，春本 要，下條真司：単一ノードに複数キーを保持可能とする Skip Graph 拡張，情報処理学会論文誌， Vol. 49, No. 9, pp. 3223–3233 (2008).
[11] Shao, X., Jibiki, M., Teranishi, Y. and Nishinaga, N.: An Efficient Load-Balancing Mechanism for Heterogeneous Range-Queriable Cloud Storage, *Future Generation Computer Systems*, Vol. 78, pp. 920–930 (2018).