Vol.2014-OS-130 No.7
2014/7/28

# ノードの負荷を考慮した Social Cache の配置手法

千野 雄貴[1,a)]　堀江 光[1]　河野 健二[1]

**概要**：Facebook のようなサーバ管理型のソーシャルネットワーク形態に対して分散型ソーシャルネットワーク (DOSN) が注目されている．DOSN はユーザがそれぞれ自身のデータを持ち，ユーザ同士でのデータのやり取りを行う．DOSN での情報配信手段として共有データをキャッシュする役割を持つ Social Cache を利用した SocialCDN がある． SocialCDN はノードの Social Cache を隣接ノードに配置することで，より少ない計算資源でのデータ配信を提供している．しかし，SocialCDN では特定のノードが多数の Social Cache を保持する傾向にあるため， そのノードに負荷が集中する．負荷の集中を防ぐために Social Cache を保持数に上限を設け，それを考慮して Social Cache 配置するノードを決定するアルゴリズム (LSEA) を提案する．提案手法を用いて Social Cache を配置することで既存手法より平均で約 32 % の負荷を軽減できた．

## 1. はじめに

現在 Facebook[1] や Twitter[2] のようなサーバ管理型のソーシャルネットワーク (Online Social Network ; OSN) が注目されている．サーバがユーザ間で共有するデータ (プロフィール画像 etc) を保持，管理し，ユーザからのリクエストに応じてデータを提供する．しかし，サーバに全ユーザのデータが置かれているため，サーバ管理者が自由にデータの変更，閲覧が可能であり，プライバシーが侵害される恐れがある．また，1 つのサーバに多数のユーザデータが保管されているため，サーバの故障やサーバへの攻撃によって多くのユーザデータの流出やユーザ数の偏りによる一部サーバへのアクセスが集中する可能性がある．

その一方で Diaspora [3] や PeerSoN [4] のようなサーバを使用しない分散型のソーシャルネットワーク ( Distributed Online Social Network ; DOSN ) がある．DOSN ではユーザが各々にデータを保持し，ユーザ同士のコネクションを通じて互いに欲しい情報のやり取りを行う．そのため，ユーザが各々のデータ管理や操作を行うことができ， データ流出の可能性も低い．また，データが一カ所に集中しないため，リスクを最小限に抑えることができる．しかし，プロフィール画像や自己紹介文などの友人と共有するデータも各ユーザが持つため，ユーザがオフラインでは，そのユーザのデータにアクセスができずに，データを得ることはできない．そこで, 互いに信頼できる隣接ノードに Social Cache を配置し，共有データ更新の際に予めデータをキャッシュし，そこからデータを取得する Social Butterfly[5] が存在する．あるユーザの持つデータが欲しい時にそのユーザがオフラインであっても Social Cache にアクセスすることで， データを取得できる．Social Cache を利用したデータ取得の例として図 1 に示す．

Social Cache をどの隣接ノードに配置するかの決定方法として SocialCDN[6] を紹介する．Social Cache からデータを取得する際のパスを減らすため，SocialCDN では 1 つのノードに多くの Social Cache を配置する．1 つのノードが多くの Social Cache を保持することで SocialCDN では Social Cache を保持するノード数を最小限に抑え，より少ない計算資源による情報配信を提供する．

SocialCDN のアルゴリズムでは 1 つのノードが保持する Social Cache の数が多くなる．SocialCDN の手法上多くのノードから 1 つのノードへ Social Cache が配置されやすいため，ノードへ負荷が集中する．Facebook のソーシャルグラフに SocialCDN で提案されているアルゴリズムで Social Cache を配置するノードを決定すると，ソーシャルグラフの全ノード数が 4039 にも関わらずあるノードに 1027 の Social Cache が配置されてしまう..

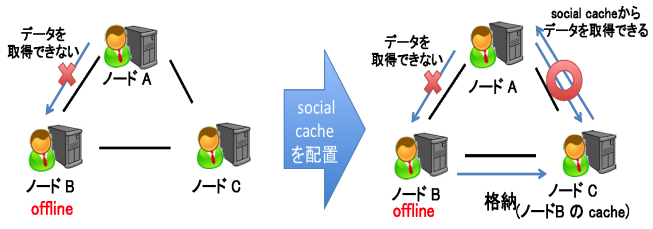


**図 1**　Social Cache を用いた情報配信

[1] 慶応義塾大学
[a)] yuuki@sslab.ics.keio.ac.jp

実存するソーシャルグラフではエッジ数に大きな偏りがあるため，負荷が極端に偏る．こうした負荷集中はデータ取得時のレイテンシ増加や Social Cache を保持するノードの性能低下の原因となる．

本論文では一部のノードへの負荷集中を避けつつ，Social Cache 配置ノードを決定するアルゴリズム (Limited Span Elimination Algorithm： LSEA ) を提案する．各ノードが保持する Social Cache の数に上限を設ける．上限に達していないノードから Social Cache を配置するノードを選出することで一部のノードへ多数の Social Cache が配置されることを防ぐ．上限を一定値に定めるのではなく，Social Cache 配置状況に合わせて可変にすることで，全てのノードに対して Social Cache を配置しつつ，負荷集中を避ける．

LSEA の有用性を示すために，複数のソーシャルグラフについてシミュレーションによる評価実験を行った．本実験ではランダムに生成したソーシャルグラフ (Random) と実存する Facebook，Enron e-mail [7]，Amazon [8] ソーシャルグラフを用い，LSEA と既存の SocialCDN のアルゴリズムで Social Cache を配置した．

4 つのソーシャルグラフで実験した結果 提案手法を用いると，ノードが保有する Social Cache の数を平均で 19 % 〜 49 % 減らすことができた．また，LSEA を用いて Social Cache 配置ノード決定するのに要した時間は既存手法より 4 〜 39 % 増加した．

本論文の構成を以下に示す．第 2 章では，SocialCDN について説明する．第 3 章では，提案手法について述べる．第 4 章では，提案手法の有用性を示すための評価実験について述べる．第 5 章では，本研究と関連する研究について述べる．第 6 章では，結論と今後の課題について述べる．

## 2. SocialCDN

本章では，提案手法のベースとなった研究を説明する．

### 2.1 概要

DOSN ではユーザがオフライン時にはそのユーザのデータを取得することができない．なぜならユーザが各々データを管理，保持しているためである．そこで，オフラインユーザのデータを取得可能にするために SocialCDN がある．DOSN 上のノードに Social Cache を配置し，予め共有するデータを格納しておくことで，ユーザがオフラインになったとしても Social Cache からデータを共有することができる．しかし，Social Cache を無作為に多くのノードに配置するとアクセス頻度の低い Social Cache も発生する．また，常にオンライン状態であるサーバのようなものが存在しないため，Social Cache を保持しているノードがオフラインになる可能性がある．Social Cache をアルゴリズムに沿って配置することによって，より少ない資源で効率の良いデータ配信を提供する．全てのノードの Social Cache を配置することで，Social Cache を保持しているノードがオフライン時もデータ取得を可能にする．

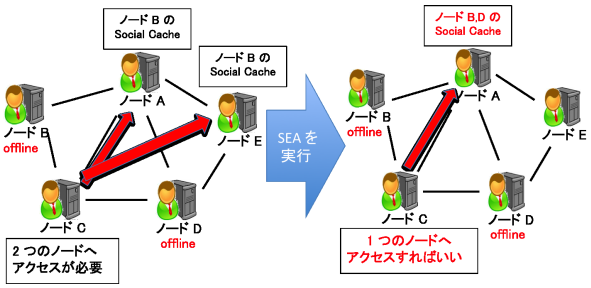


**図 2** Social Cache を配置するノードの違い

共有するデータは主に，プロフィールデータやコメントなどである．データのキャッシュはオフラインになる前だけでなく，プロフィールの変更などデータを更新する度に行う．

第三者からの不正なアクセスによるデータの悪用を防ぐため， Social Cache はお互いが信頼関係にあるユーザ，つまり隣接ノードにのみ配置される．ソーシャルグラフはユーザ同士の友人関係によって形成されるネットワークを示す．そのため，本研究で用いるネットワークは Social Cache を配置するのに最適なためソーシャルグラフを対象としている．また，プライバシーの問題を防ぐため Social Cache へのアクセスは Social Cache を配置したノードの隣接ノードのみに制限する．

Social Cache を配置するノードを決定する方法として Span Elimination Algorithm を使用する．

### 2.2 Span Elimination Algorithm

SocialCDN では Social Cache を配置するノードを Span Elimination Algorithm (SEA) に沿って決定する．SEA では隣接ノードの中でもエッジ数の多いノードに Social Cache を配置することでデータ取得時に必要なアクセス回数を減らすことを目的としている．例として図 2 のようなソーシャルグラフを挙げる．ノード B，D がオフラインでノード B の Social Cache をノード A に配置し，ノード D の Social Cache をノード E に配置されている．このとき，ノード C がノード B，D のデータを取得したい時，ノード C はノード A，E にそれぞれアクセスしなければならない．しかし，ノード B，D の Social Cache をノード C に配置した場合，ノード C はノード A にのみアクセスすれば取得できる．

このように Social Cache を保持するノードが多いと，複数のノードからデータを取得したい際に，複数のノードに配置された Social Cache にアクセスする必要がある．トラヒックの増加を引き起こしパフォーマンスが低下に起因する．そのため，1 つのノードが多くの Social Cache を保

**表 1** アルゴリズム内で使用する言葉の定義

| Word | 意味 |
|---|---|
| Node v | アルゴリズム実行対象ノード |
| Node u | ノード v の隣接ノード |
| commonfriend | ノード v, u それぞれと隣接するノード |
| size 値 | ノードに繋がるエッジ数と隣接ノード同士を繋ぐエッジ数の和 |
| TSC | Social Cache を配置するノードの候補 |

持するように配置することで無駄な資源を削減している

SEA の具体的な動作を示す．SEA は Selection Phase と Elimination Phase の 2 つのフェイズから構成されている．すべてのノードに対してこのアルゴリズムを適用し Social Cache を配置するノードを決定する．尚，アルゴリズム中に使用する言葉の定義を表 1 にまとめる．

Selection Phase では Social Cache を配置するノードの候補 (Temporary Social Cache ; TSC ) を選出する．ノードに繋がっているエッジの数を基準に TSC を決定する．まず，全ノードの size 値 を計算する． Node v に繋がっているエッジの 1 つに注目し，そのエッジによって繋がっている隣接ノード を Node u とする．Node v と Node u のどちらとも繋がっているノードを commonfriend とし，Node u, commonfriend の中から最も size 値 の大きいノードを tsc に選出する．Node v の他のエッジに対しても同様に TSC を選出する．

Elimination Phase では，TSC に選ばれたノードの中から，Social Cache を配置するノードを決定する．Selection Phase によって TSC に選出された回数を比較し，最も回数が大きかったノードに Node v の Social Cache が配置される．Elimination Phase で選出頻度を考慮することによって Social Cache を保持するノードを少なくし，できる限り 1 つのノードに多くの Social Cache を配置できる．

Span Elimination Algorithm の疑似コードを Algorithm 1 に示す．

SEA を使用して Social Cache 配置ノードを決定すると，より少ないアクセス数によるデータ取得のため， 1 つのノードに多くの Social Cache が配置され，Social Cache 保持数が大きくなる．しかし，Social Cache は配置されたノードの資源を利用して共有データを保存するため，保持数が過度に多いと Social Cache を保持しているノードに多大な負荷がかかってしまう．

## 3. 提案

本章では，Social Cache によるノードの負荷を考慮した Social Cache 配置ノード決定アルゴリズム : Limited Span Elimination Algorithm ( LSEA ) について説明する．LSEA は Social Cache 保持数が集中しないように Social Cache 配置ノードを決定する．

### 3.1 基本動作

LSEA はノードの Social Cache 保持数に上限を与える．

**Algorithm 1** SEA の疑似コード

```
// 初期条件
Node v, u, commonfriend, tsc, candidate[], socialcache[];
int size[];
int frequency[]; // TSC に選出された回数
boolean flag;
// size 値計算
for all v do
  size[v.Getid()] = SizeCalculate(v);
end for
for all v do
  // Selection phase
  //初期化
  frequency[] = 0;candidate[] = φ;
  for all Edge(v) do
    u = AnotherVertex(edge);
    TSC=HighestVertex(size[u.Getid()],size[commonfriend
    .Getid()]);
    frequency[tsc.Getid]++;
    candidate.Add(tsc)
  end for// Elimination phase
  SocialCache[v.Getid()]=
  HighestVertex(frequency[candidate[].Getid()]);
  //ノード V の Social cache 配置ノード決定
end for
```

なぜなら, 隣接するノードの中から size 値を考慮して Social Cache 候補を選出すると，同じノードが高頻度で選出されやすく Social Cache 保持数が大きくなる．そのため，もし上限に達しているノードがある場合，そのノード以外から Social Cache の候補を size 値を考慮して選出する．その後， SEA と同じように全てのエッジの Social Cache 候補を選出し，候補に選出された頻度から Social Cache を決定する．

### 3.2 最適な上限値

最適な上限値を動的に決定するため上限値を可変にし，エッジ数の少ないノードから Social Cache 配置ノードを決定する．なぜなら，上限が一定であると隣接ノード全てが上限に達し Social Cache を配置できず，上限を可変にしただけでは上限値が過度に上昇するためである．例えば，予め一定の上限を設定しても DOSN はユーザ数やエッジ数が増減するため，それが常に最適とは限らない．また，エッジ数が多いノードとエッジ数 1 のノードでは Social Cache 候補になりうるノードの数に大きな差がでる．もし，エッジ数の多いノードが先に Social Cache 配置ノードを決定すると，エッジ数 1 のノードは Social Cache 候補になりうるノードが上限に達しているため，上限を上げ再度選出を行う．しかし，エッジ数 1 のノードから配置ノードを決定すると，エッジ数の多いノードは他にも候補となるノードがいるため，上限を上げずに Social Cache 配置ノードを決定できる．

そのため，LSEA では上限を可変に設定し，エッジ数の

**Algorithm 2** LSEA の疑似コード

```
// 初期条件
Node v, u, commonfriend, tsc, socialcache[], candidate[];
int size[],i;
int frequency[];
Node noupper[]; // 上限に達していないノードの集合
Strictcount = 1 // 上限の初期値は 1 とする
for all v do
  // 全ノードの size 値計算
  Size[v.Getid()] = SizeCalculate (v) ;
end for
for all v do
  AscendingSort(v);//エッジ数が少ない順に並び替える
end for
for all v do
  // Selection phase
  //初期化
  frequency[] = 0; candidate[] = φ;
  noupperedcandidate[] = φ
  for all Edge(v) do
    u = AnotherVertex(edge);
    noupper.add(u,v,commonfriend);
    for all i = 0 to noupper.length() do
      if Strictcount == noupper[i].Havesocialcache() then
        noupper[i].remove; // 上限に達したノードを候補から
        外す
      end if
    end for
    TSC = HighestVertex(noupper[]);
    frequency[TSC.Getid()]++;
    candidate.add(TSC);
  end for
  // Elimination phase
  SocialCache[v.Getid()]
  =HighestVertex(frequency[candidate[].Getid()]);
  //ノード v の Social cache 配置ノード決定
  if SocialCache[v.Getid()] == NULL then
    Strictcount++;
    Jumpselectionphase(); // 再び social cache 配置を行う
  end if
end for
```

少ないノードから Social Cache 配置ノードを決定していくことで, 全てのノードの Social Cache を配置可能にし, 最適な上限値を得ることができる.

これらの方法全て取り込んだ LSEA の動作を疑似コードで Algorithm 2 に示す.

## 4. 実験

提案手法 ( LSEA ) を用いて Social Cache を配置することでノードへの負荷集中を軽減できることを示す. また, LSEA のオーバーヘッドを評価するために, SEA を用いた場合の配置にかかる処理時間との差を比較する.

### 4.1 実験方法

4 つのソーシャルグラフに対して SEA と LSEA を用いて Social Cache を配置するシミュレーションを行い, 得られた結果を比較する. 本実験では, ノードへの負荷集中を軽減できたかをノードの Social Cache を保持している数から評価する. 本実験で使用するソーシャルグラフは必ずエッジが 1 以上のノードのみで構成されており, Social Cache は互いに信頼関係にあるノードのみに配置されるため, エッジは双方向で表される.

**表 2** 使用するソーシャルグラフ

| ソーシャルグラフ | 総ノード数 | 総エッジ数 | 最大エッジ数 | 最小エッジ数 |
|---|---|---|---|---|
| Random | 4.039 | 88,234 | 66 | 21 |
| Facebook | 4,039 | 88,234 | 1,045 | 1 |
| Enron e-mail | 36,692 | 183,831 | 1,383 | 1 |
| Amazon | 334,863 | 925,872 | 549 | 1 |

本実験の対象となるソーシャルグラフは, Random, Facebook, Enron e-mail, Amazon の 4 つである. Facebook, Enron e-mail, Amazon のソーシャルグラフはスタンフォード大学で集計されたデータセット [9] を使用した. また, これら 4 つのソーシャルグラフを対象に SEA, LSEA を用いて Social Cache 配置ノードを決定するのにどのくらいの時間を要したかをそれぞれ計測する.

使用するソーシャルグラフの総ノード数, 総エッジ数などの情報を表 2 に示す. Random ソーシャルグラフは, ランダム関数を用いて双方向のエッジを設定し自作したソーシャルグラフである. Facebook ソーシャルグラフとの対比を明確にするため, 総ノード数, 総エッジ数を同じにした. Random , Facebook ソーシャルグラフのエッジ数を降順に並び替えグラフ化し, 図 3 に示す. それぞれ総ノード数, 総エッジ数は同じだが Random ソーシャルグラフとは違い, Facebook ソーシャルグラフはエッジ数に大きな偏りが見られた. 友人関係にあるユーザの数はユーザ毎に大きく変わるため, Facebook ソーシャルグラフではエッジ数が大きく偏っていると考えられる.

Enron e-mail ソーシャルグラフは Enron 社での一定期間内に送受信された e-mail の様子をグラフ化したグラフで, Amazon ソーシャルグラフは, Web サイト Amazon でよく同時購入される商品同士をエッジで表したグラフを示している. Enron e-mail, Amazon ソーシャルグラフも Facebook, Random と同様にそれぞれグラフ化し, 図 4, 図 5 に示す. これらも Facebook 同様にエッジ数に大きな偏りが見られた. Facebook, Enron e-mail, Amazon ソーシャルグラフのような実存するソーシャルグラフではノード毎のエッジ数は大きく偏っている.

### 4.2 実験結果

全てのソーシャルグラフにおいて LSEA を用いて Social Cache 配置ノードを決定した場合, 特定ノードへの Social Cache 保持数の集中を改善することができた. LSEA を使用して Social Cache を配置するノードを決定するのに要

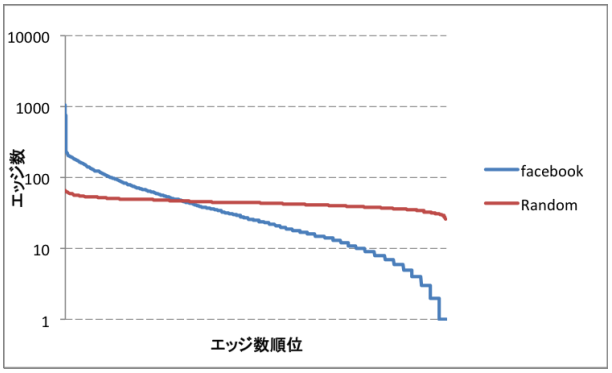


**図 3** Random ソーシャルグラフのエッジ数の分布

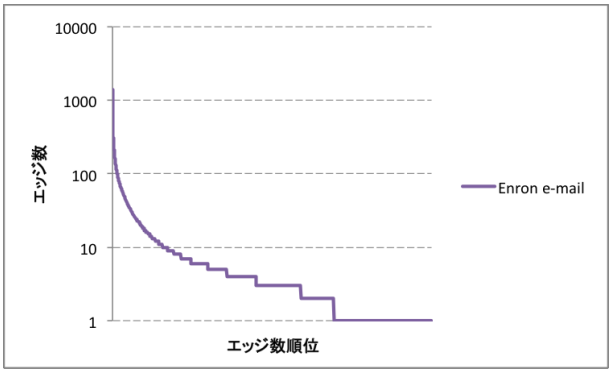


**図 4** Enron e-mail ソーシャルグラフのエッジ数の分布

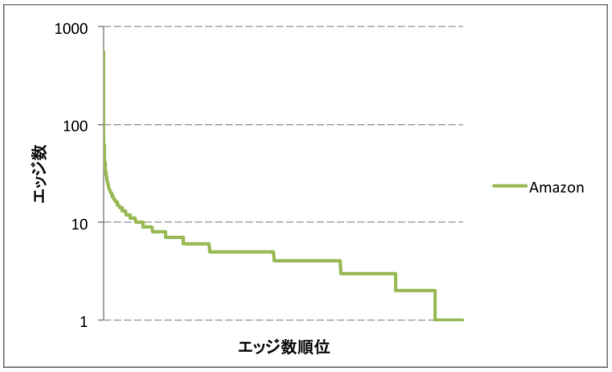


**図 5** Amazon ソーシャルグラフのエッジ数の分布

**表 3** Social Cache は位置ノード決定に要した時間（ms）

| ソーシャルグラフ | LSEA | SEA |
|---|---|---|
| Random | 27,010 | 25,914 |
| Facebook | 156,445 | 152,484 |
| Enron e-mail | 488,576 | 352,192 |
| Amazon | 39,635 | 33,055 |

した時間を SEA を使用した場合と比較すると表 3 の結果を示した．表 3 より Social Cache 配置ノード決定に要した時間は LSEA の方が 4 〜 39 % 増加した．これは SEA に比べてアルゴリズム内で上限を考慮するのに要した時間，上限に達してしまい Social Cache が配置されなかったノードの上限を増やして再度実行するのに要した時間が主な増加の原因だと考えられる．

### 4.2.1 Random ソーシャルグラフ

Random ソーシャルグラフに LSEA，SocialCDN の手法 SEA を用いて Social Cache 配置ノードを決定した場合の Social Cache 保持数を以下の図 6 に示す．また，それぞれの Social Cache を保持しているノード数，平均 Social Cache 保持数，最大 Social Cache 保持数を表 4 にまとめた．

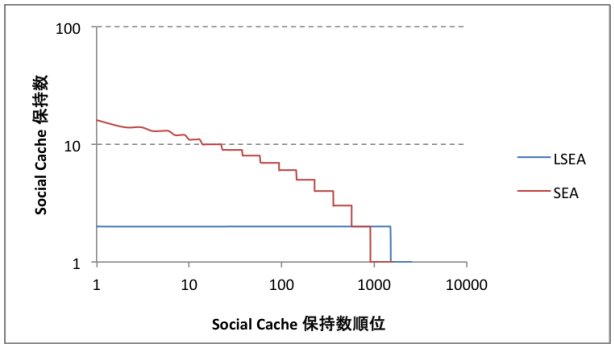


**図 6** Random ソーシャルグラフに SEA,LSEA を用いた場合の Social Cache 保持数

**表 4** Random における Social Cache の最大保持数と平均保持数

| Random ソーシャルグラフ | Social Cache を保持するノード数 | 平均保持数 | 最大保持数 |
|---|---|---|---|
| 制限あり | 2,531 | 1.596 | 2 |
| 制限なし | 1,587 | 1.962 | 16 |

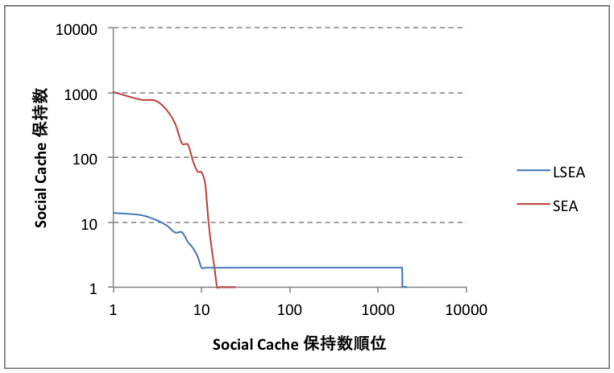


**図 7** Facebook ソーシャルグラフに SEA,LSEA を用いた場合の Social Cache 保持数

**表 5** Facebook における Social Cache の最大保持数と平均保持数

| Facebook ソーシャルグラフ | Social Cache を保持するノード数 | 平均保持数 | 最大保持数 |
|---|---|---|---|
| 制限あり | 2,101 | 1.922 | 14 |
| 制限なし | 24 | 3.801 | 1027 |

SEA を使用する時より最大 Social Cache 保持数を 88 % 削減し，平均で 19 % 減少した．しかし，SEA を使用した場合でも Social Cache 保持数に大きな偏りは見られなかった．これはソーシャルグラフ自体をランダム関数を用いて作成したため，エッジ数に大きな偏りがなかったことが起因する．

### 4.2.2 Facebook ソーシャルグラフ

Random ソーシャルグラフに LSEA ，SEA を用いて Social Cache 配置ノードを決定した場合のそれぞれの Social Cache 保持数を以下の図 7 に示す．また，それぞれの Social Cache を保持しているノード数，平均 Social Cache 保持数，最大 Social Cache 保持数を表 5 にまとめた．

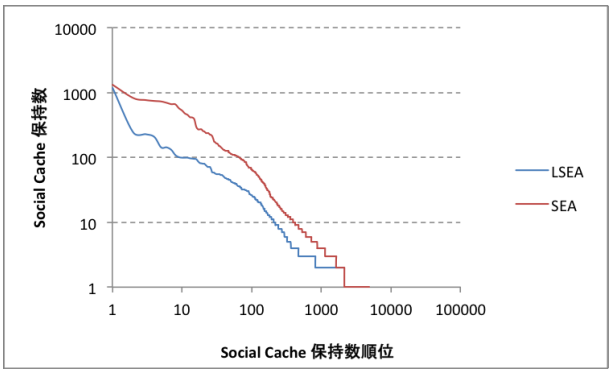


**図 8** Enron ソーシャルグラフに SEA,LSEA を用いた場合の Social Cache 保持数

**表 6** Enron e-mail ソーシャルグラフにおける Social Cache の最大保持数と平均保持数

| Enron e-mail ソーシャルグラフ | Social Cache を保持するノード数 | 平均保持数 | 最大保持数 |
|---|---|---|---|
| 制限あり | 16,603 | 2.210 | 1,187 |
| 制限なし | 4,843 | 3.422 | 1,317 |

SEA を使用する時より最大 Social Cache 保持数を 99 % 削減し，平均で 49 % 減少した．Random ソーシャルグラフとは違い，Facebook ソーシャルグラフに対して LSEA を用いた場合の方がよりノードの Social Cache 保持数を分散することができ，ノードへの負荷が分散できていることがわかる．実存するソーシャルグラフではエッジ数に大きな偏りがあるため SEA では特定のノードに Social Cache が集まる傾向がある．そこで，LSEA を用いることで，特定ノードへの負荷の集中を避けつつ Social Cache を配置にすることを可能にした．

#### 4.2.3 Enron e-mail ソーシャルグラフ

Enron e-mail ソーシャルグラフに LSEA , SEA を用いて Social Cache 配置ノードを決定した場合の Social Cache 保持数を以下の図 8 に示す．また，それぞれの Social Cache を保持しているノード数，平均 Social Cache 保持数，最大 Social Cache 保持数を表 6 にまとめた．

SEA を使用する時より最大 Social Cache 保持数を 10 % 削減し，平均で 35 % 減少した．Facebook ソーシャルグラフとは違い，LSEA を使用しても最大 Social Cache 保持数を大きく削減することができず，一部のノードだけ Social Cache 配置による負荷が集中したままとなってしまった．Facebook 同様にエッジ数に大きな偏りがでたソーシャルグラフにも関わらず，特定のノードに対して LSEA が効果を発揮しなかった理由はエッジ数 1 のノードの集中だと考えられる．LSEA を用いても Social Cache 保持数にあまり変化がみられなかったノードはどのような隣接ノードであるかを調べるために Social Cache 保持数上位 3 つのノードの隣接ノードを表 7 にま示す．

表 7 から LSEA を使用しても負荷が分散できなかったノードの隣接ノードはエッジ数 1 のノードであることがわかる．Social Cache はどのノードがオフラインになっても

**表 7** LSEA 使用時の Social Cache 保持数の高いノード上位 3 つの隣接ノードの特徴

| Social Cache 保持数 | ノード番号 | 隣接ノードの平均エッジ数 | 隣接ノードの最大エッジ数 |
|---|---|---|---|
| 1187 | 5038 | 1 | 1 |
| 242 | 588 | 1 | 1 |
| 228 | 893 | 1 | 1 |

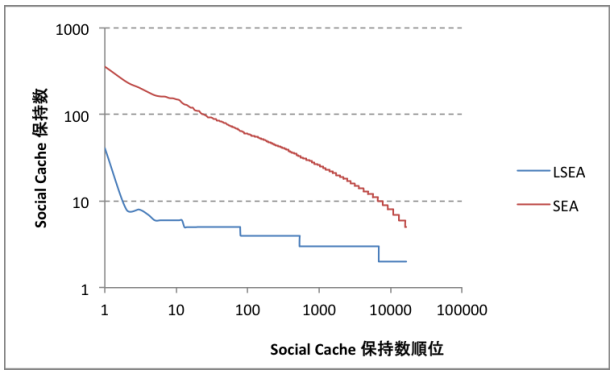


**図 9** Amazon ソーシャルグラフに SEA,LSEA を用いた場合の Social Cache 保持数

**表 8** Amazon における Social Cache の最大保持数と平均保持数

| Amazon ソーシャルグラフ | Social Cache を保持するノード数 | 平均保持数 | 最大保持数 |
|---|---|---|---|
| 制限あり | 194,738 | 1.720 | 41 |
| 制限なし | 91,841 | 2.337 | 361 |

データ取得を可能にするため全てのノードに対して Social Cache を配置する．そのため，エッジ数が 1 しかないノードも Social Cache を配置する．しかし，それらのノードは Social Cache 候補となるノードが 1 つしかないため，上限を設けても必ずそのノードが選出されてしまう．そのため，特定のノードは Social Cache 保持数を抑えることができず，負荷の集中が起きてしまう．

#### 4.2.4 Amazon ソーシャルグラフ

Amazon ソーシャルグラフに LSEA を用いて Social Cache 配置ノードを決定したときのそれぞれの Social Cache 保持数を以下の図 9 に示す．また，それぞれの Social Cache を保持しているノード数，平均 Social Cache 保持数，最大 Social Cache 保持数を表 8 にまとめた．

SEA を使用時より最大 Social Cache 保持数を 89 % 削減し，平均で 26 % 減少した．Facebook ソーシャルグラフ よりも約 80 倍近くノード数が増えたにも関わらず，Facebook 同様に LSEA を使用することで Social Cache 保持数を減らすことができ，ノードへかかる負荷の集中も改善することができた．

### 4.3 まとめ

4 つのソーシャルグラフそれぞれに LSEA を用いて Social Cache 配置ノードを決定すると Social Cache 保持数を大幅に削減することができた．特に，最も負荷が集まっていたノード，Social Cache 保持数が最大のノードは保持数を全てのソーシャルグラフにおいて約 70 % 削減することができた．Random ソーシャルグラフよりも実

存するソーシャルグラフの方が LSEA を用いることでより Social Cache による負荷の集中を抑えることができた．Enron e-mail ソーシャルグラフ内の特定ノードのようにエッジ数が多くても隣接ノードがエッジ数 1 のノードで構成されているノードは，Social Cache 候補になりうるノードが 1 つしかないため LSEA を使用しても Social Cache 保持数を下げることはできなかった． また，LSEA を使用して Social Cache 配置ノードを決定すると SEA を使用したときよりも 4 〜 39 % 決定に要した時間が増加した．

## 5. 関連研究

Social Butterfly は，DOSN での通信トラヒックを削減しつつ，オフラインノードへの対応する効率の良いデータ配信方法を提供する．ソーシャルネットワーク上のエッジを利用して， 隣接ノードに Social Cache を配置していくことで，他の従来あるデータ配信手法よりもスループットの向上，トラヒックの削減を可能にした．しかし，Social Cache の配置するノードの決定方法としてソーシャルグラフ全体を俯瞰して最もデータ取得効率の良いノードに配置するような理想的アルゴリズムを採用している． 一方，本研究では各ノードがグラフ全体の構造を知ることなく効率的に Social Cache の配置ができるため，DOSN ではより向いている．

DOSN において高いスループットでデータを散布するために HFLOODING [10] という手法が提案されている．これはデータを更新した際に，どのノードにすでに更新情報を送ったノードを示すリストを同時に送ることで，同じ相手に更新情報を送信することを回避することができる．しかし，この通信では更新情報が必要ないユーザにも配信されるため，ネットワーク全体に大きな負荷がかかってしまう．一方，本研究では 更新時に Social Cache に格納し，データが欲しいユーザのみが Social Cache にアクセスするため，ネットワークにかかる負荷を少なく済む．

P2P ネットワークの共有データの速い伝播方法に PROB [11] がある．PROB は重みを考慮た WOM propagation でデータの送受信を行う．WOM propagation は共有データを更新，アップロードした際に，隣接ノードに知らせ，その隣接ノードが他の隣接ノードに知らせることで，高いスループットで伝播を行う．PROB では隣接ノードに自身の保持するデータや過去ダウンロードしたデータなどいろいろな情報からノードに重み付けをし，速く効率の良い通信伝達を提供する．しかし，この研究ではノードの重み計算を行う際に，ネットワーク中から情報を集めるためサーバを必要とする．一方，本研究では各ノードがグラフ全体の構造を知る必要がないため，サーバのない DOSN にはより適している．

My3 [12] という分散型 OSN では，データのレプリカを作成してオフライン時でも高いスループットでデータ取得を可能にする．これはデータのレプリカをノード間で物理的距離の近いノードに配置し，オフライン時でもレプリカからデータ取得可能にする．しかし，この手法では予めオンライン時間が定まったネットワークを対象としている．DOSN ではいつどのノードがオフラインになるかの予測が難しいため，全てのノードに対して Social Cache を配置する本研究の方が向いている．

## 6. 終わりに

### 6.1 結論

Social Cache 保持数に上限を設けることでノードへの負荷の集中を抑制する Social Cache 配置アルゴリズム ( Limited Span Elimination Algorithm : LSEA ) を提案した．4 つのソーシャルグラフをシミュレートした結果，既存手法より平均で負荷を 32 % 減らすことができた．また，Social Cache 配置ノードを決定するのに要した時間は 4 〜 39 % の増加であった．

### 6.2 今後の課題

DOSN ではノード数，エッジ数が増減するため，ソーシャルグラフが動的に変化する場合にも本手法を適用できるようにする必要がある．シミュレーションではなく Social Cache の保持数がノードに及ぼす実際の負荷を計測し，提案手法の効果を評価する．Enron e-mail ソーシャルグラフで存在したように，エッジ数 1 のノードが多数隣接しているノードは LSEA の手法上，負荷を軽減することができなかった．そのようなノードへの対応方法を検討する必要がある．

### 参考文献

[1] Facebook. http://www.facebook.com/.
[2] Twitter. http://www.twitter.com/.
[3] Diaspora. https://joindiaspora.com/.
[4] Sonja Buchegger, Doris Schioberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2p social networking: early experiences and insights. In *SNS '09 Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52, March 2009.
[5] Lu Han, Badri Nath, Liviu Iftode, and S.Muthukrishnan. Social butterfly: Social caches for distributed social networks. In *IEEE International Conference on Social Computing 2011*, pages 81–86, October 2011.
[6] Lu Han, Magdalena Punceva, Badri Nath, S.Muthukrishman, and Liviu Iftode. Socialcdn : Caching techniques for distributed social networks. In *IEEE International Conference on Peer-to-Peer Computing*, pages 191–202, September 2012.
[7] Enron. http://www.enron.com/.
[8] Amazon. http://www.amazon.com/.
[9] STANFORD UNIVERSITY. http://snap.stanford.edu/data/.
[10] Giuliano Mega, Alberto Montresor, and Gian Pietro Picco. Efficient dissemination in decentralized social networks. In *IEEE International Conference on Peer-to-*

*Peer Computing*, pages 338–347, August 2011.

[11] Zhi Yang and Yafei Dai. Prob : a lightweight approach for fast content propagation in p2p networks. In *IEEE International Conference on Peer-to-Peer Computing*, pages 1–10, September 2013.

[12] Rammohan Narendula, Thanasis G. Papaioannou, and Karl Aberer. My3: A highly-available p2p-based online social network. In *IEEE International Conference on Peer-to-Peer Computing*, pages 166–167, August 2011.