Vol.2013-HPC-141 No.17
2013/10/1

Pregel グラフ処理系におけるメッセージ配送最適化に向けて

上野晃司[†1]　　鈴村豊太郎[†2]

Pregel は，バルク同期並列をベースとしたプログラミングモデルの採用した大規模分散グラフ処理フレームワークである．Pregel では，グラフの頂点間でメッセージをやり取りして，計算を進めていく．本発表では，Pregel 処理系において，各頂点が全ての隣接頂点に同じメッセージを送る場合に適用できる最適化手法を提案する．PageRank と BFS で行った性能評価の結果，PageRank では性能が 1.57 倍と向上したが，BFS では性能が 0.93 倍と下がってしまった．

Toward the Optimized message communication method for Pregel Graph Processing System

KOJI UENO[†1]　TOYOTARO SUZUMURA[†2]

Pregel is a framework for large-scale graph processing that utilizes BSP (Bulk Synchronous Parallel) based programming model. We propose optimization techniques for the message communication of the Pregel framework. This technique can be applied when all the vertexes send the same message to all of their neighbors. Our performance evaluation shows that our method can compute 1.57 times faster for PageRank but only 0.93 times for BFS.

1. はじめに

グラフ処理とは，グラフで表された構造のデータに対する処理である．近年，ビッグデータの解析の重要性が叫ばれる中，グラフ処理でも，ソーシャルネットワーク解析，金融マーケットの解析，サイバーセキュリティ，脳科学シミュレーション，など大規模なグラフの処理が重要になってきている．１ノードのメモリに入りきらない大規模グラフを，PC クラスタなどの分散環境で処理するため，2010 年に Google の Malewicz らが Pregel[1]グラフ分散処理系を提案した．Pregel のシンプルでありながら強力なプログラミングモデルから，大規模グラフの分散処理におけるプログラミングモデルとして，一般に受け入れられ，Pregel のオープンソース実装である Giraph[2], GPS[3], XPregel[4]などが開発されるなど，Pregel の産業利用に向けた開発，処理系の高速化に関する研究，ともに活発になされるようになって来ている．

本論文は，Pregel 処理系における，ノード間通信データ量を削減し，メッセージ配送を高速化する手法を提案する．本論文の貢献は，Pregel 処理系における新しいメッセージ配送方法，及び，それを実現するアルゴリズム，データ構造の提案である．

以降，２章では，Pregel 処理系を説明し，メッセージ配送高速化の必要性を述べる．３章では，提案手法の詳細を解説する．４章では，提案手法の XPregel への実装方法について説明する．５章で，４章で行った実装の性能評価を示す．６章で関連研究，７章でまとめと今後の展望を述べる．

†1 東京工業大学／JST CREST
Tokyo Institute of Technology / JST CREST
†2 東京工業大学／IBM 東京基礎研究所／JST CREST
Tokyo Institute of Technology / IBM-Research Tokyo / JST CREST

2. 背景

2.1 Pregel

まず，Pregel の計算方法を説明する．Pregel は，スーパーステップを繰り返して計算する．各スーパーステップ S で，グラフの各頂点 V に対して，ユーザ定義関数を呼び出す．この関数は，前のスーパーステップで頂点 V に送られたメッセージを受け取り，頂点 V の内部状態を変更し，他の頂点にメッセージを送信することができる．メッセージはいくつ，どの頂点に送信してもよい．ただし，通常，グラフ処理においては，隣接頂点にメッセージを送ることが多い．

また，送信されたメッセージはすぐに届くのではなく，次のスーパーステップで，メッセージが送られた頂点に対してユーザ定義関数が呼び出された時に，初めて受け取ることができる．繰り返しは，メッセージが１つも送信されなくなったら終了，または，繰り返しの回数を指定するなど，何らかの条件が満たされたら終了するようにする．

Pregel 処理系において，最も重要であり，時間のかかる処理は，メッセージの配送である．複数のノードを使って分散処理している場合，メッセージはノードを超えて送信されるようになるので，一般に，メッセージの数は，ほぼそのままノード間通信データ量となる．

2.2 大きな差をつけられた Pregel

近年，Graph500 ベンチマーク[5]の登場により，分散 BFS（Breadth-first search, 幅優先探索）を高速に計算する手法が研究されてきた．Graph500 ベンチマークでは，新しいアルゴリズムが次々と提案され，性能が急速に伸びた経緯が

ある．現在，最適化された BFS の実装と，Giraph などの Pregel 処理系で実装した BFS とでは，速度が数桁異なるという，大きな性能差が出ている状態である．確かに，Pregel は汎用グラフ処理系なので，特定のアルゴリズムに最適化された実装よりある程度遅くなってしまうのは仕方ないが，数桁という大きな性能差を克服し，特定のアルゴリズムに最適化された実装の性能に近づけるには，メッセージ配送手法にある程度抜本的な改革が必要である．

3. 提案手法

グラフアルゴリズムの中には，すべての隣接頂点に同じメッセージを送るという計算方法を使うものが多くある．例えば，PageRank や BFS である．PageRank は，現在の頂点の値を，メッセージとして，すべての隣接頂点に送信する．BFS の場合は，頂点の ID を，メッセージとしてすべての隣接頂点に送信する．このような場合，従来の Pregel 処理系では，メッセージを行き先の頂点ごとにコピーして，転送していたが，同じメッセージが同じノードに複数届く場合，重複したメッセージは減らせるはずである．

本論文では，すべての隣接頂点に同じメッセージを送る場合の，最適化手法を提案する．この提案手法は，一部の隣接頂点にしか送らない場合や，頂点ごとに異なるメッセージを送る場合には，適用できない．

3.1 ランダムグラフにおけるメッセージ減少割合

もし，各頂点がすべての隣接頂点に同じメッセージを送ると仮定して，必要最小限のメッセージ通信量を計算してみる．行列 A を，処理したいグラフの隣接行列，P をグラフの分割数（＝Pregel を走らせるノード数），n をグラフの頂点数，k をグラフの平均次数とする．グラフは，通信データ量を減らすという目的からは不利だと思われる，ランダムグラフであるとする．Pregel では，頂点数がほぼ均等になるように頂点を各ノードに分散するので，各ノードは，頂点を n/P 個担当している．もし，すべての隣接頂点にメッセージを 1 つずつ送信したとすると，各ノードがノードをまたがって送信するメッセージの数は，

$$\frac{nk}{P}\cdot\frac{P-1}{P} \tag{1}$$

となる．

行列 A'を，行列 A から適当な m 列を抜き出して作った行列であるとする．行列 A'のある行に，エッジが 1 つ以上存在する確率は，グラフがランダムグラフであることから

$$\gamma(m)=1-\left(\frac{n-1}{n}\right)^{mk} \tag{2}$$

である．ある頂点が，すべての隣接頂点にメッセージを送るとき，あるノードにメッセージを送る必要がある確率は，

$$\gamma\left(\frac{n}{P}\right) \tag{3}$$

である．よって，各頂点が，すべての隣接頂点に同じメッセージを送る場合の，必要最小限の通信する必要のあるメッセージ数は，

$$n\cdot\gamma\left(\frac{n}{P}\right)\cdot\frac{P-1}{P} \tag{4}$$

となる．

頂点数 n=100 万，平均次数 k=32 とした場合，$\frac{(4)}{(1)}$ の値を表 1 に示した．なお，平均次数 32 は，Graph500 ベンチマークで使われるグラフを同じである．

表 1　頂点数 n=100 万，平均次数 k=32 のランダムグラフの場合に削減可能なメッセージ数の割合

Table 1　Reduction of transferring messages for a 1 million vertices random graph whose average vertex degree is 32.

ノード数	通信メッセージ数 減少割合
2	0.0625
4	0.125
8	0.245
16	0.432
32	0.632
64	0.787

8 ノード以下の場合，通信されるメッセージ数を，4 分の 1 以下にすることができることが分かる．また，ここではランダムグラフで計算したが，スケールフリー性のあるグラフの場合は，次数の偏りが大きいことから，より大きく減少させることが可能であると予想される．

3.2 アルゴリズム・データ構造

次に，このメッセージ通信量を実現するアルゴリズム・データ構造を説明する．やりたいことは，短く述べると次のようなことである．メッセージの送信者ノードは，メッセージの送り先頂点を指定してメッセージを送るのではなく，メッセージを必要としているノードに向けてメッセージを送信する．メッセージは必要なノードが受け取り，受け取ったノードが頂点まで配送する．

API として，SendMessageToAllNeighbors 関数[1]が呼ばれた時に，本論文の提案手法によるメッセージ配送が行われるようにすれば良い．まず，メッセージを記憶しておくための入れ物として，担当する頂点数 $\frac{n}{P}$ の長さの配列 HM を

用意しておく．また，その配列にメッセージが入っているかどうかを表すため，担当する頂点数分のビット数があるビットマップ HB を用意しておく．SendMessageToAllNeighbors が呼ばれた時，HM の送信元の頂点に対応する場所にメッセージを入れ，HB の対応するビットを立てる．

メッセージの送信処理は，アルゴリズム 1 で行う．アルゴリズム 1 において，NBp は，自ノードが担当する各頂点から，ノード p が担当する頂点のどれかにエッジが張られているかどうかを計算しておいたビットマップである．自ノードが担当しているある頂点 v について，NBp[v]のビットが立っている場合，頂点 v から，ノード p が担当する頂点のどれか 1 つ以上にエッジが張られていることになるので，頂点 v からすべての隣接頂点にメッセージが送られる場合，ノード p にメッセージを送信する必要がある．SMp は，自ノードからノード p に向けたメッセージを格納するバッファ．SBp は，自ノードからノード p に向けたメッセージがあるかどうかを表すビットマップ．SM/SB は，全ノード分 SMp/SBp のことを表している．また，ワードは，計算機が整数演算をする場合に得意とする長さのワードである．（最近の 64bit 対応マシンなら，1 ワード=64bit）

アルゴリズム 1 の通信処理は，受信したメッセージ RM，受信したメッセージがどの頂点から来たかを表すため，グラフ全体の頂点数分の長さを持ったビットマップ RB，そして，頂点から対応するメッセージを高速に引くためのデータ RO の 3 つを返す．

アルゴリズム 1: 通信処理アルゴリズム

```
1   foreach p in 全ノード
2   |  foreach v in 担当する頂点
3   |  |  if(HB[v] & NBp[v])
4   |  |  |  HM[v]を SMp に追加
5   |  |  |  SBp[v] <- true
6   RM <- alltoallv(SM)
7   RB <- alltoall(SB)
8   RO[0] <- 0
9   foreach w in RB の全ワード
10  | RO[v+1] <- RO[v] + (w の立っているビット数)
```

各頂点に届いたメッセージを受け取る処理は，アルゴリズム 2 に示す．BPW は 1 ワードあたりのビット数，popcount(word)は word の立っているビット数を返す関数，受け取ったメッセージ M を返す．

アルゴリズム 2: メッセージ取得アルゴリズム

```
1   foreach u in 頂点 v の隣接頂点
2   |  if(RB[u])
3   |  |  mask <- (1 << (u % BPW)) - 1
4   |  |  word <- RB の (u/BPW) 番目のワード
5   |  |  index <- RO[u/BPW] + popcount(word & mask)
6   |  |  RM[index]を M に追加
```

4. 実装

前章で提案した手法を，我々が開発している Pregel 処理系 XPregel[4]に実装した．XPregel は，性能に特化して開発している処理系で，スパコンを使って動かした時，スパコンの高速なインターコネクトを十分活かすことができることを目標に開発が続けられている．Giraph や GPS は，Java で実装されているので，スパコンの高速なインターコネクトを十分に活かすことは難しいが，XPregel は，X10[6]で実装されているので，その点が有利である．X10 では，通信に MPI の集団通信を使うことも可能で，実際に XPregel は，MPI の集団通信を使って実装されている．本提案手法の実装にあたっては，アルゴリズム 1 の alltoall(v)のところを，MPI の集団通信を使って実装した．

5. 性能評価

XPregel を使って，提案手法の性能評価を行った．環境は，東京工業大学のスパコン TSUBAME2.0[8] 4 ノード，1 ノードに付き Xeon X5670 (2.93 GHz) が 2 ソケット，ネットワークは，QDR InfiniBand x2 (8GB/s) である．

表 2 は提案手法と，提案手法を使わない naive な方法とを比較したものである．naive な手法では，combine は用いていない．グラフは，頂点数 1677 万，エッジ数 2 億 6844 万（Scale 24）の RMAT グラフ，RMAT の生成パラメータは，A=0.45, B=0.15, C=0.15, D=0.25 とした．RMAT は，スケールフリー性のあるグラフを生成するジェネレータである．

表 2　提案手法と naïve な手法による，PageRank と BFS の計算時間（秒）

Table 2　Computing time for the PageRank and the BFS with the proposed method and the native method (seconds)

	提案手法	Naïve
PageRank	115.05	180.98
BFS	16.280	15.072

PageRank だと，提案手法の方は naive の 1.57 倍の速度であり，提案手法が有効であることが分かる．しかし，BFS だと提案手法の方が遅いという結果になった．これは，PageRank だと，スーパーステップ毎に全エッジをメッセージが流れるのに対して，BFS は，各スーパーステップでは一部のエッジにしかメッセージが流れないからである．提案手法だと，流れるメッセージが多くても・少なくても，固定分の計算量がかかってしまい，少ないと不利になってしまう．それに対して，naive な手法だと，計算量はほぼメッセージ数に依存するので，メッセージが少ない場合，高速に終了させることができる．

6. 関連研究

Pregel の最適化に関する研究としては，GPS[3], XPregel[4]がある．GPS で使われている最適化の１つ LALP (Large Adjacency List Partitioning) がある．LALP は，次数の大きい頂点のエッジを分散させて，次数の大きい頂点から，すべての隣接頂点に同じメッセージを送る場合は，メッセージを各ノードに１つずつ送信して，各頂点への配送を，行き先のノードで行うというものである．これは，本論文の提案手法と似ている部分はあるが，本論文の提案手法は，次数の大きさにかかわらず，すべての頂点のエッジを分散させていることと，高速な計算手法を提案しているところなどが異なる．

大規模グラフの分散処理に関しては，Graph500 ベンチマークの登場により，BFS の高速な計算手法が，盛んに研究されてきた．Beamer ら[7]は，top-down 探索と bottom-up 探索をハイブリッドする手法を提案したが，Pregel 処理系に BFS を計算させると，細かな違いはあるが，ほぼ，naive な手法が top-down 探索，本論文の提案手法が bottom-up 探索となっていることが分かる．本論文は，BFS だけでなく，汎用的に bottom-up 探索を適用できるようにした点が，彼らの論文とは異なる．

7. まとめと今後の展望

本論文では，Pregel 処理系において，各頂点がすべての隣接頂点に同じメッセージを送る場合の最適化手法を提案した．提案手法を X10 で実装した Pregel 処理系 XPregel に実装し，PageRank と BFS（幅優先探索）で性能評価を行った．その結果，PageRank では高速化されたが，BFS では逆に遅くなってしまった．

今後の展望として，

1. 本論文の提案手法の計算部分を最適化することにより BFS でも高速化することができないか検討する．
2. Beamer ら[7]が BFS で行ったように，Pregel でも提案手法と naive な手法を動的に変えることで高速化できないか検討する．

などがある．

謝辞 本研究の一部は JST CREST「ポストペタスケールシステムにおける超大規模グラフ最適化基盤」の援助による．

参考文献

1) G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in Proceedings of the 2010 international conference on Management of data, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 135–146.
2) Giraph: http://giraph.apache.org/
3) S. Salihoglu and J. Widom. GPS: A Graph Processing System. Technical Report, April 2012. (to appear on SSDBM, July 2013).
4) Bao Nguyen and Toyotaro Suzumura, "Towards Highly Scalable Pregel-based Graph Processing Platform with X10", The 2nd International Workshop on Large Scale Network Analysis (LSNA 2013) In conjunction with WWW 2013, 2013/05, Rio de janeiro, Brazil
5) The Graph500: http://www.graph500.org/
6) X10 language: http://x10-lang.org/
7) Scott Beamer, Krste Asanović, and David Patterson. 2012. Direction-optimizing breadth-first search. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12). IEEE Computer Society Press, Los Alamitos, CA, USA, Article 12.
8) Toshio Endo, Akira Nukada, Satoshi Matsuoka, and Naoya Maruyama. *Linpack Evaluation on a Supercomputer with Heterogeneous Accelerators.* In IEEE International Parallel & Distributed Processing Symposium (IPDPS 2010)