最適化分野におけるクラウド技術の利用

藤澤　克樹，安井雄一郎，高宮　安仁，佐藤　　仁

最適化問題に対するクラウド・コンピューティングの適用には様々な方法が提案されている．例えば大規模最適化問題に対して数値実験等を行うために，必要なときに，必要な量だけ計算機資源をインターネット上から調達してくるIaaSと呼ばれる技術の利用等がある．本解説ではこの利用方法に関連するクラウド技術による計算資源の動的な確保について触れてから，最適化問題として大規模なネットワークデータにおけるグラフ探索と応用，およびクラウド・コンピューティングの技術を用いた高速化などに関する話題について説明していく．

キーワード：最適化問題，クラウド・コンピューティング，グラフ探索

1. はじめに

60年以上前に最適化アルゴリズムとコンピュータが誕生してから両者共に著しい発展を遂げていることは良く知られているが，今後，実社会で要求される最適化問題に対してはこれまでの研究の単なる延長では対応できないという認識も徐々に広まりはじめている．多くの既存の研究では実データの不足，解決手法や計算能力の性能の低さによって，本来期待されているレベルに達する成果を上げることが難しかったが，近年，データに関しては収集技術および集積技術の向上により，超大規模データを扱うことが可能となってきている．また最適化理論に関しても，計算機科学や数学などの近隣の分野との融合を経て，さらなる進歩を見せている．そして，計算能力に関しては計算機自体の性能の向上および実装方法の工夫や並列化などにより，現在では大規模な問題に対しても超高速で安定な計算が可能となった．よって，これからの研究においては

最先端理論（Algorithm Theory）＋大規模実データ（Practice）＋最新計算技術（Computation）

の三つを有機的に組み合わせることによって，実用に耐え得る解決策の提示と大規模最適化問題を扱う際の先例となることが求められている．本解説では，数年前から大きな注目を受けているクラウド・コンピューティングの技術と最適化問題（特にグラフ探索問題）への適用に焦点をあてて，上記の三つの組み合わせの一例として説明を行う．

クラウド・コンピューティングとはクライアント側の端末はできるだけ軽く，安く抑えておいて，計算や検索などの処理およびデータの保存や管理はインターネット上のサーバで行うための技術の集合である．こういった思想は古くから存在していたが，近年のインターネットの高速化，安定化によってようやく実現ができるようになった．クラウド・コンピューティングでは，クライアント端末はユーザの元にあり，サーバはインターネットを超えてユーザから遠くの場所に配置されている場合が多いのだが，最近では手持ちのサーバ上で仮想マシンを立ち上げて，ユーザからの要求（サーバの数や用途）に合わせて柔軟に対応していくプライベートクラウドなども注目を集めている．また，普段はプライベートクラウドを用いて，急に需要が増えた場合には外部のパブリッククラウドの資源を追加する使い方も提案，実施されている．

最適化問題に対するクラウド・コンピューティングの適用には様々な方法を考えることができるが，代表的な方法は以下の通りである．

1. サーバ上に最適化ソフトウェアをインストールした後に，Webアプリケーションとしてユーザにサービスだけを提供する（以前はASP，現在はSaaSと呼ばれる）
2. 大規模最適化問題に対して数値実験等を行うために，必要なときに，必要な量だけ計算機資源をインターネット上から調達してくる（IaaSと

ふじさわ　かつき
中央大学 理工学部
やすい　ゆういちろう
中央大学 理工学研究科
〒112-8551 文京区春日1-13-27
たかみや　やすひと
日本電気 システムプラットフォーム研究所
〒211-8666 川崎市中原区下沼部1753番地
さとう　ひとし
東京工業大学 学術国際情報センター
〒152-8550 目黒区大岡山2-12-1

呼ばれ Amazon EC2，S3 などのサービスが有名）

本解説では最初に上記の 2 と関連するクラウドによる計算資源の動的な確保について触れてから，最適化問題として大規模なネットワークデータにおけるグラフ探索と応用，およびクラウド・コンピューティングの技術を用いた高速化などに関する話題について説明していきたい．

2. クラウドによる計算資源の動的な確保

クラウド・コンピューティングの一種である Infrastructure as a Service（Iaas）が HPC のユーザを増やしつつある．その大きな理由は，使った分だけ使用料を払うという課金モデルや，使いたい台数と環境を指定するだけで即座にさまざまな HPC 環境を構築できるという手軽さにある．一方で，償却期間の終わっていない手持ちのクラスタを優先して利用するユーザも依然として多い．このようなクラスタをパブリッククラウドと対比してプライベートクラウドと呼ぶことが多い．

実行したい HPC ジョブの特性やデッドラインによっては，プライベートクラウドとパブリッククラウドを合わせてひとつのハイブリッドなクラウドとして使うことが有用である．例えば，パラメータサーチなどノード間のデータ交換が少なく計算資源が多いほど良い問題にとっては，必要に応じてパブリッククラウドを投入することで早くジョブを終わらせることができる．

本節では，こうしたハイブリッドクラウドの技術的課題と関連研究を紹介する．クラウドの技術的課題を理解するためには，クラウドとよく似た既存技術であるグリッドコンピューティング[1]やユーティリティコンピューティングとの対比が役に立つ．グリッドコンピューティングとの大きな違いは標準化の有無である．グリッドは異機種を相互接続した環境であるため，標準化団体[2]によって API などの技術標準化が進められている．一方でクラウドは主要なクラウドベンダがそれぞれ数万台規模の計算資源を提供しているため，標準化の必要が無く異環境との融合がむずかしい．ユーティリティコンピューティングとの大きな違いは資源の仮想化の有無である．ユーティリティコンピューティングではあらかじめ利用可能なコンピュータの物理的な位置やスペックが公開され，ベンダとの厳密な契約ののち数日後に利用可能となる場合がほとんどであった．一方でクラウドでは計算資源を仮想マシン（VM）で仮想化することによりクレジットカード決済で即座に利用可能である一方で，仮想化のオーバヘッドや利用可能なリソースの変動という問題がある．

2.1 パブリックとプライベートクラウドの相互接続

パブリッククラウドとプライベートクラウドを混在させるためには，プライベートクラウド上のマシンをプライベートクラウドと同じ API でソフトウェア的に操作できる必要がある．パブリッククラウド上のマシンのセットアップはソフトウェアから制御可能な API が提供されており，VM のイメージファイルを指定するだけで必要な台数を確保できる．しかしプライベートクラウドは実マシンで構成されるため，物理的に電源を入れて OS やソフトウェアをインストールする手順を踏まなくてはならない．

相互接続の問題を解決するため，筆者らはプライベートクラウドの実マシンを Amazon EC2 と同じ API を使って起動できるシステムを提案している[3]．このシステムでは，Amazon EC2 の VM イメージを実マシンにインストールする機能や，実マシンの電源を EC2 と同じ API で投入し起動できる機能を提供することでパブリックとプライベートクラウド間の垣根を無くし，人手を経由せずにスケジューラなどでアルゴリズミックにマシン数を増減できる．技術的には，EC2 の VM イメージを実マシンにインストールするために，いったん EC2 上で VM イメージを使ったノードを起動しマシンイメージを吸い出してプライベートクラウドに転送する．吸い出したイメージはプライベートクラウド上のイメージ変換サーバによって実マシン用に変換しインストールする．また，実マシンが起動に失敗した場合でも確実に台数を確保するために，要求された台数にプラスして予備マシンを起動する．また，マシンの起動シーケンス（ネットワークブート用のパケット送受信や，各種デーモンの起動など）を外部から監視サーバで監視し，起動に失敗した場合は予備マシンでフェイルオーバーする．

2.2 仮想化による変動とオーバヘッド

パブリッククラウドでの仮想化による大きな影響は性能低下である．同じマシンを共有する他のユーザの利用状況によって利用可能なリソース量が時々刻々と変化するため，性能がばらつきやすく性能予測や最適化がむずかしい．特に，多くの HPC アプリケーションは CPU キャッシュやメモリバスを占有する前提で

最適化されているため，こうした変動の影響を受けやすい．もうひとつの大きな違いは資源の均一性である．パブリッククラウドはユーザに“無限の均一なマシンリソース”という抽象化を与えるため，利用料金を払えば均一な資源を事実上無制限に追加できる．一方でプライベートクラウドは購入時期の違いなどにより様々なスペックのマシンが混在することが多い．

3. グラフ探索と応用

大規模なネットワークデータ上における最短路計算などのグラフ探索アルゴリズムには以下のように非常に幅広い応用があることが知られている．

1. 交通ネットワークにおける経路検索（カーナビゲーションシステム等）
2. 災害時における避難や誘導等の支援システム
3. Twitter，Facebook などのソーシャルネットワークデータの解析

これらの応用においては，今後は超大規模ネットワークデータ（点数1億以上）が対象となることが予想されるので，超高速な探索アルゴリズムとソフトウェアの開発が急務となっている．例えば上記のソーシャルネットワークデータの解析においては，ネットワーク内での各点の重要度を計算することによって，各点の周辺，および広域内における影響（情報の伝播力）を推定することができる．しかし，そのためには大規模ネットワークデータ上で短時間に大量のグラフ探索を行う必要があり，メモリ要求量を抑えた上で，クラウド・コンピューティングの技術などを用いて大量のスレッドを生成，同時並行的にグラフ探索を行うことによって，極めて高いスループットを得ることができる．

4. 最短路を用いたグラフ解析

本節では，ネットワーク内の各点の重要度判定のための中心性指標を取り上げる．中心性指標には多くの定義が存在するが最短路探索を用いた4種類に焦点を当て説明を行う．各点の最短路に寄与する割合を指標とした Betweenness Centrality はソーシャルネットワークなどのグラフ解析等で幅広く応用されている．

最短路を利用した4種類の中心性の定義を行う．点数 n，枝数 m である有向グラフ $G=(V, E)$ と非負の重み関数 $w:(i, j)\to\mathbf{R}_+$，$\forall(i, j)\in E$ に対して，始点 s，終点 t 間の最短路長を $d_G(s, t)$，最短路数を σ_{st} とする．ここで自分自身への最短路長は $0(d_G(s, s)=0)$，最短路数は $1(\sigma_{ss}=1)$ とする．

- Closeness Centrality（Sabidussi, 1966）

$$C_C(v)=\frac{1}{\sum_{t\in V}d_G(v, t)}$$

- Graph Centrality（Hage and Harary, 1995）

$$C_G(v)=\frac{1}{\max_{t\in V}d_G(v, t)}$$

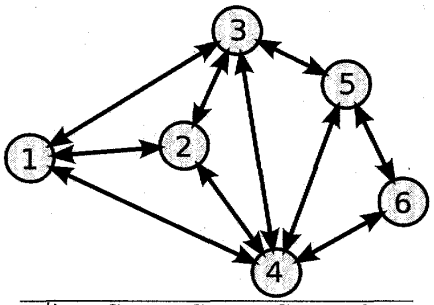
- Stress Centrality（Shimbel, 1953）

$$C_S(v)=\sum_{s\neq v\neq t\in V}\sigma_{st}(v)$$

- Betweenness Centrality（Freeman, 1977；Anthonisse, 1971）

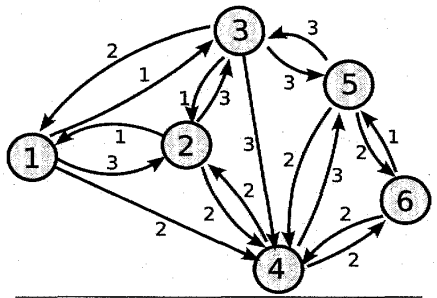
$$C_B(v)=\sum_{s\neq v\neq t\in V}\frac{\sigma_{st}(v)}{\sigma_{st}}$$

これらは最短路長を用いた中心性指標 Closeness Centrality（C_C），Graph Centrality（C_G）と，最短路数



点	C_C	C_G	C_S	C_B
1	0.14	**0.50**	0.00	0.00
2	0.14	**0.50**	3.00	2.00
3	**0.17**	**0.50**	5.00	3.00
4	0.14	**0.50**	**8.00**	**5.50**
5	0.14	**0.50**	4.00	2.50
6	0.11	0.33	0.00	0.00

図1 重みなしグラフに対する中心性指標の例



点	C_C	C_G	C_S	C_B
1	**0.077**	**0.25**	3.00	2.50
2	0.071	0.20	**7.00**	4.67
3	**0.077**	**0.25**	6.00	3.67
4	0.071	**0.25**	**7.00**	**5.33**
5	0.067	0.20	2.00	2.00
6	0.056	0.20	0.00	0.00

図2 重み付グラフに対する中心性指標の例

を用いた中心性指標 Stress Centrality（C_S），Betweenness Centrality（C_B）に分類できる．図1，2に重みなし/付グラフに対する中心性指標の例を示す．

各点 $\forall v \in V$ に対する $C_C(v)$，$C_G(v)$ を得るためには点 v を始点とした1対全最短路検索を計算すれば良い．一方，$C_S(v)$，$C_B(v)$ を得るために点 v 以外のすべての2点間最短路計算（全対全最短路問題に対応する）が必要となる．そこでBrandesによって提案された C_B 計算に対する効率的なアルゴリズム[4]を紹介する．

Brandesのアルゴリズム（Algorithm 1）は，「1対全最短路探索（3行目）」と「探索結果を用いた中心性計算（4〜8行目）」で構成され，点数回繰り返し計算される．Brandesのアルゴリズムにおける1対全最短路検索では最短路が複数本ある場合にもすべて列挙する必要があり，各点 $\forall v \in V$ に対する直前点集合 $P(v)$ によって表現される．同時に最短路数 $\sigma(v)$ も算出される．中心性計算は，探索点の履歴FILOキュー S を用いて，最短路探索における訪問順の逆順（始点から離れている点から順）に行われる（5行目）．点数 n，枝数 m の有向グラフに対する1対全最短路検索に要する計算量は，重みなし（幅優先探索）では $O(m)$，各枝の重みを考慮した場合にバイナリヒープを適用したダイクストラ法を用いると $O(m \log n)$ となる．中心性計算はいずれも $O(m)$ となるので，C_B に対するBrandesのアルゴリズムに要する計算量は，重みなしでは $O(nm)$，重み付では $O(nm \log n)$ である．

Algorithm 1 Brandes's algorithm (2001)

入力: 有向グラフ $G = (V, E)$
出力: Betweenness Centrality $C_B[v], \forall v \in V$

```
1: C_B[v] ← 0, ∀v ∈ V
2: for s ∈ V do
3:     σ, S, P ← SSSP(G, s)
4:     while S ≠ ∅ do
5:         w ← pop(S)
6:         for v ∈ P[w] do
7:             δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w])
8:         C_B[w] ← C_B[w] + δ[w]
9: return C_B
```

しかしながら，Brandesのアルゴリズムは効率的ではあるものの厳密な中心性計算において要求される計算量は依然として大きいため，Baderらによって提案されたランダムサンプリング手法[5]を紹介する．この手法は非常にシンプルであり Algorithm 1 の2行目をランダムに選択された始点集合 V' に対して繰り返し，得られた中心性指標を $\frac{|V|}{|V'|}$ 倍すれば良い．

中心性計算でのボトルネックは最短路探索であるため，Algorithm 2 のように1度の最短路探索で各中心性指標 C_C，C_G，C_S，C_B の関連ある要素を計算すると良い．計算量は，重みなしでは $O(nm)$，重み付では $O(nm \log n)$ となる．Algorithm 1，Algorithm 2 はいずれも各反復を独立に計算可能であるので容易に並列計算を行うことが可能である．

Algorithm 2 C_C, C_G, C_S, C_B 同時計算

入力: 有向グラフ $G = (V, E)$
出力: $C_C[v], C_G[v], C_S[v], C_B[v], \forall v \in V$

```
1: C_C[v] ← 0, C_G[v] ← 0, ∀v ∈ V
2: C_S[v] ← 0, C_B[v] ← 0, ∀v ∈ V
3: for s ∈ V do
4:     σ, S, P, d_G ← SSSP(G, s)
5:     C_C[s] ← 1 / Σ_{t∈V} d_G(s,t)
6:     C_G[s] ← 1 / max_{t∈V} d_G(s,t)
7:     δ_S[v] ← 0, δ_B[v] ← 0, ∀v ∈ V
8:     while S ≠ ∅ do
9:         w ← pop(S)
10:        for v ∈ P[w] do
11:            δ_S[v] ← δ_S[v] + (1 + δ_S[w])
12:            δ_B[v] ← δ_B[v] + (σ[v]/σ[w]) · (1 + δ_B[w])
13:        C_S[w] ← C_S[w] + δ_S[w] · σ[w]
14:        C_B[w] ← C_B[w] + δ_B[w]
15: return C_C, C_G, C_S, C_B
```

それでは，中心性計算の数値実験結果を示していく．全米道路ネットワークをグラフ化したUSA-road-d.NY.gr（264,346点/733,846枝）インスタンスを用いる．なお計算機環境はIntel（R）Xeon（R）CPU X5460 3.16 GHz（L2：6 MB）×2，用いたCコンパイラはGCC-4.1.2，最適化オプションは-O2である．表1はAlgorithm 1（Alg. 1），Algorithm 2（Alg. 2）の1反復あたりの実行時間（ミリ秒）をまとめたものである．括弧内はグラフ内の各枝の重みの有無を示している．Alg. 1（無），Alg. 2（無）の最短路探索には幅

表1 中心性計算1反復あたりの実行時間（ミリ秒）

スレッド数	Alg.1(無)	Alg.2(無)	Alg.2(有)
1	36.19	42.84	77.42
2	19.24	25.18	40.14
4	11.87	14.49	22.01
8	11.36	14.03	16.36

優先探索を，Alg. 2（有）ではバイナリヒープを用いたダイクストラ法を選択し，各反復をマルチスレッド化している．

Alg. 1（無）と Alg. 2（無）の実行時間の比較から，小さなオーバヘッドで4種類の中心性を同時に扱うことが可能であること，計算機資源に対する要求の衝突によりマルチスレッド計算での性能向上が難しいこと（4，8スレッド計算時）が確認される．また，Alg. 2（無）と Alg. 2（有）から，バイナリヒープを用いたダイクストラ法は幅優先探索に比べスレッド並列計算向きであり，スレッド数が増えるに従い，両者の実行時間の差は縮まっている．

5. Graph 500

近年，計算中心であったハイパフォーマンスコンピューティングの分野においても大規模なデータ処理を中心に扱うアプリケーション（データ・インテンシブアプリケーション）が増加している．例えば，バイオインフォマティクスの分野では，次世代シーケンサーの登場によりテラ～ペタバイトオーダの大量のゲノムシーケンスデータがストリーミングとして生成され，クラウドやスーパーコンピュータなどの高性能な大規模並列計算機による解析が行われている．このような背景もあり，大規模並列計算機も従来は高速な演算処理に特化していた設計であったものから，高速な大規模データ処理に特化した設計のものも登場しはじめている．例えば，米国サンディエゴスーパーコンピューティングセンターでは，Dash[6]，Gordon と呼ばれるメモリとディスク間の遅延の違いやフロップスあたりのバンド幅を重視した大規模データ処理向けのスーパーコンピュータの研究開発を進めていたり，Amazon Web Services[7]のような商業クラウドベンダーにおいても大規模ストレージや MapReduce 型のデータ処理のサービスを提供していたりする．これまで，計算機の演算性能を公平に評価するためには，Linpack と呼ばれる連立一次方程式を解くベンチマークの結果に基づき TOP 500[8]と呼ばれるランキングを決定していたが，一方でデータ処理に特化して計算機の性能を公平に評価するためのベンチマークおよびランキングは存在してこなかった．このような観点により，2010年6月の ISC'10 において Graph 500[9]と呼ばれるデータ処理の性能に基づく計算機のランキング付けの試みがアナウンスされ，2010年11月の SC 10 において初めてランキングリストが公表された．表2に2010年11月地点でのランキングリストの Top 5 を示す．Graph 500 は並行探索，最短路探索をはじめとする最適化，極大独立集合などのグラフ解析などの複数のグラフ処理カーネルからなるベンチマークにより計算機の性能を評価しランキングを行う．グラフ解析はサイバーセキュリティ，創薬，データマイニング，ネットワーク解析などの分野において必要とされる重要な計算カーネルとして位置づけられている．

表2 Graph 500 リスト（2010年11月現在）

順位	計算機名 計算機センター名	SCALE　TEPS 値 ノード数 (コア数)
1	Intrepid(BlueGene/P) Argonne National Lab.	Scale 36　6.6 GE/s 8192(32768)
2	Franklin(Cray XT4) NERSC	Scale 32　5.2 GE/s 500(2000)
3	cougarxmt(Cray XMT) Pacific NW National Lab.	Scale 29　1.2 GE/s Cray XMT×128
4	graphstorm (Cray XMT) Sandia National Lab.	Scale 29　1.17 GE/s Cray XMT×128
5	Endeavor (Xeon X5670) Intel Corporation	Scale 29　533 ME/s 256(512)

執筆時点（2011年3月）での最新の仕様（バージョン 1.1），および，参照実装（バージョン 1.2）では，Graph 500 はグラフの構築（Kernel 1）と構築されたグラフに対する幅優先探索（Kernel 2）の2種類の計算カーネルから構成される．以下に Graph 500 で行う処理の流れを示す．

1. Kronecker 積を用いて枝リストを生成
2. 生成された枝リストからグラフを構築（Kernel 1）
3. 構築されたグラフに対して次数が1以上で一意的な64の始点を選択
4. 各始点から幅優先探索（Kernel 2）を実行
5. 探索の実行結果の検証，計測した性能の表示

Kernel 1 では，（始点，終点，重み）の組で表現された枝リストからグラフの構築を行う．構築されるグラフは自分自身への枝を取り除いた無向グラフであり，隣接リスト表現によって表現される．Graph 500 で扱うグラフは各点の出次数（隣接する枝数）の平均が16程度となる Kronecker Graph [10] である．Kronecker Graph は，ソーシャルネットワークにおける人の隣接関係をうまく表現した「局所的に密，大域的には疎」といった特性をもつ．これはソーシャルネットワークにおいて，同一のコミュニティに属した人たちの関係はある程度密接であり，コミュニティ間の隣接関係は稀薄であるといった特徴と合致している．参照実装のグラフ生成プログラムでは，2種類のパラメ

ータ SCALE と edgefactor（デフォルトでは 16）を入力として，点数 $n=2^{SCALE}$，枝数 $m=$edgefactor$\times n$ の重みなし無向グラフを出力する．Kronecker Graph の生成は，再帰的に隣接関係を 2×2 に分割していき，生成確率に対して 1 つずつ枝を生成する．各枝の生成確率は初期隣接関係として K_1 を与え，$SCALE$ 回の Kronecker 積により決定される．式(1)のような 2×2-行列 K_1 に対する Kronecker 積 $K_2=K_1\otimes K_1$ は式(2)のようになり，生成確率は式(3)となる．枝 (i,j) の生成確率は行列 $K_{SCALE}=\underbrace{K_1\otimes K_1\otimes\cdots\otimes K_1}_{SCALE}$ の (i,j)-要素である．Kronecker Graph の初期隣接関係 K_1 は図 3 のような隣接関係を意味している．この生成プログラムでは，同一の始終点をもつ複数の枝や，自己ループ，および非連結点などの以降のカーネルでは使用されない枝リストも生成されることに注意されたい．

$$K_1=\begin{pmatrix}a & b\\ c & d\end{pmatrix}=\begin{pmatrix}0.57 & 0.19\\ 0.19 & 0.05\end{pmatrix} \tag{1}$$

$$K_2=K_1\otimes K_1$$

$$=\begin{pmatrix}a\cdot K_1 & b\cdot K_1\\ c\cdot K_1 & d\cdot K_1\end{pmatrix} \tag{2}$$

$$=\begin{pmatrix}0.3249 & 0.1083 & 0.1083 & 0.0361\\ 0.1083 & 0.0285 & 0.0361 & 0.0095\\ 0.1083 & 0.0361 & 0.0285 & 0.0095\\ 0.0361 & 0.0095 & 0.0095 & 0.0025\end{pmatrix} \tag{3}$$

一方，Kernel 2 では，Kernel 1 で構築された無向グラフを入力とした幅優先探索を行う．ランダムに決定した 64 の点を始点とした幅優先探索を行い，各探索の実行時間や TEPS 値の詳細（最小値，1/4 値，中央値，3/4 値，最大値，平均，標準偏差，調和平均，調和標準偏差）な計測結果を出力する．この TEPS 値の最大値に基づき Graph 500 ランキングが決定される．TEPS（Traversed Edges Per Second）は，単位時間あたりに処理した枝数を表し，点数 n のグラフに対して処理した枝数 m と処理に要した時間（秒）($\text{time}_{K2}(n)$) から，$\text{TEPS}(n)=m/\text{time}_{K2}(n)$ として定義される．連結性が仮定されたグラフに対する幅優先探索での到達枝数は入力グラフの枝数と等しい．幅優先探索は最も基礎的なグラフ探索であり，始点から開始して，隣接した点をすべて訪問するという操作を未訪問の点がなくなるまで繰り返し行う．各反復における計算手順には依存性があり，要求されるグラフデータ（隣接関係）は高々 1 度ずつ参照されるため，計算機のメモリ参照のスループット性能が反映されやすい．特に並列実行では予測困難な動的負荷分散や非同期計算による副作用（ペナルティ，コスト）などへの律速が予想される．これらの現象は計算手順が決定的で同一領域に対してデータ参照を頻繁に行うといったピーク性能が出しやすい典型的な計算での状況と大きく異なる．

Graph 500 は対象するデータサイズによっていくつかの問題クラスに分類される．表 3 にそのクラスの種別を示す．最も小さな問題クラス（Toy）では 17.2 GB のデータサイズを対象するのに対し，最も大きな問題クラス（Huge）では 1.1 PB を超えるデータサイズを対象にする．このため，問題のクラスに応じてグラフ解析のアプローチ方法も異なってくる．例えば，問題クラスが Mini までは対象とするデータが 1 台の計算機上のメインメモリに収まってしまうため 1 台のマルチコアマシン上の実行でも有効であるのに対し，問題クラスが Small～Medium では複数の計算機を集約して用いないと実行ができず，問題クラスが Large～Huge まで及ぶと複数の計算機を使用してもすべてのグラフデータをメインメモリに収めることができないため，SSD，HDD などのストレージを活用

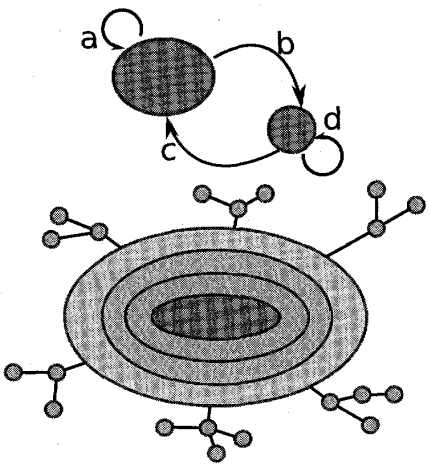


図 3 Kronecker Graph の初期隣接関係 K_1 の表現する隣接関係（上）と実際のネットワーク（下）

表 3 Graph 500 の問題クラスとデータサイズ

問題クラス	点数 n	枝数 m	データサイズ
toy	2^{26}	$2^{26}\times 16$	17 GB
mini	2^{29}	$2^{29}\times 16$	140 GB
small	2^{32}	$2^{32}\times 16$	1 TB
medium	2^{36}	$2^{36}\times 16$	17 TB
large	2^{39}	$2^{39}\times 16$	140 TB
huge	2^{42}	$2^{42}\times 16$	1.1 PB

表4　クラスタ計算機（16ノード）の構成

クラスタ計算機の構成	
計算ノード数	16
コア数	192コア (=6 × 2 × 16)
RAM	2TB(=128GB × 16)
クラスタ計算機の各ノードの構成	
CPU	Intel Xeon X5670 2.93 GHz × 2
RAM	128GB
OS	CentOS 5.5 for x86_64

表5　参照実装バージョン1.2（mpi版）の性能

入力パラメータ	
SCALE	18
edgefactor	16
グラフサイズ	
点数 n	2^{18}
枝数 m	2^{22}

プロセス数	TEPS
1	70.58 M
2	47.39 M
4	91.99 M
8	185.02 M
16	330.75 M
32	524.80 M

する必要がある．参考までに，表5にクラスタ計算機16ノードでmpi版の参照実装（graph 500_mpi_simple）を実行した結果を示す．使用したクラスタ計算機の諸元は表4である．実験では，グラフはSCALE 18，edgefactor 16（点数2^{18}，枝数2^{22}）のKronecker Graphを使用し，1ノードあたり1プロセス（ただし，32プロセスの実験では1ノード，1ソケットあたり1プロセス）を割り当てて実行した．表5ではプロセス数に対する実行時間を示している．対象とするデータが1ノード上のメインメモリに収まる非常に小さな問題ではあるが，プロセス数の増加に従いTEPS値も増加しているのが確認できる．

参考文献

[1] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid : Enabling Scalable Virtual Organizations," International Jounral of Supercomputer Applications, Vol. 15, No. 3, 2001.

[2] OpenGridForum, http://www.gridforum.org/

[3] 高宮安仁，田浦健次朗，安井雄一郎，藤澤克樹，"Bare Metal Cloud：実マシンを提供するクラウドサービス，" 情報処理学会研究報告，2010-HPC-126 (39)，2010.

[4] U. Brandes, "A Faster Algorithm for Betweenness Centrality," Journal of Mathematical Sociology, Vol. 25, No. 2 : 163-177, 2001.

[5] D.A. Bader, S. Kintali, K. Madduri and M. Mihailm, "Approximating Betewenness Centrality," The 5th Workshop on Algorithms and Models for the Web-Graph (WAW 2007), San Diego, CA, December 11-12, 2007.

[6] J. He, A. Jagatheesan, S. Gupta, J. Bennett and A. Snavely, "DASH : a Recipe for a Flashbased Data Intensive Supercomputer," 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC10), pp. 1-11, 2010.

[7] Amazon Web Services, http://aws.amazon.com/jp/

[8] TOP 500 Supercomputer Sites, http://www.top500.org/

[9] Graph 500. org, http://www.graph500.org/

[10] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos and Z Ghahramani, "Kronecker Graphs : An Approach to Modeling Networks," Journal of Machine Learning Research, Vol. 11 (2010) 985-1042.