実時間ゲリラ豪雨予測システムのための ファイルI/O調停ミドルウェアの試作

Jianwei Liao[1]　Gerofi Balazs[1]　石川 裕[1]　Guo-Yuan Lien[1]　三好 建正[1]　大塚 成徳[1]　富田 浩文[1]
西澤 誠也[1]

概要：30 秒毎の最新気象観測データから 30 分後のゲリラ豪雨を予測するシステムを開発している．本システムでは，100 ケースの 30 秒アンサンブル気象シミュレーション結果と 30 秒毎の最新気象観測データを同化する．気象シミュレーションプログラムとデータ同化プログラムは別々に開発されており，データ交換はファイル渡しになっている．これら 2 つのアプリケーションプログラムの周期実行起動方式を検討すると共に、プロセス間で効率の良いデータ転送方式を設計・試作・評価する．

Prototyping File I/O Arbitrator Middleware for Real-Time Severe Weather Prediction System

Jianwei Liao[1]　Gerofi Balazs[1]　Yutaka Ishikawa[1]　Guo-Yuan Lien[1]　Takemasa Miyoshi[1]
Shigenori Otsuka[1]　Hirofumi Tomita[1]　Seiya Nishizawa[1]

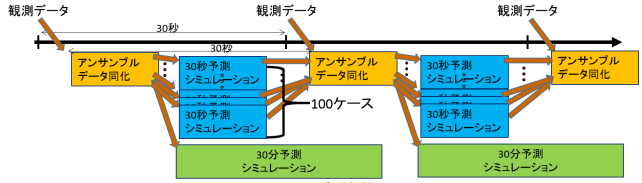
Abstract: A high-resolution 30-minute numerical weather prediction system, using weather observable data that is available every 30 seconds, is being developed. In this system, 100-case ensemble simulations and data assimilation of results and observed data are performed every 30 seconds to predict 30 minute weather. Because the simulation and data assimilation programs are independently developed, data is exchanged via files. In this paper, after examining a mechanism to recycle two application jobs, an efficient data transfer method between application processes is designed, implemented and evaluated.

1. はじめに

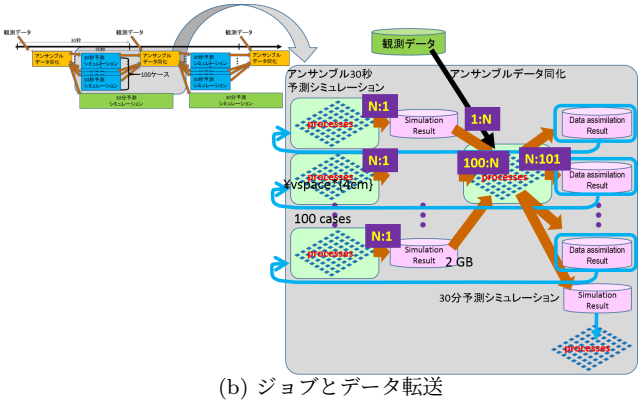
我々が研究開発を進めている 30 秒毎に更新する 30 分天気予報を実現する実時間天気予測システム [1], [2] [*1] では，30 秒サイクルで以下のジョブを実行する：i) 天気予測シミュレーションを同時に 100 ケースアンサンブルシミュレーションするジョブ群実行，ii) これら結果と次世代型フェーズドアレイ気象レーダーで取得可能な 30 秒毎の 3 次元観測データや次期気象衛星ひまわり 8 号・9 号で可能となる 2 分 30 秒毎の日本周辺雲画像データを用いてデータ同化するジョブ実行、iii) 30 分後の予測シミュレーションジョブ実行．現在，シミュレーションプログラムも同化プログラムも独立して開発されており，データはファイルから入出力している．計算ノードの性能向上に比べファイル I/O の性能向上は今後も大きな飛躍はない．実時間処理のためにはファイル I/O によるジョブ間のデータ交換でなく，プロセスのメモリ領域を直接 RDMA (Remote Direct Memory Access) 機能を用いて交換するミドルウェアを実現しデータ交換時間を短縮する必要がある．

論文 [2] では, 気象・気候シミュレーション系で使われている netCDF API[3] を拡張し，プロセスが保持するデータをそのデータを必要とする他のジョブのプロセスに非同期で直接転送する機構を提案した．本論文では，実時間天気予測システムのために必要とされる 100 ケースシミュレーションジョブとデータ同化ジョブの周期実行起動方法について検討を行った後、netCDF API を拡張せずにアンサンブルジョブ群とデータ同化ジョブ間でのデータ転送を効率的に提供する MPI 実装方式を検討する．

[1] 理化学研究所 計算科学研究機構
[*1] 科学技術振興機構 CREST「「ビッグデータ同化」の技術革新の創出によるゲリラ豪雨予測の実証」（研究代表者：三好建正）



(a) 全体像



(b) ジョブとデータ転送

図 1 実時間ゲリラ豪雨予測システム

1.1 全体像

2. 実時間ゲリラ豪雨予測システムの概要

図 1(a) にシステムの全体像を示す．30 秒サイクルで以下の処理を繰り返す．

- シミュレーション
  アンサンブル 30 秒気象予測シミュレーションの数値天気予報モデルとしては，理研で開発されている LES 気象モデル SCALE[4]，気象庁で現業で運用されている NHM モデルの利用が想定されている．同時に 100 ケースの気象予測シミュレーションを実行する．100 ケースの予測結果はデータ同化に渡される．
- データ同化
  100 ケース予測結果と観測データとの同化には，局所アンサンブル変換カルマンフィルタ（LETKF）[5] を使用する．観測データとしては，次世代型フェーズドアレイ気象レーダー [6] や気象衛星ひまわりのデータがインターネット経由で受信されることを想定している．次世代型フェーズドアレイ気象レーダーからは，30 秒間隔で反射強度，ドップラー速度，速度幅の 3 パラメータ (37.6MB/parameter) が取得できる．
  各予測ケース結果と観測データが同化し，同化結果は次のフェーズのシミュレーション初期値として使われる．また，100 ケースのうちの一つは 30 分予測シミュレーションの初期値として使われる．

これらプログラムは MPI 並列化されており，いずれも独立して開発されており，入力データ，出力データはファイルに格納される．

本稿で想定する問題サイズは，水平方向 100 メートル解像度で 120 Km x 120 Km の領域，垂直方向 80 グリッドである．モデルが扱う変数は 16 であり，生成されるデータ量は 13.7GiB となる．

$$1200 \times 1200 \times 80 \times 16 \times 8Byte = 13.7GiB$$

100 ケースシミュレーションでは，約 1.3TiB のデータ量となる．

データ転送の観点で気象予測シミュレーションコードを概観する (図 1(b))．気象予測シミュレーションの並列化は領域分割され，各 MPI プロセスは担当領域を計算する．一つのファイルから各 MPI プロセスが担当する領域データが分配される．シミュレーション結果をファイルに格納するために，各 MPI プロセスが保持しているデータが集約される．ファイルアクセスパターンを入力ファイル数：プロセス数：出力ファイル数で表現すると，気象予測シミュレーションコードは，1:N:1 アクセスパターンといえる．なお，SCALE では並列ファイル I/O ライブラリ netCDF[3] を使ったファイルアクセスも可能である．

LETKF データ同化コードは，アンサンブルシミュレーション結果と観測データを同化する．観測データにはノイズが載っているため，品質保証のためにノイズ除去すると使える観測データ領域は動的に変わる．このため，シミュレーションと同じ領域分割による並列化では，モデルが扱っている分割領域周辺に観測データが存在しない領域を扱っているプロセスの計算負荷は少なくなり，負荷がアンバランスになる．第 2.1 節および第 2.2 節において，アンサンブルシミュレーションジョブとデータ同化ジョブの計算ノード割り当ておよびデータ交換パターンについて説明する．

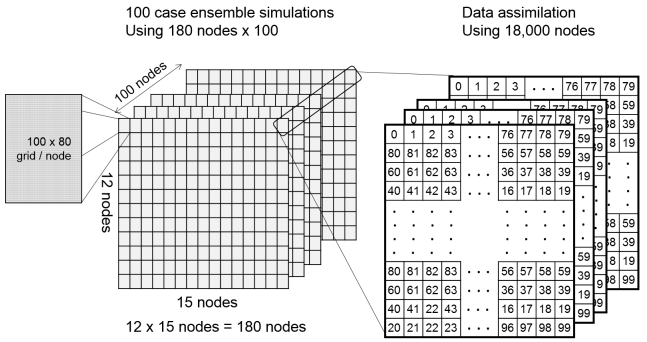


図 2 実行イメージ

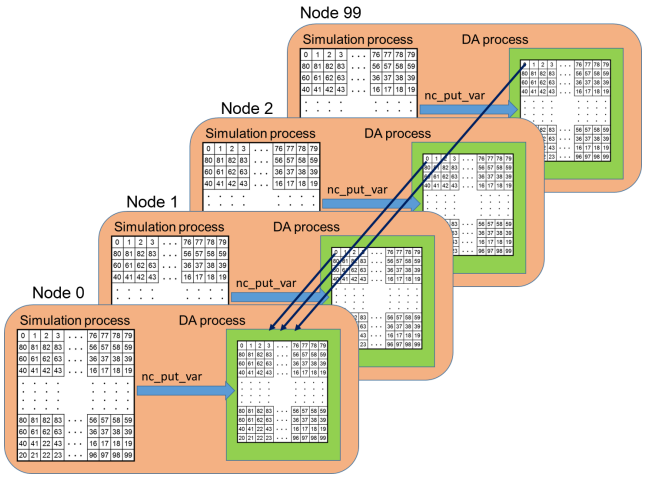


図 3 データ交換パターン

2.1 ジョブ割り当て

本論文では，1 シミュレーションは 180 ノードで計算し，データ同化は 18,000 ノードで計算することを想定する．100 アンサンブルシミュレーションジョブ群とデータ同化ジョブは同時に実行しないため，18,000 計算ノードを交互に利用できる．図 2 に 100 アンサンブルシミュレーションジョブ群とデータ同化ジョブがシミュレーション領域をどのように分割して実行するかを示す．

図 2 の左側は，アンサンブルシミュレーションジョブ群が実行している時のジョブ割り当てを示している．ここで，1 つの面は 1 つのシミュレーションの領域を示している．1 シミュレーションは 180 計算ノードで実行され，計算領域が分割されて処理される．1200 x 1200 の格子に対して 180 計算ノードを割り当てる．すなわち，1 計算ノードあたり 100 x 80 の格子を計算させる．1 計算ノードあたりのデータサイズは

$$80 \text{ 格子点} \times 80 \text{ 垂直方向} \times 16 \text{ 変数} \times 8\text{Byte} = 819 \text{ KB}$$

となる．

100 アンサンブルシミュレーションにより，各 100 x 80 格子領域は 100 通りの結果をそれぞれのプロセスが保持している．100 通りの 100 x 80 格子領域毎に 100 個の計算ノードがデータ同化する．図 2 の右側は，データ同化ジョブの計算ノードがアンサンブルシミュレーションプロセスが保持している各 100 x 80 格子領域のどの領域を計算するか示している．本図においてシミュレーション領域の右上の 100 x 80 領域に対する 100 ケースのシミュレーション結果と観測データの同化は同じ 100 の計算ノードで計算される．それぞれの計算ノードを 0 から 99 の番号でラベル化したとすると，各格子点の数字はその格子点をデータ同化する計算ノード番号を示している．

2.2 データ転送パターン

100 アンサンブルシミュレーションプロセス群 (180 プロセス x100) が保持している格子領域をデータ同化プロセス 18,000 にどのように分配するかを説明するために，図 2 の右側で扱っている 100 x 80 格子領域の分配方法を図 3 に示す．図 3 は，各計算ノード上で実行しているシミュレーションプロセスとデータ同化プロセスを図示している．各計算ノードには，同一 100 x 80 格子領域を異なる初期値で計算しているシミュレーションプロセスとデータ同化するプロセスが同居している．各シミュレーションプロセスは同居しているデータ同化プロセスに格子領域全体のデータを送る．データ同化プロセス側でそれらデータを他のデータ同化プロセスに配布する．図 3 では，計算ノード 0 のデータ同化プロセスに集められるデータを表現している．

本ジョブ割り当てでは，100 アンサンブルシミュレーションプロセス群が保持している格子領域に対して 100 のデータ同化プロセスを割り当てており，100 計算ノードに閉じていることになる．言い換えると図 2 の右側の各 100 x 80 格子領域毎に 100 個の計算ノードを使って 100 アンサンブルのシミュレーションプロセスと 100 データ同化プロセスがデータ交換するだけで，他の領域を処理しているアンサンブルプロセスやデータ同化プロセスとはデータ交換しない．すなわち，このような領域分割では，対象領域が大きくなっても計算ノード数を増やすことによってウィークスケールすることになる．

3. 設計・試作

単純に独立して開発されたアプリケーションを同時に実行し共有ファイルシステム経由で連成計算させることはジョブを同時に実行するだけであるが，実時間予測を行うために，netCDF API を保持しながら通信によりデータ転送を行うためには以下の課題を解決する必要がある．

(1) 図 3 で示したとおり各ノードにシミュレーションプロセスとデータ同化プロセスを割り当てて，同時に実行できること．
(2) 複数ジョブ間で通信路を確立すること．
(3) 30 秒以内にアンサンブルシミュレーションおよびデータ同化が終了するよう高速ジョブ起動・消滅機構を実現すること，あるいは最小限のプログラム変更で，そ

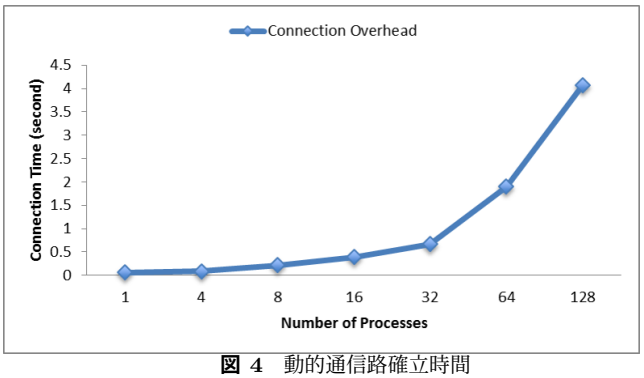


図 4 動的通信路確立時間

```
01: /* Initialization */
02:
03: nc_create("data1.nc", ncmode, &ncid1);
04: nc_create("data2.nc", ncmode, &ncid2);
05: /* Another Initialization **/
06: /* main computation */
07: /* data output */
08: for (i = 0; i < 2; i++)
09:  nc_put_vara(ncid, vid[i], startp, countp, buf+i*N);
10: for (i = 0; i < 2; i++)
11:  nc_put_vara(ncid, vid[i], startp, countp, buf+i*N);
12: nc_close(ncid1);
13: nc_close(ncid2);
14:
15:
16: /* Finalization */
```

図 5 プログラム例

れぞれのアプリケーションジョブが繰り返し実行できる機能を提供すること．

3.1 ジョブ間の通信路確立

最初の課題はジョブスケジューラに対してジョブの割り当てを制御できる機能を有していれば良い．2 番目の課題であるシミュレーションジョブとデータ同化ジョブの間で通信路を確立するために，本論文では，MPI が提供する動的通信路確立機能（MPI_Comm_Accept，MPI_Comm_Connect）を用いる．

スーパーコンピュータ「京」におけるプロセス数による動的通信路確立時間の違いを計測した結果を図 4 に示す．以降，計測に利用したコンピュータは「京」である．128 ノードでは約 4 秒の時間がかかっていることが分かる．30 秒周期でシミュレーションプロセスとデータ同化プロセスを実行するために毎回ジョブを生成して動的に通信路を確立するのはコストが高すぎることが分かる．

3.2 周期実行起動方式

毎週期毎にシミュレーションとデータ同化ジョブを起動しジョブ間の通信路を確立するのはコストが高い．そこで，最小限のプログラム変更で，それぞれのアプリケーションジョブが繰り返し実行できる機能を検討する．

```
01: /* Initialization */
02: while(1) {
03:   nc_create("data1.nc", ncmode, &ncid1);
04:   nc_create("data2.nc", ncmode, &ncid2);
05:   /* Another Initialization **/
06:   /* main computation */
07:   /* data output */
08:   for (i = 0; i < 2; i++)
09:    nc_put_vara(ncid, vid[i], startp, countp, buf+i*N);
10:   for (i = 0; i < 2; i++)
11:    nc_put_vara(ncid, vid[i], startp, countp, buf+i*N);
12:   nc_close(ncid1);
13:   nc_close(ncid2);
14:
15: }
16: /* Finalization */
```

図 6 プログラム例の修正版

図 5 は netCDF を使用したアプリケーション例である．プログラムの初期化の後，3，4 行目で nc_create 関数によりファイルを生成している．ファイル生成後，データフォーマット定義等の初期化を行った後主計算を行う．9，11 行目の nc_put_vara 関数によりデータをファイルに書き出している．12，13 行目の nc_close 関数でファイルを閉じた後，プログラムの後処理を行っている．

本アプリケーションが nc_create 関数を除けば，図 6 のように 3 行目から 13 行目を繰り返しても動作が保証できる，すなわち，1 行目の初期化は主計算を繰り返しても変更されない初期化であり，主計算を繰り返す度に初期化しなければいけない変数群は 4 行目の初期化で行われていると仮定する．このような仮定をおけば，図 5 のプログラムは，図 6 のプログラムのように変更できる．

ここで問題になるのは，nc_create 関数の扱いである．最初の繰り返し時に出現する nc_create 関数は通常と同じ動作をするが，それ以降に出現する nc_create 関数およびその関数によって作られるファイルに対するデータ型定義は，通常の netCDF 関数処理を行わず，メタ情報を再利用しファイルを生成時と同じ状態に戻す操作を施す必要がある．netCDF ライブラリ側では，nc_create 関数が呼ばれた時に最初の繰り返し時なのかそれ以降の繰り返し時に呼び出されたのか判別できない．そこで，netCDF に繰り返しを通知する関数として，next_cycle 関数を導入する．図 6 プログラムでは，14 行目に next_cycle を挿入することになる．

netCDF ライブラリに対して通常の File I/O かプロセス間データ転送かを選択できるように nc_create, nc_open のアーギュメントに新しいモード NC_COMM を導入する．NC_COMM モードにおいては，nc_create 側の MPI ジョブは初めて当該 API が呼ばれた時に MPI_Comm_Accept を，nc_open 側の MPI ジョブは初めて当該 API が呼ばれた時に MPI_Comm_Connect を発行して通信路を確立させる．最初の回の実行時間は長くなるが 2 回目以降は動的通信路確

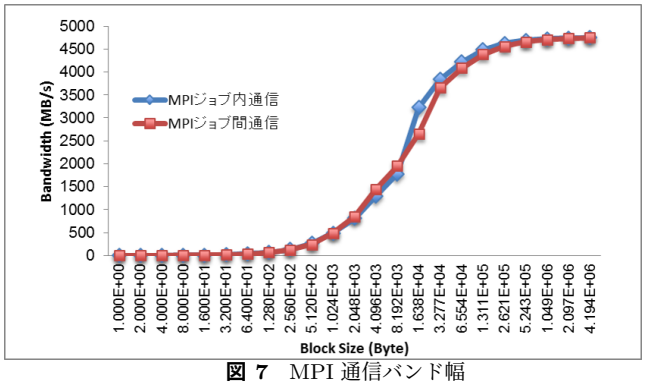


図 7 MPI 通信バンド幅

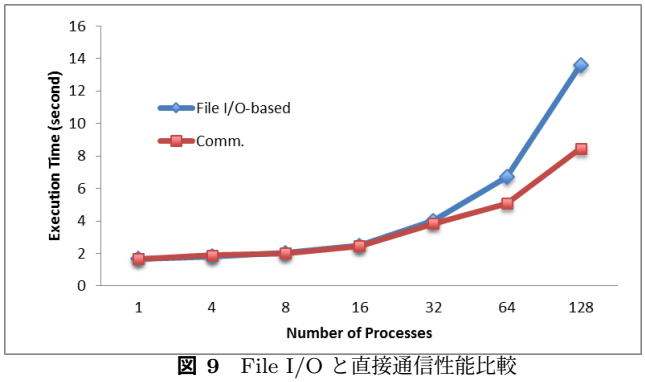


図 9 File I/O と直接通信性能比較

```
/* Simulator */
nc_create(...);
nc_def_dim(...);
nc_def_var(...);
nc_put_var_double(...);
nc_close()
```

```
/* Assimilation */
nc_open(...);
nc_inq_dimid(...);
nc_inq_varid(...);
nc_get_var_double(...);
nc_close(...);
```

図 8 評価プログラム概要

立時間は削減される．

3.3 データ転送

netCDF のファイル書き出し，ファイル読み出し API を修正し，MPI 通信ライブラリの Two sided 通信機能によりデータ交換を実現した．まず，MPI ジョブ内と MPI ジョブ間での MPI 通信バンド幅に差がないことを確認する．図 7 はオハイオ州立大学が提供している OSU ベンチマークのなかからバンド幅を計測する osu_bw による実行結果である．Two sided 通信において「京」では大きな差異はないといえる．

図 3 におけるシミュレーションプロセスとデータ同化プロセス間のデータ転送部分について，ファイル I/O 渡しと MPI 通信ライブラリの Two sided によるプロセス間データ転送とのコストを評価する．評価に使用したプログラムの概要を図 8 に示す．シミュレーションプロセス側はデータ構造を定義し，nc_put_var_double 関数で格子領域を書き込んでいる．データ同化プロセス側はファイルをオープンし，データ構造を読み出した後，nc_get_var_double 関数で格子領域を読み込んでいる．

図 9 に想定しているデータ転送サイズ 819KB における評価プログラムの実行時間結果を示す．X 軸はプロセス数であり，Y 軸は実行時間である．ファイル渡しによるデータ交換ではプロセス数の増加と共に実行時間が増し 128 プロセスでは 13.6 秒となっている．通信によるデータ交換では動的通信路確立時間が含まれているため，32 プロセスまではファイル渡しと同程度実行時間がかかっている，通信路確立時間を差し引くと 128 プロセスでは 4 秒程度に収まっている．

4. おわりに

本稿では，ゲリラ豪雨予測のための実時間天気予測システムに必要とされる周期実行起動方式および netCDF API を拡張することなくジョブ間で直接データ転送する方式の検討を行い，データ転送方式に関して予備実験を行った．本予備実験では，ジョブ間の起動に MPI 通信ライブラリが提供する動的通信路確立機能を用い，Two sided 通信を用いた実装を行った．

今回の実装では，netCDF ファイルを書き出すプロセスと読み込むプロセスの 2 つのプロセスに限定している．今後，複数のプロセスが一つの netCDF ファイルを読み書きするような，より一般的な netCDF を使った連成計算アプリケーションで利用できるよう検討する．また，RDMA 通信による高速化，非同期 I/O 処理の検討も行っていく．

謝辞

本研究の一部は，科学技術振興機構 CREST「科学的発見・社会的課題解決に向けた各分野のビッグデータ利活用推進のための次世代アプリケーション技術の創出・高度化」領域のなかの課題名「「ビッグデータ同化」の技術革新の創出によるゲリラ豪雨予測の実証」（研究代表者：三好建正）による．

参考文献

[1] 齋藤智之，石川　裕，Balazs, G.，三好建正，大塚成徳，富田浩文，西澤誠也：ジョブ間データ転送方式の検討，情報処理学会研究報告，Vol. 2014-HPC-143(2), pp. 1–6 (2014).
[2] 齋藤智之，石川　裕，Balazs, G.，Lien, G.-Y.，三好建正，大塚成徳，富田浩文，西澤誠也：実時間ゲリラ豪雨予測システムのためのファイル I/O 調停ミドルウェア試作，情報

処理学会研究報告, Vol. 2014-HPC-143(2), pp. 1–6 (2014).
[3] Li, J., Liao, W.-k., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B. and Zingale, M.: Parallel netCDF: A High-performance Scientific I/O Interface, *Supercomputing, 2003 ACM/IEEE Conference*, IEEE, pp. 39–39 (online), available from ⟨http://www.ece.northwestern.edu/ choudhar/Publications/LiLia03A.pdf⟩ (2003).
[4] Tomita, H.: SCALE-LES: Strategic development of large eddy simulation suitable to the future HPC, *Solution of Partial Differential Equations on the Sphere* (2012).
[5] Takemasa, Miyoshi, Yamane, S. and Enomoto, T.: Localizing the Error Covariance by Physical Distances within a Local Ensemble Transform Kalman Filter (LETKF), *Scientific Online Letters on the Atmosphere (SOLA)*, Vol. 3 (2007).
[6] Ushio, T., Wu, T. and Yoshida, S.: Review of Recent Progress in Lightning and Thunderstorm Detection Techniques in Asia, *Atmospheric Research*, Vol. 154, pp. 89–102 (2015).