CF-005

分散処理基盤のための粒子フィルタ実装手法の検討

佐藤 哲 †

NHN テコラス株式会社 データサイエンスチーム †

1. はじめに

粒子フィルタは，状態空間モデルを用いてオンライン学習をする手法として，動物体追跡，自己位置推定，データ同化，平滑化，欠損データ処理，周期予測，トレンド推定など，様々な分野で利用されている．特徴は汎用性及び実装の容易さであるが，一方で計算コストの高さが問題になることがある．粒子フィルタの計算コストは主に粒子の数に依存し，各粒子の確率密度計算などは独立に計算できるため並列分散処理の導入が容易であるが，再サンプリング処理は粒子を集約することが一般的であるため負荷が大きいとされている．そこで本論文では，分散処理基盤である Apache Spark†† に適した再サンプリング処理のコスト削減方法を提案する．提案手法は，(1) 各粒子毎に独立に計算できるアルゴリズムの導入，(2) 全粒子を複数の粒子群に分割し並列に処理するアルゴリズムの導入の 2 ステップから構成される．

2. 粒子フィルタの基本アルゴリズム

粒子フィルタの概念には様々なアルゴリズムが考案されているが，本論文ではモンテカルロフィルタ [1] と呼ばれるシンプルなアルゴリズムを対象とする．

モンテカルロフィルタのアルゴリズムは以下である．次のようなシステムモデルと観測モデルを考える:

$$\begin{cases} x_k &= F(x_{k-1}, v_k) \\ y_k &= H(x_k, \omega_k) \end{cases} \tag{1}$$

ここで，x_k 及び y_k は，時刻 k での状態ベクトル及び観測ベクトル，v_k 及び ω_k はシステムノイズ及び観測ノイズと呼ばれる量である．状態ベクトルは観測できない隠れ変数であり，1 ステップ前の状態ベクトルと観測ノイズによって決まると仮定し，観測可能な観測ベクトルは状態ベクトル及び観測ノイズによって決まると仮定する．このモデルに対し，次のアルゴリズムにより，観測ベクトルから状態ベクトルを推定する：

(1) システムモデルに基づき状態ベクトルを予測する．

$$p_k^{(i)} = F(f_{k-1}^{(i)}, v_k)$$

ここで，f_{k-1} は 1 ステップ前の状態ベクトルを近似する粒子ベクトルである．

(2) 観測値 y_k を得る．

(3) 予測値 $p_k^{(i)}$ と観測値 y_k より，観測モデルを用いて予測値の信頼度を表す尤度 $\alpha_k^{(i)}$ を求める：

$$\alpha_k^{(i)} = r(H^{-1}(y_k, p_k^{(i)}))|\partial H^{-1}/\partial y_k|$$

ここで，r は観測ノイズの確率密度関数であり，H^{-1} は関数 H の逆関数で状態ベクトル x_k と観測ベクトル y_k の関数であるが，状態ベクトルの代わりに予測値である $p_k^{(i)}$ を代入する．

(4) 粒子の尤度 α_k に基づき，粒子 p_k を再サンプリングし状態ベクトルを近似する粒子ベクトル f_k を求める：

$$f_k^{(i)} = 確率\ \alpha_k^{(j)} / \sum_{l=0}^{n-1} \alpha_k^{(l)} で\ p_k^{(j)}, 0 \le j < n$$

ここで，n は粒子の個数である．

(5) 粒子群 $f_k^{(0)}, f_k^{(1)},\ \ldots$ の期待値等を求め，状態ベクトルの推定値とする．

最後の処理 (5) では目的により期待値以外の量を使うこともあるが，マルコフ連鎖モンテカルロ法により粒子数が無限大の元で任意の関数の期待値が得られるため，期待値を計算することが多い．

このアルゴリズムのうち，(1) 及び (3) については各粒子 $p_k^{(i)}$ について独立に計算できる．(2) は計算の必要がない．(4) と (5) は集計の必要があるため分散処理が困難であり，特に (5) は出力が通常はベクトル 1 つであるが，(4) は粒子の個数だけのベクトルの入出力が必要となるため，負荷が高い処理となる．そこで本論文では，(4) の再サンプリングの負荷軽減手法を検討する．

3. 分散再サンプリング法の検討

前節で説明したアルゴリズムの再サンプリングステップ (4) を素朴に実行した例が図 1 である．左から 1 番目と 4 番目の粒子は尤度が低いために選択されず消滅している．3 番目の粒子は尤度が高いため選ばれる確率が高く，3 回選択されている．ところが 2 番めの粒子は低い確率ながら 4 番目の粒子として選択されている．このように 1 つの粒子を選択するために全ての粒子の情報を必要とするため，素朴な方法では記憶コスト・通信コストが高くなる．ま

A Study of Efficient Implementations for Particle Filters on Distributed Computing Platforms

†Tetsu R. Satoh, NHN Techous Corp.

††http://spark.apache.org/

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

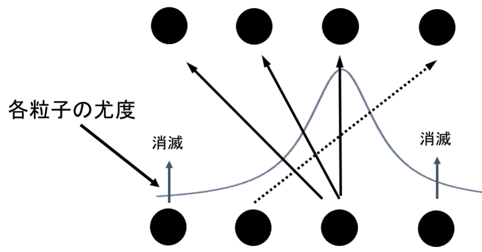

図 1: 素朴な再サンプリングの実行例

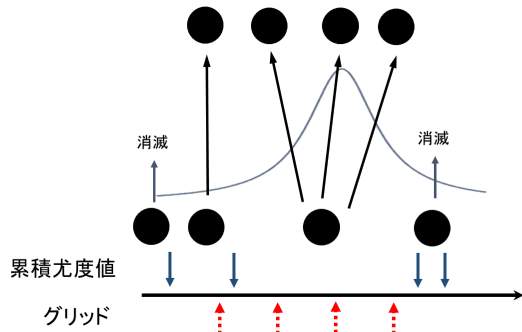

図 2: 層化抽出の例

た，アルゴリズム上でも，目的は尤度に基づいた粒子配列にすることであり，2 番目の粒子を 4 番目に移動させることに特別な意味はない．そこで図 2 に例を示すような層化抽出を利用した手法がしばしば採用される．層化抽出を利用した手法では，尤度の累積値：

$$w_k^{(i)} = \sum_{j=0}^{i} \alpha_k^{(j)} / \sum_{l=0}^{n-1} \alpha_k^{(l)} \tag{2}$$

及びグリッドを考える．累積尤度の列は，高い尤度 $\alpha_k^{(i)}$ の属する区間 $(w_k^{(i-1)}, w_k^{(i)}]$ が広くなるため，区間 $(w_k^{(i-1)}, w_k^{(i)}]$ により多くのグリッドが入る．そのため，区間中のグリッド数をカウントすることにより，図のように尤度の大きさに基づいたサンプリングが可能となる．しかしながら，層化抽出ではグリッドがどの尤度が属する区間にあるか探索しなければならず，粒子数が増えるとサーチアルゴリズムに応じた計算量が必要となる．この問題を緩和する手法としては，残差サンプリング法やマルコフ連鎖モンテカルロ法による手法があげられる [2]．しかし残差サンプリング法は全ての再サンプリングを避けることはできず，また，マルコフ連鎖モンテカルロ法は素朴な再サンプリングと同様に全ての粒子の情報を必要とするため，記憶コスト・通信コストが高い．

そこで本研究では，累積複製数から複製数を計算する手法 [3] を採用し，各粒子が独立に自身の複製個数を計算することで層化抽出における探索処理を不要とし，効率的に残差サンプリングを実行する手法を提案する．累積複製数及び複製数の計算は分散処理基盤 Spark の Window 関数を用いることで効率的

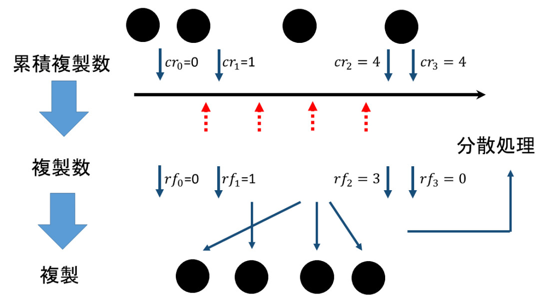

図 3: 累積複製数からの残差サンプリング

に計算できる．さらに，粒子群を確定的なサンプリングにより複数のグループに分割し，それぞれ独立に再サンプリングを実行することにより処理速度向上と精度向上を図る．既に述べたように粒子フィルタにおいて再サンプリングの目的は尤度に基づいた粒子配列にすることであるため，処理は粒子毎に独立に行ってよく，層化抽出の数値的な処理を考慮すると全粒子を処理するよりも分割する方が精度が良くなる可能性がある．全粒子を複数のグループに分割する従来研究 [4] と比べると，各グループ内での再サンプリングも分散処理されることが特徴と言える．

4. 提案手法

提案する再サンプリング手順は以下である：

(1) 粒子にユニークな数値 ID を割り当てる
(2) ID を元に，決められた分割数 m による剰余を求めグループ分けする
(3) 各グループに対し並列に以下を実行する
  (1) 各粒子の尤度から累積尤度 $w_k^{(i)}$ を求める
  (2) 各粒子に対し，独立に累積複製数 CRF_i を求める
  (3) 累積複製数より，各粒子の複製数 RF_i を求める
  (4) 複製数が 0 の粒子に対し，複製数が 2 以上の粒子を配分する
(5) 全グループの粒子を集約する

累積尤度 $w_k^{(i)}$ の定義は式 (2) である．Spark SQL API では，例えば次のような擬似コードにより計算できる：

```
val cond = Window.partitionBy(‘partition)
 .orderBy(‘id)
 .rowsBetween(Window.unboundedPreceeding,0)
val cumweight = particles
 .select(sum(‘weight).over(cond))
```

累積複製数 CRF_i(Cumulative Replication Factors) は，次のように計算することができる．まず，層化抽出のためのグリッド間隔 U_{size} を計算する：

$$U_{size} = \frac{\max(w_k^{(0)}, w_k^{(1)}, \cdots, w_k^{(n-1)})}{n} = \frac{w_k^{(n-1)}}{n} \tag{3}$$

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

次に，グリッドの先頭位置を計算する：

$$u_0 = \frac{w_k^{(n-1)} - w_k^{(n-1)}/2}{n} = \frac{w_k^{(n-1)}}{2n} \tag{4}$$

続いて各粒子の累積複製数 CRF_i を計算する．グループの先頭の粒子の累積複製数を 1 に初期化し，

$$CRF_0 = 1 \tag{5}$$

各粒子に対し，累積複製数を以下のように求める：

$$CRF_i = \left\lfloor \frac{w_k^{(i)} - u_0}{U_{size}} \right\rfloor + CRF_0 \tag{6}$$

記号 $\lfloor\rfloor$ は床関数を表す．これらの計算は他の粒子の情報に依存しないため，独立に計算可能である．

全ての累積複製数が求まれば，差分を取ることにより各粒子の複製数 RF_i(Replication Factors) を計算できる．Spark SQL API では，例えば次のような擬似コードになる：

```
val cond = Window.partitionBy(‘partition)
 .orderBy(‘crf)
val rf = cumweight
 .withColumn(‘‘lag’’, lag(‘cumweight , 1, 0)
 .over(cond)
 .select(‘cumweight-’lag))
```

複製数が求まった後，粒子を分配する分散処理アルゴリズムは開発されていない．ただし，提案手法では全粒子を複数の粒子グループに分ける処理を行うため，適切な粒子数と分割数を選択することにより逐次処理負荷を分散することができる．いずれにせよ，複製数の合計が粒子数に等しくなるよう設計されており，複製数が 0 の粒子に対し複製数が 2 以上の粒子をコピーすることで，再サンプリングを実現できる．

以上に述べた提案手法では，全粒子を複数のグループに分割するためのサンプリングと，粒子を尤度に基づき再配置するためのサンプリングの 2 段階のサンプリングを実施するが，いずれも乱数を使わない確定的な方法である．各粒子の累積複製数を求めてから複製数を計算し，粒子を複製するまでの処理の概念図の例を図 3 に示す．分散並列処理については，複数のグループに対する処理を同時に呼び出す部分はマルチスレッドを利用し，呼び出された処理，すなわちグループ内の各粒子に対する計算を並列に実行する部分は分散メモリ型分散処理基盤である Spark の機能を用いる．

5. 実験例

提案手法は任意の次元のデータ系列に適用可能であるが，簡単のために時間をパラメータとしてスカラー値を対応させる 1 次元の時系列データに対し提案手法を適用した例を紹介する．

入力データはネットワークのパケットキャプチャデータであり，約 58 万レコードである．システムモデル，観測モデルともに状態ベクトル及び観測ベクトルとノイズの線形和であると仮定し，システムノイズはコーシー分布，観測ノイズはガウス分布に従うとした．すなわち状態空間モデルは以下である：

$$\begin{cases} x_k &= x_{k-1} + v_k, v_k \leftarrow \tau/(\pi(t^2+\tau^2)) \\ y_k &= x_k + \omega_k, \omega_k \sim N(\mu, \sigma^2) \end{cases} \tag{7}$$

システムノイズにコーシー分布を採用した理由は，対象とするネットワークのトラフィックはしばしば大きな変動が発生することが分かっているため，値域の広い分布が必要であるためである．パラメータは，粒子数 n は $500, 1000, 2000$ と変化させ，システムノイズのパラメータは $\tau = 50.0$，観測ノイズのパラメータは $\mu = 0, \sigma = 5.0$ とした．

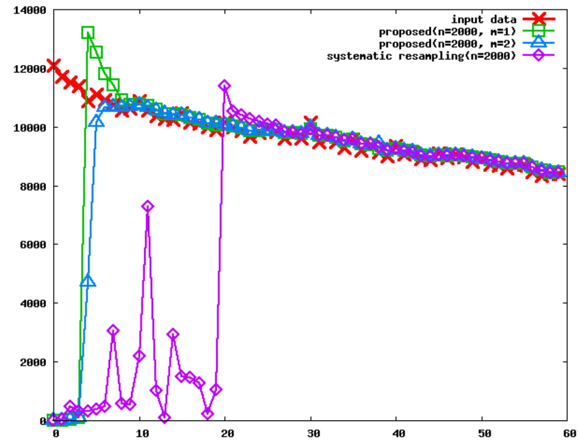

図 4: 時系列データのフィルタリング例

これらの条件の元で，データに対し粒子フィルタを適用した例のうち，粒子数 $n = 2000$ の場合の結果を図 4 に示す．この粒子数を選んだのは，どの粒子数でも傾向が同じであったので見やすさのために 1 つを選んだだけで特に理由は無い．マーカーが X 印（赤）が入力データ，四角印（緑）が分割数 $m = 1$ での提案手法によるもの，三角印（青）が $m = 2$ での提案手法によるもの，ダイヤ型（マゼンタ）が図 2 で説明した通常の層化抽出によるものである．縦軸は毎分当たりのキャプチャパケット数，横軸は観測開始からの時間（分）である．事前知識を仮定しないため，状態ベクトルの初期値は $x_0 = 0.0$ とした．結果は，$m = 2$ での提案手法が最も速く観測値を捉え，次が $m = 1$ の提案手法でありどちらも 10 分以内である．通常の層化抽出では 30 分ほどの位置で捉えることができている．ただしここで示している時間は入力データの位置であり，計算時間では無い．

次に，フィルタリングの計算時間を図 5 に示す．「分割無し」「分割数 2」が提案手法によるもので，「層化抽出」が図 2 により説明した手法を用いた結果である．図中で，層化抽出によるものは粒子数が増えると非常に時間がかかったため，グラフの上部を省略している．グラフ描画に用いた数値データは表 1 に

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

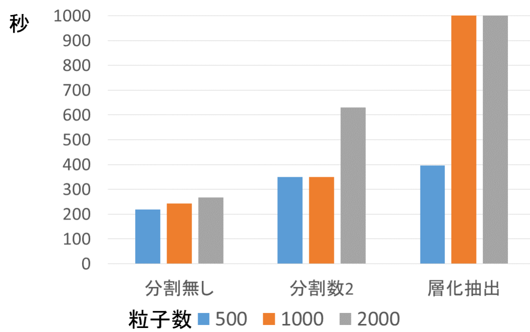

図 5: 計算時間

示す．図と表より，以下のことが分かる．

(1) どの条件下でも提案手法の処理速度の方が速く，単純な層化抽出と比べて 1.7%から 55%の時間で終了している

(2) 提案手法の中では，全粒子を 2 分割した方が処理時間が増大している

(3) いずれの手法でも粒子数の増大とともに処理時間が増大しているが，ある粒子数から急激に処理時間が増大している

(4) 提案手法の中では，全粒子を 2 分割して各独立に再サンプリングを実行する方が若干速く観測値近辺に収束している

これらの項目について考察する．(1) は，グリッドが累積尤度のどの区間に含まれるか検索する時間を削減できるので明らかである．(2) は実装方法に問題があることが分かっており，今後解決できる見込みである．(3) は，処理系の Apache Spark が Java の JVM（Java Virtual Machine）上で動作しており，JVM のメモリガベージコレクション処理が原因と考えている．この問題がシステムのスケールアップ・スケールアウトで解決できるのか，実装の改善により解決できるのかは分かっておらず，今後の課題である．(4) について調べるために，粒子数 $n = 2000$ で分割数 4 の場合を加えて比較した結果を図 6 に示す．この図ではダイヤ型のマーカーが分割数 4 の場合を表す．分割数が 2 と 4 では差はほとんど見られず，1 ステップほど分割数 2 の方が速く観測値に収束している．このことから，再サンプリングを実施するグループ内の粒子数（分割数 2 の場合，2000/2=1000）が適切であったことが示唆される．一般に，外れ値に対応するためにコーシー分布のような広い値域を持つ関数を利用する場合，尤度が数値的に計算機誤差に近づくような小さな値になることが多く，粒子の密度が高すぎると再サンプリングの効果が落ちることがある．そのため，提案手法の再サンプリング法が性能向上につながることがあり得る．

以上の実験は Apache Spark-2.3.0 クラスタ上で実施し，プログラミング言語は主に Scala-2.11(Java-1.8) を用いた．クラスタのマシンのうち，主要な計算処理を実施するワーカノードは 9 台で，スペックは

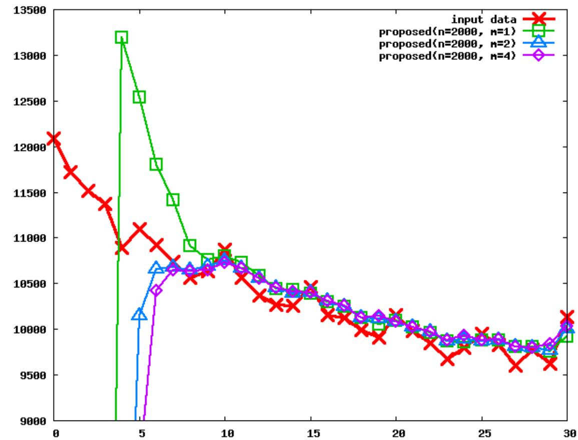

図 6: 分割数 4 の計算結果を加えた例

CPU は Intel Xeon X5690，メモリ 72G バイトである．集約処理を実施するクライアントノードは CPU は Intel Xeon E5-2643，メモリ 64G バイトである．

6. おわりに

本論文では，粒子フィルタの実装における再サンプリング処理について，(1) 各粒子毎に独立に計算できるアルゴリズムの採用，(2) 全粒子を複数のグループに分割し並列に処理するアルゴリズムの採用，の 2 段階の手法により，計算コストを削減する手法を提案した．今後の課題は，メモリや通信処理の最適化など実装上の問題の解決，より集約修理を削減する分散処理アルゴリズムの開発があげられる．

参考文献

[1] 北川源四朗, モンテカルロ・フィルタおよび平滑化について, 統計数理, Vol. 44, No. 1, pp. 31-48, 1996.

[2] T. S. Liu and R. Chen, Sequential Monte Carlo Methods for Dynamic Systems, J. Am. Stat. Assoc., Vol. 93, No. 443, pp. 1032–1044, 1998.

[3] Q. Gan, J. M. P. Langlois and Y. Savaria, A Parallel Systematic Resampling Algorithm for High-Speed Particle Filters in Embedded Systems, Circuits Syst. Signal Process., Vol. 33, No. 11, pp. 3591–3602, 2014.

[4] M. Bolić, P. M. Djurić and S. Hong, Resampling Algorithms and Architectures for Distributed Particle Filters, IEEE Trans. Signal Process., Vol. 53, No. 7, pp-2442–2450, 2005.

表 1: 計算時間（秒）

粒子数	分割無し	分割数 2	層化抽出
500	219	349	397
1000	242	350	1018
2000	268	629	14903

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.