Vol.2012-DBS-156 No.6
2012/12/12

ストリームデータ処理における異常検知手法の共有化に関する検討

川島英之[†2]　大桶真宏[†1]　北川博之[†2]

本論文ではストリーム処理システムをもとにマルウェア検知基盤システムを開発する取組に関して述べた後，異常検知手法である Change Point Detection (CPD) を我々が開発しているストリーム処理システム SS*へ実装する方法を述べる. そして複数の CPD を効率的に実行するために CPD の一部を共有計算する手法を提案する. 簡易実験の結果, 共有計算により最大で 206%程度の性能向上が得られることを述べる.

A Consideration on Sharing of Outlier Detections on Stream Data Processing

HIDEYUKI KAWASHIMA[†2]　MASAHIRO OHKE[†1]
HIROYUKI KITAGAWA[†2]

This paper first describes our work on constructing a malware detection infrastructure system based on stream processing system. Then we describe how we implement an anomaly detection method “Change Point Detection (CPD)” onto our stream processing system SS*. Then we propose a method to share computations among multiple CPD runs. The result of initial experiment showed that shard computation achieved 206 % of performance improvement at the maximum case.

1. はじめに

トラフィック分析によるマルウェア検知の手法には様々なものが現存する. 新種のマルウェアが毎年現れてくる点, ならびにマルウェアを防ぎたいという要求がなくならない点を鑑みれば，検知手法はこれからも増加すると考えられる．また，ある検知手法にパラメータが存在し，そのパラメータ設定の是非により検知率が異なる場合もある．すなわち, k 個の手法があり，各手法に l 個のパラメータ設定数があるならば，運用すべき検知手法は $k \times l$ 個と多数になることがわかる．なお，l は複数の変数により決定されることもあり得る点に注意されたい.

多数のマルウェア検知手法を効率的に運用することは容易ではない．運用作業は煩雑であるし，手法数の増加に伴い性能が劣化するからである．複数のマルウェア検知手法を運用する際，ナイーブな方式として，各手法を別のプログラムとして実装し，別々にコンパイルして，それらを別プロセスとして動作させる方式が考えられる．この方式は単純であるために実現が容易だが，次の 2 つの問題が存在する.

第一の問題は，運用に関する問題である．プログラムの整理・起動が煩雑であることに加えて，処理結果を受信するインタフェースが統一化されているとは考えられないため，処理結果を利用したプログラムの作成は複雑・困難となる.

第二の問題は，性能に関する問題である．システムの入力であるパケットストリームは別々の形式で処理されるため，N 個のプロセスに同じパケットデータをフィードせざるを得ない. プロセス間通信にはシステムコールを要する. パケットストリームは非常に頻繁に到着するため，システムコールに伴う性能の劣化度合いを，小さいと看過することはできない.

これらの問題を解決するために，我々はストリーム処理システムをベースにしたマルウェア検知基盤システムを開発してきた．提案システムはパケットストリームをリレーショナルストリームとしてモデル化し，複数の分析手法を SQL ライクな問合せ言語により記述可能にする．これによりユーザは SQL を発行すれば結果をタプルストリームとして受信可能になる．これにより分析手法は連続的問合せとして一覧可能になり，分析手法の起動と停止は分析基盤を通して提供されるために実現され，ユーザプログラムから分析基盤へのインタフェースは統一化される．従って，我々の提案基盤を用いることで, 第 1 の問題は解決される.

また，我々のマルウェア検知基盤システムにおいては，パケットストリームと各分析手法へのパケット転送はシステムコールを介さずにアプリケーションメモリ空間内で実現される．この理由は次の通りである．パケットがシステムに到着すると入力アダプタへルーティングされる．ここでパケットはバイナリイメージからタプルイメージへとメ

†1 筑波大学 情報科学類
College of Information Science, University of Tsukuba.
oke@kde.cs.tsukuba.ac.jp
†2 筑波大学システム情報系
Faculty of Information, Systems and Engineering, University of Tsukuba.
{kawasima, kitagawa}@cs.tsukuba.ac.jp

モリ形式が変更される．入力アダプタは当該タプルを分析に用いる連続的問合せの葉ノードを検索し，見つかったノードの入力キューへとタプルを enqueue する（物理的にはコピー処理）. この処理においてシステムコールは不要であることは明らかである．それゆえ，第 2 の問題はある程度の解決を得る．しかしながら更に複数パラメータ設定に伴ってスケールに関する問題が発生する．例を通して，この問題について述べる．

マルウェアによるアクセスパターンをトラフィックにおける異常アクセスと見做すことがある．その異常アクセスを検知する手法として，時系列回帰分析は有力な手段の 1 つである．例えば NICTER においては Change Point Detection (CPD)と呼ばれる，忘却の概念を導入した回帰分析手法を用いて異常アクセスを検知し，異常アクセスをマルウェアによるアクセスと見做す研究がおこなわれている [1,2]．CPD は強力な手法であるが，パラメータ設定が容易ではないという欠点がある．即ち，パラメータ設定を誤ると侵入検知が不能になる可能性がある．これを防ぐためには, 異なるパラメータを有する複数の CPD を並列に動作させることが求められる．

このような並列動作を高速化するために，本論文では CPD の共有計算技法を検討する．ストリーム処理システムにおいては common sub expression を共有することで高性能化を達成する手法が広く研究されてきた [15]．しかしながらそれらの手法が対象とする演算はリレーショナル演算に限定されてきた．本研究ではリレーショナル演算ではない CPD について，その内部処理である SDAR を分析し，共有可能な部分についての考察を行う．

本論文の構成は次の通りである．2 節では関連研究について述べる．3 節では多数の検知手法を管理することに伴うコストについての考察を行う． 4 節では，まず異常分析手法である Change Point Detection について述べると共に，その SS*への実装方式について述べる. そして Change Point Detection の共有化方式についての検討を行うと共に，SS*への実装方式についても検討を行う．最後に 5 節では本論文をまとめる．

2. 関連研究

2.1 異常検知手法

異常検知には様々な手法がある．距離に基づく外れ値, 密度に基づく外れ値など種々の技法が知られている．パケットストリームから異常を検知する研究には回帰分析が使われることがあり，AR モデルに忘却の概念を導入した CPD [1]などもある．また，異常データのペイロードを学習させておき分類する方式も存在する．教師付学習のコストは通常高価であるため，様々な分散処理ライブラリが存在する．その例としては Hadoop を用いる Mahout [3]，単一マシン上の複数コアを用いる Mallet [4], 非同期並列処理により高性能化を達成する Jubatus [5]などが挙げられる．

これらのシステムは強力であるが，いずれも管理システムではない. それゆえ, 複数の手法を同時に走らせるとき，共有計算を行うことができない．また，これらの手法は DBMS ならば提供する選択・射影・結合・集約などの基本的演算を提供しないため，それらの処理と組み合わせるためには, DBMS や Hadoop などとの連携作業が必要になる．この場合，運用が煩雑になると共にシステム間データ転送に起因する性能下落が生じうる．

2.2 In-DBMS Analytics

通常，DBMS は選択・射影・結合・集約などの基本的演算しか提供せず, 分析システムは分析処理のみを提供する．これらをまとめてしまい，DBMS 内部で解析処理を行う方式は in-DBMS analytics と呼ばれる [6, 18, 19]．DBMS は特定の演算のみを提供する．例えば RDBMS の場合には選択・射影・結合・集約などの限られた種類の演算のみを提供する．そのため，機械学習・データマイニングなどの分析を行いたいユーザにとっては処理機能が不足する．そこで DBMS 内部に機械学習・データマイニング等に関する様々な処理機能を追加するというコンセプトが in-DBMS analytics である．この例には MauveDB [7], WaveScope [17], MADlib [6, 19], Bismarck[18] が挙げられる．

上とは逆に解析処理を DBMS 外部で行う方式を out-DBMS analytics 方式と記す．Out-DBMS analytics の例としては SciDB と Python or R クライアントとの連携，PostgreSQL と Mathematica の連携などが考えられる．

In-DBMS analytic 方式が out-DBMS analytics 方式に対して優れる点は，性能とユーザインタフェースである．性能差はデータ移動に起因する．out-DBMS analytics 方式においては, 分析を行うには DBMS 内部のデータを外部へ移動しなければならない．データ量が小さければ，この移動に関するコストは小さい．しかしながら巨大なデータを扱う場合には，その移動コストは莫大になる．

次にユーザインタフェースに差が存在する．ここで我々の意図するユーザインタフェースとは，分析プログラムにおいて巨大データを論理的に記述可能か否か，である．R や Matlab では最大でもメモリに乗るだけのデータサイズでなければ配列を記述できない．それゆえプログラマはこの制約を満足させるために DBMS アクセスを考慮したプログラムを記述することを余儀なくされる．一方 in–DBMS analytics 方式ではメモリサイズを超える量のデータについても論理的には取扱いが可能になる．それゆえプログラマはメモリサイズを気にせずに分析ロジックを記述可能である．

ただし in-DBMS analytics 方式の実現には大きなコストを要することを欠点として指摘されなければならない．

上記では DBMS における in-DBMS analytics について述べた．一方，in-SPS analytics は筆者らの知る限り存在しな

い.

2.3 ストリーム処理システム (SPS)

マルウェア検知に際してはパケットを処理する必要がある．パケットは連綿と終わりなく連続的に到着するデータである．このようなデータを処理する基盤システムとしてストリーム処理システムがある．

ストリーム処理システムとは，問合せを永続的に保持し，データがシステムに到着する度に 問合せを評価するシステムである．一部の演算（例：結合，集約）についてはシノプシスと称される中間結果を保持することがある．問合せには宣言的言語 [8, 9]，あるいはデータフロー記述言語 [10, 11]が用いられる．ストリーム処理システムをデータモデルから分類すると，非順序集合を基礎としてリレーショナル演算系 [8-11]を提供するものと，順序集合を基礎として正規表現の一部を提供するもの [12]とに二分できる．

ユーザはまず連続的問合せを登録する．そして実行したい連続的問合せを稼働させる．当該連続的問合せはシステム内部においては演算子をノード，演算子間のキューをエッジとするグラフへと変換される．これは演算グラフなどと呼称される．演算グラフのルートはユーザへの転送処理を担い，演算グラフのリーフは入力アダプタと接続している．演算グラフのリーフに該当するノードの入力キューへ入力アダプタからタプルがルーティングされると，リーフノードから上部へと順々に処理結果がルーティングされる．なお, リレーショナルストリーム処理システムにおいては，エッジに存在するデータは全てタプルとなる．

2.4 SS*

SS*は我々が開発しているストリーム処理システムである．SS*はStreamSpinner [9]の後継として開発されている．SS*はSQLライクな宣言的問合せ言語を提供する. 各問合せは登録され，そしてユーザの指示により起動される．起動した問合せはパケットが到着する度に評価され，その結果を出力する．

問合せはリレーショナルスキーマに対して実行される．マルウェア検知システムの場合，パケットを次の属性を有するリレーショナルスキーマとして表現する．パケットは次の属性を有するタプルとして表現される：source IP, destination IP, source port, destination port, TCP sequence number, packet size, arrival time, protocol (ex: TCP = 6), {ACK|FIN|SYN|URG|PUSH|RESET} flags. なお, パケットをNICから取得するにあたってSS*はNEGI [16] を利用しており，SS*とNEGIの接続においては同一プロセス別スレッドとしての実装を行っている．

パケットがタプルに変換された後，各問合せの入力キューにタプルの複製が挿入される．この複製作業はプロセス内部で実行される．それゆえ性能に関する問題を提案基盤は解決している．

3. 検知手法の管理方式

多数のマルウェア検知手法を管理することを考えるとき，次の 2 つの手法が考えられる．

3.1 個別プロセス方式

この方式は各手法を個別に実装し，同時に動作させる方式である．各手法は別のプログラムとして開発され，別プロセスとして動作する．

この方式の長所は開発と運用の柔軟性である．各方式を異なる言語で開発可能であるため，開発の柔軟性は高い．各手法の開始・停止は個別に行えるため，運用の柔軟性が高い．

この方式の欠点は性能劣化である．パケットを各プロセスに複製・配送する必要があるため，パケットのプロセス間移送に時間がかかってしまう．

3.2 同一プロセス・一括コンパイル方式:

各手法を同じ言語（C++/Java 等）で個別に実装した後，まとめてコンパイルを行い，そして 1 つのプロセスとして動作させる方式である．

この方式の長所は性能である．全手法は同一プロセス中で別スレッドとして実現されるため，プロセス間データ移送が発生しない．

この方式の欠点は柔軟性である．各手法は同一言語で開発される必要があるため, 開発言語の柔軟性が低い. また，各手法はまとめてコンパイル・起動・停止される必要があるため，運用の柔軟性が低い．

3.3 SPS を用いた管理方式

求められる方式は，高い開発・運用柔軟性と高い性能を同時に提供する方式である．これを実現するためには次のことを行えばよい．

(1) 高い性能を実現するために, 各手法を同一プロセス中の別スレッドとして実現する．
(2) 高い運用柔軟性を実現するために，各スレッドの開始・停止を管理可能にする．
(3) 最後に高い開発柔軟性を実現するために, 各スレッドから分析手法を呼び出す機構を用意する．

これを同一プロセス・動的手法管理方式と本論文では記す．同一プロセス・動的手法管理方式を実現するシステムには様々なものが考えられるだろうが，本論文ではストリーム処理システム(SPS)を用いて同一プロセス・動的手法管理方式が実現できることを示す．SPS 内部では各問合せは別スレッドとして実現されるため，上記 (1) は満足される．各問合せは専用コマンド（run/stop 等）により開始・停止可能であるため (2) は満足される．問合せから分析処理を行う組込関数を呼出して演算子として実現するため，(3) は満足される．

4. 異常検知手法の共有化

4.1 異常検知手法：Change Point Detection

本論文では Change Point Detection (CPD) [1]と呼ばれる技法を実装する．CPD は AR モデルに逐次学習と忘却機能を導入した SDAR アルゴリズムを用いる．逐次学習とはデータを 1 つ読み込む度に AR モデルを学習する．忘却機能とは i 時点前のデータの影響が $(1-R)\times i$ 倍に減少するようにする機能である．

SDAR (Sequentially Discounting AR model learning)アルゴリズムとは現在までの T 時間のデータから AR モデルを学習するアルゴリズムである．SDAR は学習した AR モデルを用いて現在のデータを推定し，実際の現在のデータに対する推定値の外れ値らしさを計算する．

時系列データに対して第一段階の SDAR アルゴリズムを適用して外れ値を検出し，第二段階の SDAR アルゴリズムを適用して変化点を検出する．アルゴリズムの手続きは下記のようになる．

1：x_Tを入力;
2：SDARでx_Tの確率密度を学習;
3：外れ値スコアを計算;
4：スコアの移動平均y_Tを計算;
5：SDARでy_Tの確率密度を学習;
6：変化点スコアを計算;
7：2 へ戻る;

アルゴリズム 1：Change Point Detection

4.2 CPD の実装

我々は Eigen [13]と C++言語を用いて CPD を実装した．Eigen を利用した理由は，CPD の計算において逆行列，転置行列，行列積などを求める必要があったからである．Eigen の利用により，CPD に要したコード行数は 300 程度で済んだ．SS*においては CPD を組込関数として実装し，リレーショナル演算として表現した．具体的には次のパラメータを演算子に管理させる実装を行った．パラメータを関数レベルで管理すると，複数の CPD がパラメータを共有してしまうことになり，本来異なるべきであるモデルが混合してしまって意味をなさなくなる．それゆえ演算子によるパラメータ管理が必要であった．

- ✓ 時刻
- ✓ 第一段階 SDAR から得られる確率密度
- ✓ 第二段階 SDAR から得られる確率密度
- ✓ 入力ベクトル$x_1 \dots x_T$
- ✓ 期待値 μ
- ✓ 新しい入力の推定値
- ✓ 共分散$C_0 \dots C_K$
- ✓ AR係数$A_1 \dots A_K$
- ✓ 共分散行列

これにより CPD は集約関数の一種として扱うことを可能にし，Grouping 演算からも呼出すことを可能にした．即ち，下記のような問合せを実行可能にした．この問合せでは CPD はデフォルトパラメータで動作する（3 つのパラメータを設定可能．パラメータは 4.3, 4.4 節を参照のこと）．そして dst_port 毎に AR モデルを構築し，1 秒間のウィンドウについて，各ポートに関する CPD スコアを計算して出力する．

```
SELECT dst_port, cpd()
FROM packet[1 sec]
GROUP BY dst_port;
```

4.3 CPD のパラメータ依存性

CPD には 3 つのパラメータ（忘却率 R，AR 次数 K，移動時間 T）をユーザが指定する必要がある．マルウェア検知に際して適切であるパラメータを自動的に見つける方法は我々が知る限り存在しない．従って 1 つのパケットストリームに対して，パラメータの異なる複数の CPD を適用することが，マルウェア検知の場合には必要であると考えられる．パラメータが異なることで検知率が変動することを図 1 と 2 に示す．両図において横軸は経過時間，左縦軸はアクセス数（青線に対応），右縦軸は CPD スコア（茶線に対応）である．青線は両図で同じであるが，茶線は異なる振る舞いを示している．図 1 では CPD に適切なパラメータを与えられているため，アクセス数が高い場合に高い CPD スコアを示している．一方，図 2 では CPD に不適切なパラメータを与えられているため，CPD スコアが高くならず，異常を検知できなくなってしまっている．

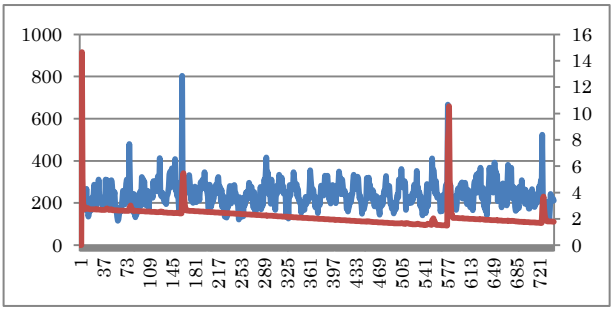

図 1：適切なパラメータを有する CPD の挙動

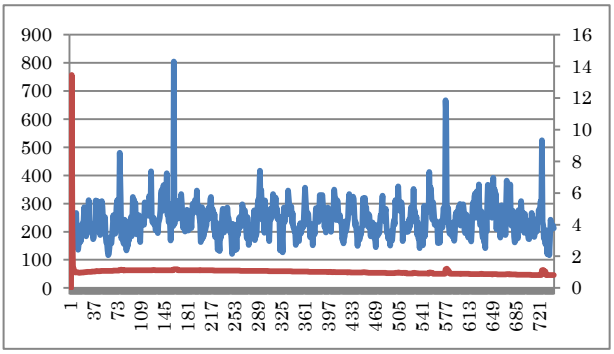

図 2：不適切なパラメータを有する CPD の挙動

4.4 CPD のパラメータ共有化に関する検討

パラメータには忘却率r，AR次数k，移動時間Tがある．これらのうち，移動時間Tは移動平均スコアy_tと変化点スコアの計算時にのみ用いられる．忘却率rおよびAR次数kは第一段階学習，第二段階学習におけるSDARアルゴリズムで用いられる．またSDARアルゴリズムの最初の計算である平均μの計算では忘却率Rのみを用いる．

ある入力に対してパラメータの異なる複数の CPD を適用する場合，以下の場合において計算結果の共有を行うことができる．

(1) 移動時間 T が異なるが，忘却率 R と AR 次数 K が等しい複数の CPD を実行する場合

この場合では第一段階学習におけるSDARアルゴリズムの結果を共有することができる．第二段階学習では第一段階学習結果からの移動平均スコアy_Tを用いるため，計算結果の共有は不可能である．N個のCPDを適用する場合，共有しない場合SDARアルゴリズムで$2N$回計算する必要があるが，共有した場合SDARアルゴリズムでの計算が最小で $(N+1)$ 回で済む．

(2)移動時間 T と AR 次数 K が異なるが，忘却率 R が等しい複数の CPD を実行する場合

この場合では AR 次数が異なるため，第一段階の SDAR アルゴリズムの結果は共有できないが，SDAR アルゴリズムの結果の最初の計算である平均μの計算結果を共有することができる．

SS*において共有化方式の実装を行う際には，共有部分を 1 つの演算子とし，非共有化部分は異なる演算子として演算グラフを構築すれば良い．この処理はシステムが動的に行う必要がある．CPD を含む複数の連続的問合せを分析して，共有部分を抽出し，そして演算グラフを自動生成するロジックを実装する必要がある．

4.5 初期的実験

上記(1)の手法について初期的な実験を行った．実験機は CPU:Intel(R) Xeon(R) CPU E5640 @ 2.67GHz (4 cores) Dual CPU, RAM: 48 GB，の機能を有する．

各実験においては 100 件のCPDを実行させ，全ての処理を終えるまでの時間を測定した．3 種類のパラメータセットにおいて実験を行った結果（実験 1…3）を表 1 に示す．なお，表 1 において，x_R, x_K, x_T,は第一段階SDARのパラメータであり，y_R, y_K, y_T,は第一段階SDARのパラメータである．

実験結果は，共有化により最大で 206%の性能向上が得られたことを示している．第一段階 SDAR において AR 次数が大きい場合には計算量が増大するため，特に顕著な性能向上が観察されたと考えられる．

なお，共有化した場合と非共有化した場合において，CPD のスコアは同一であることを全ての実験において確認した．

表 1：共有化に関する初期的実験の結果

	実験 1	実験 2	実験 3
忘却率x_R	0.005	0.005	0.005
AR次数 x_K	2	3	5
移動時間 x_T	5	5	10
忘却率y_R	0.005	0.02	0.02
AR次数 y_K	2	2	2
移動時間 y_T	5	5	10
処理時間（共有無）	3269 ms	3424 ms	4517 ms
処理時間（共有有）	2123 ms	2130 ms	2190 ms
性能向上率	153 %	161 %	206 %

5. まとめ

本論文ではストリーム処理システムをもとにマルウェア検知基盤システムを開発する取組に関して述べた後，異常検知手法である Change Point Detection (CPD) を SS*へ実装する方法を述べた. そして複数の CPD を効率的に実行するために CPD の一部を共有計算する手法を提案した. 簡易実験の結果，共有計算により最大で 206%程度の性能向上が得られることを観察した．

今後の課題はさらなる高性能化と，SS*への実装である．

謝辞 本研究成果の一部は科研費[#24500106]および独立行政法人 情報通信研究機構（NICT）の委託研究「新世代ネットワークを支えるネットワーク仮想化基盤技術の研究開発」により得られたものである．

参考文献

1) Jun’ichi Takeuchi and Kenji Yamanishi, “A nifying Framework for Detecting Outliers and Change Points from Time Series,” IEEE transactions on Knowledge and Data Engineering, pp.482-492, 2006.
2) Daisuke Inoue, Katsunari Yoshioka, Masashi Eto, Masaya Yamagata,Eisuke Nishino, Jun’ichi Takeuchi, Kazuya Ohkouchi, and Koji Nakao, “An Incident Analysis System NICTER and Its Analysis Engines Based on Data Mining Techniques”, ICONIP 2008, Part I, LNCS 5506, pp. 579–586, 2009.
3) Mahout: http://mahout.apache.org/
4) Mallet: http://mallet.cs.umass.edu/
5) Jubatus: http://jubat.us/
6) MADlib: http://madlib.net/
7) Amol Deshpande and Samuel Madden. “MauveDB: supporting model-based user views in database systems”, SIGMOD 2006.
8) Arvind Arasu, Shivnath Babu, and Jennifer Widom. “The CQL continuous query language: semantic foundations and query execution”, The VLDB Journal 15, 2 (June 2006), 121-142.
9) StreamSpinner: www.streamspinner.org/
10) Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. “SPADE: the system s declarative stream processing engine”, SIGMOD 2008.
11) Yanif Ahmad, Bradley Berg, Uğur Cetintemel, Mark Humphrey, Jeong-Hyon Hwang, Anjali Jhingran, Anurag Maskey, Olga Papaemmanouil, Alexander Rasin, Nesime Tatbul, Wenjuan Xing, Ying Xing, and Stan Zdonik. “Distributed operation in the Borealis stream

processing engine”, SIGMOD 2005.
12) Eugene Wu, Yanlei Diao, and Shariq Rizvi. “High-performance complex event processing over streams”, SIGMOD 2006.
13) Eigen: http://eigen.tuxfamily.org
14) Patrick Leyshock , “Agrios: A Hybrid Approach to Scalable Data Analysis Systems”, XLDB 2012.
15) Yousuke Watanabe, Hiroyuki Kitagawa, “Query Result Caching for Multiple Event-driven Continuous Queries”, Information Systems, Elsevier, Vol.35, No.1, pp.91-110, January 2010.
16) NEGI: https://github.com/westlab/negi
17) WaveScope: http://www.cs.indiana.edu/~rrnewton/wavescope/WaveScope_+_WaveScript/WaveScope_Homepage.html
18) Aaron Feng, Arun Kumar, Benjamin Recht, and Christopher Ré, “Towards a Unified Architecture for In-Database Analytics”, SIGMOD, 2012
19) Joseph M. Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleks Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar, “The MADlib Analytics Library or MAD Skills, the SQL”, PVLDB 2012