# ストリーム処理システムを用いたマルウェア検知基盤システム

大桶 真宏† 川島英之‡ 北川博之‡

†筑波大学情報科学類
305-8573 茨城県つくば市天王台 1-1-1
oke@kde.cs.tsukuba.ac.jp

‡筑波大学システム情報系
305-8573 茨城県つくば市天王台 1-1-1
{kawasima, kitagawa}@cs.tsukuba.ac.jp

**あらまし** 多数のマルウェア検知手法を効率的に運用することは管理者にとっては容易ではない．なぜなら各手法は別のプログラムとして個別に実装されるからである．個別実装には2つの問題がある．第一の問題は効率である．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個のプログラムに同じデータをフィードせざるを得ない．第二の問題は運用である．プログラムの整理・起動が面倒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．そこで本研究で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提案する．提案システムはパケットをリレーショナルストリームとしてモデル化することで，上記の効率に関する問題を解決する．次に提案システムは検知手法を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ため，第二の問題は解決される．

# A Malware Detection Infrastructure System based on Stream Processing System

Masahiro Ohke† Hideyuki Kawashima‡ Hiroyuki Kitagawa‡

†College of Information Science
1-1-1 Tennodai, Tskuba, 305-8573, JAPAN
oke@kde.cs.tsukuba.ac.jp
‡Faculty of Information, Systems and Engineering
1-1-1 Tennodai, Tskuba, 305-8573, JAPAN
{kawasima, kitagawa}@kde.cs.tsukuba.ac.jp

**Abstract** This paper proposes a malware detection infrastructure system based on stream processing system. The proposed system models packets as relational streams to solve the efficiency problem and generates results as relational streams to solve operation problem. We realize the proposed system based on SS* which we have developed, and we implement conventional methods.

# 1　はじめに

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人類の望みがなくならない点を鑑みれば, それらの手法はこれからも増加すると考えられる.

残念なことに, 多数の手法を運用することは面倒であり, 管理者にとっては容易ではない. なぜならば, 一般的に各々の手法は別のプログラムとして実装されるからである. この個別実装方式には 2 つの問題がある.

第一の問題は効率である. 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 個のプログラムに同じパケットデータをフィードせざるを得ない. 第二の問題は運用である. プログラムの整理・起動が面倒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.

そこで本研究で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提案する. 提案システムはパケットをリレーショナルストリームとしてモデル化することで, 上記の効率に関する問題を解決する. 次に提案システムは検知手法を 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ため, 第二の問題は解決される. 提案システムを我々が開発してきたストリーム処理システム SS*上に実装し, 既存のマルウェア検知手法である[ref]を動作させ, 性能と運用に関して得られた知見を述べる.

本論文の構成は次の通りである. 2 節でマルウェア検知プログラムの管理に関する問題を述べる. 3 節でストリーム処理システムを用いた問題解決方法を述べる. 4 節ではストリーム処理システム SS*にマルウェア検知手法である group-by-aggregate と ChangeFinder を実装する方法を述べる. 5 節では分析手法の実装方法について述べる. 6 節では提案手法を評価する. ChangeFinder を個別実装した場合と SS* を用いる場合について論じる. また, RDBMS と SPS の性能差についても述べる. 最後に 7 節で本論文をまとめる.

# 2　多数の検知手法の管理コスト

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, 次の手法が考えられる.

1. 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. 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.

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.

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) まず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，各スレッドから外部ライブラリ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呼ぶ. これ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・動的手法管理方式を実現するシステムを提案する. 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 UDF を経由して外部ライブラリを呼び出し可能であるため(3)は満足される. この詳細について本論文で記述する.

# 3 関連研究

## 3.1 異常検知手法

パケットストリームから異常を検知する研究には AR モデルの利用をはじめとして[1]など様々なものがある. 教師有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. Hadoop を用いる Mahout [3]，単一マシン上の複数コアを用いる Mallet [4]，非同期計算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ため，複数の手法を同時に走らせることができない. また，データベースならば提供する選択・射影・結合・集約などの処理を提供しない.

## 3.2 In-DB 解析

DBMS 内部で解析処理を行う方式は in-DB 解析と呼ばれる. DBMS は選択・射影・結合・集約などの限られた種類の演算のみを有するため，機械学習・データマイニングなどの処理を提供しない. そこで DBMS 内部に機械学習・データマイニングに関する様々な処理を追加することで容易に複雑な処理を実現する方式である. この例には MauveDB[8], MADlib[6]が挙げられる. この方式は他の方式に比べて高い性能を提供できる. なぜならば他の方式では DBMS 内部の巨大なデータを外部へ移動しなければならず，その移動コストはとても大きいからである.

# 4 提案

## 4.1 ストリーム処理システム

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. 問合せには宣言的言語[9, 10]，あるいはデータフロー記述言語[11, 12]が用いられる. ストリーム処理システムにはリレーショナル演算系[9-12]を提供するものと，正規表現の一部を提供するもの[13]とがある.

## 4.2 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[10]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 TIME, SRCIP, DSTIP, SRCPORT, DSTPORT, DIRECTION, PAYLOADという

属性を有するリレーショナルスキーマとして表現する．パケットはこのスキーマを有するタプルとして表現される．

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方式は解決する．

各実行方式はスレッドにより実現されるが，起動と停止はコマンドにより実現されている．それゆえ提案方式は運用に関する問題を解決する．

SS*が提供する問合せ記述言語の計算能力はTuring完全ではない．それゆえSS*が提供できない処理については共有オブジェクトとして実現する必要があるが，SS*は共有オブジェクトとのリンクを実現している．それゆえ提案方式は開発の柔軟性に関する問題を解決する．

# 5　分析手法の実装

## 5.1　Group-by-Aggregate

攻撃を検知する単純な手法として，特定ポートへの一定時間ごとのアクセス数を数え上げる方法が考えられる．これを group-by-aggregate と記載する．この方法では，アクセス数が閾値を超えている場合を攻撃だとみなす．この率直な手法を宣言的言語により実現した．

分析システムとしてはリレーショナル DBMS (RDBMS)とストリーム処理システム(SPS)を用いた．両者の違いを図 1 に示す．RDBMS の場合，ストリームは分析前に永続的デバイスに書きこまれる必要がある．一方，SPS の場合，ストリームはオンメモリで分析処理される．必要ならば分析結果を永続的デバイス上のテーブルと結合する．実装に関する詳細を下記に述べる．

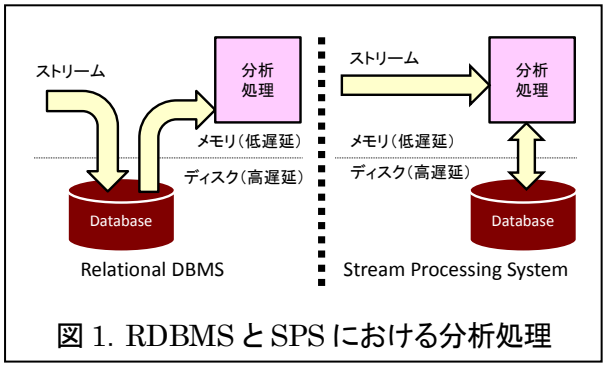

図 1. RDBMS と SPS における分析処理

### 5.1.1　RDBMS での実装

上述の方法を実現する手法として RDBMS の利用が考えられる．パケットデータをいったん RDBMS に格納し，そして問合せを用いて分析を行う手法である．データを RDBMS に格納するには永続的デバイスへの書込みが必要である．なぜならデータベースへのアクセスはトランザクションである必要があり，トランザクションは ACID 性を有しており，それには atomicity と durability が含まれるからである．

また，分析処理を行うためには，問合せを何等かの方法で発行する必要がある．ストリームデータは時々刻々と発生するため，問合せは周期的に実行される必要がある．この実行は DBMS 外部のクライアントによって管理される必要がある．従ってこの方式を用いるとクライアント側の問合せ管理コストが増大する．

また，この方式は問合せ結果を得るまでに時間がかかる．なぜならば問合せがクライアントから発行されると，それがサーバに送信される必要があり，それから問合せが分析されて処理木が構築され，処理木が永続的デバイス(通常はディスク)中の膨大なデータから所望のデータを検索しなければならないからである．

### 5.1.2　Stream Processing System(SPS) での実装

上述の方法を実現する手法として SPS の利用が考えられる．SPS はシステムに到着したデータを永続的デバイスに書くことなくオンメモリで処理するシステムである．問合せはシステムに常駐し，新規データが到着すると即座に評価される．RDBMS のようにクライアント側で問合せを管理する必要はない．また，問合せは処理木の状態でサーバに保持される．従って SPS

の問合せ処理は DBMS に比べて高速である.

### 5.2 ChangeFinder

本論文では ChangeFinder[1]と呼ばれる技法を実装する. ChangeFinder とは異常値と変化点を検出する技法である. ChangeFinder は AR モデルに忘却性を導入した SDAR モデルを用いる. そして二段階の学習により異常値と変化点を検出する.

我々は Eigen[14]と C++言語を用いて ChangeFinder を実装した. コード行数は 300 程度である.

SS*において ChangeFinder をユーザ定義関数 cf()として実装した. cf は dstport を引数にとり ChangeFinder のスコアを返す. そして連続的問合せからは次のように cf()を呼出せるようにした.

これを図 2 に示す. 図 2 において RANGE r はストリームの時間幅を表す. 無限長のストリームに対して cf()を適用することはできないため, ストリームを有限長に切り取る必要がある. 切り取り幅が r で示されている. また, t は閾値を表している. ChangeFinder が返戻したスコアが閾値 t を超えた場合に, 異常が発生したとこの問合せは判定する.

```
SELECT *
FROM packet[RANGE r]
WHERE cf(dstport) > t
```

**図 2. ChangeFinder を含む問合せ**

## 6 評価

### 6.1 環境

評価データには MWS データセット[2]を用いた. 評価項目として, 性能, 開発柔軟性, 運用柔軟性を用いた.

マシンスペックは次の通りである. CPU: Intel(R) Xeon(R) CPU E5640 @ 2.67GHz (4 cores) Dual CPU, RAM: 48 GB.

データセットとしては IIJ MITF DATAset 2012 を用いた. これには次の 2 種類のデータが含まれる.

1. ipcomm: 日付, 時刻, 送信元(攻撃者)アドレス, 送信元(攻撃者)ポート, 宛先(honeypot)ID, 宛先(honeypot)ポート, プロトコル(1=ICMP,6=TCP,17=UDP)
2. mal: 日付, 時間, 送信元(攻撃者)アドレス, 送信元(攻撃者)ポート, 宛先(honeypot)ID, 宛先(honeypot)ポート,通信プロトコル,マルウェア取得先 URL,マルウェア MD5hash, マルウェア名,ウイルス対策ソフト名(ClamAV),ファイルタイプ, 最初に取得した日付, 悪用された脆弱性(honeypot が判断できた場合のみ)

具体的には 2011 年 7 月から 2011 年 9 月までのデータを用いた. 各テーブルのサイズは ipcomm: 約 2797MB, mal: 約 2884MB であった.

### 6.2 分析結果(Group-By-Aggregate)

RDBMS(PostgreSQL(version 8.4.13)を用いた分析結果について述べる. 2 種類の分析を行った. 第一の分析はアクセスに関するものである. 2011-07-01 の 00:00:00 から 01:00:00 間のデータについて各 dstIP の各 dstport に対する攻撃回数を図 3 に示す SQL により PostgreSQL から出力させた.

その結果, 639 件のアクセスが観測された. Honeypot は 96 台あり, 平均アクセス回数は 6.6, 最大アクセス回数は 12, 最小アクセス回

```
SELECT  dstip, dstport, COUNT(*)
FROM  (SELECT  *
        FROM  ipcomm
        WHERE date ='2011-07-01' AND
        time BETWEEN
        '00:00:00' AND '01:00:00') AS t
GROUP BY dstport, dstip
ORDER BY dstip,dstport;
```

**図 3. アクセス数を求める問合せ**

数は 1，アクセスされたポートは 81 種だった．アクセス数が多かったポートは，137, 139, 445 であり回数はそれぞれ 290, 4713, 14488 だった．445 番がとりわけ多かったことが観察された．

### 6.3 性能(Group-By-Aggregate)

#### 6.3.1 RDBMS の性能

図 3 に示した問合せに要した時間は 8 秒程度であった．8 秒が長いか短いかはユーザにより判断の分かれるところであろうが，攻撃検知処理に要する時間は短いほど好ましい．また，この処理時間においては 2 点注意されなければならない点がある．

第一の注意点は，図 3 に示される問合せ文ではデータ挿入処理時間が省かれている事である．パケットは DBMS に到着すると，まず永続的デバイスに書き込まれる．この処理が終わってから問合せ処理が実行可能になる．従って実際に攻撃検知処理に RDBMS を用いる場合には，より長い時間が必要になる．

第二の注意点は時間幅である．図 3 では 1 時間にしている解析時間幅を変えた結果を図 4 (青線)に示す．時間幅を変動させても処理時間はさほど大きく変動しない．横軸は時間幅(秒数)，縦軸は処理時間を表す．図 4 に示されているグラフは横軸が対数である点にご注意頂きたい．なおデータ件数は 1 秒，10 秒，100 秒，1000 秒の場合に各々12, 319, 917, 5923 である．

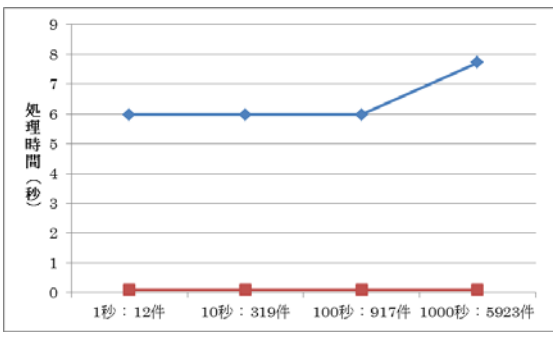


図 4. 問合せ処理性能
青:RDBMS，茶:SPS

図 4 に示されている処理時間が大きく変動しない理由として，index scan ではなく sequential scan が使われていることが考えられる．PostgreSQL において explain コマンドを用いて得た問合せ実行計画の概要を図 5 に示す．図 5 では最下層がデータアクセスであり，上に行くほど問合せ結果に近づく．最下層において sequential scan が使われていることが図 5 には示されている．

一方，index を用いることで処理時間は短縮する可能性がある．しかしながらパケットのように連綿を到着するデータの場合には index 構築コストが余計なオーバヘッドとなる可能性も

```
        Sort [Sort Key: ipcomm.dstport]
->      HashAggregate
->      Seq Scan on ipcomm
```

**図 5. 問合せ実行計画**

ある．それゆえ本研究で想定する環境においては，index 構築により処理時間が短縮されるとは断定できない．

#### 6.3.2 SS*の性能

問合せ処理について，SS*が要した時間を図 4(茶線)に示す．これはクライアントへのタプル集合到着間隔を表している．

茶線は青線より少なくとも 9 倍以上の性能向上を示している．1 秒，10 秒，100 秒，1000 秒の場合の性能向上率は，各々60 倍，60 倍，60 倍，77 倍である．

この結果より，ストリーム処理システムを用いることで RDBMS よりも大幅な高速化が実現できること言える．

### 6.4 開発柔軟性

SS*は C++言語により実装されている関係で，UDF を C++で開発し，ライブラリとして実装することで問合せから呼び出し可能にした．これより，提案方式が開発柔軟性を部分的に有することを確認した．現状の欠点は C++以外の言

語による UDF 実装がない点である. これは今後の課題である.

### 6.5　運用柔軟性に関する議論

SS*を用いた場合, run コマンドにより問合せを起動し, stop コマンドにより問合せを停止できる. また, どのようなマルウェア検知手法がシステムにより管理されているか, 一目瞭然である. 2 節で述べた別プロセス方式, 同一プロセス・一括コンパイル方式ではこれらは実現不能である. 従って, 本論文のアプローチである同一プロセス・動的手法管理方式のみが求められる運用柔軟性を有する.

## 7　まとめ

本論文ではストリームデータ処理システムを用いたマルウェア検知システムを提案した. 提案システムは宣言的言語を用いることで容易にポート毎のアクセス数を分析できることを示し, SQL では記述不能な処理を UDF として実現可能であることを示した. また, 提案システムは SPS を利用することで RDBMS を用いる方式よりも 60 倍程度, 処理時間を高速化することが実験的に示された.

今後の課題は分析手法を増やすこと, 分析結果を効率的に保存できるストレージシステムを開発することである.

## 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] J. Takeuchi and K. Yamanishi, “A Unifying Framework for Detecting Outliers and Change Points from Time Series,” IEEE transactions on Knowledge and Data Engineering, pp.482-492, 2006.

[2] MWS2012 実行委員会, 研究用データセット MWS 2012 Datasets について, http://www.iwsec.org/mws/2012/about.html#datasets

[3] Mahout: http://mahout.apache.org/

[4] Mallet: http://mallet.cs.umass.edu/

[5] Jubatus: http://jubat.us/

[6] MADlib: http://madlib.net/

[7] PostgreSQL: http://www.postgresql.org/

[8] Amol Deshpande and Samuel Madden. 2006. MauveDB: supporting model-based user views in database systems. ACM SIGMOD 2006.

[9] 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.

[10] StreamSpinner: www.streamspinner.org/

[11] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. SPADE: the system s declarative stream processing engine. SIGMOD 2008.

[12] 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.

[13] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. SIGMOD 2006.

[14] Eigen: http://eigen.tuxfamily.org