Vol.2013-IOT-22 No.2
2013/8/1

Indexer Bullet
インターネットの情報リソースを集約するシステムの設計

藤田 昭人[1,a)]　石橋 勇人[2]

概要：Indexer Bullet はインターネット上に存在する独立した複数の情報リソースを集約して利用者の目的に応じたデータオブジェクトを生成し、そのインデックスを管理するシステムである。例えば SNS で流通するメッセージを取得し、その発言時間をキーとするインデックスを生成して保管することができる。このようにインターネットに展開されているさまざまな情報リソースを集約する事により、それを活用したビッグデータ処理を支援することが目的である。

Indexer Bullet
Design of a System for Collecting Internet Information Resouces

Fujita Akito[1,a)]　Ishibasi Hayato[2]

Abstract: Indexer Bullet is a system for creating and managing enormous data that are aggregated from several information resources on the Internet and tailored for each user. For example, messages on SNSs are acquired and indexed by times as their keys. Indexer Bullet supports processing *big data* for social listening.

1. はじめに

インターネットでは膨大なコンテンツが流通しており、それらは様々な目的に活用されている。例えば、サーチエンジンでは全文検索エンジンを使ってコンテンツの検索を行うために、インターネット上に公開されているウェブページを取得し、その情報を元にインデックスを生成している。これはインターネットの情報リソースの典型的な活用事例として従来から良く知られている。

このようなインターネットの情報リソースの活用方法は今日抜本的な変化を迎えている。例えばソーシャルリスニング [1] では情報リソースの取得対象として SNS にフォーカスし、取得データに独自の解析手法を適用して、各種の社会的な変化を理解する手段として利用される。従来の活用方法はサービスとして運用可能なごく少数のプロバイダーが、独自のプロプラエタリな解析アルゴリズムを利用して、社会的に一般性の高いランキング等を生成していたのに対し、今日の新たな試みでは任意の解析ニーズを持つ多数の利用者が、インターネット等で流通する解析アルゴリズムを利用して、利用者のニーズにあった解析結果を得ることを目的としてる。

Indexer Bullet (iBullet) はインターネットの情報リソースの収集・管理を目的としたシステムである。利用者の解析ニーズにマッチした情報リソースを取得し、解析アルゴリズムに適した形に加工、格納するビッグデータ向けのプラットフォームを目指している。

本論文では特に iBullet の設計について述べる。

2. 設計

2.1 開発の背景

iBullet の開発の背景にはソーシャルリスニングへの需要が存在する。ソーシャルリスニングとはマイクロブログや SNS などのソーシャルメディアでの会話に注目し実生活に基づいた意見を汲み取る新しいマーケティング手法で

[1] IIJ イノベーションインスティテュート
IIJ Innovation Institute
[2] 大阪市立大学大学院創造都市研究科
Graduate School for Creative Cities, Osaka City University
[a)] akito-fujita@iij-ii.co.jp

あり、ソーシャルメディアで流通するメッセージ等を収集し様々な分析手法を適用する事により、社会的トレンドを明らかする。

我々が想定するソーシャルリスニングの身近な 1 例として Wikipedia の Page view statistics [2] が上げられる。このページからは Wikimedia プロジェクトに関わるページの 1 時間毎のページビュー情報を入手することができるが、このページビューカウントをランキング集計するとその上位にはこの 1 時間内に多くの人が Wikipedia を使って調べたトピックがランクされる [3]。ランク上位のトピックについてサーチエンジンを使って追跡すると、その時間帯に放送されたテレビ番組や電子版新聞記事で扱ったニュースなど上位にランキングされた理由を比較的容易に調べる事が可能である。

同様にマイクロブログや SNS などのソーシャルメディアから入手できる情報を活用して、各種情報を抽出・集約すれば、対象となるソーシャルメディアに基づく社会的トレンドが把握できると我々は考えている。

iBullet はこのようにインターネットから入手できる情報リソースを対象に各種情報の抽出・集約を行うためのシステムとして構想された。

2.2 iBullet のアイデア

iBullet のアイデアは抽出・集約した情報をインデックスに蓄積することを基本としている。このインデックスは Key-Value スタイルのストレージであり、Key による Value アクセスだけでなく任意の Key パターンを指定した Range Query も可能である。例えば、前述の Wikipedia のランキングはページビューカウントを Key、ページタイトルを Value とするインデックスとして表現される。

一般的なインデックスは様々な目的に利用されるが現時点では次のような利用方法を想定している。

- ウェブページのインデックス
  サーチエンジンなどで用いられる単語とその単語が出現するページ URL のリストからなるインデックスである。転置インデックスを作成すれば任意のウェブページで出現する単語のリストを得る事ができる。
- 巨大でスパースなベクトル
  ウェブコンテンツの解析等では PageRank [4] や TF-IDF [5] といった 2 次元配列を利用する様々なアルゴリズムが提案されているが、インデックスは 2 次元配列を表現する実装としても活用できる。巨大かつスパースな 2 次元配列であることが多く、その実装として Key-Value スタイルのストレージを用いるメリットがある。

iBullet はインターネット上の様々な情報リソースを取り込み、生成した様々なインデックスを管理するシステムである。

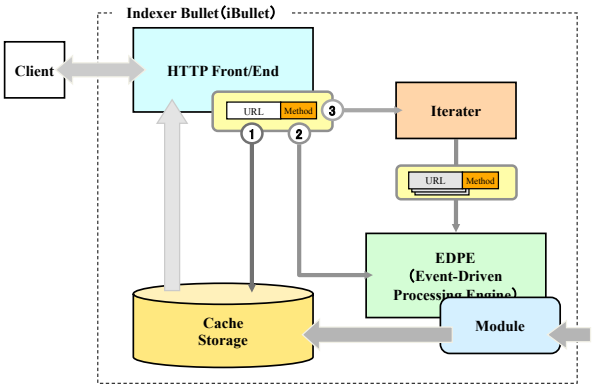

図 1 iBullet の基本構造

2.3 設計上の選択

iBullet は HTTP プロトコルによりアクセス可能なサーバソフトウェアとして実現する。これはウェブアプリケーションとの親和性を重視した結果で、ウェブアプリケーションでは一般的なマッシュアップの等の手法を用いて容易に iBullet が管理するインデックスにアクセスすることが可能となる。

iBullet のアイデアを実現する情報リソースの取得、データの抽出・集約、インデックスの生成の iBullet 独自の機能はサーバソフトウェアに対する拡張モジュールとして実現する。これは対象となる情報リソースによって、その取得方法やデータフォーマットが異なるためである。拡張モジュールとして仕様化することにより、対象情報リソース毎の機能追加が容易となる。

2.4 iBullet の基本構造

iBullet はイベント駆動型アーキテクチュア [6] として設計している。

図 1 に示すとおり次の 4 つの内部コンポーネントから iBullet は構成される。

- HTTP Front End:
  HTTP プロトコルによるアクセスの受付
- CacheStorage:
  キャッシュ・データを格納するストレージ
- Event-Driven Processing Engine(EDPE):
  キャッシュ・データの取得およびデータ抽出・集約を実行するエンジン

iBullet は原則としてウェブコンテンツ・キャッシュと同様の挙動を行う。HTTP リクエストは HTTP Front End

によって受け付けられ、まず CacheStorage に問い合わせ対応するキャッシュデータの存在を確認する。キャッシュデータが存在しない場合は EDPE を呼び出す。EDPE は HTTP リクエストを解釈して対応する拡張モジュールを選択、実行し、データの取得やデータの抽出・集約を行った後、その結果を CacheStorage に格納する。

2.5 Method 識別子

iBullet がユニークな点はキャッシュデータの識別子として URL だけではなく Method と呼ばれる補助識別子を使用する事である。Method は EDPE が起動するべき拡張モジュールを識別するために使用される文字列であるが、Cache Storage へのアクセスの際には両者を結合した文字列を Key として使用する。iBullet は 1 つの URL に対して Method の異なる複数のデータをキャッシュできる。

例えば Method “html” で HTML コンテンツを取得する拡張モジュールを、Method “outlink” で HTML データ内のアンカーで定義されている URL を抽出する拡張モジュールを対応づける事ができる。クライアントはリクエスト時に指定する Method を選択する事により HTML コンテンツ全体やアウトリンクを参照する事ができる。iBullet はこの機構を使用してインターネット上の情報リソースの取得、データ抽出・集約を実現する。

2.6 Cache Storage

Cache Storage は PUT/GET/DELETE による Key-Value スタイルのインターフェースを備える。Key は URL と Method を結合した文字列を使用する。Value は任意のバイトストリームと規定されるが、キャッシュデータを格納するのは拡張モジュールであるため Method 毎に実際に格納されるデータ形式が決定される。

Cache Storage の設計で課題となるのはデータの格納先である。一般的なウェブコンテンツ・キャッシュではインメモリへの格納されることが多いが、iBullet が取り扱うインターネット経由で入手できるデータのサイズは仮定できないこと、また拡張モジュール内において子プロセスが起動することも想定していること、更に Key-Value スタイルでのアクセスを行うインデックスも取り扱う必要がある事から、ファイルシステムなどの永続ストレージへの格納を前提とし、可能な場合は Cache Storage 内部でメモリにキャッシュする。

利用する永続ストレージは現時点でローカルファイル、および KyotoCabinet [7] などのデータベースライブラリの利用を想定している。

2.7 Event-Driven Processing Engine(EDPE)

EDPE は URL と Method で指定されたコンテンツを取得する拡張モジュールの実行エンジンである。

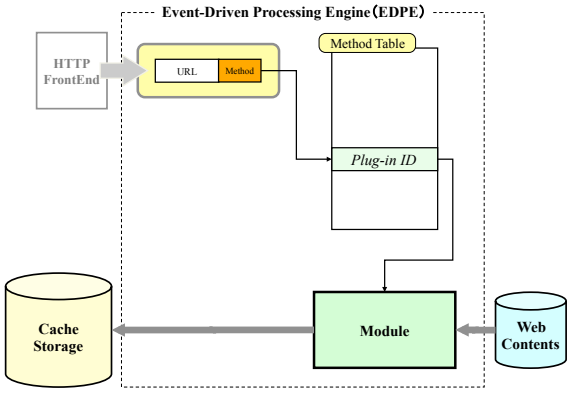

図 2 Event-Driven Processing Engine(EDPE)

Method は拡張モジュールの識別に利用される。EDPE は Method と拡張モジュールを対応づける Method テーブルを内蔵しており、リクエストを受け付けると指定された URL、Method に対応する拡張モジュールを探索し、これを起動する（図 2）。

拡張モジュールは次の Fetch と Carry の 2 つ API で定義される。

```
int fetch(char *url, char *method,
          rCache *rc, mCache *mc);
int carry(char *url, char *method,
          rCache *rc, mCache *mc);
```

Fetch は指定された url/method のコンテンツを取得する API で、主に情報リソースの取得やデータ抽出の機能を実装するために用いる。一方、Carry は url/method のコンテンツの取得直後に起動される API で、url/method のコンテンツの内容が変更された際に連動する更新を定義するために使用する。通常は url/method のコンテンツの内容に依存するインデックスの更新に用いる。

```
class rCache {
public:
        int get(string url, string method,
                char **val);
};

class mCache: public rCache {
public:
        int put(char *url, char *method,
                char *val, int nval);

        int delete(string url, string method);
};
```

Fetch/Carry の引数として定義されているクラス rCache/mCache は Cache Storage を抽象化したクラスで、クラス rCache を介して iBullet 内に存在する全てのキャッシュデータに参照する事ができる。クラス mCache

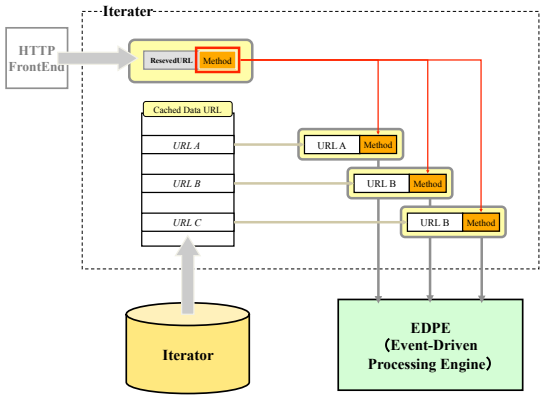

図 3 Iterator

は Cache Storage にデータを登録する際に使用する rCache の派生クラスである。mCache は永続ストレージへの対応付けのないメモリベースのインスタンスを生成する。拡張モジュールで mCache に格納されたデータは、その後 iBullet が任意のタイミングで Cache Storage に反映する。

2.8 インデックス

クラス index は iBullet が管理するインデックスを表現するクラスで Fetch および Carry の内部で任意に使用する事ができる。

```
class index {
public:
        index(string name);
        int get(char *key, char **val);
        int put(char *key, char *val, int nval);
        int delete(char *key);
};
```

クライアントは Fetch/Carry で生成・更新したインデックスを iBullet にアクセスする場合には “iter:://<index name>” のように URL のプロトコル名に “iter” を、パス名にはインデックス名を指定する。

この URL を使って iBullet からインデックスの内容を参照する事ができるが、さらに Method も指定すると iBullet は Iterater の機構を起動する。Iterator は URL の異なる複数のキャッシュデータに対し同一の拡張モジュールを起動するための EDPE の補助機構である。Iterator の展開メカニズムを図 3 に示す。

3. 実装の現状

iBullet は現在 CentOS 5.9 で稼働するサーバソフトウェアとして開発を進めている。実装では libevent [8] を利用し、イベント駆動型アーキテクチュアを実現している。

目下の開発目標は弊社が公開する Wikipedia ランキングシステム [3] でのバックエンド処理に iBullet を活用する事である（図 4）。

図 4 Wikipedia Ranking

現時点では Wikipedia Pageview データの取得、ランキング集計（1 時間、２４時間）の対応を完了しており、最終的にバックエンド処理全てを iBullet に置き換える予定である。

4. 類似研究

グーグルが開発した対話型検索システムである Dremel [9] はビッグデータ解析のプロセスを支援する開発目的と MapReduce [10] などのビッグデータ処理の分散実行フレームワークとの相互補完的な性質を持つことでは iBullet と共通している。しかしながら Dremel は対話性に優れた検索機能を提供するシステムであり、その検索対象として nested data model に基づくデータを仮定しており、独自のクエリー言語と実行エンジンを定義しているところが iBullet とは異なる。

iBullet は検索処理に必要なインデックスの管理をサポートするが、インデックスの生成機能およびそれを活用した検索機能は拡張モジュールとして実装すること想定している。また対象データの構造や表現に仮定を設けていない。これは iBullet の設計上の選択であり、インターネットから入手可能な様々な構造や表現によるビッグデータ解析のニーズを持つ多数のユーザーに広く活用されることを想定しているからである。例えば、ビッグデータの統計解析を考えるユーザーには R 言語 [11] の利用を望むであろう。このような多様なニーズに対応するため、iBullet では対象データの特性に依存する仕様を iBullet の外部の拡張モジュールにて規定することとした。

5. おわりに

iBullet はインターネットの情報リソースの収集・管理を目的としたシステムで、情報リソースから抽出・集約したデータをインデックスに蓄積することにより、利用者のビッグデータ解析を支援する。

現状はシステムの初期のプロトタイプが稼働しており Wikiepedia 日本語版（710 万ページ）が収容できることを

確認している。

今後、さらに実際的なビッグデータ解析に適応するためには拡張モジュールの処理能力や収容データ量の向上が必要になる。そのため iBullet の分散並列化を検討したいと考えている。

参考文献

[1] Rappaport, S. D.: *Listen First!: Turning Social Media Conversations Into Business Advantage*, Wiley (2011).
[2] Wikipedia: *Page view statistics for Wikimedia projects*, `http://dumps.wikimedia.org/other/pagecounts-raw/` (2013).
[3] Project, I.-I. G.: *Wikipedia Pageview Ranking*, `http://www.gryfon.iij-ii.co.jp/ranking/` (2013).
[4] Page, L., Brin, S., Motwani, R. and Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web., Technical Report 1999-66, Stanford InfoLab (1999).
[5] Manning, C. D., Raghavan, P. and Schütze, H.: *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, UK (2008).
[6] Wikipedia: *Event-driven architecture*, `http://en.wikipedia.org/wiki/Event-driven_architecture` (2013).
[7] Labs, F.: *Kyoto Cabinet: a straightforward implementation of DBM*, `http://fallabs.com/kyotocabinet/index.ja.html` (2013).
[8] Mathewson, N. and Provos, N.: *libevent – an event notification library*, `http://libevent.org/` (2013).
[9] Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M. and Vassilakis, T.: Dremel: Interactive Analysis of Web-Scale Datasets, *Proc. of the 36th Int'l Conf on Very Large Data Bases*, pp. 330–339 (online), available from ⟨http://www.vldb2010.org/accept.htm⟩ (2010).
[10] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *OSDI*, USENIX Association, pp. 137–150 (2004).
[11] R Development Core Team: *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria (2005).