Groongaの可変型Ngramトークナイザーについて

Naoya Murakami

Groonga "Tokenizer" Talks
2015/3/20

自己紹介

- Naoya Murakami
- 1月から知財のWeb系のスタートアップに転職
  - 東京に引っ越しました!
  - その前は数年ほど関西の特許事務所勤務
  - 仕事でプログラミングするのは初めて

twitter: @naoa_y
blog: http://blog.createfield.com

Groongaのおかげで転職ができました

- グローバルな知的財産権の流通市場プラットフォーム　https://www.ipnexus.com
- IPの専門家と顧客とを繋げるサービス
- 個人発明家やスタートアップでも知財を有効活用できるように
- まだ事業を立ち上げている段階

最近やっていること

- Railsとかその辺が多い
  仕事ではまだGroongaを使っていない
  - 圧倒的に人材不足
  - 事業内容や開発に興味がある方は気軽に声をかけてください
- 個人的には検索技術やデータマイニングとかに興味がある

個人でやっていたこと

- 特許の全文検索サービス Patent Field
  http://patentfield.com
- 特許検索では内容を絞り込むための分類体系がたくさんある IPC, FI, FTERM, UC, CPC ...
- 特許文献では固有名詞はほとんど使われずあまり特徴的なワードを取ることはできない

  パソコン⇒情報処理装置, バイオリン⇒弦楽器 etc
- 検索してすぐにそれっぽいものが見つかるよりも検索漏れ防止のが重要
  ⇒Ngramのトークナイザーが使いたい

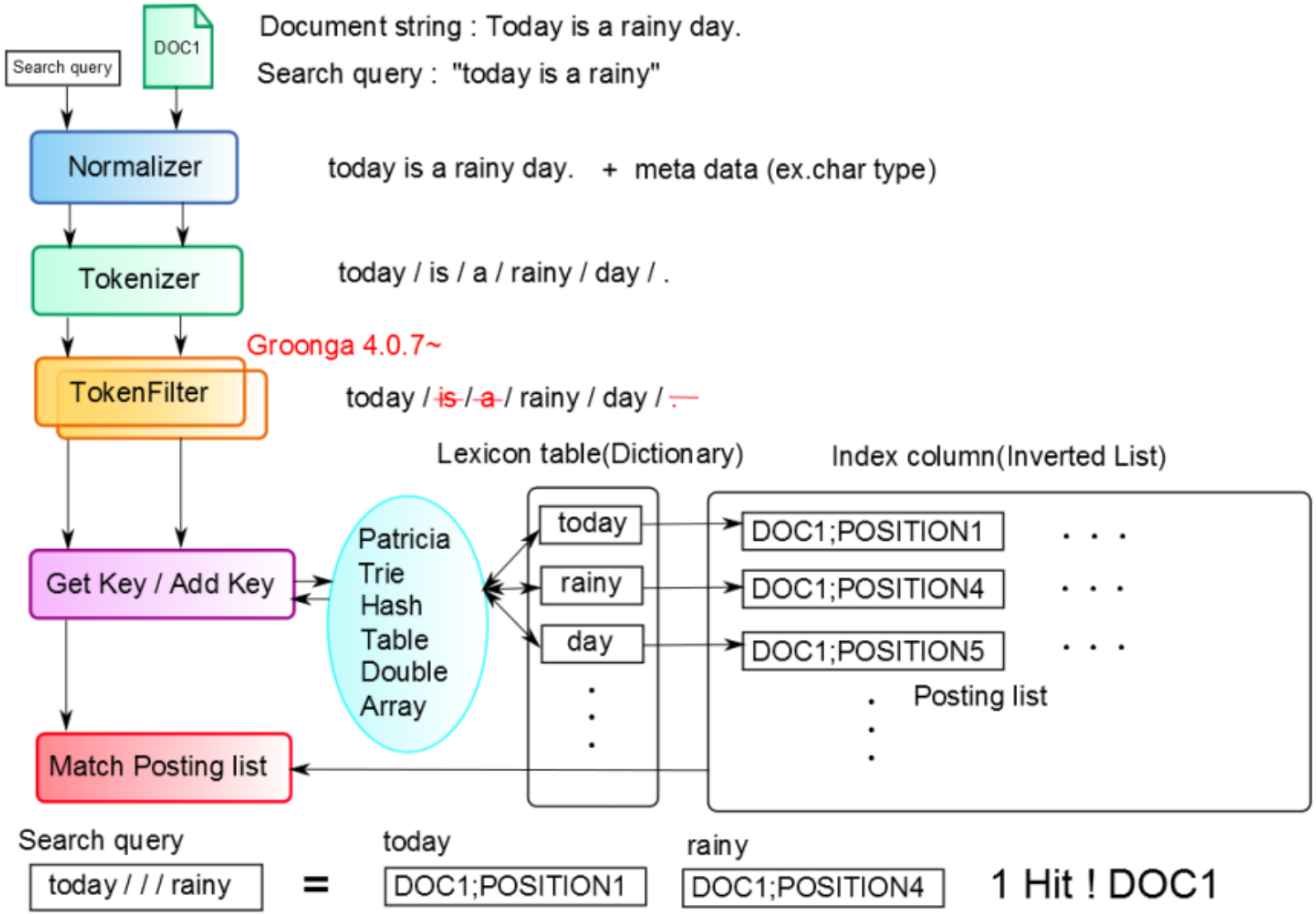
個人でやっていたこと

- ✓ 一番大きな**日本語**のデータベースで
  - ✓ 数百GiB超(カラム非圧縮時)
  - ✓ 一千万レコード超
- ✓ Ngramトークナイザーは検索漏れに強いが一般的に**速度DOWN**　サイズUP
  ⇒Ngramのトークナイザーを高速化

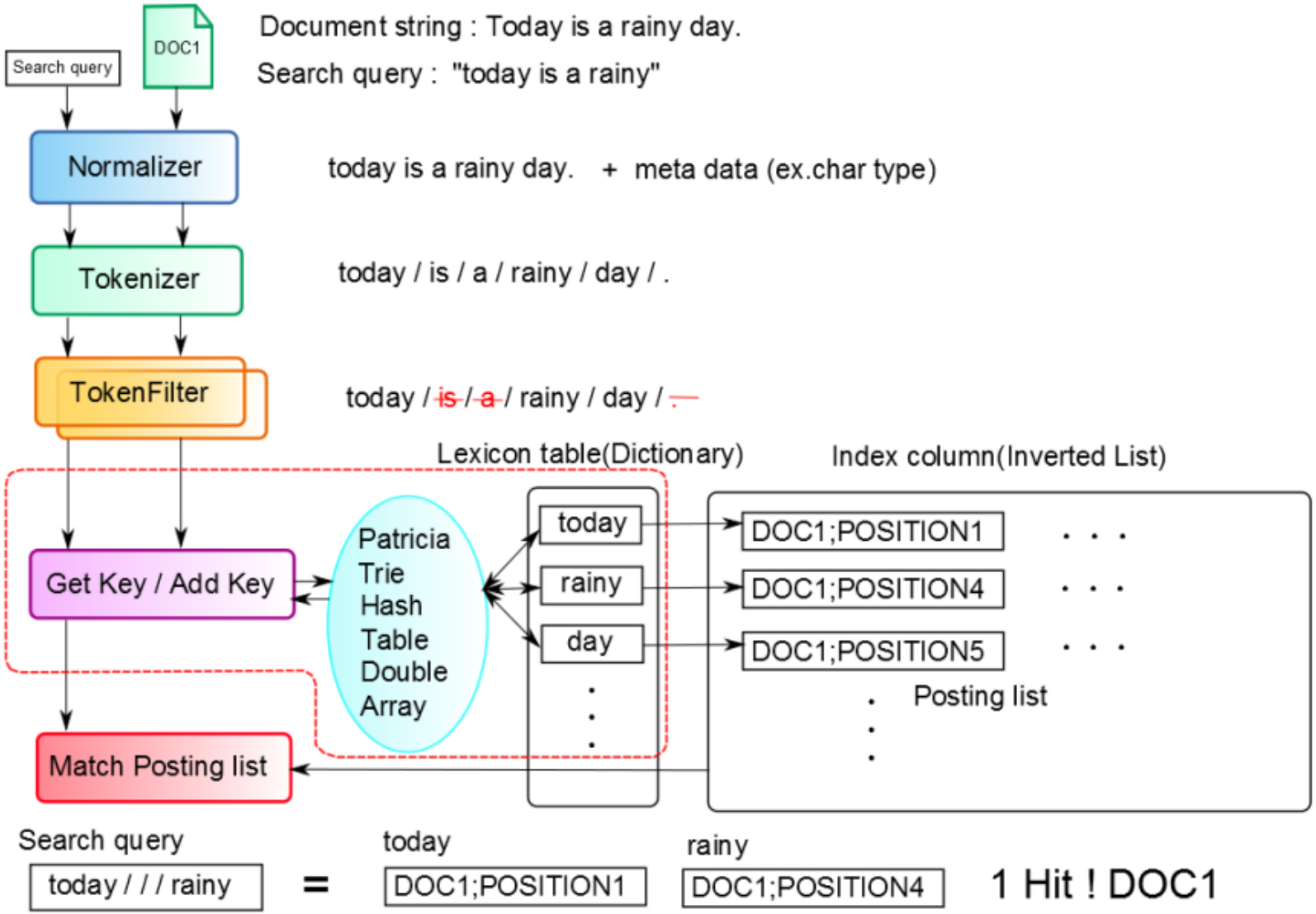
改良型Ngramトークナイザー

- YaNgram - Yet another Ngram tokenizer plugin
  https://github.com/naoa/groonga-tokenizer-yangram
- 検索時のオーバラップスキップ
- 静的な頻度情報に応じた可変Ngram (Vgram)
- 既知フレーズのグループ化

Groongaの全文検索の流れ



キー探索


Search query
DOC1
Document string : Today is a rainy day.
Search query : "today is a rainy"
Normalizer
today is a rainy day. + meta data (ex.char type)
Tokenizer
today / is / a / rainy / day / .
TokenFilter
today / is / a / rainy / day / .
Lexicon table(Dictionary)
Index column(Inverted List)
Get Key / Add Key
Patricia Trie Hash Table Double Array
today
rainy
day
DOC1;POSITION1
DOC1;POSITION4
DOC1;POSITION5
Posting list
Match Posting list
Search query
today / / / rainy
=
today
DOC1;POSITION1
rainy
DOC1;POSITION4
1 Hit ! DOC1

キー探索

- ハッシュ表やパトリシアトライなどを使って語彙表のキーとして登録されたトークンを探す
- いわゆる辞書引き・KVS
- Groongaではインデックス≠キー
- キー探索は非常に速くμsecオーダー

キーの種類数が増える要因

- 文字の種類数が多いこと
  組み合わせが増えるためキーの種類数は多くなる
  - 日本語の文字の種類は多い
    ひらがなカタカナ50種 漢字いっぱい
  - 英語の文字の種類は非常に少ない
    アルファベット26種
- 文字数が多いこと
  組み合わせが増えるためキーの種類数は多くなる
  - NgramはNが大きいほどキーの種類数が多い

キーの種類数増によるキー探索速度への影響

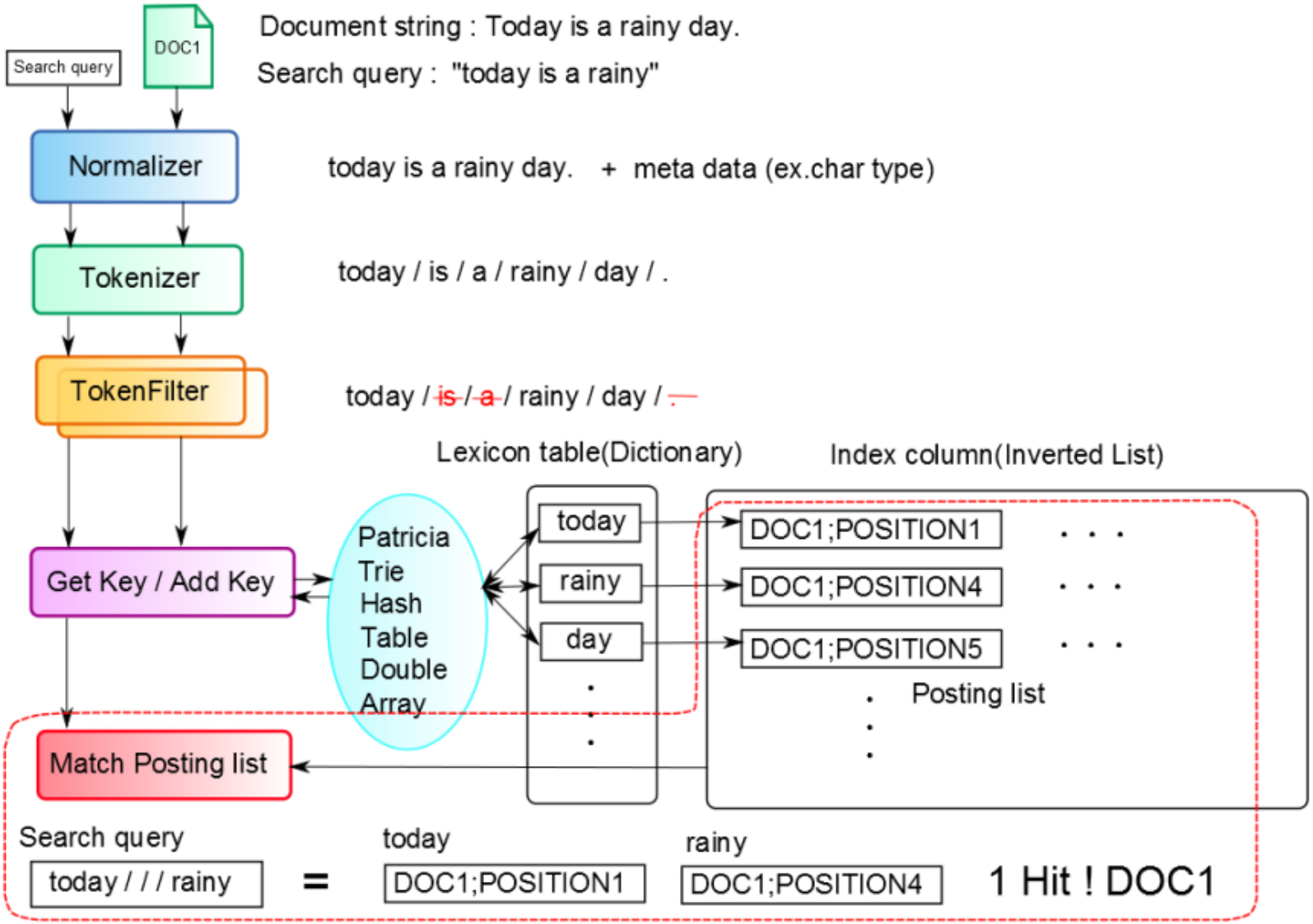
パトリシアトライ(ADD後⇒GET)

キー種類数	キー1件取得秒数
1万	21 μsec
1千万	37 μsec

キーの種類数増によるキー探索速度への影響

- ✓ キー探索はキーの種類が増えても線形的に時間が増えない
  例：ハッシュ表O(1)、パトリシアトライO(k)
- ✓ キー種類増による検索速度への影響は非常に軽微

ポスティング探索


Search query
DOC1
Document string : Today is a rainy day.
Search query : "today is a rainy"
Normalizer
today is a rainy day. + meta data (ex.char type)
Tokenizer
today / is / a / rainy / day / .
TokenFilter
today / is / a / rainy / day / .
Lexicon table(Dictionary)
Index column(Inverted List)
Get Key / Add Key
Patricia Trie Hash Table Double Array
today
rainy
day
DOC1;POSITION1
DOC1;POSITION4
DOC1;POSITION5
Posting list
Match Posting list
Search query
today / / / rainy
=
today
DOC1;POSITION1
rainy
DOC1;POSITION4
1 Hit ! DOC1

ポスティング探索

- キー探索によって取得したポスティングリスト中のトークンの出現位置と検索クエリのトークンの出現位置の並びが一致するかどうかを比較
- トークンの出現頻度が増えるとポスティングリストが長くなる
  - 一番時間がかかるところ
    シーケンシャルサーチを除く

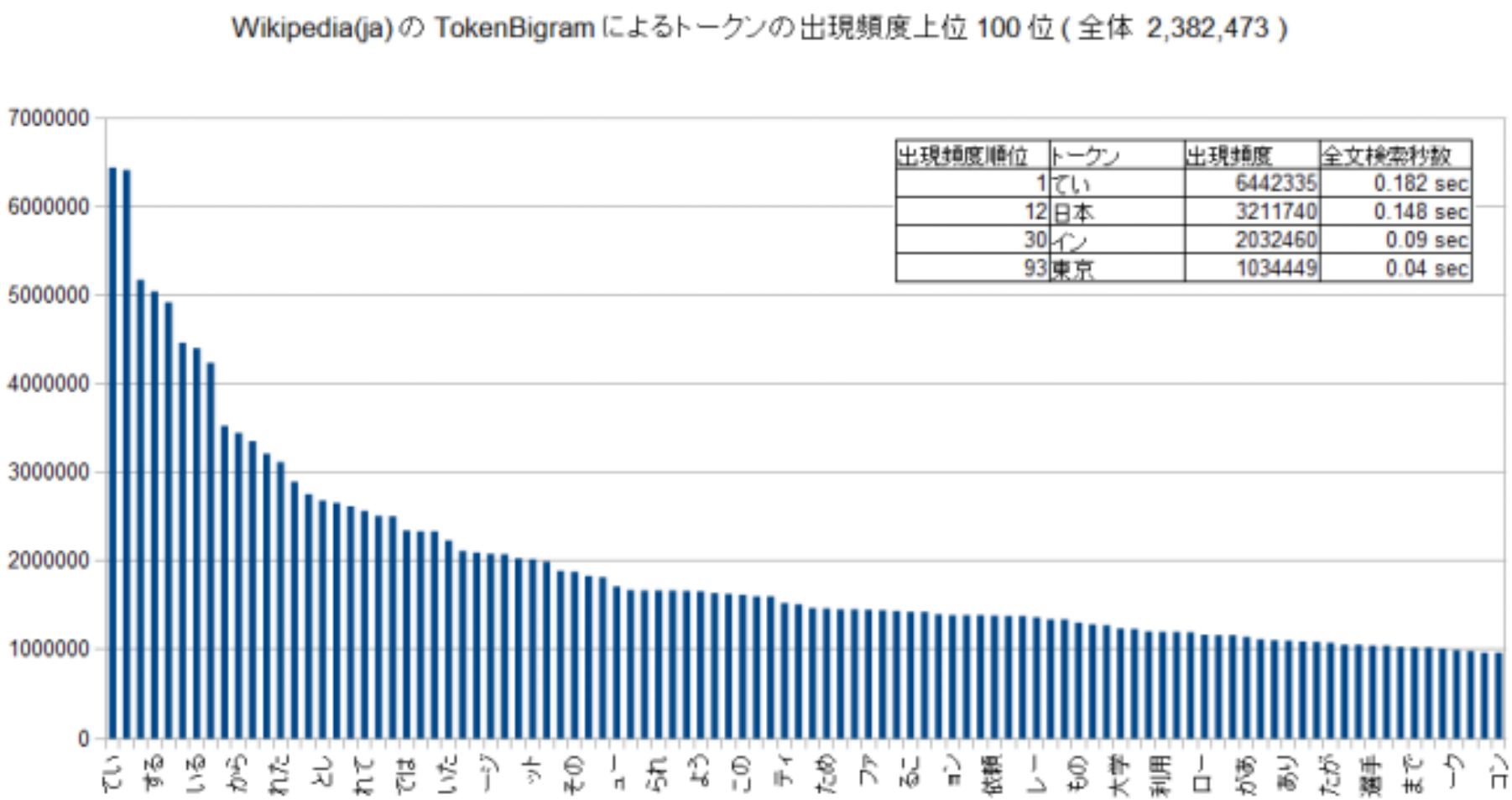
トークンの出現頻度が増える要因

- キーの種類数が少ないほどトークン1個あたりの出現頻度は大きくなる
  - Unigramや英語の文章をBigramでトークナイズするとトークン1個あたりの出現頻度は非常に大きい
- 「**の、が、は**」などの助詞は１文書中にたくさん出現する

頻出トークンのポスティング探索速度への影響

- トークンの出現頻度の増加に応じて検索時間が伸びる
  できるだけ飛ばせるところはとばしているが、規模が大きくなってくるとほぼ線形に検索速度に影響してくる
  - 大抵の場合、ポスティングリストの探索でCPUがボトルネック
- 文書数/サイズに応じて頻出トークンのポスティングリストが長くなる
  - クエリ間の検索速度の差が広がる

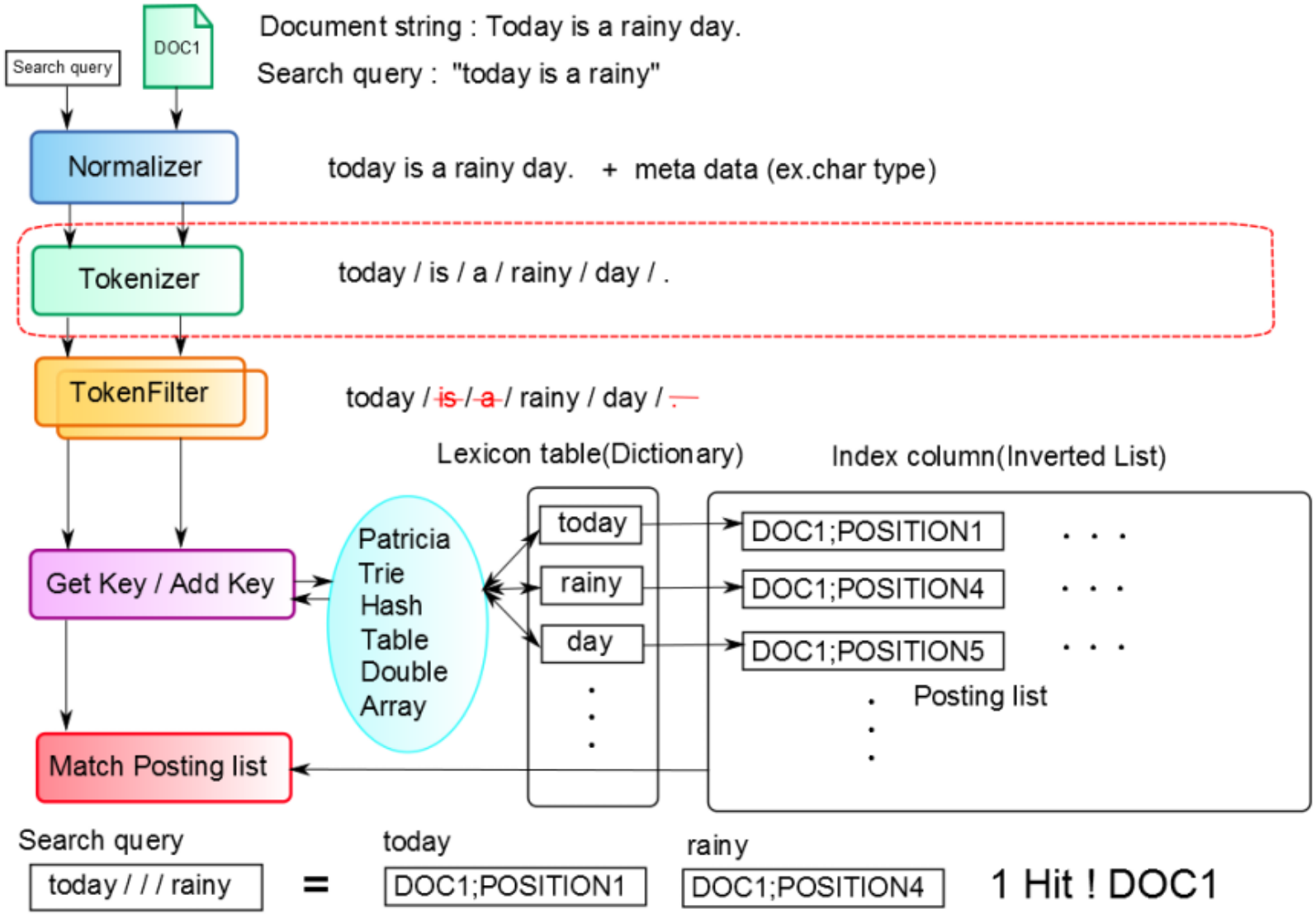
Bigramトークナイザーの検索速度例



検索速度を高速に保つために重要なこと

- キーの種類数よりもポスティングリストが長くなりすぎないようにする
- CPUクロック数を上げる
- なお、最新のGroongaでは頻出トークンとレアトークンの組み合わせ時に一部の探索をスキップする高速化処理が組み込まれているが、このスライドの実験結果には反映されていない
  http://sourceforge.jp/projects/groonga/lists/archive/dev/2015-February/003097.html

Groongaのトークナイザー



Ngramトークナイザー

- 所定の長さのユニットサイズで1文字ずつずらす
  1:Unigram 2:Bigram 3:Trigram
- 「今日は雨だ」⇒「今日/日は/は雨/雨だ/**だ**」
- Groongaでは1文字でも検索できるように末尾1文字も含まれる
  検索時は末尾1文字は含まれない

Ngramトークナイザー

- デフォルトではアルファベット、記号、数字はグループ化
  - 検索ノイズ低減、検索速度向上のため
- アルファベット、記号、数字もNgramにしたいのであれば、TokenBigramSplit系を使う

Ngramトークナイザーのメリット

- ✓ 原則 漏れのない検索が可能
- ✓ 辞書のメンテナンスコスト不要

Ngramトークナイザーのデメリット

- 1文字ずつずらすためトークンの総数が多くなり転置索引のサイズが大きくなる
  - 転置索引のサイズはほぼトークンの総数によって決まる
- 検索ノイズが含まれることがある
  - 例：東京都に対して京都でヒットする
- トークンの比較回数が多く形態素解析よりも検索速度が遅くなることがある

形態素解析トークナイザー

- 形態素解析器を使って文脈に応じて単語単位に分かち書き TokenMecab
- 分割ルールは学習モデルと辞書による Unidicであれば短く分かち書き
- 例：「今日は雨だ」⇒「今日/は/雨/だ」

形態素解析トークナイザーのメリット

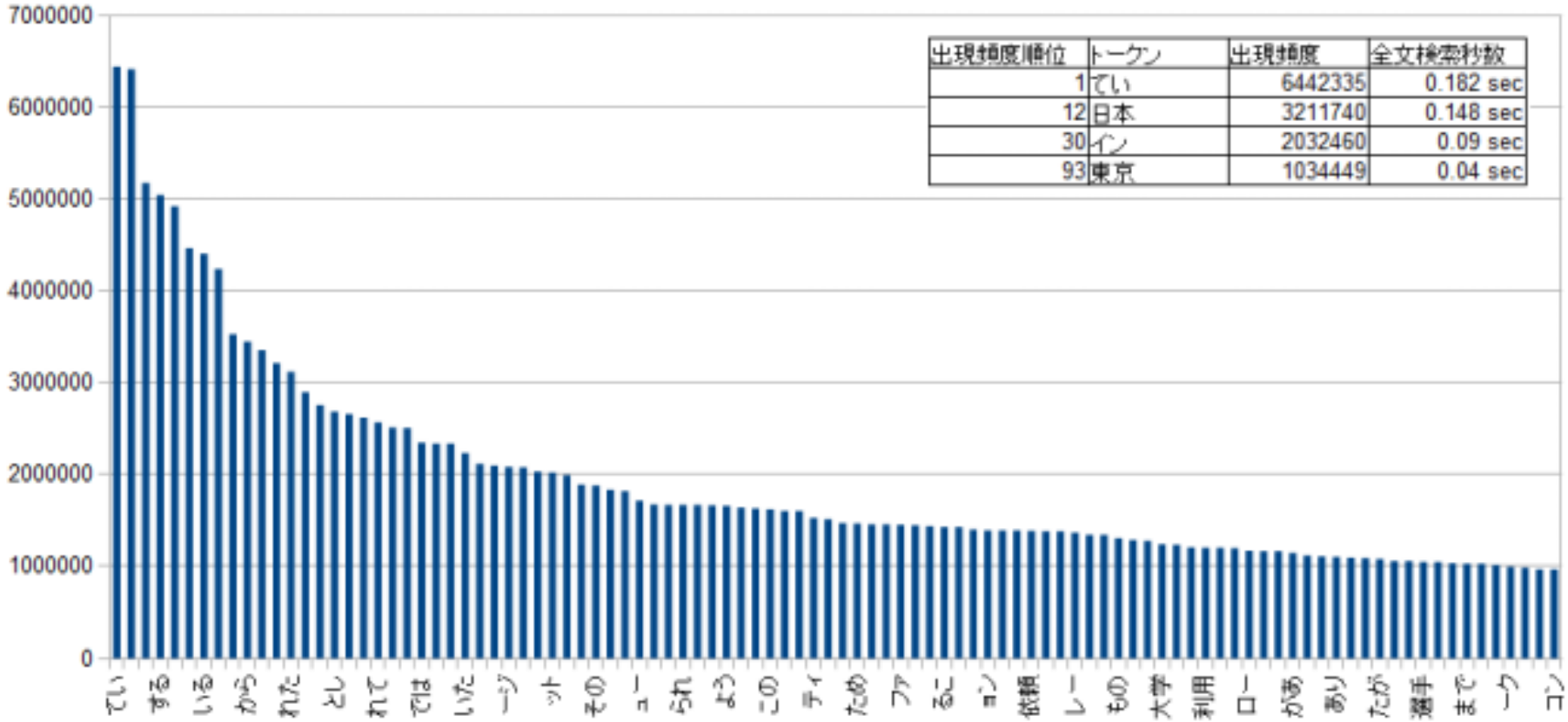
- ✓ 検索ノイズの低減
  例：東京都に対して京都がヒットしない
- ✓ 単語ごとにずらせるため転置索引のサイズがコンパクト
  形態素解析の場合「転置索引」⇒「転置索引」1つ
  Bigramの場合「転置索引」⇒「転置/置索/索引/引」4つ
- ✓ おおむね検索が高速
  基本的に比較するトークンの数がNgramよりも少ない

形態素解析トークナイザーのデメリット

- ✓ 検索漏れ有
- ✓ 辞書の追加やモデルの再学習などメンテナンスコスト大
- ✓ 検索クエリと文章中では文脈が異なり分割ルールが異なることがまれによくある
  チューニングが大変。検索クエリでは左の文脈がない
- ✓ クエリによって検索速度にムラがでやすい
  形態素解析では1文字トークンもあり出現頻度の最大値がNgramよりもかなり大きくなりやすい
  ⇒ストップワードが重要

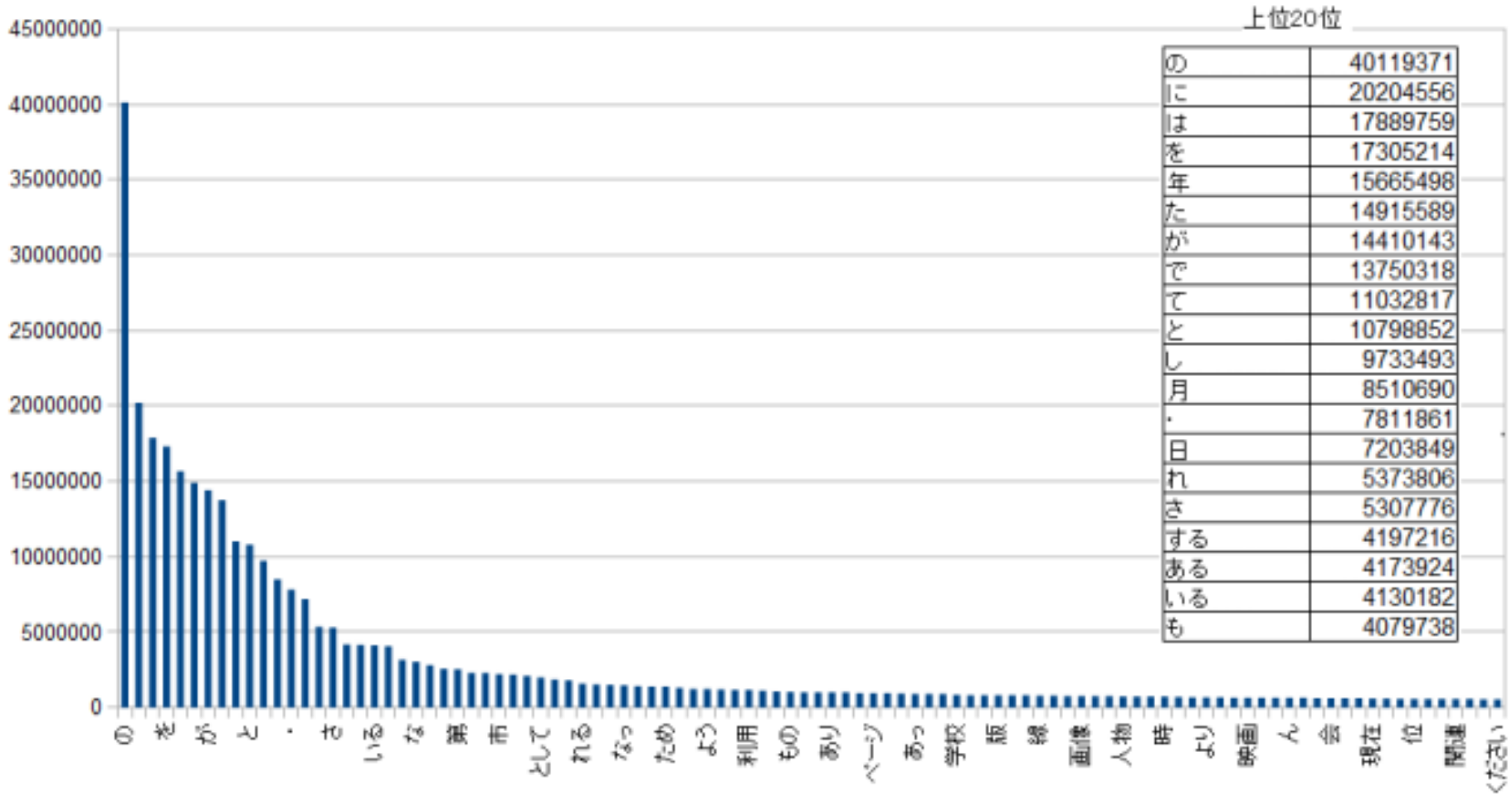
Bigramトークナイザーの出現頻度例

Wikipedia(ja) の TokenBigram によるトークンの出現頻度上位 100 位 (全体 2,382,473)



出現頻度順位	トークン	出現頻度	全文検索秒数
1	ている	6442335	0.182 sec
12	日本	3211740	0.148 sec
30	イン	2032460	0.09 sec
93	東京	1034449	0.04 sec

形態素解析トークナイザーの出現頻度例



Ngramトークナイザーの高速化1
検索時のオーバーラップスキップ

- Ngramの文書追加時は1文字ずらしですべてのポジションのキーを登録
- 検索時も1文字ずらしでキー探索、ポスティングリスト比較
- 例：「今日は雨だ」⇒「今日/日は/は雨/雨だ」
- トークンの比較回数が多い
  - 検索速度が**劣化**

Ngramトークナイザーの高速化1
検索時のオーバーラップスキップ

- Ngramの文書追加時は1文字ずらしですべてのポジションのキーを登録
- 検索時も1文字ずらしでキー探索、ポスティングリスト比較
- 例：「今日は雨だ」⇒「今日/日は/は雨/雨だ」
- トークンの比較回数が多い
  - 検索速度が**劣化**

Ngramトークナイザーの高速化1
検索時のオーバーラップスキップ

- ✓ 検索時は開始位置が決まっているので1文字ずらしする必要はない
  - ✓ 原則:オーバラップ部分をスキップ

従来:「今日は雨」⇒「今日/日は/は雨」
改良:「今日は雨」⇒「今日/　　/は雨」

- ✓ トークンの比較回数が3回から2回に減る
  - ✓ **高速化**

Ngramトークナイザーの高速化1 検索時のオーバーラップスキップ

- 例外:末尾や字種境界で短くなるところは1つ手前の長い方を採用する
  - 短いやつはポスティングリストが長く検索が遅い

「今日は雨だ」⇒「今日/　　/は雨/　　/だ」 ×
「今日は雨だ」⇒「今日/　　/は雨/雨だ」 ○

Ngramトークナイザーの高速化1
検索時のオーバーラップスキップ

- これを追加実装したのが以下のトークナイザー
  - TokenYaBigram
  - TokenYaTrigram

TokenBigram/TokenYaBigramの速度比較

Wikipedia(ja)で1000回検索

トークナイザー	検索秒数平均
TokenBigram	0.0508 sec
TokenYaBigram	**0.0325 sec**

※5文字以上の日本語のみのカテゴリ

TokenTrigram/TokenYaTrigramの速度比較

Wikipedia(ja)で1000回検索

トークナイザー	検索秒数平均
TokenTrigram	0.0146 sec
TokenYaTrigram	**0.0063 sec**

TokenYaBigram/TokenYaTrigramの速度比較

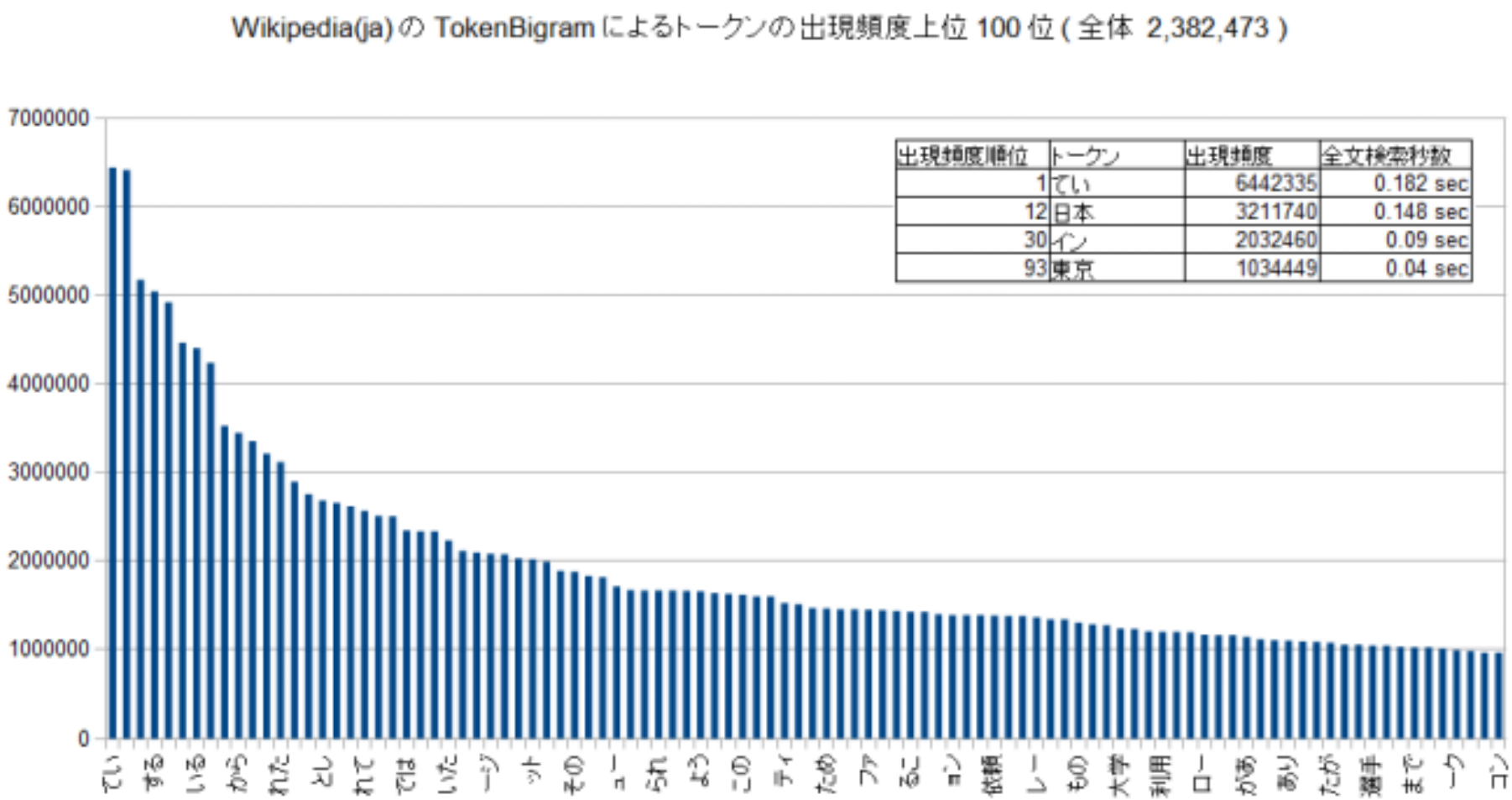
- YaBigramはBigramに比べ1.5倍ほど速い
- YaTrigramはTrigramに比べ2倍ほど速い
  - オーバーラップ部分を飛ばせる量が増える

Bigram「今日は雨だな」⇒「今日/　/は雨/　/だな」3
Trigram「今日は雨だな」⇒「今日は/　　/　　/雨だな」2

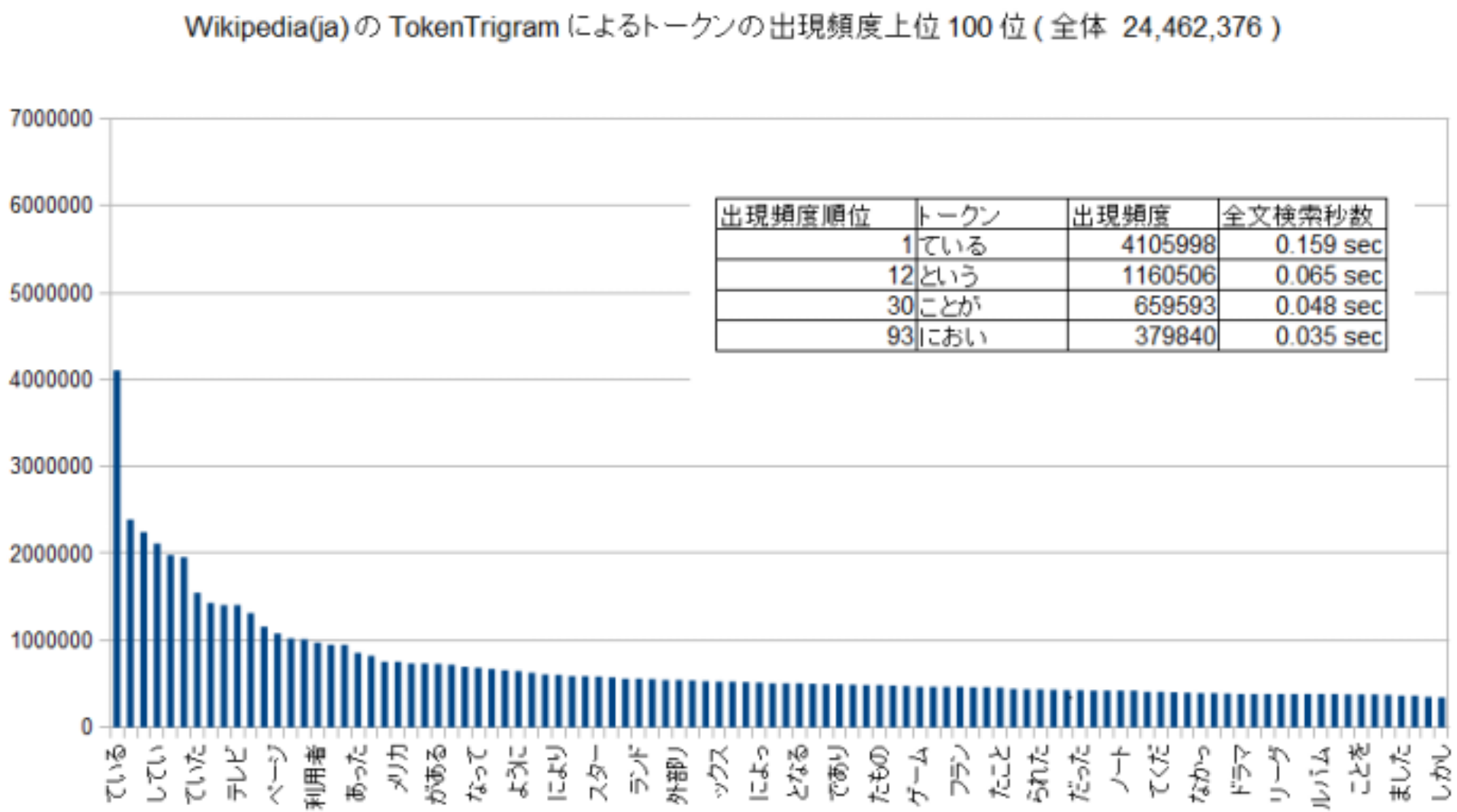
Ngramトークナイザーの高速化2
静的な頻度情報に応じた可変Ngram
(Vgram)

- ✓ 原則、Nのサイズが大きくほどトークンの種類が増えてトークン１つあたりのポスティングリストは短くなる
- ✓ BigramをTrigramにすれば3文字以上の検索で**速くなる**
  - ✓ 2文字以下で検索する場合は速くはならない

Bigramトークナイザーの検索速度例



Trigramトークナイザーの検索速度例



出現頻度順位	トークン	出現頻度	全文検索秒数
1	ている	4105998	0.159 sec
12	という	1160506	0.065 sec
30	ことが	659593	0.048 sec
93	におい	379840	0.035 sec

TokenTrigramのデメリット

- TokenTrigramはTokenBigramに比べキー数とキーサイズが増大
- メモリ使用量が増大
  - キーサイズは小さいほうが望ましい

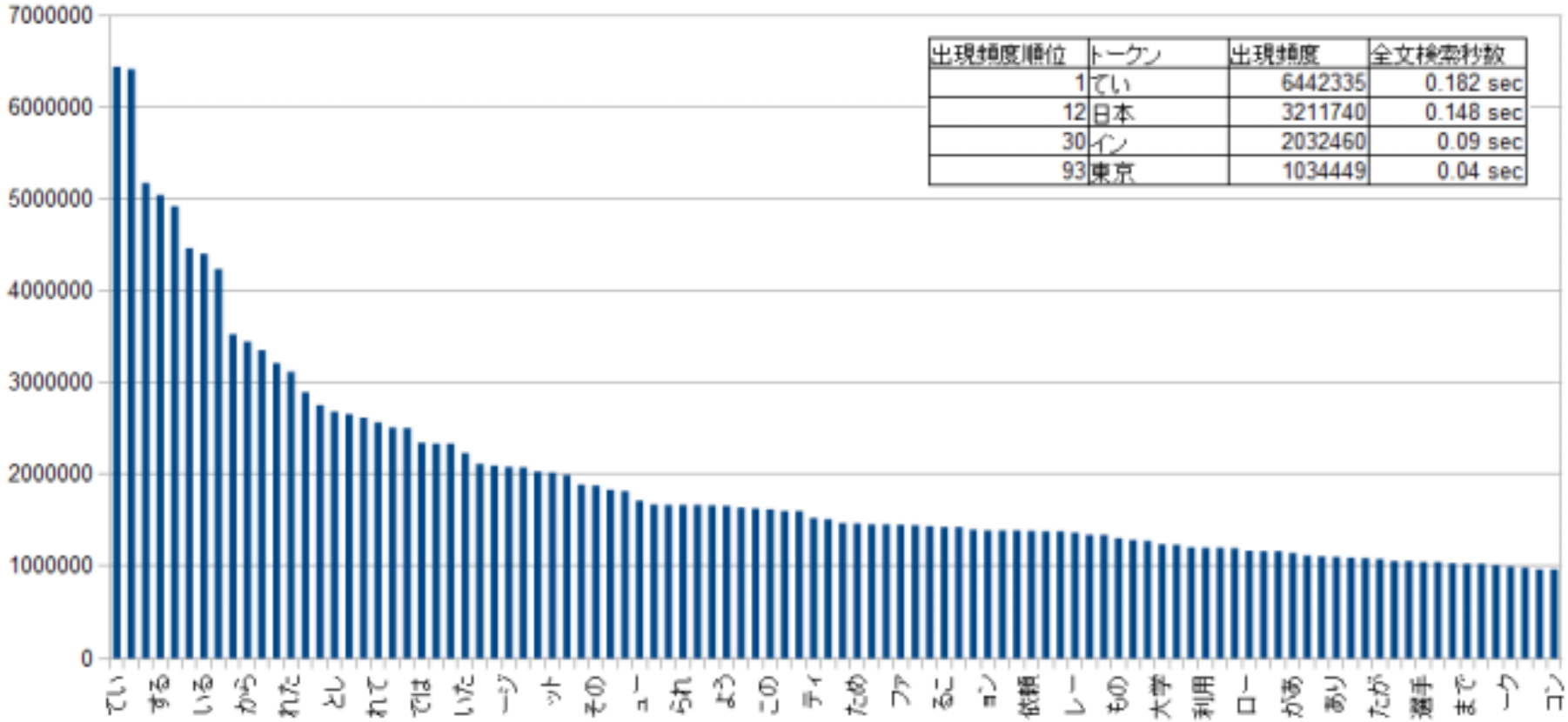
TokenBigram/TokenTrigramのキー

Wikipedia(ja)

トークナイザー	キーの数	キーサイズ
TokenBigram	5767474	136.047MiB
TokenTrigram	28691883	684.047MiB

Bigramトークナイザーの出現頻度例

Wikipedia(ja) の TokenBigram によるトークンの出現頻度上位 100 位 (全体 2,382,473)



出現頻度順位	トークン	出現頻度	全文検索秒数
1	ている	6442335	0.182 sec
12	日本	3211740	0.148 sec
30	イン	2032460	0.09 sec
93	東京	1034449	0.04 sec

Ngramトークナイザーの高速化2 静的な頻度情報に応じた可変Ngram (Vgram)

- ✓ トークンの出現頻度はNgramといえ大きく偏っている
- ✓ 大半のトークンは出現頻度が高くなく十分な検索速度が得られている
- ✓ Bigramでの**出現頻度が高いトークンだけ**をTrigramにすれば良い

Ngramトークナイザーの高速化2
静的な頻度情報に応じた可変Ngram
(Vgram)

- これを追加実装したのが以下のトークナイザー
  - TokenYaVgram
  - TokenYaVgramBoth
  - TokenYaVgramQuad

TokenYaVgram

- **原則**:管理テーブルのキーと一致するBigramトークンのみを後ろに伸ばしてTrigramにする
- 「処理」を登録
  「画像処理装置」をトークナイズ
  - 「画像/像処/**処理装**/理装/装置/置」
- 「処理装」が出現する頻度は「処理」よりも低い
  - **高速化**
- 検索時は上記と同様にオーバラップを飛ばす

TokenYaVgram

- 管理テーブルに出現頻度の高いBigramトークンのみを登録しておく
  - あらかじめ、ある程度の文章量が必要
  - 同分野であれば頻出トークンの傾向はほぼ同じ
  - 通常のBigramトークナイザーでインデックスを構築しAPIを使えばBigramの出現頻度を取得できる
  - Rroonga: https://gist.github.com/naoa/fac2cff05fa9113bf5d6
  - C-API: https://gist.github.com/naoa/8d862028e23e45e23304

TokenYaVgram

- **例外**:検索クエリの末尾ではTrigram対象のBigramトークンであっても後ろに伸ばせない
  - 本文中では伸ばされている可能性がある
  - この場合は強制的に前方一致検索させる
  - vgram対象でも2文字で検索が可能
    ただしTokenBigramのときより速くはならない
- 「処理」を登録
  文書登録:「画像処理装置」　検索クエリ:「画像処理」
  - 文書登録:「画像/像処/**処理装**/理装/装置/置」
  - 検索クエリ:「画像/　/処理*」

TokenYaVgraの速度

Wikipedia(ja)で1000回検索

トークナイザー	検索秒数平均
TokenBigram	0.0444 sec
TokenYaBigram	0.0325 sec
TokenYaTrigram	0.0063 sec
TokenYaVgram	**0.0166 sec**

TokenYaVgramのキー

Wikipedia(ja)

トークナイザー	キーの数	キーサイズ
TokenBigram	5767474	136.047MiB
TokenTrigram	28691883	684.047MiB
TokenYaVgram	**7425198**	**172.047MiB**

TokenYaVgramの効果

- キーサイズの増大を抑えつつ、検索の高速化を実現
- TokenYaTrigramほど速くはならなかった
  - 検索クエリ末尾がVgram対象の場合、2文字トークンで前方一致探索する必要がある

TokenYaVgramBoth

- **原則**:管理テーブルのキーと一致するBigramトークンのみを後ろに伸ばしてTrigramにする
- **さらに**:1つ後ろのBigramトークンが管理テーブルのキーと一致するトークンも後ろに伸ばしてTrigramにする
- 「処理」を登録
  文書登録:「画像処理装置」　検索クエリ:「画像処理」
  - 文書登録:「画像/**像処理**/**処理装**/理装/装置/置」
  - 検索クエリ:「画像/像処理」
  - 「処理」じゃなく「像処理」で探索できる⇒**高速化**

TokenYaVgramBoth

- **例外**:検索クエリの末尾では次のトークンがないため判断できない
- 全ての場合で検索クエリ末尾のトークンは強制的に前方一致検索させるせざるを得ない
  - 前方一致が必要のないケースで前方一致をしたとしてもあまり影響はない
    vgram対象でないやつはもとより遅くないという想定
- 「理装」を登録
  文書登録:「画像処理装置」　検索クエリ:「画像処理」
  - 文書登録:「画像/像処/**処理装**/**理装置**/装置/置」
  - 検索クエリ:「画像/　/処理*」

TokenYaVgramBotの速度

Wikipedia(ja)で1000回検索

トークナイザー	検索秒数平均
TokenBigram	0.0444 sec
TokenYaBigram	0.0325 sec
TokenYaTrigram	0.0063 sec
TokenYaVgram	0.0166 sec
TokenYaVgramBoth	**0.0065 sec**

TokenYaVgramBothのキー

Wikipedia(ja)

トークナイザー	キーの数	キーサイズ
TokenBigram	5767474	136.047MiB
TokenTrigram	28691883	684.047MiB
TokenYaVgram	7425198	172.047MiB
TokenYaVgramBoth	**8560779**	**200.047MiB**

TokenYaVgramBothの効果

- 出現頻度が高いもののみTrigramにすることでキーサイズの増大を抑えつつ、検索速度の高速化を実現
- TokenYaTrigram並の検索速度ながらもキーサイズをTokenTrigramの1/3以下に抑えられた

TokenYaVgramQuad

- 3文字にしてもまだ速度が満足いかなかったケースがあったのでつくってみた
- 管理テーブルに3文字のキーを登録しておくと、その対象のみを4文字トークンにする
- 現在はこのトークナイザーを適用中

既知フレーズのグループ化

- あらかじめ既知のフレーズを管理テーブルに登録しておき、そのフレーズのみをグループ化してトークナイズ
  - パトリシアトライのLCPサーチを利用して高速にフレーズ抽出
  - 「１２月は寒い」で「１２月」を登録
  - 「１２月／は寒／寒い」

既知フレーズのグループ化の効果

- ✓ 検索ノイズの低減
- ✓ 見出しタグや頻出語を含む複合語を登録することにより頻出トークン数低減
  - ✓ **高速化**

まとめ

- ✓ 検索速度を高速に保つのためにはポスティングリストが長くなりすぎないようにする
- ✓ Ngram検索時はオーバラップさせなくても良い
- ✓ Ngramといえどトークンの出現頻度は偏る
- ✓ 検索速度に影響のある頻出トークンのみを選択的に伸ばすVgramトークナイザーを作った

今後の課題

- Vgramは検索クエリが3文字以上のときは高速化できるが2文字以下の場合の高速化にはならない
  前方一致検索しないといけないのでむしろ少し遅くなる
- 前処理でいらないフレーズ除去、一部フレーズ化やストップワードという対応方法もあるがどれも制約が発生
- 1文字、2文字があまり遅くならないように特化したインデックスを別につくり検索クエリに応じてそちらのインデックスを使うようにできないかと検討中
- 形態素解析をスコアリングに特化した形でVgramとハイブリッドで使えないか検討中

TokenBigramとTokenYaVgramQuadの速度差デモ(時間があれば)