# FAST LARGE-SCALE APPROXIMATE GRAPH CONSTRUCTION FOR NLP

Amit Goyal, Hal Daume III, Raul Guerra

EMNLP 2012

読み手: 岡﨑　直観

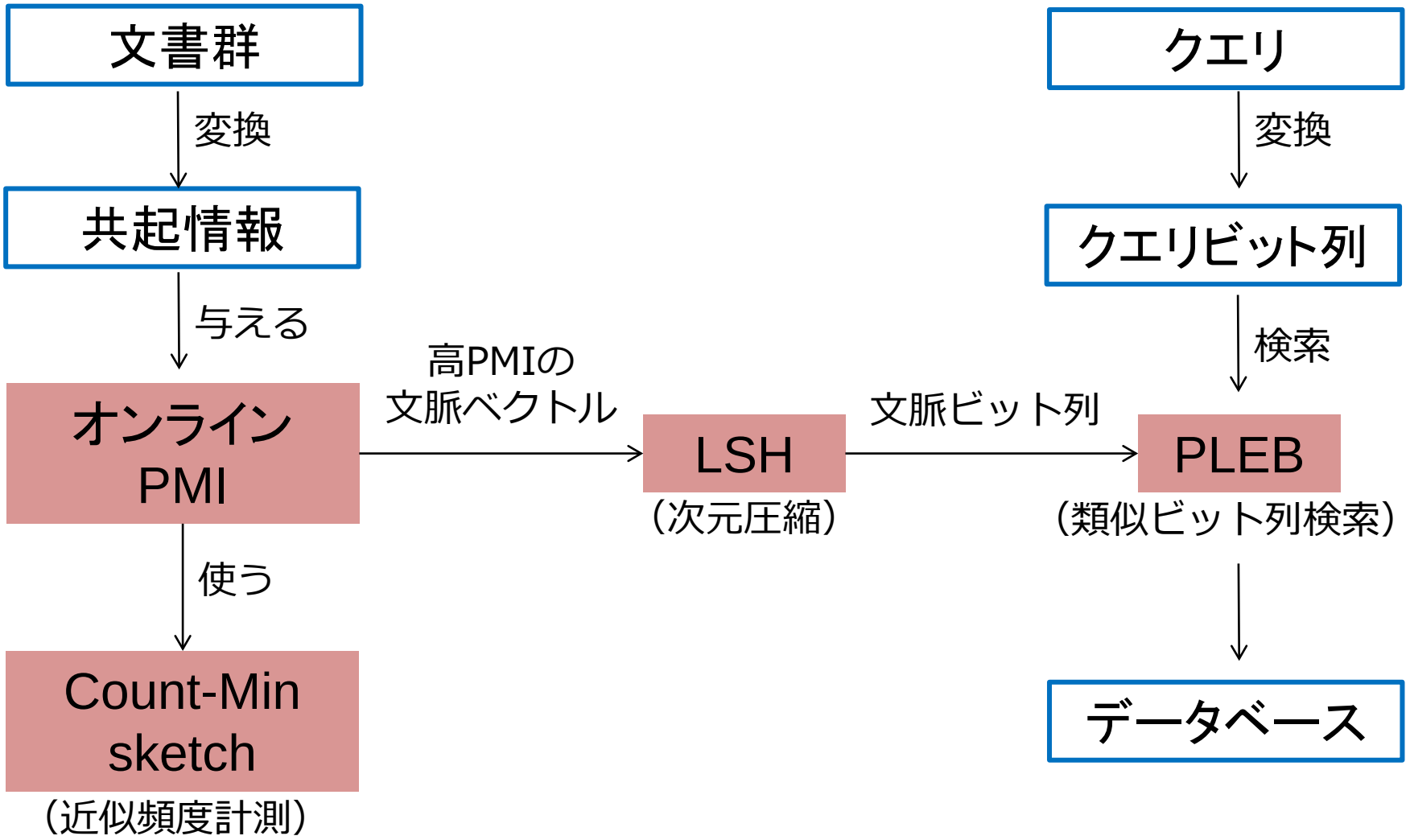
# 背景・研究の動機

- 多くのNLPの問題は類似度計算に落とし込める
  - 評価極性辞書の構築，集合拡張，教師無し品詞タグ付け，・・・
- 全ての単語ペアに対して類似度を計算するのは重い
  - 計算量は最低でも二乗オーダー
- 既存手法は，クラスタPC環境を使った力技
  - 例えば，Pantelら (2009) は100ノード・４コアのマシンを使って，５億語のペアの類似度を50時間で計算
- 本研究の目的は，最新の技術（近似，乱択，ストリーミング）を使って，大規模な類似度計算問題を解くこと！（←具体的な動機は書かれていません）

# 本研究の貢献

- 新しい分散オンラインPMI計算を提案
  - Count-Minアルゴリズムで頻度を近似計測
  - 分散・オンライン・近似をしてもPMI計算に与える影響が少ないことを確認
- 近傍探索アルゴリズムPLEBの改良を提案
  - 類似度DBの構築時間を短縮
- 集合拡張のアプリケーションを報告
  - Google Set問題，抽象／具体名詞の分類
- 最新の技術をうまく寄せ集め，有効性を示した
  - 本研究単体で見たときの新規性は微妙だと思います
  - 徹底的に実験を行っています

# 本研究の登場技術相関図

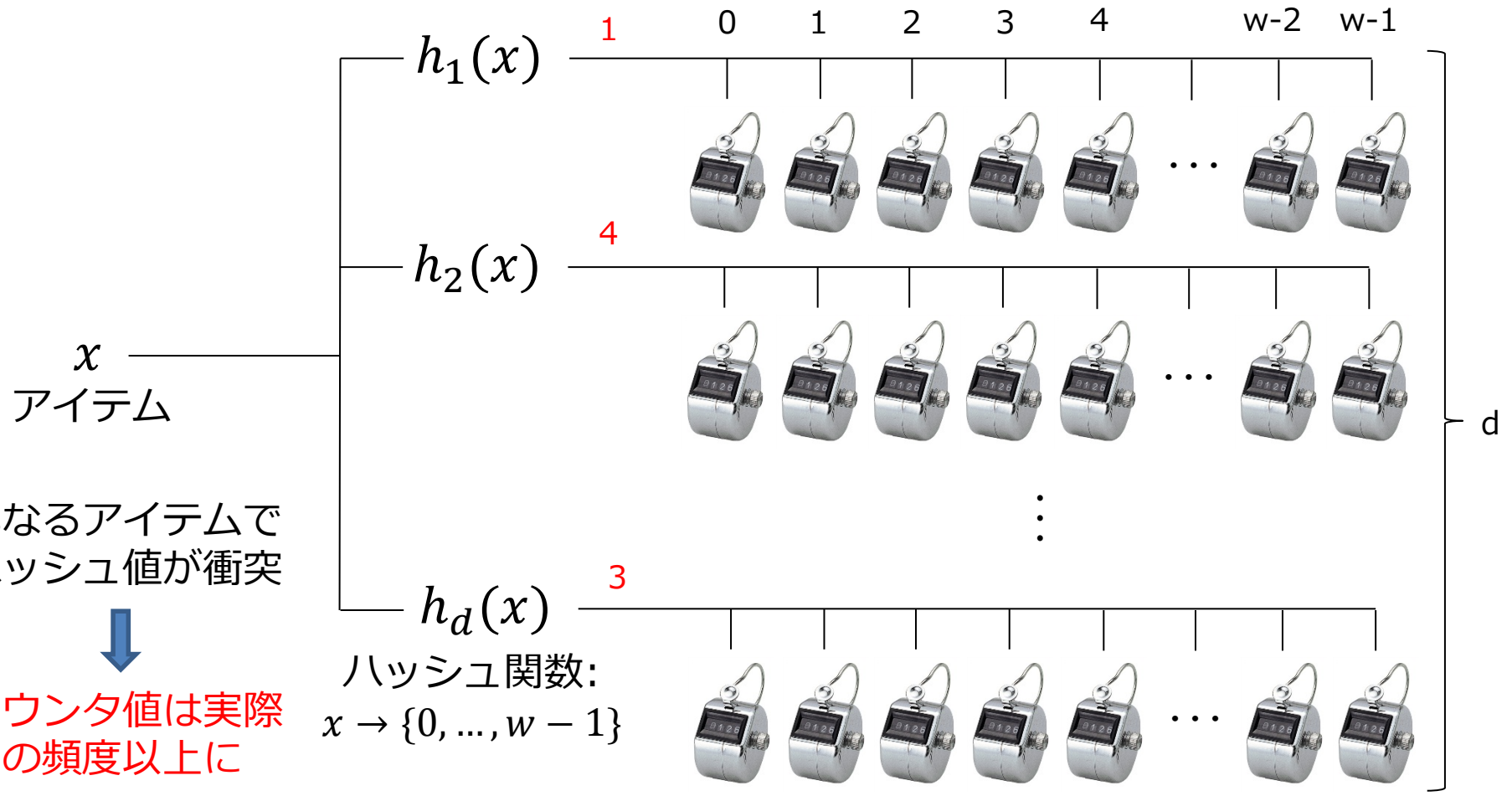


# Count-Min sketch

近似頻度計測アルゴリズム

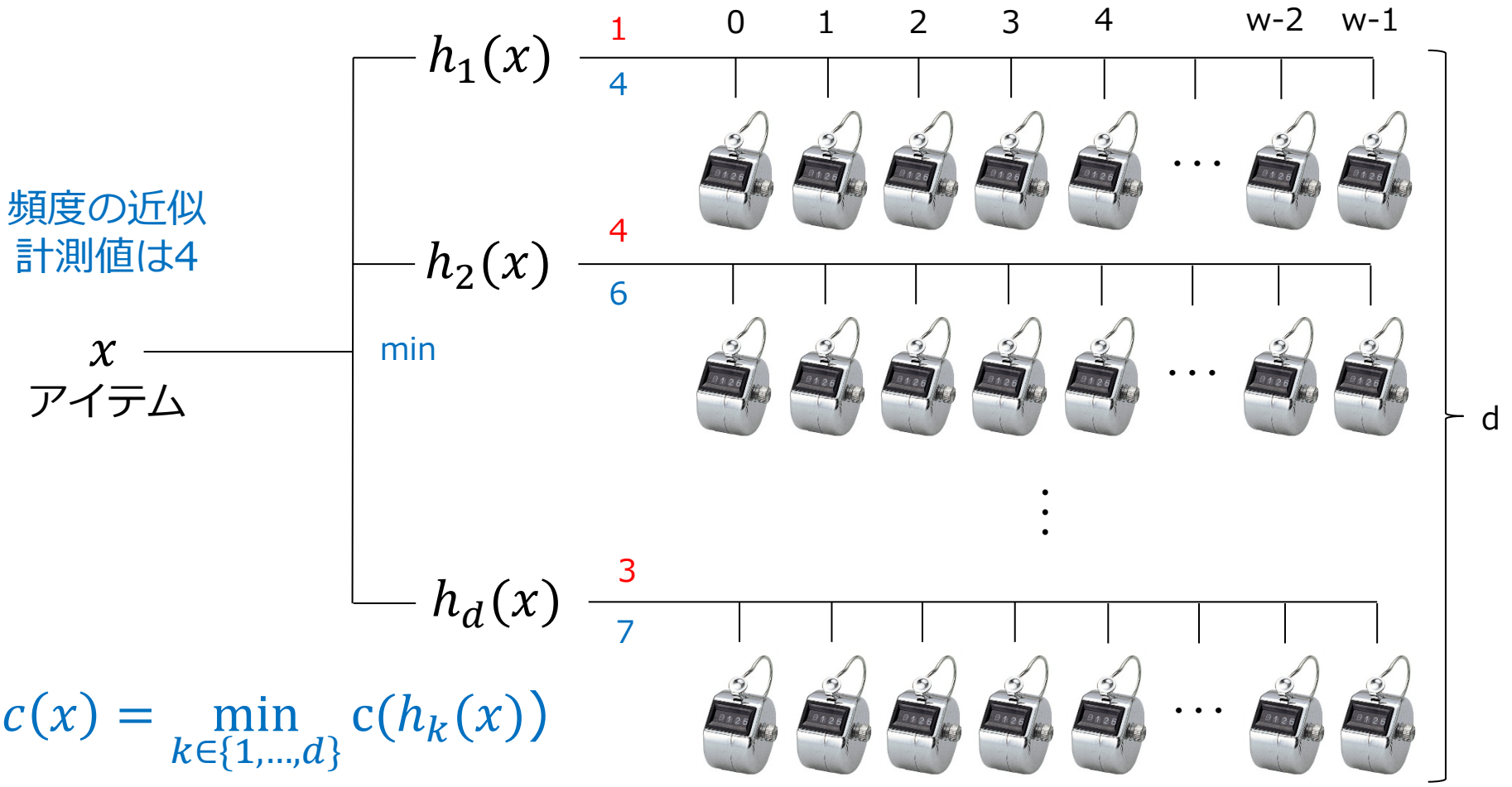
# Count-Min sketch (更新編)

- 近似頻度計測アルゴリズムの一種



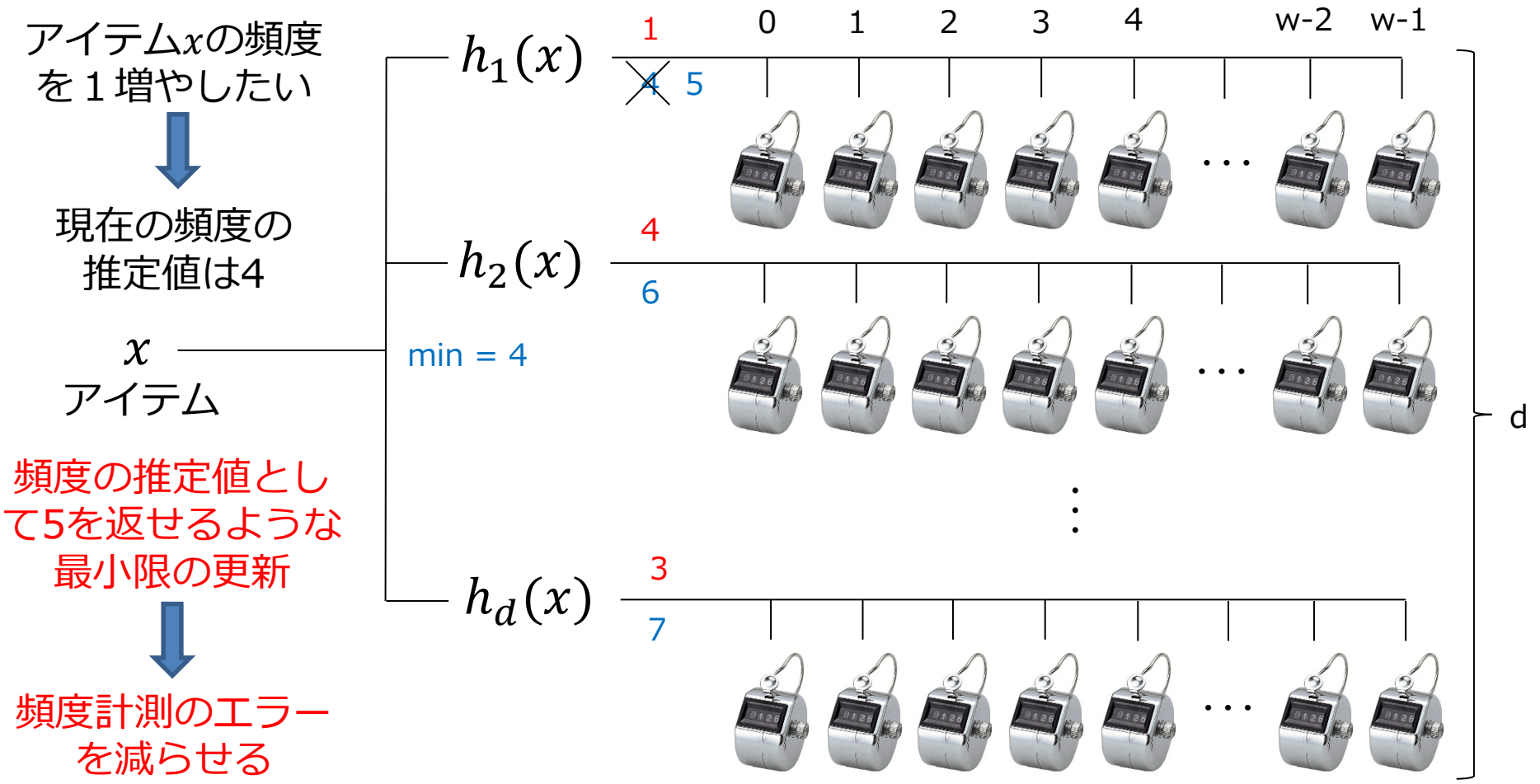
# Count-Min sketch (頻度取得編)

- ハッシュ値で選んだカウンタ値の最小値



# CM sketch with conservative updates

- 値が増えすぎているカウンタは増やさない

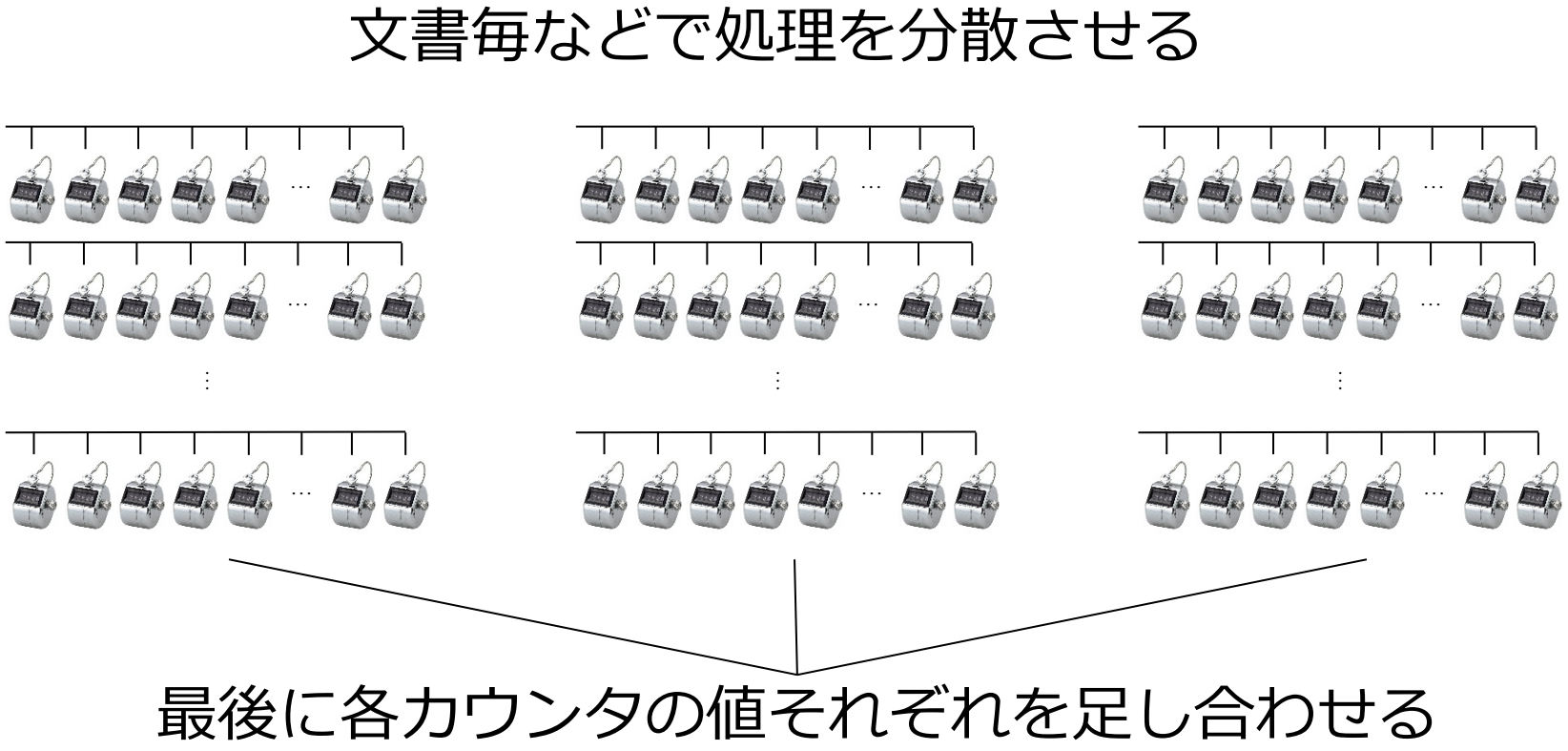


# Count-Min sketchの性質

- アイテムの頻度の総和を$N$とする
- 理論的なパラメータ
  - $\epsilon N$: 実際の頻度からの乖離の許容範囲
    - 頻度の乖離は，常にover-estimation
  - $\delta$: 頻度の乖離が$\epsilon N$を超えてしまう割合
    - $(1-\delta)$の割合のアイテムの頻度のエラーは許容範囲内
- パラメータの設定方法
  - $w = 2/\epsilon;\ d = \log(1/\delta)$
- 計算量: $O(d) = O(\log(1/\delta))$
- 空間利用量: $O(wd) = O(\frac{1}{\epsilon}\log(1/\delta))$

# Count-Min sketchの分散・並列化

- 同じハッシュ関数の組を用いて並列化したならば，各カウンタの値の和を取ればよい



# 分散
# オンラインPMI

PMI値が高いアイテムペアを省メモリで求める

# オンラインPMI (Van Durme+, 2009)

- 解きたいタスク:
  - アイテム$x$に対して，$PMI(x,y)$の高い$d$-bestなアイテム群$y$を返す
  - これを全てのアイテム$x \in X$に対して行うことで，PMIの近似行列を作成する
- 要件:
  - すべての$x,y$の組に対して$PMI(x,y)$を計算・保持するのではなく，省メモリにする

# オンラインPMI（改変版）

1. $C_x, C_y, C_{xy}$: 近似頻度計測器（Count-Min sketch）
2. $L_x$: $x$に対してPMIの高い$d$-bestなアイテム$y$を保持する優先順位付きキュー$\langle y, PMI(x,y)\rangle$
3. **for each** buffer $B$ **in** data $D$
4. $I \leftarrow$ an empty set
5. **for** $\langle x, y\rangle$ **in** $B$
6. insert $\langle x, y\rangle$ to $I$
7. increment $C_x(x)$; increment $C_y(y)$
8. increment $C_{xy}(x,y)$
9. **for each** $x$ **in** $I$
10. re-compute $L_x$ using $y \in L_x$ and $y \in I$

# オンラインPMIの分散・並列化

- 処理文書を適当なサイズに分割する
  - 分割された文書毎にオンラインPMIを適用
- 処理結果を集約
  - Count-Min sketch $C_x, C_y, C_{xy}$ を集約
  - $d$-best優先順位キュー$L_x$の和集合をとる
  - 集約された$C_x, C_y, C_{xy}$とキューの和集合から，最終的な$d$-bestリストを作成する

# Locality Sensitive Hashing

文脈ベクトルの次元圧縮

# 本研究におけるLSHの役割

- オンラインPMIで構築されたもの
  - 各アイテム$x$に対して，$PMI(x, y)$の高い$d$個の文脈ベクトル$\langle (c_1, v_1), (c_2, v_2), \ldots, (c_d, v_d) \rangle$
  - 文脈$c_y$の異なり数を$D$とすると，文脈ベクトルの次元は$D$（疎ベクトル非ゼロ要素は高々$d$個）

- 文脈ベクトルを用いて類似アイテムのペアを見つけたい

- $D$次元のベクトルを$k$次元のベクトルに圧縮することで，類似アイテムペアを高速に検索

# Locality Sensitive Hashing (コサイン型)

- コサイン距離をシミュレーションで求める

$$\begin{cases} u = (0.2 \quad \dots \quad 0.01) \\ v = (0.3 \quad \dots \quad 0.1) \end{cases} \longrightarrow \cdot \text{(内積)}$$

$$\begin{array}{l} r_1 = (0.1 \quad \dots \quad -0.4) \\ r_2 = (-0.3 \quad \dots \quad 0.2) \\ r_{\dots} = (\dots \quad \dots \quad \dots) \\ r_k = (0.5 \quad \dots \quad -0.1) \end{array} = \begin{array}{cc} h_r(u) & h_r(v) \\ + & - \\ - & - \\ \dots & \dots \\ + & + \end{array}$$

$$h_r(u) = \begin{cases} 1 & (r \cdot u \geq 0) \\ 0 & (r \cdot u < 0) \end{cases}$$

$D$次元の単位ランダムベクトル $r$を$k$個用意する

符号の一致する確率

- 定理より，$\Pr[h_r(u) = h_r(v)] = 1 - \frac{\theta(u,v)}{\pi}$
- したがって，$u$と$v$のコサイン係数は，

$$\cos(\theta(u,v)) = \cos\left((1 - \Pr[h_r(u) = h_r(v)])\pi\right)$$

# 定理の簡単な証明（２次元の場合）

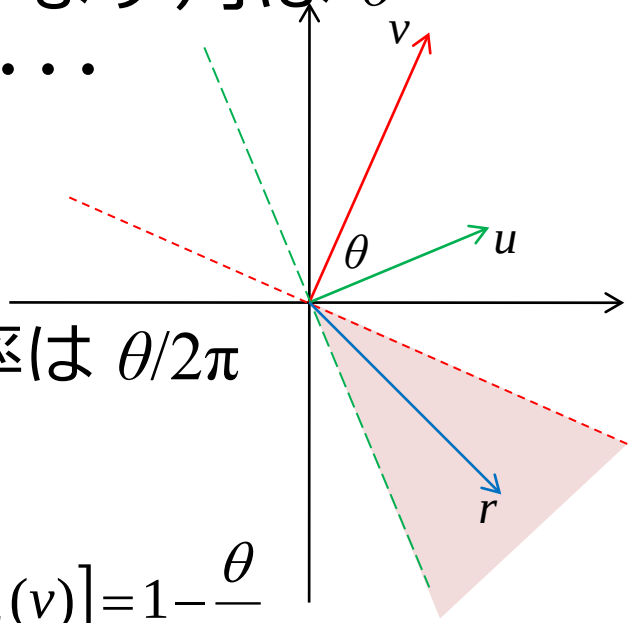
- 対称性により，

$$\Pr[h_r(u) \neq h_r(v)] = 2\Pr[u \cdot r \geq 0 \wedge v \cdot r < 0]$$

- このとき，$u$ と $r$ のなす角は90度以内，$v$ と $r$ のなす角は90度以上
- $u$ と $r$ の定義により，これらのなす角は $\theta$
- この状況を2次元で図示すると･･･
- したがって，$r$ は網掛けの領域に存在していればよい
  - $r$をランダムに選ぶとき，その確率は $\theta/2\pi$
- よって，

$$\Pr[h_r(u) \neq h_r(v)] = 2 \cdot \frac{\theta}{2\pi} = \frac{\theta}{\pi}$$

$$\Pr[h_r(u) = h_r(v)] = 1 - \Pr[h_r(u) \neq h_r(v)] = 1 - \frac{\theta}{\pi}$$



# これまでの話を整理すると・・・

- $k$ 個のランダムベクトルを作成
- LSHを用いて単語を $k$ ビットのストリームに変換
  - SaaS　　0100010010111000
  - クラウド 0110010010111010
  - 雲　　1100101011010011
- 似ている単語はビットストリームも似ている
  - LSHのビットストリームのハミング距離が近い（ビット毎の相違数が少ない）ものは，コサイン距離も近い
- つまり，この操作はコサイン係数を保持したまま次元削減（$D$ から $k$ へ）を行う処理
  - $D$ 次元上で距離の近い単語を探すよりも，$k$ 次元の0-1ベクトル上で距離の近い単語を探す方が，はるかに楽

# Point Location in Equal Balls (PLEB)

アイテムの文脈ビットストリームと似ているものを
高速に探すアルゴリズム（名前があったのか・・・）

# 類似アイテム検索

- これまでのおさらい
  - オンライン分散PMIアルゴリズムで，PMI値が高い文脈で構成された疎文脈ベクトルを獲得
  - 疎文脈ベクトルをLSHで低次元ビット列に圧縮
- 類似アイテム検索
  - 単語の文脈ビット列と似ているビット列を持つ単語を検索する
    - 例
      - クラウド: (0110010010111010)b 25786
      - 雲: (1100101011010011)b 51923
    - 以下のクエリから「雲」を探すのは意外と難しい
      - SaaS: (0100010010111000)b 17592

# 置換＋ソート＋二分探索＋周辺探索

- ハッシュ値のビット列の並び替え方をランダムに決めて，ビット列の辞書順（数値順）にソートしたリストを作る
  - 例えば3ビット目と右端のビット列を置換
    - クラウド　(0110010010111010)b: (0100010010111011)b = 17595
    - 雲　　(1100101011010011)b: (1110101011010010)b = 60114
- クエリのビット列も同様の方法で並び替える
  - 「SaaS」のビット列の数値は「クラウド」に近くなる
    - SaaS　　(0100010010111000)b: (0100010010111000)b = 17592
- ソートしたリストに対して，クエリのビット列に最も近い物を二分探索法で探し，周辺$b$個のアイテムをチェック
- 以上の探索処理で類似アイテムが見つかるかどうかは置換とクエリのビット列次第
  - 置換方法をいくつか（$q$個）試し，同じ事をやる

# ビット列の置換の高速化

- 置換操作はコストが高いので，普通は以下の線形変換で近似する
  - $\rho(x) = (ax + b) \bmod p$
  - $x$: 入力ハッシュ値
  - $a, b\ (< p)$: ランダムに決める数
  - $p$: 適当に決める素数

- 本論文はFAST-PLEBを提案
  - $k$ビット全体を並び替えるのではなく，$q$ビットだけ並び替えることとする（$q \ll k$）
  - これだけ？？？