1F-5

秘匿性を持つ ID の効率的生成方式

小山　武士　　松尾　俊彦　　鴨田　浩明
株式会社 ＮＴＴデータ 技術開発本部 セキュリティ技術センタ

1　はじめに

近年，情報システムにおいて Twitter のように 1 投稿ごとに識別子（ID）を付与して管理するシステムおよびサービスが普及してきている．このように不特定多数のノードから随時大量の処理要求を受けるシステムでは，大量の ID を高速に生成する必要がある．また ID の生成に際し，過去に生成した ID の履歴から次に生成される ID を類推することが困難であるという性質（秘匿性）を要する場合がある．例えば行政サービスにおいて個人情報を名寄せするために用いる共通番号，EC サイトなどで用いられるセッション ID や仮 ID など，ID をキーとして個人情報へ直接的にアクセスすることが可能である場合に，ID 生成に関する秘匿性を求められることがある．

秘匿性を持った ID は，乱数生成器等から生成した乱数を ID として割り当てることで生成することができるが，新たに生成した乱数が過去に生成されたそれと同一となった場合，別の乱数を割り当てる必要がある．秘匿性を持った ID の随時大量生成が要求される場合，生成した乱数と生成済乱数との重複チェックを高速に行う必要がある．

そこで本稿では，4 つの重複チェック方式を比較評価し，各方式の選択基準を示す．

2　重複チェック方式

2.1　対象とする重複チェック方式

以下に本稿で対象とする重複チェック方式を示す．各方式における重複チェックは，疑似乱数生成器で生成した 2 進乱数を固定長 10 進整数へ変換した後に実施する．

- 方式 1：生成済乱数と順次比較する方式
- 方式 2：B-tree を利用する方式
- 方式 3：ハッシュテーブルを利用する方式
- 方式 4：配列に格納されたフラグを確認する方式

方式 1 は，生成する乱数と生成済の全ての乱数を順次比較することで重複を検出する．この方式では，乱数の生成数の二乗に比例して累計比較回数が増えるため，比較処理が処理性能のボトルネックとなる可能性がある．

方式 2 は B-tree 構造に格納した乱数を検索し，重複を確認する方式である．そのため，重複の検出に要する累計比較回数は乱数の生成数と生成数の階乗の対数（底 2）の合計となる．

方式 3 は生成した各乱数に対するハッシュテーブルを構築し，ハッシュ値の有無およびバケットの中身を比較して重複を検出する方式である．ハッシュテーブルであるため，ハッシュ値の衝突が発生しない限りにおいて，重複検出に要する累計比較回数は乱数の生成数と同一になる．

方式 4 は，固定長 10 進整数で表現可能な ID の集合の元の総数（ID の空間）の配列を事前に確保し，生成した乱数を index とするバケットにフラグを立て，乱数を生成する毎に新たに生成した乱数を index とするバケット内のフラグをチェックする方式である（図 1 参照）．よって，当方式における重複チェックにかかる累計比較回数は乱数の生成数と同一になる．

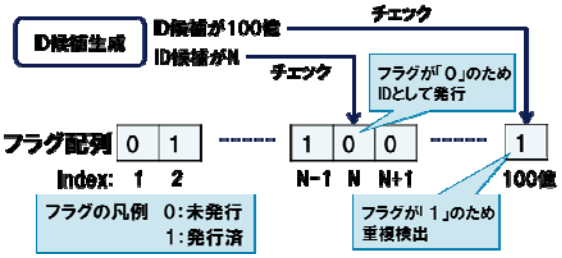

図 1 方式 4 の重複チェックの仕組み

2.2　重複チェック方式の実装

方式 1 は生成済乱数を格納する配列をメモリ上に確保し，新たに生成した乱数と順次比較することで実現する．

方式 2 は B-tree 構造の DB を用いて実現する．

方式 3 はハッシュテーブルを実装したインメモリ DB を用いて実現する．

方式 4 は，ID 空間分の 1 ビットの重複チェックフラグ（初期値は全て「0」）の配列を用意し，乱数が生成されるごとに乱数を index とする配列の要素であるフラグのチェックをメモリ上で

A study on creating huge number of distinct random strings
Takeshi KOYAMA, Toshihiko MATSUO, Hiroaki KAMODA
NTT DATA CORPORATION
koyamatk@nttdata.co.jp

行うことで実現する．チェック時にフラグが「1」であれば重複した乱数が生成されたため，再度乱数を生成する．

3　比較評価

3.1　評価実験・結果

前章の 4 種類の重複チェック方式について，実機を用いた ID 生成実験により比較する．

実験環境は表 1 の通りである．

表 1 実験環境

HW	マシン	DELL OptiPlex™ 980
	CPU	Intel® Core™ i7 920 2.66GHz
	メモリ	DDR3-1333 16GB
MW	OS	Ubuntu 10.04
	DB	PostgreSQL 8.4
	インメモリ DB	Redis 2.2.4
その他	擬似乱数生成器	Mersenne twister

また，実験は以下に従って実施した．

- ID のフォーマット：10 進数で 10 桁（ID の空間は 100 億）
- 生成する ID 数：1.3 億件
- 実験フロー
  1. 擬似乱数生成器より 2 進乱数を生成
  2. 乱数を ID のフォーマットに整形
  3. 重複チェック
- 測定メトリクス：生成完了までの経過時間

以上の実験条件により実験を行った結果を図 2 に示し，必要メモリ量の算出結果を表 2 に示す．

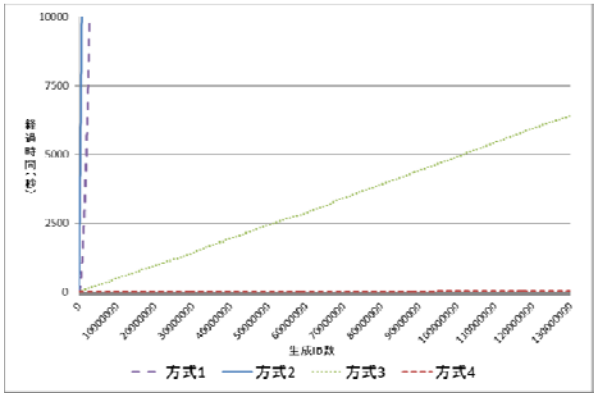

図 2 実験結果：1.3 億件生成までの経過時間

以下，「生成完了までの経過時間」および「メモリ使用量」の観点で結果を説明する．

【生成完了までの経過時間】

生成に要する時間は方式 3，4 が短く，現実的な時間内（それぞれ約 1.5 時間，約 1 分）に 1.3 億件を生成できた．一方，方式 1 は生成済みの ID 数が増加するとチェック時間を多く要し，現実的な時間内に生成が完了しなかったため途中で実験を終了した．仮に現在の生成速度から 1.3 億件生成時の経過時間を算出したとしても数千万年もかかることになり現実的な時間内には生成が終了しない．また，方式 2 については採用した DB の仕様上，ディスクアクセスが生じる実装であるため，他方式と一律に比較することは不可能であった．

表 2 重複チェックに必要なメモリ量

	メモリ量(GByte)
方式 1	0.97
方式 2	4.8
方式 3	7.7
方式 4	1.2

【メモリ使用量】

方式 1，2，3 では生成する ID の件数に比例してメモリ使用量が増加する．また，方式 2，3 は DB を用いるため ID そのもののデータ量に加え DB を構成するためのメモリ量が必要となる．一方，方式 4 は生成する ID の件数によらず，ID の空間に比例してメモリ使用量が増加する．

3.2　考察

重複チェック方式は重複チェックに利用可能なメモリ量によって選択できる方式が異なる．

生成する ID の空間分のメモリ量を確保できる場合は，方式 4 で重複チェックを行うと高速にチェック可能である．

方式 4 に必要なメモリ量を確保できない場合，即ち ID の空間が大きい場合は生成する ID 数に応じて選択できる方式が制限される．この場合，方式 3 は他方式と比べてメモリ使用量が多いため，最終的に発行する ID 分のメモリ量を確保可能であれば，他方式に比べ高速にチェック可能である．一方，メモリ量の確保が不可能な場合は，方式 1，2 が利用できるが，大量の ID を発行する場合は生成速度が低速という制限が残る．

4　おわりに

本稿では秘匿性を持った ID の生成における重複チェック方式に関して複数方式を実装し，評価実験を実施した．特定の実験環境下で 1.3 億件の ID の生成時間と重複チェックに必要なメモリ使用量を計測し比較評価を行った．その結果，ID の空間の大きさ，および生成する ID 数によって方式毎にメモリ使用量が異なることを示し，ID 生成に求められる要件に応じて適切な方式を選択するための基準を示した．