Adapting MapReduce for Efficient Watermarking of Large Relational Dataset

Sapana Rani, Dileep Kumar Koshley and Raju Halder
Indian Institute of Technology Patna, India
{sapana.pcs13, dileep.pcs15, halder}@iitp.ac.in

***Abstract*—In the era of big-data when volume is increasing at an unprecedented rate, structured data is not an exception from this. A survey in 2013 by TDWI says that, for a quarter of organizations, big-data mostly takes the form of the relational and structured data that comes from traditional applications. In this reality, watermarking of large volume of structured relational dataset using existing watermarking techniques are highly inefficient, and even impractical in the situations when periodic rewatermarking after a certain time frame is necessary. As a remedy of this, in this paper, we adapt MapReduce as an effective distributive way of watermarking of large relational dataset. We show how existing algorithms can easily be converted into an equivalent form in MapReduce paradigm. We present experimental evaluation results on a benchmark dataset to establish the effectiveness of our approach. The results demonstrate significant improvements in watermark generation and detection times w.r.t. existing works in the literature.**

***Keywords*-Large Relational Dataset; Watermarking; MapReduce;**

I. Introduction

Dealing with large dataset generated from various sectors like health care, census, survey, web-based retail shops such as e-bay or amazon, etc., in the order of terabytes or even petabytes is a reality now-a-days. For example, as reported in [1], U.S. health care data alone reached 150 exabytes (10^{18}) in 2011 and it is predicted that it will exceed the zettabytes (10^{21}) and the yottabytes (10^{24}) in the near future. This is observed that more than 20% of large volume of data are structured in nature and are stored in the form of relational database [2]. Intuitively, like any other databases, these databases also suffer from various attacks like copyright infringement, data tampering, integrity violations, piracy, illegal redistribution, ownership claiming, forgery, theft, etc. [3].

Database watermarking has been introduced as one of the most effective solutions to address the above mentioned threats [4], [5], [6], [7]. The basic idea of this technique is to embed a piece of information (known as watermark) in an underlying data and to extract it later from any suspicious content in order to verify the absence or presence of any possible attacks. The former phase is known as *Embedding* phase, whereas the later phase is known as *Detection* or *Verification* phase [8].

This work is partially supported by the Council of Scientific and Industrial Research (CSIR), India.

A. Related Works

Watermarking of relational databases was first proposed by Agrawal et al. in [5]. Subsequently many works have been proposed in this direction [4], [6], [7], [9], [10], [11], [12], [13], [14], [15]. Broadly the watermarking of relational databases can be categorized into *distortion-based* [5], [6], [7], [9], [12] and *distortion-free* [4], [10], [11], [13], [14], [16], [17]. In general, distortion-based techniques embed watermark keeping in mind the usability of data, whereas distortion-free techniques generate watermark based on various characteristics of data. A comprehensive survey on various types of watermarks and their characteristics, possible attacks, and the state-of-the-art can be found in [3]. A recent survey by Xie et al. is reported in [18] giving special attention to the distortion-based watermarking.

The existing proposals on distortion-based watermarking techniques include random bit flipping [5], [12], fake tuple insertion [19], random bit insertion [7], [9], etc. On the other hand, distortion-free watermarking techniques in the literature are based on tuple reordering [4], binary image generation [10], [11], generation of local characteristics like range, digit and length frequencies [14], matrix operations [13], etc. To the best of our knowledge, the first proposal to address watermarking of distributed databases is proposed in [20] which supports database outsourcing and hybrid partitioning.

B. Motivations and Contributions

Although all the existing techniques work well in watermarking of small dataset, however they are highly inefficient in case of watermarking of large relational dataset. The primary reason is the involvement of various computations like hash calculation, grouping of tuples, group-wise watermark generation, etc., which consume a significant amount of time in sequential processing of all database-tuples, thus making the existing watermarking techniques highly inefficient and even impractical for very large databases. This concern also pertains to the situations where databases go through frequent updation due to which periodic rewatermarking after certain time frame is necessary.

To ameliorate this performance bottleneck, this paper presents an efficient way of watermarking of large scale relational dataset exploiting the benefits of parallel and distributed computing environment. In particular, we adopt the potential of the MapReduce paradigm identifying the key computational steps involved in watermarking approaches.

To summarize, our major contributions in this paper are:

- We adapt MapReduce for fast and efficient watermarking of large scale relational databases, as an alternative to the existing sequential algorithms and their challenges.
- We design a generic MapReduce-based watermarking algorithm identifying the key computational steps involved in watermarking approaches.
- A case study describing the design of MapReduce-based variant of an existing sequential form of watermarking algorithm is presented.
- Finally, we perform rigorous experiments on benchmark dataset. The evaluation results are really encouraging and provide an evidence of significant improvements over the existing sequential approaches.

To the best of our knowledge, till now there exists no watermarking technique for big-data in the literature and this is the first work towards this direction. In this preliminary work, we effectively deal with only the "volume" characteristic of big relational dataset, and we restrict only to the distortion-free techniques which generate watermark from the underlying database content.

The structure of the paper is as follows: Section II briefly introduces MapReduce algorithm. Section III presents our proposed approach of database watermarking using MapReduce. A case study describing the design of MapReduce variant of sequential watermarking algorithm is presented in section IV. The experimental results are analyzed in section V. Finally we conclude our work in section VI.

II. MapReduce algorithm at a glance

Jeffrey et al. [21], [22] introduced MapReduce programming model as highly efficient and reliable solution to process large dataset in distributed computing environment. In general, MapReduce algorithm is implemented based on two primary functions: *map* and *reduce*. The map function processes a key-value pair to generate a set of intermediate key-value pairs. The output of a map function then acts as input to the reduce function which merges all intermediate values associated with the same intermediate key. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. In particular, distributed and parallel computation of these functions results into a significant reduction of the overall data processing time. A pictorial form of MapReduce algorithm to compute word frequency is depicted in Figure 1. The map function emits each word plus an associated count of occurrences (just '1' in this example). The reduce function sums together all counts emitted for a particular word.

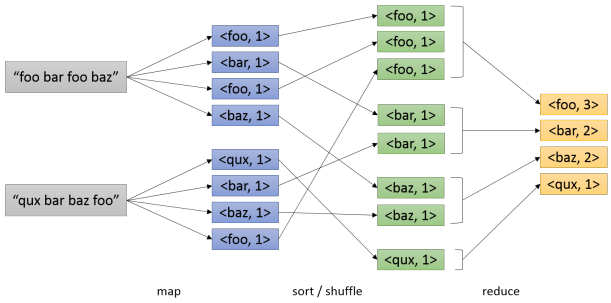


Figure 1: Word frequency computation using MapReduce

III. Adapting MapReduce Algorithm for Distortion-free Watermarking

In this section, we provide an insight on how to adapt MapReduce-based computing paradigm as an alternative to the existing watermarking algorithms, leading to an efficient watermarking of large relational dataset. In this paper, although we mainly focus on distortion-free watermarking techniques, however this can easily be extended to the case of distortion-based watermarking as well.

As observed, the existing distortion-free watermarking techniques [4], [10], [11], [13], in general, mainly focus on the generation of watermarks from database-contents. Precisely, they consist of two prime phases: (i) Partitioning of tuples into groups, and (ii) Watermark generation from each group. The obtained watermarks from all groups may finally be combined together to generate watermark for the whole database.

Watermark Generation using MapReduce: While designing MapReduce algorithm for watermarking of large dataset, our primary focus is how to adapt parallel computation in an efficient way into the above-mentioned two prime phases – *partitioning* and *watermark generation*. This results a significant improvement in the computational costs involved in those phases.

Algorithm 1 and 2 in Figure 2 refer to the watermark generation process in MapReduce framework. On assigning each mapper a fragment R from the large dataset, the MAP function applies partition-function f on each tuple t and computes the group-ID g_{id} to which the tuple belongs. An example of f is a modulo function on hash [23] of the tuple values [5]. This group-ID along with the tuple itself acts as an intermediate key-value pair emitted as the output from the mapper. Observe that this way multiple mappers process large scale dataset in parallel and partition them in efficient way.

The outputs of all mappers are then fed to a reducer which finally computes the watermark W for whole database by computing group-wise watermarks W_g and by combining them together using suitable operation (denoted by $||$) . This

Algorithm 1 Watermark Generation: MAPPER

1: **procedure** MAP (R)
2: **for** each tuple $t \in R$ **do**
3: Compute $g_{id} = f(t)$
4: $emit\ (g_{id}, t)$
5: **end for**
6: **end procedure**

Algorithm 2 Watermark Generation: REDUCER

1: **procedure** REDUCE ($g_{id}, list := [t_1, t_2, ...]$)
2: Compute watermark W_g for g_{id} using $list$
3: $emit\ (g_{id}, W_g)$
4: **end procedure**
5: Compute $W = ||W_g$

Figure 2: Watermark Generation algorithm using MapReduce

is worthwhile to mention that the proposed MapReduce-based watermarking framework is generic in the sense that any existing watermarking technique is easily adaptable to this.

Watermark Detection using MapReduce: The watermark detection process for a suspicious database fragment $R^{'}$ is formalized in Algorithm 3 and 4 in Figure 3.

The input of detection phase is a suspicious database fragment and the output is a reasoning about the success of watermark detection/verification.

Algorithm 3 Watermark Detection: MAPPER

1: **procedure** MAP ($R^{'}$)
2: **for** each tuple $t \in R^{'}$ **do**
3: Compute $g_{id} = f(t)$
4: $emit\ (g_{id}, t)$
5: **end for**
6: **end procedure**

Algorithm 4 Watermark Detection: REDUCER

1: **procedure** REDUCE ($g_{id}, list := [t_1, t_2, ...]$)
2: Compute $W_g^{'}$ for g_{id} using $list$
3: $emit\ (g_{id}, W_g^{'})$
4: **end procedure**
5: Compute $W^{'} = ||W_g^{'}$
6: **if** ($W^{'} == W$) **then** Verification = $TRUE$
7: **else** Verification = $FALSE$
8: **end if**

Figure 3: Watermark Detection algorithm using MapReduce

As like watermark generation phase, on assigning each mapper to a fragment $R^{'}$ of the large suspicious dataset during detection, the MAP function applies on each tuple $t \in R^{'}$ the same partition-function f as used in the generation phase to compute its group-ID g_{id}. The MAP function this way emits $\langle g_{id}, t \rangle$ as intermediate key-value pair which is then used as input to the reducer. The REDUCE function then computes the watermark $W_g^{'}$ for each group. The complete watermark $W^{'}$ for the suspicious database is obtained by following similar operations as in the watermark generation phase. Finally the detection phase concludes a success if $W^{'}$ matches with the original watermark W.

IV. A Case Study: Transforming Sequential to Distributed Watermark Computation

This section considers the watermarking algorithm proposed by Li et al. in [4] and describes how to design its MapReduce version making it suitable for large scale databases. The notations and parameters used in the algorithms are depicted in Table I.

ω	number of tuples in the relation
r_i	i^{th} tuple
$r_i.A_j$	j^{th} attribute of the i^{th} tuple
h_i	tuple hash of the i^{th} tuple
$h_i{}^P$	primary key hash of the i^{th} tuple
g	number of groups after partitioning of tuples
G_k	k^{th} group
g_{id}	group-ID of a particular group
q_k	number of tuples in group k
H	group hash
K	watermarking key
W	watermark generated from a group
$W^{'}$	watermark generated from a suspicious group
V	result of watermark detection

Table I: Notations and Parameters

Algorithms 5, 6 and 7 in Figure 4(a) depict the sequential version of the watermarking algorithm by Li et al. [4]. The Algorithm 5 initially computes tuple hash and primary key hash for each tuple in steps 5 and 6 respectively. In steps 7 and 8, the database tuples are divided into various groups. In algorithm 6, group hash is calculated from each group in step 2. The watermark is then generated from group hash by calling `ExtractBits` in Algorithm 7. The Algorithm 6 then finally reorders the tuples in a group based on these watermark bits. Observe that sequential processing of tuples to partition and to generate watermarks absorb a significant amount of time. This makes the algorithm highly inefficient in case of big relational databases.

The corresponding MapReduce algorithms for partitioning and watermark-generation for each group are depicted in Figure 4(b). Given a fragment R, the mapper in Algorithm 8 computes tuple hash and primary key hash for each tuple (in steps 3 and 4 respectively) and computes the group to which a tuple belongs (step 5). The mapper then emits group-ID and $\langle$tuple, tuple hash, primary key hash$\rangle$ as intermediate key-value pairs in step 6. These key-value pairs act as input to the reducer depicted in Algorithm 9. The reducer first sorts the tuples in the group according to their primary key hash values in steps 3, and computes the group hash in step 4. The watermark bits from this group hash are extracted by calling `ExtractBits` as depicted in Algorithm 10. Finally, the tuples are reordered based on the watermark in steps 7-12 of Algorithm 9.

Algorithm 5 Li2004 Watermark Generation

for k = 1 to g **do**
  q_k = 0
end for
for i = 1 to ω **do**
  $h_i = \texttt{HASH}(K, r_i.A_1, ..., r_i.A_\gamma)$ // tuple hash
  ${h_i}^P = \texttt{HASH}(K, r_i.P)$ // primary key hash
  $k = {h_i}^P mod\ g$
  $r_i \rightarrow G_k$
  q_k++
end for
for $k = 1\ to\ g$ **do**
  watermark generation in G_k // See Algorithm 6
end for

Algorithm 6 Watermark Generation in G_k

Sort tuples in G_k in ascending order based on primary key hash // virtual operation
$H = \texttt{HASH}(K, h_1, h_2, ..., h_{q_k})$
W = $\texttt{ExtractBits}(H, qk/2)$ // See algorithm 7
for $i = 1$ to q_k - 1 **do**
  if ($W[i/2] == 1\ and\ h_i < h_{i+1}$) or ($W[i/2] == 0\ and\ h_i > h_{i+1}$) **then**
    Switch r_i and r_{i+1}
  end if
  i = i + 2
end for

Algorithm 7 $\texttt{ExtractBits}(H, l)$

if length(H) $> l$ **then**
  W = concatenation of first l selected bits from H
else
  $m = l$ - length(H)
  W = concatenation of H and $\texttt{ExtractBits}(H, m)$
end if
return W

(a) Sequential Watermark Generation Algorithm by Li et al. [4]

Algorithm 8 Watermark Generation using MapReduce: Mapper

procedure MAP (R)
  for i = 1 to ω **do**
    $h_i = \texttt{HASH}(K, r_i.A_1, ..., r_i.A_k)$ // tuple hash
    ${h_i}^P = \texttt{HASH}(K, r_i.P)$ // primary key hash
    $g_{id} = {h_i}^P mod\ g$
    $emit\ (g_{id}, \langle r_i, h_i, {h_i}^P \rangle)$
  end for
end procedure

Algorithm 9 Watermark Generation using MapReduce: Reducer

procedure REDUCE ($g_{id}, list := [r_1, r_2, ..., r_k]$)
  $k = |list|$
  Sort tuples in ascending order according to ${h_i}^P$. // virtual operation
  $Group\_hash$ = $\texttt{HASH}(K, h_1, ..., h_k)$ // group hash
  W = $\texttt{ExtractBits}(Group\_hash, k/2)$ // algorithm 10
  $emit(g_{id}, W)$
  for i = 1 to k - 1 **do**
    if ($W[i/2] == 1\ and\ h_i < h_{i+1}$) or ($W[i/2] == 0\ and\ h_i > h_{i+1}$) **then**
      Switch r_i and r_{i+1}
    end if
    i = i + 2
  end for
end procedure

Algorithm 10 $\texttt{ExtractBits}(H, l)$

if length(H) > 1 **then**
  W = concatenation of first l selected bits from H
else
  $m = l$ - length(H)
  W = concatenation of H and $\texttt{ExtractBits}(H, m)$
end if
return W

(b) MapReduce-based Watermark Generation Algorithm

Algorithm 11 Li2004 Watermark Detection

for k = 1 to g **do**
  $q_k = 0$
end for
for $i = 1\ to\ \omega$ **do**
  $h_i = \texttt{HASH}(K, r_i.A_1, ..., r_i.A_\gamma)$ // tuple hash
  ${h_i}^P = \texttt{HASH}(K, r_i.P)$ // primary key hash
  $k = {h_i}^P mod\ g$
  $r_i \rightarrow G_k$
  q_k++
end for
for $k = 1\ to\ g$ **do**
  watermark verification in G_k // See Algorithm 12
end for

Algorithm 12 Watermark Verification in G_k

Sort tuples in G_k in ascending order based on primary key hash
$H = \texttt{HASH}(K, h_1, h_2, ..., h_{qk})$
$W = \texttt{ExtractBits}(H, qk/2)$ // See algorithm 7
for i = 1 to q_k - 1 **do**
  if ($h_i \leq h_{i+1}$) **then**
    $(W'[i/2] = 0)$
  else
    $(W'[i/2] = 1)$
    if $(W' == W)$ **then**
      $(V = TRUE)$
    else
      $(V = FALSE)$
    end if
  end if
  $i = i + 2$
end for

(c) Sequential Watermark Detection Algorithm by Li et al. [4]

Algorithm 13 Watermark Detection using MapReduce: Mapper

procedure MAP (R)
  for i = 1 to ω **do**
    $h_i = \texttt{HASH}(K, r_i.A_1, ..., r_i.A_k)$ // tuple hash
    ${h_i}^P = \texttt{HASH}(K, r_i.P)$ // primary key hash
    $g_{id} = {h_i}^P mod\ g$
    $emit\ (g_{id}, \langle r_i, h_i, {h_i}^P \rangle)$
  end for
end procedure

Algorithm 14 Watermark Detection using MapReduce: Reducer

procedure REDUCE ($g_{id}, list := [r_1, r_2, ..., r_k]$)
  $k = |list|$
  Sort tuples in ascending order according to ${h_i}^P$. //virtual operation
  $Group\_hash$ = $\texttt{HASH}(K, h_1, ..., h_k)$ // group hash
  $W = \texttt{ExtractBits}(Group\_hash, k/2)$
  $emit(g_{id}, W)$
  for i = 1 to k - 1 **do**
    if ($h_i \leq h_{i+1}$) **then**
      $(W'[i/2] = 0)$
    else$(W'[i/2] = 1)$
      if $(W' == W)$ **then**
        $(V = TRUE)$
      else$(V = FALSE)$
      end if
    end if
    $i = i + 2$
  end for
end procedure

(d) MapReduce-based Watermark Detection Algorithm

Figure 4: MapReduce-based Watermarking Algorithm corresponding to the Li et al.'s sequential algorithm [4]

Similarly, Figure 4(d) depicts the MapReduce-based watermark detection algorithm corresponding to the sequential version in Figure 4(c) proposed by Li et al. This way any existing watermarking technique can be converted into its equivalent MapReduce form.

This is to observe that in the process of designing MapReduce watermarking algorithm one has to identify all possible prime computations (for example, hash value computation, grouping of database tuples, etc.) which involve high computational complexity for large dataset. A careful assignment of these computations to mappers, of course, reduce watermarking time significantly.

V. Experimental Analysis

We have searched the literature exhaustively and identified five significant proposals [4], [10], [11], [13], [14] for our experiments. We have implemented all the five algorithms in both sequential and MapReduce framework using Java. The experiments are performed using the Hadoop framework (version 2.7.0)[1] installed on a server equipped with six core Intel Xeon Processor, 2.4 GHz Clock Speed, 128 GB RAM and Linux Operating System. We use benchmark dataset obtained by modifying the Forest Cover Type dataset[2] into a large volume.

The following notations are used in this section to describe the experimental results:

T_g : Watermark generation time in minute
T_d : Watermark detection time in minute
M : Number of Mappers
G : Number of groups formed from the database tuples

This is worthwhile to mention here that, although we have varied the number of mappers in our experiments, but we have used single reducer in all the cases.

Li et al., 2004 [4]: The sequential and MapReduce algorithms for this technique have been discussed in section IV. In these algorithms, the database is partitioned into various groups and the watermark is generated for each group separately.

Table II depicts the watermark generation and detection times in both sequential- and MapReduce-based implementations by considering G equal to 10000 and 50000 in each of these cases.

[1]Installation details at http://hadoop.apache.org/
[2]https://kdd.ics.uci.edu/databases/covertype/covertype.html

Existing algorithm	No. of Tuples	Size in MB	Sequential		MapReduce		
			T_g	T_d	M	T_g	T_d
Li et al. [4]	5810120	276	23.19	24.43	2	5.26	5.23
	11620240	556	83.36	75.62	5	9.70	9.18
	17430360	840	172.44	169.66	7	14.64	13.44
	23240480	1124	294.74	291.60	9	19.96	18.28
	34860720	1692	586.00	566.83	13	31.72	28.61

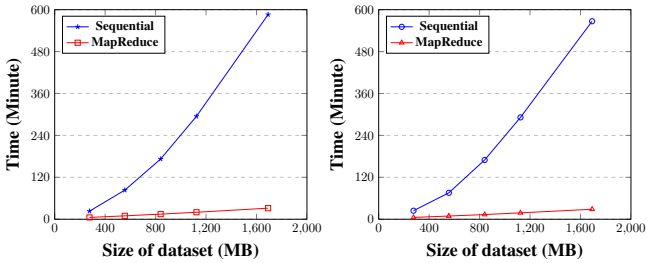
(a) Watermarking results for G = 10000

Existing algorithm	No. of Tuples	Size in MB	Sequential		MapReduce		
			T_g	T_d	M	T_g	T_d
Li et al. [4]	5810120	276	7.16	6.87	2	5.11	4.89
	11620240	556	22.99	19.22	5	9.26	8.52
	17430360	840	41.75	39.08	7	13.55	12.25
	23240480	1124	69.19	66.60	9	18.26	16.23
	34860720	1692	124.02	120.25	13	25.64	23.83

(b) Watermarking results for G = 50000

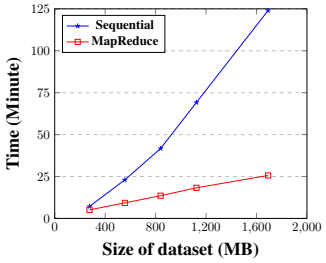
Table II: Results by sequential watermarking algorithm of Li et al. [4] and by its corresponding MapReduce Version

The performance comparison in terms of watermark generation time and detection time for G equal to 10000 and 50000 are shown in Figures 6.

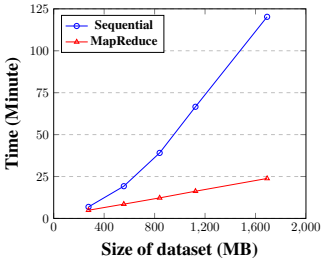


(a) Watermark Generation Time for G=10000

(b) Watermark Detection Time for G=10000



(c) Watermark Generation Time for G=50000



(d) Watermark Detection Time for G=50000

Figure 6: Performance comparison between Li et al.'s sequential [4] and its corresponding MapReduce Version

The observations are as follows:

- The watermark generation time and detection time reduce significantly in MapReduce framework as compared to the sequential one.
- From Figure 6(a) and 6(c), we observe that the watermark generation time decreases as the number of

groups increases. Similarly, we observe from Figure 6(b) and 6(d) that the detection time decreases as the number of groups increases.

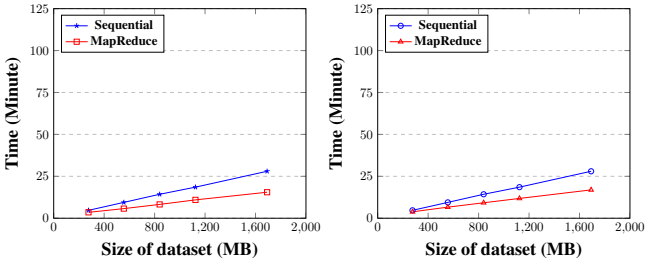
Bhattacharya and Cortesi, 2009 [10]: The watermark generation technique proposed in [10] is based on the following two steps: (i) grouping of tuples in the relation and (ii) generation of watermark for each group independently. In the former step, the message authenticated code (MAC) using HMAC for each tuple is calculated by seeding a secret key along with its primary key. This resultant value is used to determine the group to which the tuple belongs. In the later step, the most significant bits (MSBs) of the attributes values are used to generate watermark.

In our MapReduce-based version, we have assigned to the mappers the task of identifying group-ID for each tuple and the extraction of MSBs from all attributes values. The group to which a tuple belongs is calculated based on the message authenticated code (MAC) using HMAC. All mappers then send the group-ID and the MSBs of tuples as key-value pair to the reducer. Finally, the reducer collects the MSBs belonging to each group and combines them to form the watermark.

The comparison of watermark generation and detection times in sequential and MapReduce framework are depicted in Table III and Figure 7.

Existing algorithm	No. of Tuples	Size in MB	Sequential		MapReduce		
			T_g	T_d	M	T_g	T_d
Bhatta-charya et al. [10]	5810120	276	3.89	4.72	2	3.52	3.83
	11620240	556	7.77	9.45	5	5.74	6.61
	17430360	840	11.13	14.29	7	8.24	9.22
	23240480	1124	15.28	18.54	9	10.95	11.81
	34860720	1692	22.38	28.01	13	15.51	16.90

Table III: Results by sequential watermarking algorithm of Bhattacharya et al. [10] and by its corresponding MapReduce Version for G = 50000



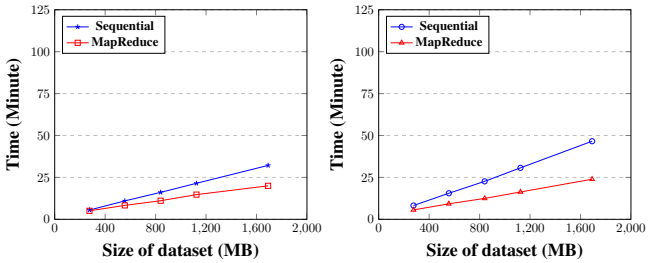
(a) Watermark Generation Time (b) Watermark Detection Time

Figure 7: Performance comparison between Bhattacharya et al.'s sequential [10] and its corresponding MapReduce Version for G = 50000

Bhattacharya and Cortesi, 2010 [11]: The grouping of tuples in this proposal follows similar steps as in [10]. However, unlike [10], the group-wise watermark generation is performed by extracting a fixed number of most significant bits (MSBs) and least significant bits (LSBs) from a selected field of each tuple in a group and by combining them together using concatenation operation.

Existing algorithm	No. of Tuples	Size in MB	Sequential		MapReduce		
			T_g	T_d	M	T_g	T_d
Bhatta-charya et al. [11]	5810120	276	5.56	8.23	2	5.08	5.61
	11620240	556	10.98	15.54	5	8.31	9.25
	17430360	840	16.09	22.65	7	11.10	12.48
	23240480	1124	21.49	30.71	9	14.73	16.31
	34860720	1692	32.16	46.59	13	19.99	23.92

Table IV: Results by sequential watermarking algorithm of Bhattacharya et al. [11] and by its corresponding MapReduce Version for G = 50000



(a) Watermark generation time (b) Watermark detection time

Figure 8: Performance comparison between Bhattacharya et al.'s sequential [11] and its corresponding Mapreduce Version for G = 50000

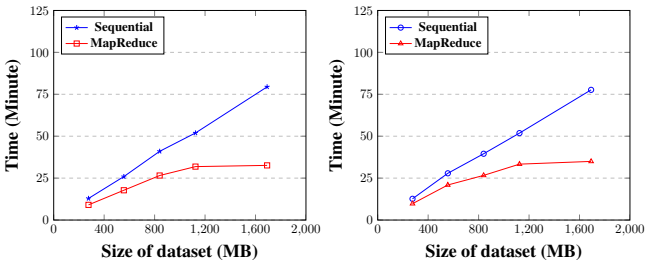
In our MapReduce-based version, like in the case of [10], the mapper computes the group-ID based on the primary key hash value and extracts the MSBs and LSBs from the selected attribute value in each tuple belonging to the dataset assigned to it. The mapper sends this group-ID along with the generated watermark bits for each tuple as key-value pair to the reducer. The reducer then combines these watermark bits according to the group-ID in order to generate watermark for each particular group.

The comparison of watermark generation and detection times in sequential and MapReduce framework are shown in Table IV and Figure 8.

Khan et al., 2013 [14]: This technique comprises of sub-watermark generation for digit count, length, and range of data values. In our MapReduce-based implementation, mappers compute digit frequency, length frequency and range frequency of data values in each tuple. The mapper then sends the tuples along with these frequencies as key-value pair to the reducer. On receiving inputs, the reducer computes digit sub-watermark, length sub-watermark and range sub-watermark, and it finally generates complete watermark by concatenating these three sub-watermarks.

Existing algorithm	No. of Tuples	Size in MB	Sequential		MapReduce		
			T_g	T_d	M	T_g	T_d
Khan et al [14]	5810120	276	12.88	12.67	2	9.05	9.75
	11620240	556	25.91	27.88	5	17.76	20.91
	17430360	840	40.98	39.54	7	26.55	26.63
	23240480	1124	51.93	51.84	9	31.87	33.34
	34860720	1692	79.43	77.65	13	32.60	34.97

Table V: Results by sequential watermarking algorithm of Khan et al. [14] and by its corresponding MapReduce Version for G = 50000



(a) Watermark generation time (b) Watermark detection time

Figure 9: Performance comparison between Khan et al.'s sequential [14] and its corresponding Mapreduce Version for G = 50000

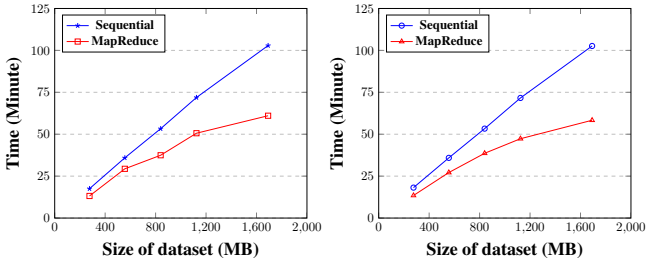
The results of watermark generation and detection times in sequential and MapReduce-based executions are shown in Table V and the comparison is depicted in Figure 9.

Camara et al., 2014 [13]: This distortion free technique is based on grouping of tuples into various square matrices of size $n \times n$, where n is the number of attributes. Individual watermark is first computed for each matrix as follows: The determinant value of the matrix and the different values of the minor of diagonal values in the matrix are computed, which are then concatenated to get the watermark for the matrix. The final watermark for the whole database follows the concatenation of matrix-wise watermarks.

Existing algorithm	No. of Tuples	Size in MB	Sequential		MapReduce		
			T_g	T_d	M	T_g	T_d
Camara et al[13]	115	276	17.52	18.16	2	14.40	13.47
	230	556	35.92	35.90	5	29.37	27.14
	345	840	53.30	53.34	7	37.46	38.63
	460	1124	71.90	71.63	9	50.60	47.32
	690	1692	102.89	102.63	13	61.03	58.33

Table VI: Results by sequential watermarking algorithm of Camara et al. [13] and by its corresponding MapReduce Version for G = 50000

Since the matrix operations are the primary reason of affecting the watermarking time, we have used two passes of computations in our MapReduce algorithm. In the first pass, the mappers compute the matrix-ID of each tuple to which it belongs and send them along with their associated tuple-values as key-value pair to the reducer. The reducer

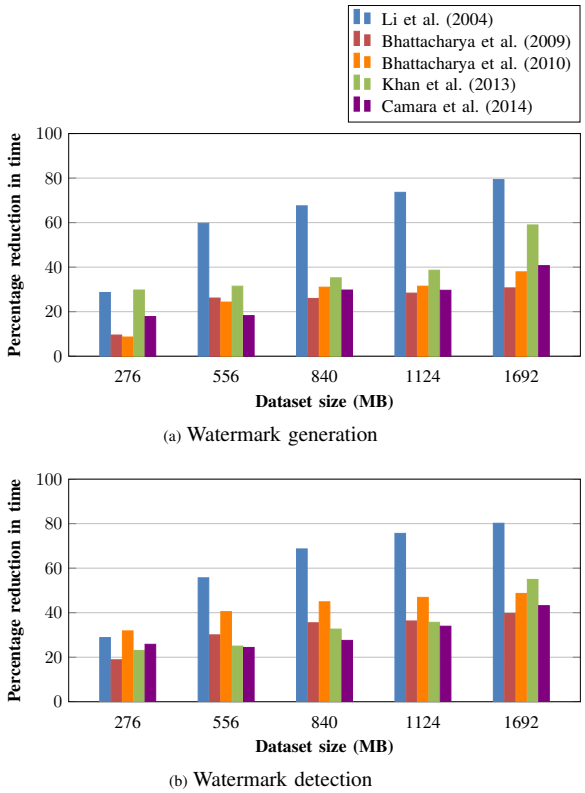


(a) Watermark generation time (b) Watermark detection time

Figure 10: Performance comparison between Camara et al.'s sequential [13] and its corresponding Mapreduce Version for G = 50000

forms square matrices based on the matrix-IDs of the tuples. The algorithm uses this information (that is, matrix-ID and values in each matrix) as input to the second pass. In the second pass, the mappers compute matrix-wise watermarks which are finally sent to the reducer to compute watermark for the whole database.

The comparison of watermark generation and detection times are depicted in Table VI and Figure 10. Note that in our experiment we have considered square matrices of size 5×5.



(a) Watermark generation

(b) Watermark detection

Figure 11: Percentage of time-reduction

Common observations: All the distortion free watermark-

ing techniques referred in this section have some common observations when we execute their MapReduce versions in Hadoop framework. These observations are as follows:

- The watermark generation and detection times reduce significantly in MapReduce as compared to their sequential executions.
- As intuitive, the performance improves as the number of mappers increases.
- Let t_s and t_m be the time required in Sequential and MapReduce-based computations respectively. Let us define the percentage of time-reduction as follows: % of time-reduction=$\frac{t_s - t_m}{t_s} \times 100$. This is observed that the percentage of time-reduction in watermark generation and detection increases as we increase the size of the dataset. This is depicted in Figure 11. Observe that the percentage of time-reduction is highest in case of Li et al. [4].

VI. Conclusions and Future Works

In this paper, we introduced an efficient watermarking approach for large relational databases adapting the potential of MapReduce paradigm. We have focused on distortion-free watermarking and implemented the existing algorithms in both sequential and MapReduce form. The experimental results showed that there is significant reduction in watermark generation as well as detection time in MapReduce. We observed that the percentage reduction of time from sequential to MapReduce increases with the increase of data size. To the best of our knowledge, this is the first work towards big relational database watermarking. As future works, we aim to extend this proposal to distortion-based watermarking and to focus on watermarking of bigdata considering other key characteristics.

References

[1] W. Raghupathi and V. Raghupathi, "Big data analytics in healthcare: promise and potential," *Health Information Science and Systems*, vol. 2, no. 1, p. 3, 2014.

[2] P. Russom, "Managing big data," *TDWI Best Practices Report, TDWI Research*, pp. 1–40, 2013.

[3] R. Halder, S. Pal, and A. Cortesi, "Watermarking techniques for relational databases: Survey, classification and comparison." *J. UCS*, vol. 16, no. 21, pp. 3164–3190, 2010.

[4] Y. Li, H. Guo, and S. Jajodia, "Tamper detection and localization for categorical data using fragile watermarks," in *Proceedings of the 4th ACM workshop on Digital rights management*. ACM, 2004, pp. 73–82.

[5] R. Agrawal and J. Kiernan, "Watermarking relational databases," in *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 155–166.

[6] R. Halder and A. Cortesi, "A persistent public watermarking of relational databases," in *International Conference on Information Systems Security*. Springer, 2010, pp. 216–230.

[7] H. Guo, Y. Li, A. Liu, and S. Jajodia, "A fragile watermarking scheme for detecting malicious modifications of database relations," *Information Sciences*, vol. 176, no. 10, pp. 1350–1378, 2006.

[8] Y. Li, "Database watermarking: A systematic view," in *Handbook of database security*. Springer, 2008, pp. 329–355.

[9] M. Kamran, S. Suhail, and M. Farooq, "A robust, distortion minimizing technique for watermarking relational databases using once-for-all usability constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 12, pp. 2694–2707, 2013.

[10] S. Bhattacharya and A. Cortesi, "A generic distortion free watermarking technique for relational databases," in *International Conference on Information Systems Security*. Springer, 2009, pp. 252–264.

[11] S. Bhattacharya and A. Cortesi, "Distortion-free authentication watermarking," in *International Conference on Software and Data Technologies*. Springer, 2010, pp. 205–219.

[12] S. Rani, P. Kachhap, and R. Halder, "Data-flow analysis-based approach of database watermarking," in *Advanced Computing and Systems for Security*. Springer, 2016, pp. 153–171.

[13] L. Camara, J. Li, R. Li, and W. Xie, "Distortion-free watermarking approach for relational database integrity checking," *Mathematical problems in engineering*, vol. 2014, 2014.

[14] A. Khan and S. A. Husain, "A fragile zero watermarking scheme to detect and characterize malicious modifications in database relations," *The Scientific World Journal*, vol. 2013, 2013.

[15] R. Halder and A. Cortesi, "Persistent watermarking of relational databases," in *Proceedings of the IEEE International Conference on Advances in Communication, Network, and Computing (CNC10)*, 2010, pp. 46–52.

[16] A. Hamadou, X. Sun, L. Gao, and S. A. Shah, "A fragile zero-watermarking technique for authentication of relational databases," *International Journal of Digital Content Technology and its Applications*, vol. 5, no. 5, 2011.

[17] S. Bhattacharya and A. Cortesi, "A distortion free watermark framework for relational databases." in *ICSOFT (2)*, 2009, pp. 229–234.

[18] M.-R. Xie, C.-C. Wu, J.-J. Shen, and M.-S. Hwang, "A survey of data distortion watermarking relational databases," *International Journal of Network Security*, vol. 18, no. 6, pp. 1022–1033, 2016.

[19] V. Pournaghshband, "A new watermarking approach for relational data," in *Proceedings of the 46th annual southeast regional conference on XX*. ACM, 2008, pp. 127–131.

[20] S. Rani, D. K. Koshley, and R. Halder, "A watermarking framework for outsourced and distributed relational databases," in *International Conference on Future Data and Security Engineering*. Springer, 2016, pp. 175–188.

[21] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[23] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons, 2007.