Adapting MapReduce for Efficient Watermarking of Large Relational Datasets

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 and relational datasets using existing database watermarking techniques are highly inefficient, and even impractical in the situations when periodic rewatermarking of datasets 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 structured relational datasets. 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 embedding and detection time w.r.t. existing works in the literature.**

Index Terms— **Large Relational Datasets, Watermarking, MapReduce**

I. Introduction

Dealing with large data-sets 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 [6], 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 [4] in nature and are stored in the form of relational database. 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* [20].

Database watermarking has been introduced as one of the most effective solutions to address the above mentioned threats [15], [18], [21], [23]. 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 presence of any possible attacks. The former phase is known as *Embedding* phase, whereas the later phase is known as *Detection* or *Verification* phase.

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

A. Related Works

Watermarking in case of relational database was first proposed by Agrawal et al. in [18]. Subsequently many works have been proposed in this direction [7], [8], [10], [12], [14], [15], [19], [21], [23]. Broadly the watermarking of relational databases can be categorized into *distortion-based* [7], [12], [18], [21], [23] and *distortion-free* [8], [10], [14], [15], [19]. In general, distortion-based techniques embed some watermark keeping in mind the usability of the 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 [20]. A recent survey by Xie et al. is reported in [24] giving special attention to the distortion-based watermarking.

The existing distortion-based techniques are based on random bit flipping [12], [18], fake tuple insertion [2], random bit insertion [7], [23], etc. On the other hand, the existing distortion-free techniques are based on tuple reordering [15], binary image generation [8], [10], generation of local characteristics like range, digit and length frequencies [19], matrix operations [14], etc. The first proposal to address watermarking of distributed databases is proposed in [13] that supports database outsourcing and hybrid partitioning.

B. Motivation and Contributions

Although all the existing techniques work well in watermarking of small datasets, they are highly inefficient in case of watermarking of large relational datasets. This is observed that the existing watermarking techniques involve various computations like MD5-based hash calculation [3], partitioning, group-based watermark generation, etc. The sequential processing of those operations on all database-tuples makes these techniques highly inefficient and even impractical for very large databases. This is also true in the situations where databases go through frequent updation and periodic re-watermarking after certain time-frame is necessary.

To ameliorate this performance bottleneck, this paper presents an efficient way of watermarking of large scale relational datasets 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 paradigm 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.
- We present a case study describing the design of MapReduce variants of existing sequential form of watermarking algorithms.
- We perform experiments on a benchmark dataset. The experimental results are 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 elaborates our proposed approach of database watermarking using MapReduce. A case study has been discussed in section IV. The experimental results have been analyzed in section V. Finally we conclude our work in section VI.

II. MapReduce algorithm at a glance

Jeffrey et al. [5] introduced MapReduce programming model as highly efficient and reliable solution to process large datasets 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 map function will then act 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.

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 datasets. 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.

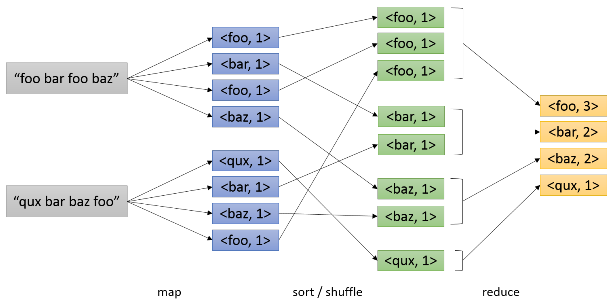


Fig. 1: Word frequency computation using MapReduce

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 $W = ||W_g$
```

Fig. 2: Watermark Generation algorithm using MapReduce

As observed, the existing distortion-free watermarking techniques [8], [14], [15], [19], in general, mainly focus on generation of the watermark from the content of the database. Precisely they consist of two prime phases: *(i)* Partitioning of data-set into groups [14], [15], [23], and *(ii)* Watermark generation from each group. The obtained watermarks in each group may finally be combined to generate watermark for the whole database.

a) ***Watermark Generation using MapReduce:*** While designing MapReduce algorithm for watermarking of large datasets, our primary focus is on "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 reduction of 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 of the tuple values [18]. This group-ID along with the tuple itself act as an intermediate key-value pair emitted as the output from the mapper. Observe that this way multiple mappers process large scale data-set in parallel, partitioning them in an efficient way.

The outputs of all MAPPERS are then fed to a REDUCER which finally computes the watermark W for whole database by computing and combining together (represented by $||$ operation) group-wise watermarks W_g. This 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.

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

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
```

Fig. 3: Algorithm for Watermark Detection using MapReduce

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

As like embedding phase, during detection, on assigning each MAPPER to a fragment $R^{'}$ of the large suspicious dataset, the MAP function applies the same partition-function f which is already used in the embedding phase, on each tuple t and computes its group ID g_{id}. The MAP function emits $\langle g_{id}, t\rangle$ as intermediate key-value pairs which is used as the input to the REDUCER. The REDUCE function then computes the watermark $W_g^{'}$ for each group. The complete watermark $W^{'}$ for database is obtained by following similar operations as in the embedding phase. The detection phase concludes a success, if $W^{'}$ and the embedded watermark W are same.

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

This section considers the watermarking algorithm proposed by Li et al. in [15] and describe how to design its MapReduce version making it suitable for large scale database watermarking. The notations and parameters used in the algorithms are depicted in Table I.

Algorithms 5, 6 and 7 in Figure 4(a) depict the sequential version of the watermarking algorithm by Li et al. [15]. The Algorithm 5 initially computes tuple hash and primary key hash for each tuple in steps 4 and 5 respectively. In steps 6 and 7, 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 watermark generation absorb a significant amount of time. This makes the algorithm highly inefficient in case of big relational database.

ω	number of tuples in the relation
h_i	tuple hash of i^{th} tuple in relation
${h_i}^P$	primary key hash of i^{th} tuple in the relation
r_i	i^{th} tuple
$r_i.A_j$	the j^{th} attribute of the i^{th} tuple
g	number of groups after partitioning
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

The corresponding MapReduce algorithms for partitioning and watermark generation for each group are depicted in Figure 4(b). In Algorithm 8, the mapper computes the tuple hash and primary key hash for each tuple (steps 3 and 4 respectively) and computes the group to which a tuple belongs (step 5). Mapper then emits group-id and ⟨tuple, tuple hash, primary key hash⟩ as intermediate key-value pairs in step 6. These key-value pairs act as the input for Reducer depicted in Algorithm 9. The reducer first sorts each tuple in the group according to their primary key hash value in steps 3 and 4, and computes the group hash in step 6. 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 9-11 of Algorithm 9.

Similarly, the MapReduce version of the sequential detection algorithm by Li et al. is depicted in Figure 5. This way any existing watermarking technique can be converted into its equivalent MapReduce form.

V. Experimental Analysis

We have searched the literature exhaustively and identified five significant proposals [8], [10], [14], [15], [19] for our experiments. We have implemented all the five algorithms in both sequential and MapReduce framework using java. We have performed the experiments on a server equipped with Intel Xeon processor, 128 GB RAM, 2.4 GHz clock speed and linux operating system installed with the hadoop[1] framework, version 2.7.0. We have performed the experiments on large relational datasets that are obtained by modifying the Forest Cover Type dataset[2] into a large dataset.

[1]installation details at `http://hadoop.apache.org/`

[2]`https://kdd.ics.uci.edu/databases/covertype/covertype.html`

Algorithm 5 Li2004 Watermark generation

```
for all $k \in [1, g], q_k = 0$ do
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 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
$H = \texttt{HASH}(K, h_1{}', h_2{}', ..., h_{qk}{}')$
$W$ = $\texttt{ExtractBits}(H, qk/2)$ // See algorithm 7
for $i = 1, i < k, i = i + 2$ 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
end for
```

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

```
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 $W$
```

(a) Watermark generation in Sequential

Algorithm 8 Watermark generation using MapReduce: Mapper

```
procedure MAP ($R$)
    for $i$ = 1 to $\omega$ 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 ($g_{id}, list := [r_1, r_2, ..., r_k]$)
    $k = |list|$
    for $i$ = 1 to $k$ do
        Sort tuples in ascending order according to $h_i{}^P$.
    end for
    $Group\_hash$ = $\texttt{HASH}(K, h_1, ..., h_k)$ // group hash
    $W$ = $\texttt{ExtractBits}(Group\_hash, k/2)$ // algorithm 10
    $emit(g_{id}, W)$
    for $i = 1, i < k, i = i + 2$ 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
    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) Watermark generation in MapReduce

Fig. 4: Algorithm for embedding in sequential [15] and MapReduce framework

Algorithm 11 Li2004 Detection

```
for all $k \in [1, g], q_k = 0$ do
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 Watermarking 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 $i = 1, i < k, i = i + 2$ 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
end for
```

(a) Detection in Sequential

Algorithm 13 Detection using MapReduce: Mapper

```
procedure MAP ($R$)
    for $i$ = 1 to $\omega$ 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 Detection using MapReduce: Reducer

```
procedure REDUCE ($g_{id}, list := [r_1, r_2, ..., r_k]$)
    $k = |list|$
    for $i$ = 1 to $k$ do
        Sort tuples in ascending order according to $h_i{}^P$.
    end for
    $Group\_hash$ = $\texttt{HASH}(K, h_1, ..., h_k)$ // group hash
    $W = \texttt{ExtractBits}(Group\_hash, k/2)$
    $emit(g_{id}, W)$
    for $i = 1, i < k, i = i + 2$ 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
    end for
end procedure
```

(b) Detection in MapReduce

Fig. 5: Algorithm for detection in sequential [15] and MapReduce framework

Li et al, 2004 [15]: The sequential and MapReduce algorithm for this technique has been discussed in section IV. In these algorithms, the database is partitioned into various groups and watermark is generated for each group separately.

Existing algorithm	No. of tuples	Size (in MB)	Sequential		MapReduce		
			Embed time (minute)	Detect time (minute)	No. of mapper	Embed time (minute)	Detect time (minute)
Li et al [15]	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	566.83	13	31.72	28.61

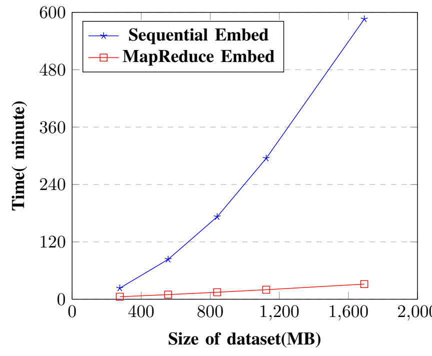
(a) Watermarking results for number of groups 10000

Existing algorithm	No. of tuples	Size (in MB)	Sequential		MapReduce		
			Embed time (minute)	Detect time (minute)	No. of mapper	Embed time (minute)	Detect time (minute)
Li et al [15]	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

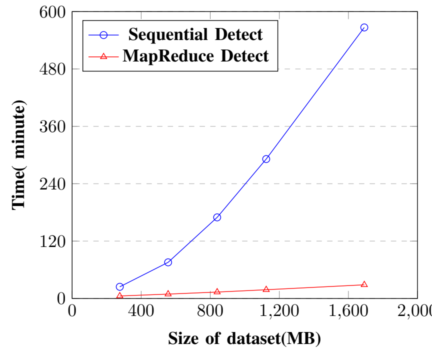
(b) Watermarking results for number of groups 50000

TABLE II: Result for sequential [15] and MapReduce

We have compared the embedding and detection time in both sequential as well as Hadoop MapReduce framework. The watermarking results for sequential and MapReduce framework for total number of groups 10000 and 50000 are depicted in Table II.

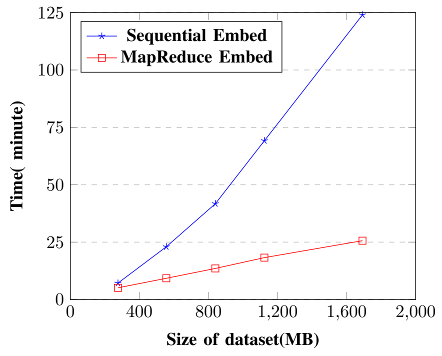


(a) Time comparison during watermark embedding for group size 10000

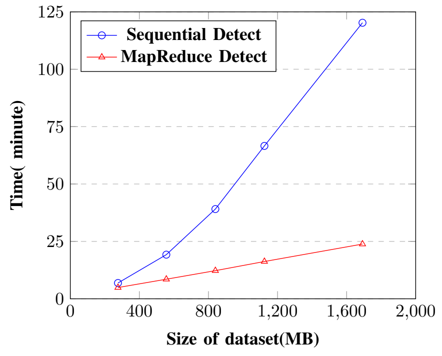


(b) Time comparison during watermark detection for group size 10000

Fig. 7: Time comparison for group size 10000 in [15]



(a) Time comparison during watermark embedding for group size 50000



(b) Time comparison during watermark detection for group size 50000

Fig. 8: Time comparison for group size 50000 in [15]

The time comparison between sequential and MapReduce framework for embedding and detection in case of group size 10000 is depicted in Figure 7(a) and Figure 7(b) respectively. Similarly, for group size 50000 the time comparison is shown in Figures 8(a) and 8(b). The observations are as follows:

- The embedding time and detection time is reduced significantly in MapReduce framework as compared to the sequential one.
- The embedding and detection time decreases significantly on increasing the number of groups.

Bhattacharya and cortesi, 2009 [8]: Authors have partitioned the dataset according to the primary key hash and the watermark for each group is computed separately. In our MapReduce framework, Mapper computes most significant bits(MSB) of all attributes for each tuple. The group to which a tuple belongs is calculated as primary key hash modulo number of groups. Mapper sends the group id and MSBs of a tuple as key-value pair to the reducer. Reducer collects the MSBs for tuples belonging to a group which act as the watermark for that group.

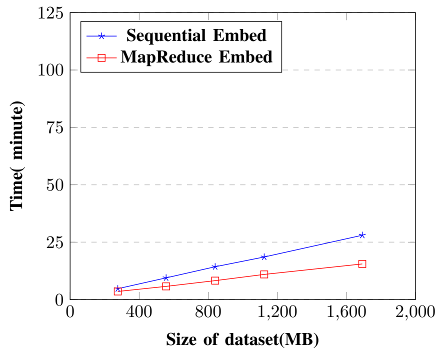
The comparison of watermark embedding and detection time in sequential and MapReduce framework is depicted in table III and figure 9.

Bhattacharya et al, 2010 [10]: Authors here partitioned the dataset into various groups and the watermark for each group is computed separately.

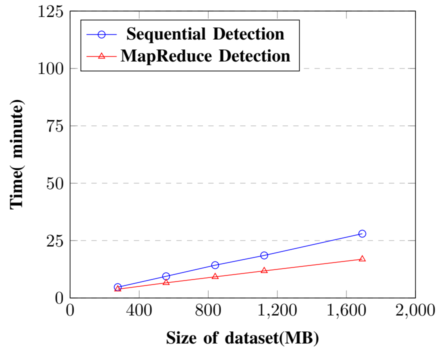
In our MapReduce framework, for each value in the tuple,

Existing algorithm	**No. of tuples**	**Size (in MB)**	**Sequential**		**MapReduce**		
			Embed time (minute)	Detect time (minute)	No. of mapper	Embed time (minute)	Detect time (minute)
Bhatta-charya et al [8]	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	16.9

TABLE III: Result for sequential [8] and MapReduce



(a) Time comparison during watermark embedding



(b) Time comparison during watermark detection

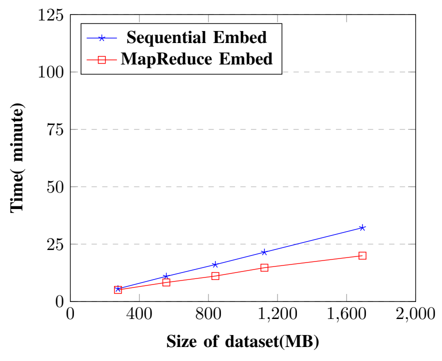
Fig. 9: Watermark embedding and detection time in [8]

Existing algorithm	**No. of tuples**	**Size (in MB)**	**Sequential**		**MapReduce**		
			Embed time (minute)	Detect time (minute)	No. of mapper	Embed time (minute)	Detect time (minute)
Bhatta-charya et al [10]	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	19.97	23.92

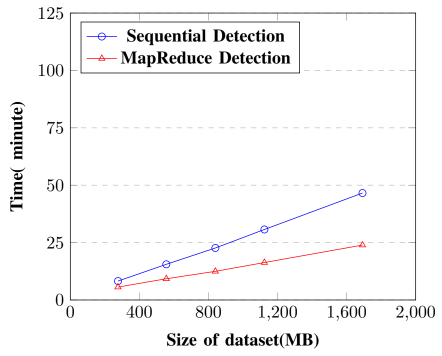
TABLE IV: Result for sequential [10] and MapReduce

Mapper computes the watermark value as the concatenation of m number of MSBs and n number of LSBs such that the sum of m and n is 8. The group to which a tuple belongs is calculated as primary key hash modulo number of groups. Mapper sends the group id and watermark of the tuple as key-value pair to the reducer. Reducer collects the MSBs for tuples belonging to a group which act as the watermark for that group. The comparison of watermark embedding and detection time in sequential and MapReduce framework is depicted in Table IV and Figure 10.

Khan et al, 2013 [19]: This technique comprises of sub-



(a) Time comparison during watermark embedding



(b) Time comparison during watermark detection

Fig. 10: Watermark embedding and detection time in [10]

watermark generation for digit count, length, and range of data values. In our MapReduce framework, Mapper computes digit frequency, length frequency and range frequency of all data values in each tuples.

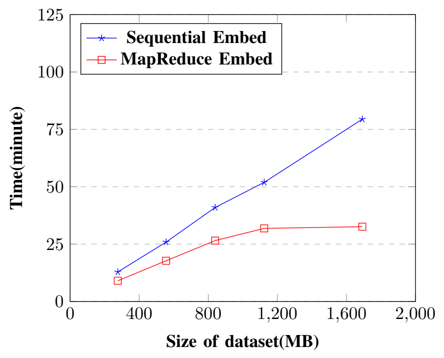
Existing algorithm	**No. of tuples**	**Size (in MB)**	**Sequential**		**MapReduce**		
			Embed time (minute)	Detect time (minute)	No. of mapper	Embed time (minute)	Detect time (minute)
Khan et al [19]	5810120	276	12.88	12.67	2	9.05	9.75
	11620240	556	25.91	27.88	5	17.76	20.91
	17430360	840	40.98	39.54	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: Result for sequential [19] and MapReduce

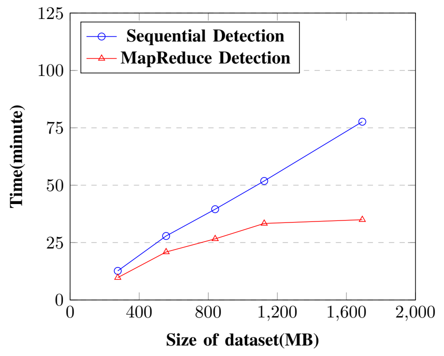
Mapper sends the tuple along with these frequencies as key-value pair to the reducer. Reducer computes the digit sub-watermark, length sub-watermark and range sub-watermark and finally concatenates the three sub-watermarks to generate the complete watermark. The comparison of watermark embedding and detection time in case of sequential and MapReduce framework is depicted in Table V and Figure 11.

We can observe from Figure 11 that the time reduces significantly in MapReduce as compared to sequential.

Camara et.al, 2014 [14]: This distortion free technique is based on grouping the tuples into various square matrices of size $n \times n$, where n is number of attributes. The total number of such square matrix groups is calculated by dividing number of tuples from number of attributes. For large n, its difficult to compute the determinant and minor of the matrix.

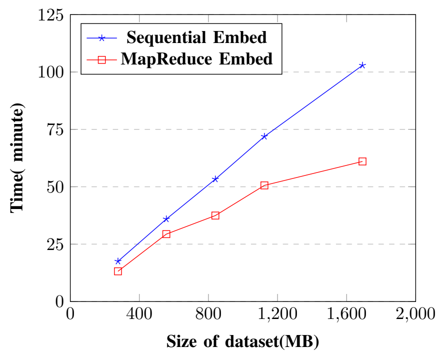


(a) Time comparison during watermark embedding

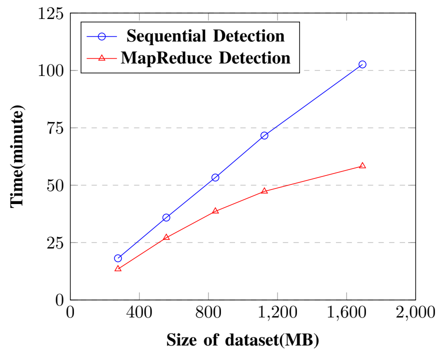


(b) Time comparison during watermark detection

Fig. 11: Watermark embedding and detection time in [19]



(a) Time comparison during watermark embedding



(b) Time comparison during watermark detection

Fig. 12: Watermark embedding and detection time in [14]

Therefore in our experiment, we have considered 5 attributes so that the square matrices are 5×5.

Since matrix operation in a single reducer will be time consuming, we have used two MapReduce frameworks. The grouping is done on mapper of first MapReduce framework based on primary key hash value. The reducer collects all the tuples that belong to a particular group and then sends these tuples to the second MapReduce framework.

Existing algorithm	**No. of tuples (in lakh)**	**Size (in MB)**	**Sequential**		**MapReduce**		
			Embed time (minute)	Detect time (minute)	No of mapper	Embed time (minute)	Detect time (minute)
Camara et al[14]	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: Result for sequential [14] and MapReduce

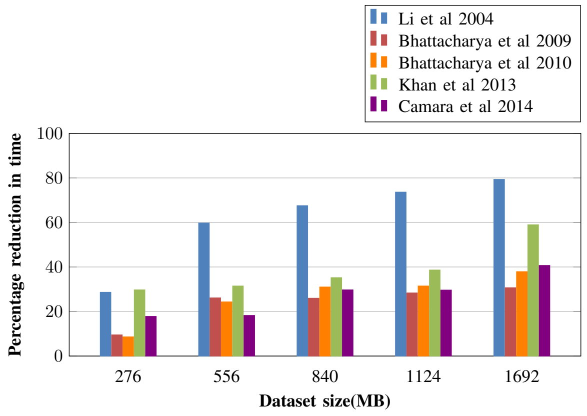
The mapper of second MapReduce framework calculates the determinant of that square matrix group and a set of values obtained from minor of all the diagonal elements. The determinant and minor are then concatenated to generate the watermark for that group. Mapper sends the group number and the corresponding watermark as intermediate key-value pairs to reducer. The complete watermark is computed by reducer by concatenating all the group watermarks. The comparison of watermark embedding and detection time in case of sequential and MapReduce is depicted in Table VI and Figure 12.

Common observations: All the distortion free watermarking techniques discussed above in this section have some common observations when we execute the watermarking algorithms in MapReduce framework. These observations are as follows:

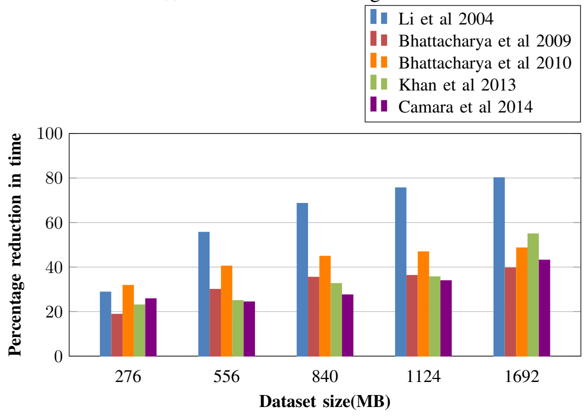
- The watermark embedding and detection time reduces significantly in MapReduce framework as compared to the sequential execution.
- In case of MapReduce, the number of mapper available for execution increases with increase in size of dataset and all the mappers execute parallely so it works better as we increase the size of dataset.
- Let t_s and t_m be the time required in Sequential and MapReduce-based computation 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 embedding and detection time increases as we increase the size of the dataset shown in Figure 13. The percentage reduction in time is highest in case of Li et al. [15] as depicted in Figure 13.

VI. Conclusions and Future Works

In this paper, we proposed a novel database watermarking technique using MapReduce framework. We have focused on distortion-free watermarking and implemented the existing algorithms both in sequential as well as in MapReduce framework. The experimental results show that there is significant reduction in embedding as well as detection time in MapReduce framework. We observe that the percentage



(a) Watermark embedding



(b) Watermark detection

Fig. 13: Percentage reduction in time for embedding and detection

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. The future work aims to extend this proposal in case of distortion-based database watermarking and also considering other characteristics of bigdata.

References

[1] Dean, Jeffrey, and Sanjay Ghemawat.: "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.
[2] Pournaghshband, Vahab.: "A new watermarking approach for relational data." Proceedings of the 46th annual southeast regional conference on XX. ACM, 2008.
[3] Schneier, Bruce. Applied cryptography: protocols, algorithms, and source code in C. john wiley and sons, 2007.
[4] Russom, Philip.: "Managing big data." TDWI Best Practices Report, TDWI Research (2013): 1-40.
[5] Dean, Jeffrey, and Sanjay Ghemawat.: "MapReduce: a flexible data processing tool." Communications of the ACM 53.1 (2010): 72-77.
[6] Raghupathi, Wullianallur, and Viju Raghupathi.: "Big data analytics in healthcare: promise and potential." Health Information Science and Systems 2.1 (2014): 3.
[7] Kamran, Muhammad, Sabah Suhail, and Muddassar Farooq.: "A robust, distortion minimizing technique for watermarking relational databases using once-for-all usability constraints." IEEE Transactions on Knowledge and Data Engineering 25.12 (2013): 2694-2707.
[8] Bhattacharya, Sukriti, and Agostino Cortesi.: "A generic distortion free watermarking technique for relational databases." International Conference on Information Systems Security. Springer Berlin Heidelberg, 2009.
[9] Bhattacharya, Sukriti, and Agostino Cortesi.: "A Distortion Free Watermark Framework for Relational Databases." ICSOFT (2). 2009.
[10] Bhattacharya, Sukriti, and Agostino Cortesi.: "Distortion-Free Authentication Watermarking." International Conference on Software and Data Technologies. Springer Berlin Heidelberg, 2010.
[11] Hamadou, Ali, et al.: "A fragile zero-watermarking technique for authentication of relational databases." International Journal of Digital Content Technology and its Applications 5.5 (2011).
[12] Rani, Sapana, Preeti Kachhap, and Raju Halder.: "Data-flow analysis-based approach of database watermarking." Advanced Computing and Systems for Security. Springer India, 2016. 153-171.
[13] Rani, Sapana, Dileep Kumar Koshley, and Raju Halder.: "A Watermarking Framework for Outsourced and Distributed Relational Databases." International Conference on Future Data and Security Engineering. Springer International Publishing, 2016.
[14] Camara, Lancine, et al.: "Distortion-free watermarking approach for relational database integrity checking." Mathematical problems in engineering 2014 (2014).
[15] Li, Yingjiu, Huiping Guo, and Sushil Jajodia.: "Tamper detection and localization for categorical data using fragile watermarks." Proceedings of the 4th ACM workshop on Digital rights management. ACM, 2004.
[16] Xie, Ming-Ru, et al.: "A Survey of Data Distortion Watermarking Relational Databases." International Journal of Network Security 18.6 (2016): 1022-1033.
[17] Khanna, Sanjeev, and Francis Zane.: "Watermarking maps: hiding information in structured data." Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2000.
[18] Agrawal, Rakesh, and Jerry Kiernan.: "Watermarking relational databases." Proceedings of the 28th international conference on Very Large Data Bases. VLDB Endowment, 2002.
[19] Khan, Aihab, and Syed Afaq Husain.: "A fragile zero watermarking scheme to detect and characterize malicious modifications in database relations." The Scientific World Journal 2013 (2013).
[20] Halder, Raju, Shantanu Pal, and Agostino Cortesi.: "Watermarking Techniques for Relational Databases: Survey, Classification and Comparison." J. UCS 16.21 (2010): 3164-3190.
[21] Halder, Raju, and Agostino Cortesi.: "A persistent public watermarking of relational databases." International Conference on Information Systems Security. Springer Berlin Heidelberg, 2010.
[22] Halder, Raju, and Agostino Cortesi.: "Persistent watermarking of relational databases." Proceedings of the IEEE International Conference on Advances in Communication, Network, and Computing (CNC 2010), October. 2010.
[23] Guo, Huiping, et al.: "A fragile watermarking scheme for detecting malicious modifications of database relations." Information Sciences 176.10 (2006): 1350-1378.
[24] Xie, Ming-Ru, et al.: "A Survey of Data Distortion Watermarking Relational Databases." International Journal of Network Security 18.6 (2016): 1022-1033.
[25] Pavlo, Andrew, et al.: "A comparison of approaches to large-scale data analysis." Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. ACM, 2009.