Institute of Information Engineering, Chinese Academy of Sciences

A Scalable Distributed Private Stream Search System

peng zhang and yan li

CNCERT/CC

Outline of The Presentation

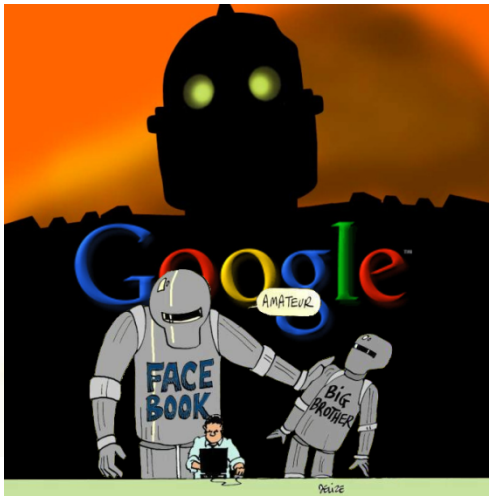
- Motivation
- Research Content
  - Private Stream Searching Model
  - Distributed Processing Architecture
  - Bitmap-based Compression Storage
- Conclusions

1. Motivation

- Private Information Retrieval (PIR)
  - allows a user to retrieve an item from a server in possession of a database without revealing which item is retrieved

- Query can be seen anywhere
  - Searching Engine
  - E-Business
  - News Feed

The big brother is watching you
when you type in a query



Privacy leaks in the query

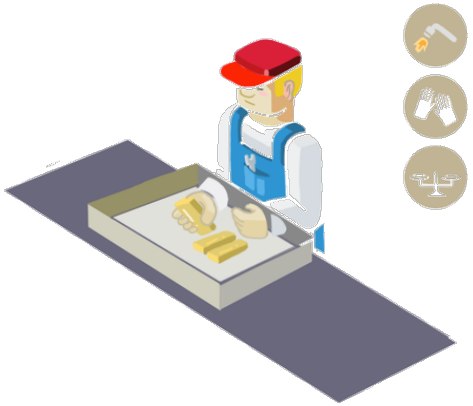
- Query input: keyword
- Keyword: reflect user's intention
- Mining the user query can be seen everywhere
  - query association analysis
  - advertising promotion
  - personalized search criteria

CNCERT/CC

1. Motivation

- How to implement private information search?
  - offline query：download the entire resource to the client machine and perform the search locally is typically infeasible
    - large size of the data
    - limited bandwidth
    - unwillingness to disclose the entire resource
  - homomorphic encryption
    - principle: It is allowed to perform a specific algebraic operation on the encryption results, and get the plain results through decryption.
    - significance：It is a solution to delegate the data and the operation to the third party, for example, for a variety of cloud computing applications

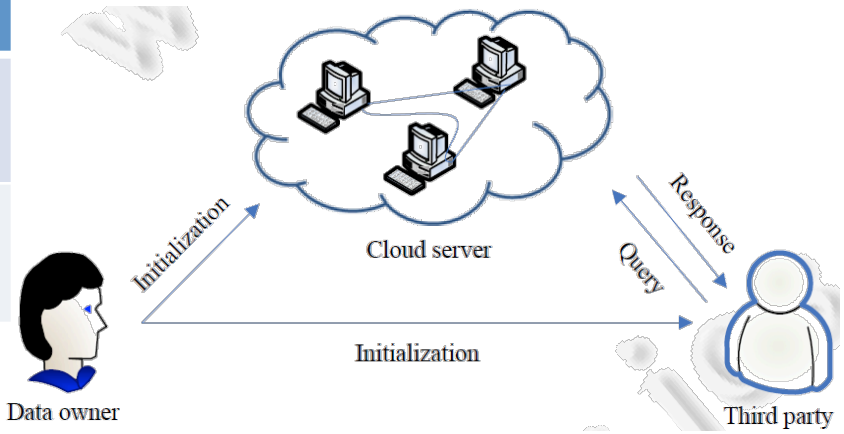
Box	• encryption algorithm
Lock	• user key
Put the bullion into the box and lock the box	• encrypt the data with the homomorphic encryption
Handling	• directly process the encryption results
Unlock	• Decrypt the processing results and get the plain results



1. Motivation

- Privacy Protection at the Age of BigData
  - “4V” character
  - privacy leak risk

	Data Owner	Third Party	Application
Storage	Data confidentiality, integrity		Cloud Storage
Computing	Data confidentiality, integrity	Data confidentiality	Searching Engine, Medical record analysis



Research Area	Example
General privacy protection technology	Perturbation, Randomization, Swapping, Encryption
Oriented-data mining privacy protection technology	Association Rule Mining, Classification, Clustering
Data publishing principle based on privacy protection	k-anonymity, l-diversity, m-Invariance, t-Closeness
privacy protection algorithm	Anonymized Publication, Anonymization with High Utility

- **Homomorphic encryption inevitably brings about additional time consumption, so how to ensure the processing efficiency**
  - Volume
  - Velocity
  - Variety

CNCERT/CC

1. Motivation

- Our Works
  - Develop a Scalable Distributed Private Stream Searching System
  - Interdisciplinary Study of Distributed System and Information Security

Private Stream Searching Model

Distributed Processing Architecture

Bitmap-based Compression Storage

Contributions

- The First Scalable Distributed Private Steam Searching System

CNCERT/CC

2. Related Works

Searching on encrypted data

- the data is hidden from the server

Private Information Retrieval

- the data is known to the server and the client's queries must remain hidden;
- communication dependent on the size of the entire database;
- database as a long bitstring and the queries as indices of bits to be retrieved

Private Stream Searching

- the data is known to the server and the client's queries must remain hidden;
- communication independent of the size of the stream or database;
- queries based on a search for keywords within text

2. Related Works

Private Stream Search's Performance Bottleneck

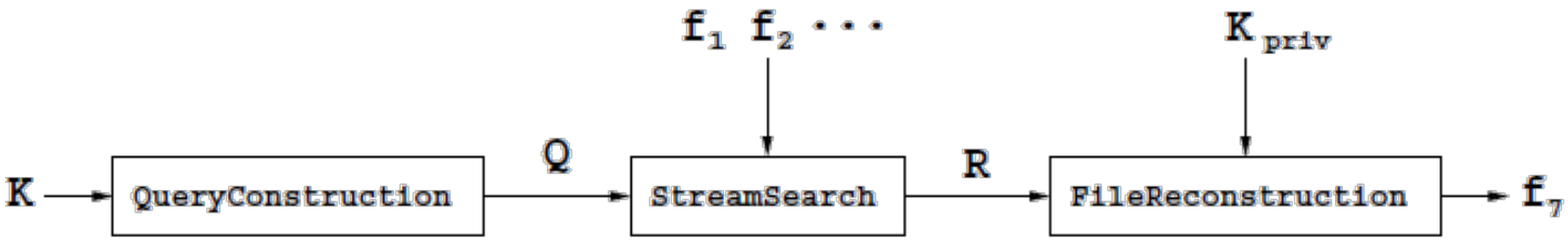
- A large number of multiplication and module operations
- Time consumption in client and server
  - server: second/file encrypted query
  - client: minute/file decrypted

How to implement efficient private stream search at the age of Big Data?

Stream Processing's Security Bottleneck

- Storm, MapUpdate , D-Streams , RAMCloud and Spark are excellent solutions to support high-performance query, however, they can't hide the search criteria

3. Private Stream Search Model



Private Stream Search Model

3.1 Client's QueryConstruction Procedure

A public dictionary of potential keywords

$$D = \{w_1, w_2, \ldots, w_{|D|}\}$$

is assumed to be available. Constructing the encrypted query for some disjunction of keywords $K \subseteq D$. The client generates a key pair, then for each $i \in 1, \ldots, |D|$, defines $q_i = 1$ if $w_i \in K$ and $q_i = 0$ if $w_i \notin K$. The values $q_1, q_2, \ldots, q_{|D|}$ are encrypted and put in the array $Q = (E(q_1), E(q_2), \ldots, E(q_{|D|}))$, which forms the final encrypted query.

3. Private Stream Search Model

3.2 Server's `StreamSearch` Procedure

State The server must maintain three buffers as it processes the files in its stream. These buffers are hereafter referred to as the *data buffer*, the *c-buffer*, and the *matching-indices buffer* and denoted F, C, and I respectively.

The data buffer will store the matching files in an encrypted form which can then be used by the client to reconstruct the matching files. In particular, the data buffer will contain a system of linear equations in terms of the content of the matching files in an encrypted form. This system of equations will later be solved by the client to obtain the matching files.

The c-buffer stores in an encrypted form the number of keywords matched by each matching file. We call the number of keywords matched for a file the *c-value* of the file. The c-buffer will be used in reconstruction of the matching files from the data buffer by the client. As in the case of the data buffer, the c-buffer stores its information in the form of a system of linear equations. The client will later solve the system of linear equations to reconstruct the c-values.

The matching-indices buffer is an encrypted Bloom filter that keeps track of the indices of matching files in an encrypted form. More precisely, the matching-indices buffer will be a encrypted representation of some set of indices $\{\alpha_1, \ldots, \alpha_r\}$ where $\{\alpha_1, \ldots, \alpha_r\} \subseteq \{1, \ldots, t\}$. Here r is the number of files which end up matching the query.

3. Private Stream Search Model

Processing Steps To process the ith file f_i, the server takes the following steps.
Step 1: Compute encrypted c-value. First, the server looks up the query array entry $Q[j]$ corresponding to each word w_j found in the file. The product of these entries is then computed. Due to the homomorphic property of the Paillier cryptosystem, this product is an encryption of c-value of the file, i.e., the number of distinct members of K found in the file. That is,

$$\prod_{w_j \in W_i} Q[j] = E\left(\sum\nolimits_{w_j \in W_i} q_j\right) = E\left(c_i\right)$$

where W_i is the set of distinct words in the ith file and c_i is defined to be $|K \cap W_i|$. Note in particular that $c_i \neq 0$ if and only if the file matches the query.

Step 2: Update data buffer. The server computes $E\left(c_i f_i\right)$ using the homomorphic property of the Paillier cryptosystem.

$$E\left(c_i\right)^{f_i} = E\left(c_i f_i\right) = \begin{cases} E\left(c_i f_i\right) & \text{if } f_i \text{ matches the query} \\ E\left(0\right) & \text{otherwise.} \end{cases}$$

The server multiplies the value $E\left(c_i f_i\right)$ into a subset of the locations in the data buffer according to the following procedure. Let G be a family of pseudo-random functions that map $\mathbb{Z} \times \mathbb{Z}$ to $\{0,1\}$. Randomly select $g \xleftarrow{R} G$ (this should be done once upon initialization and the same g used for all files). The algorithm multiplies $E\left(c_i f_i\right)$ into each location j in the data buffer where $g(i,j) = 1$.

3. Private Stream Search Model

Step 3: Update c-buffer. The value $E(c_i)$ is multiplied into each of the locations in the c-buffer in a similar fashion as $E(c_i f_i)$ was used to update the data buffer. In particular, the server multiplies the value $E(c_i)$ into each location j in the c-buffer where $g(i, j) = 1$.

Step 4: Update matching-indices buffer. The server then multiplies $E(c_i)$ further into a fixed number of locations in matching-indices buffer. This is done using essentially the standard procedure for updating a Bloom filter. Specifically, we use k hash functions $h_1, \ldots, h_k$ to select the k locations where $E(c_i)$ will be added. The locations of the matching indices buffer that a matching file i is multiplied into are take to be $h_1(i), h_2(i), \ldots, h_k(i)$. Again, if the f_i does not match, $c_i = 0$ so the matching-indices buffer is effectively unmodified.

After completing the aforementioned steps for a fixed number of files t in its stream, the server sends its three buffers back to the client. Also, the server should return the function g.

3.3 Client's `FileReconstruction` Procedure

Step 1: Decrypt buffers. The client first decrypts the values in the three buffers using the Paillier decryption algorithm with its private key K_{priv}, obtaining decrypted buffers F', C', and I'.

Step 2: Reconstruct matching indices. For each of the indices $i \in \{1, 2, \ldots, t\}$, the client computes $h_1(i), h_2(i), \ldots, h_k(i)$ and checks the corresponding locations in the decrypted matching-indices buffer; if all these locations are non-zero, then i is added to the list $\alpha_1, \alpha_2, \ldots, \alpha_\beta$ of potential matching indices. Note that if $c_i \neq 0$, then i will be added to this list. However, due to the false positive feature of Bloom filters, we may obtain some additional indices.

3. Private Stream Search Model

Step 3: Reconstruct c-values of matching files. Given our superset of the matching indices $\{\alpha_1, \alpha_2 \ldots, \alpha_{\ell_F}\}$, the client next solves for the values of $c_{\alpha_1}, c_{\alpha_2}, \ldots, c_{\alpha_{\ell_F}}$. This is accomplished by solving the following system of linear equations for $\vec{c}$,

$$A \cdot \vec{c} = C' \tag{1}$$

where A is the matrix with the i, jth entry set to $g(\alpha_i, j)$, C' is the vector of values stored in the decrypted c-buffer, and $\vec{c}$ is the column vector $(c_{\alpha_i})_{i=1,\ldots,\ell_F}$.[4] Now the exact set of matching indices $\{\alpha'_1, \alpha'_2 \ldots, \alpha'_r\}$ may be computed by checking whether $c_{\alpha_i} = 0$ for each $i \in \{1, \ldots, \ell_F\}$. Before proceeding, we replace all zeros in the vector $\vec{c}$ with ones.

As an example of Step 3, suppose there are four spots in the decrypted c-buffer (i.e., $\ell_F = 4$), seven files are processed, and we have established the following list of potentially matching indices: $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} = \{1, 3, 5, 7\}$. Then given

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}, \quad C' = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 3 \end{pmatrix}$$

we may compute

$$\begin{aligned} c_{\alpha_1} &= c_1 = 1 \\ c_{\alpha_2} &= c_3 = 2 \\ c_{\alpha_3} &= c_5 = 1 \\ c_{\alpha_4} &= c_7 = 0 \ . \end{aligned}$$

We then see that there were three matching files ($r = 3$): f_1, f_3, and f_5.

3. Private Stream Search Model

Step 4: Reconstruct matching files. Finally, the content of the matching files $f_{\alpha'_1}, f_{\alpha'_2}, \ldots, f_{\alpha'_r}$ may be determined by solving the linear system

$$A \cdot \mathrm{diag}(\vec{c}) \cdot \vec{f} = F' \tag{2}$$

where

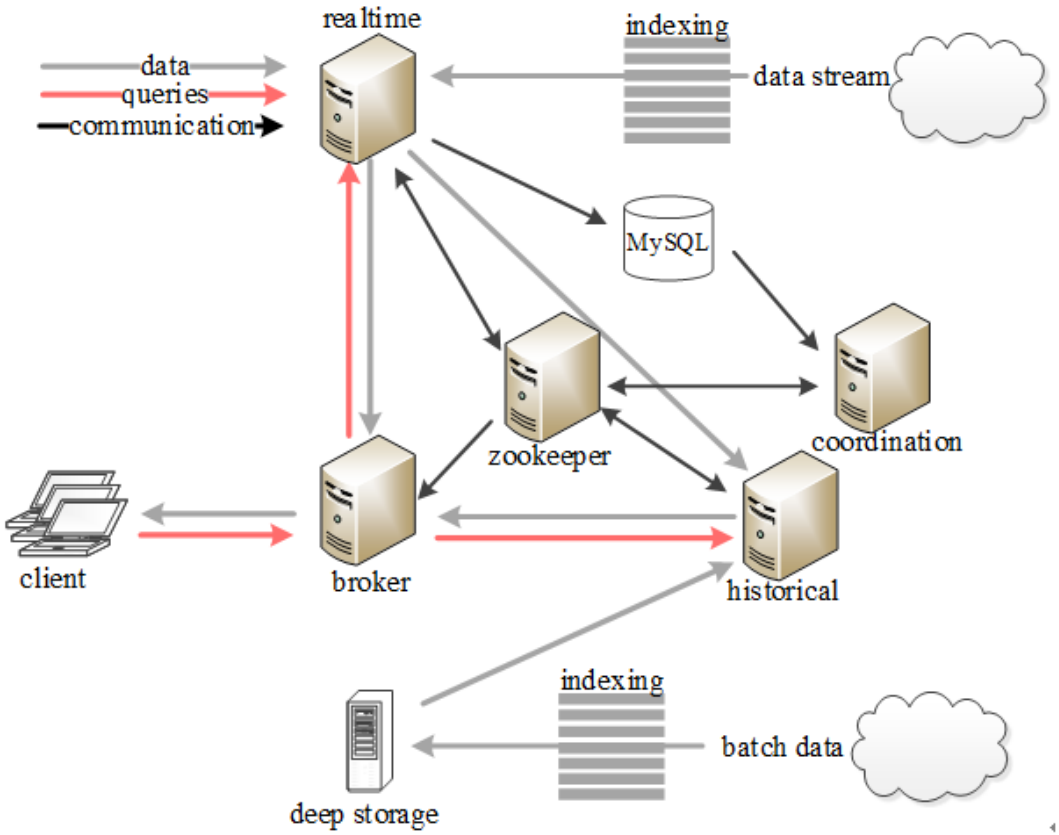
$$\mathrm{diag}(\vec{c}) = \begin{pmatrix} c_1 & 0 & \cdots \\ 0 & c_2 & \\ \vdots & & \ddots \end{pmatrix} .$$

We directly compute $\vec{f} = \mathrm{diag}(\vec{c})^{-1} \cdot A^{-1} \cdot F'$.

$$\begin{aligned} f_1 + f_5 &= 32 \\ f_1 + 2f_3 + f_7 &= 32 \\ f_1 + f_7 &= 10 \\ 2f_3 + f_5 &= 44 \ , \end{aligned}$$

thereby determining that $f_1 = 10$, $f_3 = 11$, and $f_5 = 22$ (and $f_7 = 0$, but this value is ignored).

4. Distributed Processing Architecture



Distributed Processing Architecture

The broker node functions as query router, it can route query to the queryable node

The real-time compute node is responsible for data injection, storage and responses to queries for the most recent data.

The historical compute node is responsible for loading and responding to queries for historical data

The coordination node is responsible for the management and distribution of compressed raw data--segment, and the management of the replicated segment and load balancing of segment

A query will firstly be sent to the broker node, which is responsible for finding and routing the query to the storage nodes containing related data, the storage nodes execute their portion of the query in parallel and return the results to the broker node, then the broker node receives the results and merges them, and finally returns the result to the users.

4. Distributed Processing Architecture

☆ Historical Compute Node

It loads historical segments from a permanent storage and make them queryable. Since historical compute nodes do not know each other, there is no competition of single point between the nodes.

The historical compute node only needs to know how to perform its assigned tasks through maintaining a connection with the Zookeeper.

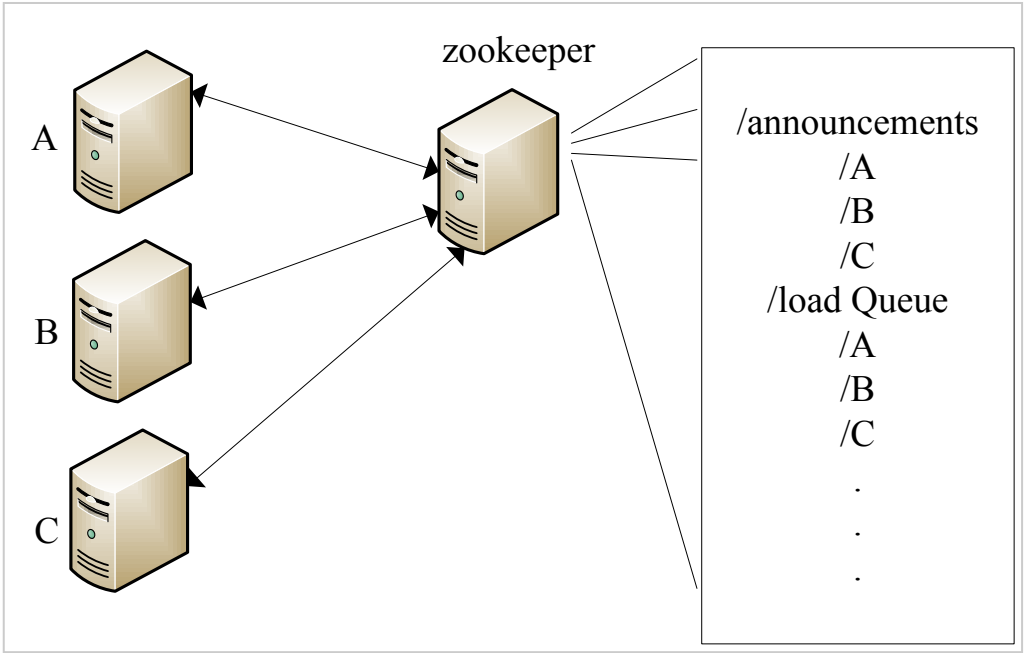


TABLE I. THE SEGMENT EXAMPLE

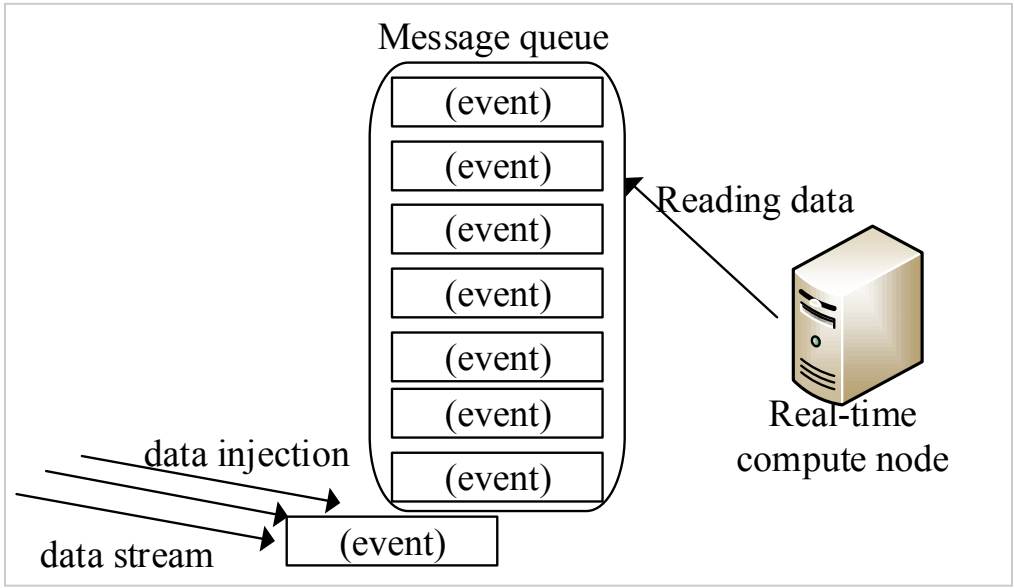
Timestamp	Publisher	Advertiser	Gender	Country	Impressions	Clicks	Revenue
2014-01-01 T01:00:00Z	sina.com	baidu.com	Male	China	1800	25	15.70
2014-01-01 T01:00:00Z	sina.com	baidu.com	Male	China	2912	42	29.18
2014-01-01 T01:00:00Z	yahoo.com	google.com	Male	USA	1953	17	17.31
2014-01-01 T01:00:00Z	yahoo.com	google.com	Male	USA	3914	170	34.01

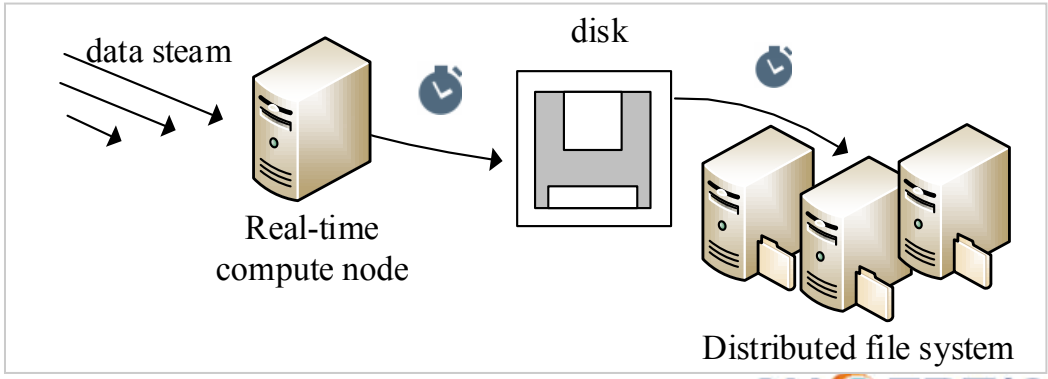
4. Distributed Processing Architecture

☆ Real-time Compute Node

The real-time compute node encapsulates the functions of data injection and query. Data indexed via real-time compute node can be queried immediately.

The real-time compute node will merge all persisted indexes, and build a historical segment, and then send the historical segment to historical compute nodes to serve. Once the segment on the historical compute node can be queried, then the real-time compute node will delete all the information of this segment and publish it will never serve this segment.





4. Distributed Processing Architecture

☆ Broker Node

Query Router

- routes encrypted query to the queryable node

Information Collector

- gathers the metadata published in Zookeeper about what segments exist and where the segments.

Result Evaluator

- merges the query results from each node, before returning final result to the users.
- provides data persistence layer through maintaining a cache for recent results.

4. Distributed Processing Architecture

☆ Coordination Node

Data Manager

- management and distribution of segments (including loading new segments, dropping outdated segments)
- management of the replicated segments and load balancing of segments.

Cluster Manager

- compares the expected state of the cluster and the actual state of the cluster to make decision.
- maintains a Zookeeper connection to obtain information of all nodes in the cluster
- maintains a MySQL database connection to get the information.

5. Bitmap-based Compression Storage

Column-oriented storage could make the CPU more efficient as a result of only needed data are loaded and scanned.

Publisher	Advertiser	Gender	Country	Impressions	Clicks	Revenue
bieberfever.com	sina.com	Male	China	1800	25	15.70
bieberfever.com	sina.com	Male	China	2912	42	29.18
ultratrimfast.com	yahoo.com	Male	USA	1953	17	17.31
ultratrimfast.com	yahoo.com	Male	USA	3914	170	34.01

Dictionary encoding is a common method to compress data. For data shown in Table, we can map each publisher into a unique integer identifier as follows:

sina.com (15B) -> 0 (4B)， yahoo.com (17B) -> 1 (4B)

For numeric columns，we compress the original value instead of the encoded dictionary representations. We store the information in a binary array, which represents the row by the array indices. If the publisher is seen in a certain row, the array indices will be marked as 1, for example:

sina.com -> rows[0, 1]->[1][1][0][0]

6. Experiments

➢Scalability Experiment

Testing datasets contains 80GB data including millions of rows. This data set includes more than a dozen dimensions, and the cardinalities ranges from double digits to string. We calculate three aggregation metrics for each row (count, sum, average). This data set is firstly divided on the time stamp, and then on dimension value to create thousands of segments, each segment is about 10,000 lines.
Testing benchmark cluster contains 6 compute nodes, and each node has 16 cores, 16GB of RAM, 10Gigabit Ethernet and 1TB disk space. Overall, the cluster contains 96 cores, 96GB of RAM, as well as enough fast Ethernet and enough disk space.

Query#	Encrypted Aggregation Query
1	SELECT count() FROM _table_ WHERE name=E(zp)
2	SELECT count(), sum() FROM _table_ WHERE name=E(zp)
3	SELECT count(), average() FROM _table_ WHERE name=E(zp)
4	SELECT arrival_date, count() AS cnt FROM _table_ WHERE name=E(zp) GROUP BY arrival_date ORDER BY cnt limit 100
5	SELECT arrival_date, count() AS cnt, sum() FROM _table_ WHERE name=E(zp) GROUP BY arrival_date ORDER BY cnt limit 100
6	SELECT departure_date, count() AS cnt, sum(), FROM _table_ WHERE name=E(zp) GROUP BY departure_date ORDER BY cnt limit 100

6. Experiments

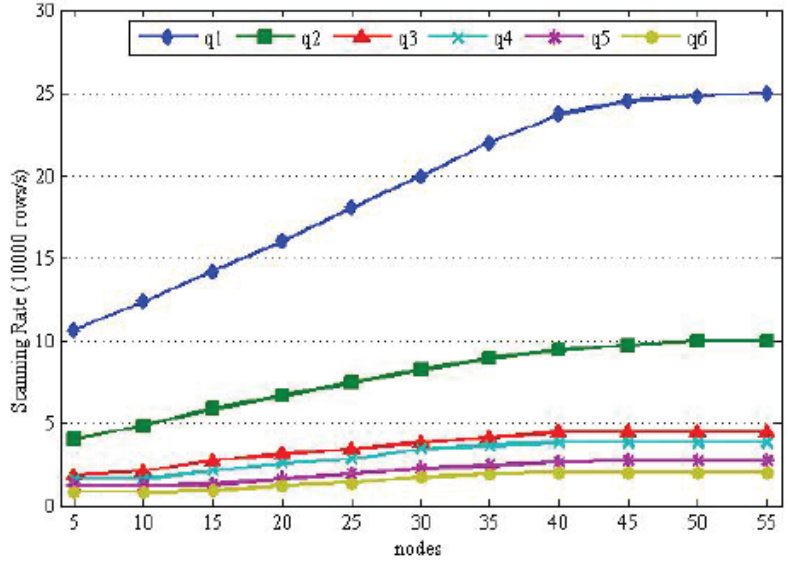


Fig. 5. The cluster scanning rate

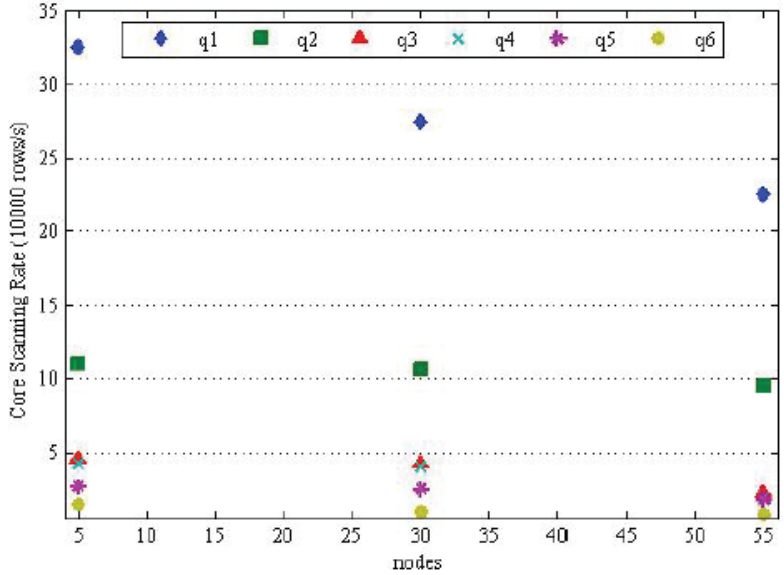


Fig. 6. The core scanning rate

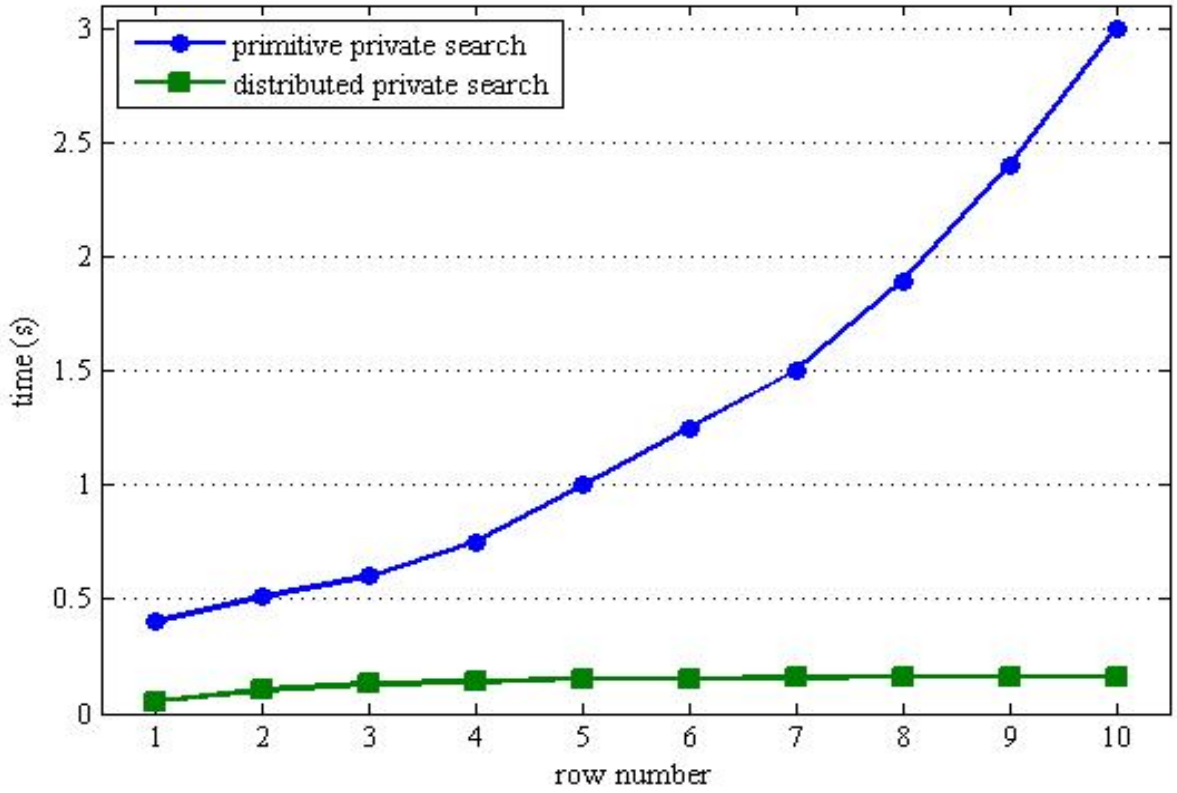
Result1: the performance of the marginal revenue decreases with the scale of the cluster increasing. Under the expected linear scaling, Query 1 on a cluster with 55 nodes would achieve scanning rate of 37 million rows per second. In fact, the scanning rate is 25 million rows per second.

Result2: the first query achieves scanning rate of 330 thousands lines per second per core. In fact, we consider that the cluster with 55 nodes is actually over provisioned for the test datasets, which explains the growth is slower than the cluster with 30 nodes.

Analysis：According to the Amdahls Law, the increase of the speed of a parallel computing system is often limited by the time requirements for the sequential operations of the system.

6. Experiments

The last experiment compares the time consumption of average aggregate function running in primitive private search system to that running in our system with the scale of the input increasing. Since the primitive private search system cannot support dynamic scalability, its time consumption keeps high increasing rate, on the contrary, our system is dynamically scalable according to the input scale, so the processing keeps stable over time.



Keyword:
zp,
ly,
google,
baidu,
car,
ship,
people,
water,
paper,
patent

CNCERT/CC

7. Conclusions

➢ Conclusion

In this paper, we propose a scalable distributed private stream search system, which adopts the distributed architecture to support the scalability. The experiments show the system has good performance and scalability on online aggregation queries. Moreover, the query can be encrypted through Paillier encryption to protect search criteria.

➢ Future Works

In some applications, the predetermined set of possible keywords D may be unacceptable. Many of the strings a user may want to search for are obscure and including them in D would already reveal too much information. We will eliminating D, we allow K to be any finite subset of $\sum^*$, where $\sum$ is some alphabet

REFERENCES

- [1] X. Meng and X. Ci, “Big data management: Concepts, techniques and challenges,” *Journal of computer research and development, vol. 50,* no. 1, pp. 146–169, 2013.
- [2] G. Jim and G. Goetz, “The five-minute rule ten years later, and other computer storage rules of thumb,” *ACM Sigmod Record, vol. 26, no. 4,* pp. 63–68, 1997.
- [3] W. Lam, L. Liu, S. Prasad, A. Rajaraman, Z. Vacheri, and A. Doan, “Muppet: Mapreduce-style processing of fast data,” *PVLDB, vol. 5,* no. 12, pp. 1814–1825, 2012.
- [4] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, “Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters,” in *Proceedings of the 4th USENIX Workshop on Hot Topics in Cloud Computing, 2012.*
- [5] J. K. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazi`eres, S. Mitra, A. Narayanan, G. M. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The case for ramclouds: scalable high-performance storage entirely in dram,” *Operating Systems Review, vol. 43, no. 4, pp. 92–105, 2009.*
- [6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing, 2010.*
- [7] J. Bethencourt, D. X. Song, and B. Waters, “New techniques for private stream searching,” *ACM Transactions on Information and System Security, vol. 12, no. 3, 2009.*
- [8] S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, “Outsourced symmetric private information retrieval,” in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security,* 2013, pp. 875–888.
- [9] S. Yekhanin, “Private information retrieval,” *Communications of the ACM, vol. 53, no. 4, pp. 68–73, 2010.*

REFERENCES

- [1] X. Meng and X. Ci, “Big data management: Concepts, techniques and challenges,” *Journal of computer research and development, vol. 50,* no. 1, pp. 146–169, 2013.
- [2] G. Jim and G. Goetz, “The five-minute rule ten years later, and other computer storage rules of thumb,” *ACM Sigmod Record, vol. 26, no. 4,* pp. 63–68, 1997.
- [3] W. Lam, L. Liu, S. Prasad, A. Rajaraman, Z. Vacheri, and A. Doan, “Muppet: Mapreduce-style processing of fast data,” *PVLDB, vol. 5,* no. 12, pp. 1814–1825, 2012.
- [4] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, “Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters,” in *Proceedings of the 4th USENIX Workshop on Hot Topics in Cloud Computing, 2012.*
- [5] J. K. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazi`eres, S. Mitra, A. Narayanan, G. M. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The case for ramclouds: scalable high-performance storage entirely in dram,” *Operating Systems Review, vol. 43, no. 4, pp. 92–105, 2009.*
- [6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing, 2010.*
- [7] J. Bethencourt, D. X. Song, and B. Waters, “New techniques for private stream searching,” *ACM Transactions on Information and System Security, vol. 12, no. 3, 2009.*
- [8] S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, “Outsourced symmetric private information retrieval,” in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security,* 2013, pp. 875–888.
- [9] S. Yekhanin, “Private information retrieval,” *Communications of the ACM, vol. 53, no. 4, pp. 68–73, 2010.*

- [10] C. B¨osch, P. Hartel, W. Jonker, and A. Peter, “A survey of provably secure searchable encryption,” ACM Computing Surveys (CSUR), vol. 47, no. 2, p. 18, 2014.
- [11] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques, 1999, pp. 223–238.
- [12] R. Ostrovsky and W. E. S. III, “Private searching on streaming data,” in Proceedings of the 25th Annual International Cryptology Conference, 2005, pp. 223–240.

Thanks

For any question or further information please contact
liyan@cert.org.cn

CN ERT/CC