Executing SQL over Encrypted Data in the Database-Service-Provider Model

Summary:
This paper attacked a problem which can be critical in the following decade -- how to enforce data confidentiality in cloud computing when the potential adversaries include the cloud provider. The paper presented a novel data encoding format along with a formal algebra framework which allows the client to execute complex SQL queries on the server without decrypting the data remotely. The evaluation preliminarily proves the feasibility of the idea.

Comments:

Pros:
The paper noticed a critical issue in the era of cloud computing and proposed a novel solution to the problem. Different from previous work, this new technique allows the client to execute very complex SQL queries on encrypted data, which significantly increases the attraction of adoption. Also, the solution is highly compatible with existing software and hardware, thus can be expected to be deployed at a low cost.

The authors developed a formal algebra framework for their algorithms which makes the solution solid and very clear to reviewers. Apart from the standard routines for executing a SQL query, the paper also provided some heuristic rules for further optimization.

Cons:
The overall quality of the paper is harmed by lacking a dedicated related work section. The authors should write more about competing techniques. For example, a homomorphic encryption algorithm allows arithmetic computation over encrypted data without having to decrypt them, thus achieving a similar objective to that of this paper.

Another shortcoming of this paper is the evaluation section. In both experiments only "select" SQL operations are tested, which means the datasets are always static. However in reality, most databases are both readable and writable, so the data patterns may change all the time. The efficiency of this technique partially depends on how evenly the data are distributed by the partition algorithm. With the data pattern constantly changing, the original partition algorithm may no longer fit the data and slow down the system after a duration of deployment.

Finally, the paper is weak at taking security issues into consideration. For instance, there is not writing about encryption key management issues. Based on the evaluation settings, I will assume the database is encrypted by a symmetric block
encryption algorithm. Since one key is shared by all the clients, the risk of encryption key theft significantly increases. The paper also did not describe their threat model, which makes it hard to evaluate the security strength of the system, e.g., if the interface of the client is public, the attackers can perform large scale chosen-cipher attacks even if the SQL translator is kept secret.

Question:

Is this technique good at managing long text data?
Human-powered Sorts and Joins

Summary:

This work implemented a programming interface for Amazon's crowdsourcing service along with algorithms and optimizations for human intelligence tasks. The proposed framework is able to perform relational data operations such as sorting and joining based on feedbacks from crowdsourcing workers. The work was evaluated in a real-world setting, showing very promising results with respect to accuracy, latency, and most importantly, the financial cost of crowdsourcing tasks.

Comments:

Pros:
The framework presented by this paper greatly reduces the burden for designing the workflow of crowdsourcing tasks. It is a novel trial to improve the productivity of the current crowdsourcing system. The new batching, joining, and sorting algorithm designed dedicatedly for crowdsourcing is straightforward while still very effective. The evaluation was very solid with real-world experiment settings and the results proved the power of this new approach.

Cons:
The paper developed a new query language for their system; however, there was no formal definition or description for the new language, which is a critical flaw for programming language design. Without a formal description it is hard to fully evaluate the capability of the language. It is not beneficial for language standardization or implementation, either.

Integrating HTML code into the query language is not a good design from the perspective of software engineering. Since HTML code is directly related to web page rendering, it usually contains code irrelevant to content (such as styles and javascripts). Mixing the frontend and backend of a web application can cause serious development and maintenance problems. The authors should develop some other approach to remove HTML from their query design.

Questions:

Does poorly performing or malicious workers in crowdsourcing share any special patterns? Can this approached be applied to AI recognition systems with random algorithms?