Privacy of the Client Protection by using Trusted Database Security System

M.Akila¹, M.Markco²

¹PG scholar, Department of Computer Science Engineering, E.G.S. pillay Engineering College, Nagapattinam. ²Assistant Professor, Department of Computer Science Engineering, E.G.S. pillay Engineering College, Nagapattinam.

Abstract: Although the remuneration of outsourcing and exhaust are well known, significant challenges yet lie in the path of large-scale adoption since such services often require their customers to inherently trust the provider with full access to the outsourced data sets. We introduce trust DB, an outsourced database prototype that allows clients to execute SQL queries with privacy and under regulatory compliance constraints by leveraging server-hosted, tamper-proof trusted hardware in critical query processing stages, thereby removing any limitations on the type of supported queries. Despite the cost overhead and performance limitations of trusted hardware, we show that the costs per query are orders of magnitude lower than any (existing or) potential future software-only mechanisms. Trusted DB is built and runs on actual hardware and its performance and costs are evaluated here. This work’s inherent thesis is that, at scale, in outsourced contexts, computation inside secure hardware processors is orders of magnitude cheaper than equivalent cryptography performed on provider’s unsecured server hardware, despite the overall greater acquisition cost of secure hardware.

Keywords: Query Processing, Trusted DB, Cryptography, Prototype.


1 Introduction

Although the benefits of outsourcing and clouds are well known significant challenges yet lie in the path of large-scale adoption since such services often require their customers to inherently trust the provider with full access to the outsourced data sets. Numerous instances of illicit insider behavior or data leaks have left clients reluctant to place sensitive data under the control of a remote, third-party provider, without practical assurances of privacy and confidentiality, especially in business, healthcare, and government frameworks. Moreover, today’s privacy guarantees for such services are at best declarative and subject customers to unreasonable fine-print clauses. For example, allowing the server operator to use customer behavior and content for commercial profiling or governmental surveillance purposes.

Ideas have also been proposed to leverage tamper-proof hardware to privately process data server-side, ranging from smart-card deployment in healthcare to more general database operations. Yet, common wisdom so far has been that trusted hardware is generally impractical due to its performance limitations and higher acquisition costs. As a result, with very few exceptions, these efforts have stopped short of proposing or building full-fledged database processing engines. However, recent insights into the cost-performance tradeoff seem to suggest that things stand somewhat differently. Specifically, at scale, in outsourced contexts, computation inside secure processors is orders of magnitude cheaper than any equivalent cryptographic operation performed on the provider’s unsecured server hardware, despite the overall greater acquisition cost of secure hardware. This is so because the overheads for cryptography that allows some processing by the server on encrypted data are extremely high even for simple operations. This fact is rooted not in cipher implementation inefficiencies but rather in fundamental cryptographic hardness assumptions and constructs, such as trapdoor functions.

Moreover, this is unlikely to change anytime soon as none of the current primitives have, in the past half-century. New mathematical hardness problems will need to be discovered to allow hope of more efficient cryptography. In shared-nothing multiprocessor database machines, the relational operators that form a query are executed on the processors where the relations they reference are stored. When a query must be executed by multiple processors, it is first sent to a Query Manager (QM) process which assumes responsibility for its execution. The QM process sends the query to each processor that the query optimizer has indicated should participate in its execution. When each processor finishes executing the query, it sends a “query done” message back to the QM process. If the QM receives a “query done” message from each processor, it sends a “commit” message to each processor and closes all of the communication ports. Finally, it notifies the process that submitted the query of its successful execution. If, on the other hand, the query is aborted at a processor (generally because of a concurrency control deadlock) that processor sends an “abort” message to the QM.

At a multiprogramming level of one, the throughput with the hash and hybrid-range declustering strategies is 16% higher than that of the range partitioning strategy.
because the hash and hybrid-range declustering strategies utilize intra-query parallelism effectively while the range partitioning strategy directs the query to the absolute minimum number of processors and performs the majority of the work in a sequential manner. With the range partitioning strategy, the throughput increases only slightly from a multiprogramming level of one to two because every time two queries are directed to the same processor the response time of each query increases significantly as the SCSI cache becomes ineffective. Since a large number of disk pages are processed by a query at a single processor, when the SCSI cache becomes ineffective, the performance of the system degrades significantly.

II. Related Work

We survey the recent wave of extensions to the popular map reduce systems, including those that have begun to address the implementation of recursive queries using the same computing environment as map-reduce. A central problem is that recursive tasks cannot deliver their output only at the end, which makes recovery from failures much more complicated than in map-reduce and it’s no recursive extensions. We propose several algorithmic ideas for efficient implementation of recursions in the map-reduce environment and discuss several alternatives for supporting recovery from failures without restarting the entire job. The introduction of map-reduce by Dean and Ghemawat for parallel computation on commodity clusters focused a great deal of commercial and intellectual interest on this model and similar approaches to managing large-scale data [1]. It is useful to reflect on what map-reduce brings to the table. Map-reduce itself is a convenient way for modestly skilled programmers to implement many data operations, including the operations of relational algebra and operations on sparse matrices and vectors (which are not too much different from joins of relations). However, the resulting parallel algorithms for relational operations are essentially the same as those known earlier. The secret sauce of map-reduce is not the algorithms it supports, but rather the way it handles failures during the execution of a large job. Rapid advances in Networking and Internet technologies have fueled the emergence of the “software as a service” model for enterprise computing. Successful examples of commercially viable software services include rent-a-spreadsheet, electronic mail services, general storage services, disaster protection services.” Database as a Service” model provides users power to create, store, modify and retrieve data from anywhere in the world, as long as they have the access to internet. It introduces several challenges, an important issue being data privacy. It is in context that we specifically address the issue of data privacy. There are two main privacy issues. First owner of the data needs to be assured that the data stored on the data stored on service provider site. Second data needs to be protected even from service providers, if the providers themselves can be trusted. The internet had made it possible for all computers to be connected to one another [2]. The influence of transaction processing system and the internet ushered in the era of e-business. The democratization of ubiquitous computing access data anywhere, anytime, anyhow, the increasing connection of corporate databases to the Internet and the today’s natural resort to Webhosting companies strongly emphasize the need for data confidentiality.

The weaknesses of the server-based approach to meet the data confidentiality requirements led us to devise client based solutions. As a preliminary remark, let us notice that the solution presented to enforce data privacy is typically client-based since the server does nothing but storing encrypted data. Unfortunately, these solutions do not support sharing. However, client-based approaches have been historically disregarded considering that users have themselves the opportunity to hack the client system, and then the sharing control in our context, with total impunity.

Preterit introduces [8] a clear distinction between the role of the DBA, administering the database resources, and the role of the SA (Security Administrator), administering user privileges, encryption keys and other related security parameters. This distinction is also made effective at the system level by separating the database server from the security server. Simple smartcard applications do not require administration because they are in some sense pre-administered (data schema, user and access rights are hardcoded). The side effect is the lack of extensibility. The concept of server typically addresses the storage and sharing issues [9]. Thus, let us consider to which extent the sphere of security provided by the smartcard could be extended to a remote server holding encrypted data.

III. Proposed System

We posit that a full-fledged, privacy enabling secure database leveraging server-side trusted hardware can be built and run at a fraction of the cost of any (existing or future) cryptography-enabled private data processing on common server hardware. We validate this by designing and building Trusted DB, a SQL database processing engine that makes use of tamperproof cryptographic coprocessors such as the IBM 4764 in close proximity to the outsourced data. Tamper resistant designs however are significantly constrained in both computational ability and memory capacity which makes implementing fully featured database solutions using secure coprocessors (SCPUs) very challenging. Trusted DB achieves this by utilizing common unsecured server resources to the maximum extent possible. E.g., Trusted DB enables the SCPU to transparently access external storage while preserving data confidentiality with on-the-fly encryption. This eliminates the limitations on the size of databases that can be supported. Moreover, client queries are pre-processed to identify sensitive components to be run inside the SCPU. Non-sensitive operations are offloaded to the entrusted host server. This greatly improves performance and reduces the cost of transactions.
Fig 1 Proposed of Block Diagram

Figure 1 shows the proposed novel system architecture. The aim of this paper is to. Trusted DB achieves this by utilizing common unsecured server possessions to the limit extent possible Non-sensitive operation are off-loaded to the entrusted host server. This greatly improves performance and reduces the cost of connections. The performance of securing catalog in server surface trusted DB, a module of client to thwart data in cryptography are tampered with for authenticated however are appreciably embarrassed in both computational facility and memory capability which make implement fully featured list solutions using secure coprocessors (SCPs) very demanding. Kernel element loading after boot to prevent any tampering with the agent, a user-space trust client opens the device and concurrently opens an opening to the trusted server, waiting for communication.

A. Query Plan Generator

A kernel module, which we call trust agent, is loaded on boot or can be compiled in the Kernel Module loading after boot to prevent any tampering with the agent, a user-space trust client opens the device and concurrently opens a socket to the trusted server, waiting for communication. The trust agent who has the signature of the correct client program verifies the integrity of the client. This verification also prevents any other program from opening the device in our security model.

B. Initial Authentication

When the remote server gets a connection from a client, it requests the initial attestation. The must client uses the write system call to request the required information from the agent. The trust agent onwards the signed TPM quote. The trust client forwards the information to The server which verifies the information.

C. Key Exchange and Monitoring

The trust agent and the remote server set up a shared key through an RSA key exchange protocol based on the TPM keys. At the trust agent, each keystroke event is encrypted and a corresponding signature is generated. Both the encrypted event and its signature are wrapped in a packet. The trust client forwards the packets to the remote server which verifies the integrity of events by checking the signatures using a keyed hash function. If signatures associated with events do not pass the server’s verification, then the trust agent is notified.

D. Performance Evaluation

This module to evaluate the malware actively makes outside connections for command and control or attacks. For example, malware may attempt to log user inputs, inject traffic bypassing the host’s firewall, forge input events, and tamper with network traffic.

IV. Results And Discussion

This result are discussed by eliminates the limitations on the size of databases that can be supported. Besides, client queries are pre-processed to classify sensitive workings to be run surrounded by the SCPU. This element to assess the malware energetically makes outside links for domination and control or attacks. Regardless of the cost transparency and performance restrictions of trusted hardware, we show that the expenditure per query are orders of enormity lower than any (to be had or) potential expectations software-only mechanism. Trusted DB is built and runs on actual hardware and its performance and costs are evaluated here. This work’s inherent thesis is that, at scale, in outsourced contexts, computation inside secure hardware processors is orders of magnitude cheaper than equivalent cryptography performed on provider’s unsecured server hardware, despite the overall greater acquisition cost of secure hardware.

Fig 2 Comparison of Different Methods

The Graph1 shows comparison of results produced by various methods and it shows clearly that the proposed method produces efficient performances, low cost operation.

V. Conclusion

We conclude by that Trusted DB is built and runs on actual hardware and its performance and costs are evaluated here. This work’s intrinsic argument is that, at measure, in outsourced contexts, computation inside secure hardware
processors is orders of magnitude cheaper than equivalent cryptography performed on provider’s unsecured server hardware, despite the overall greater acquirement cost of make safe hardware. This future work’s inherent thesis is that, at scale, in outsourced contexts, working out inside secure hardware processors is orders of enormity cheaper than the same cryptography perform on provider’s unsecured member of staff serving at table hardware, despite the overall greater attainment cost of vulnerable hardware.

References


