Risk-Aware Data Processing in Hybrid Clouds

Technical Report UTDCS-31-11
The University of Texas at Dallas
Department of Computer Science
November 2011

Vaibhav Khadilkar, Kerim Yasin Oktay, Bijit Hore,
Murat Kantarcioglu, Sharad Mehrotra, Bhavani Thuraisingham

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
NOV 2011		**00-00-2011 to 00-00-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Risk-Aware Data Processing in Hybrid Clouds	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **The University of Texas at Dallas,Department of Computer Science,Richardson,TX,75080-3021**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This paper explores query processing in a hybrid cloud model where a user?s local computing capability is exploited alongside public cloud services to deliver an efficient and secure data management solution. Hybrid clouds offer numerous economic advantages including the ability to better manage data privacy and confidentiality, as well as exerting control on monetary expenses of consuming cloud services by exploiting local resources. Nonetheless, query processing in hybrid clouds introduces numerous challenges, the foremost of which is, how to partition data and computation between the public and private components of the cloud. The solution must account for the characteristics of the workload that will be executed, the monetary costs associated with acquiring/operating cloud services as well as the risks affiliated with storing sensitive data on a public cloud. This paper proposes a principled framework for distributing data and processing in a hybrid cloud that meets the conflicting goals of performance, disclosure risk and resource allocation cost. The proposed solution is implemented as an add-on tool for a Hadoop and Hive based cloud computing infrastructure.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**21**	

Risk-Aware Data Processing in Hybrid Clouds

Vaibhav Khadilkar [#1], Kerim Yasin Oktay [*1], Bijit Hore [*2],
Murat Kantarcioglu [#2], Sharad Mehrotra [*3], Bhavani Thuraisingham [#3]

[#] *The University of Texas at Dallas*
[1]vvk072000@utdallas.edu, [2]muratk@utdallas.edu, [3]bxt043000@utdallas.edu

[*] *University of California, Irvine*
[1]koktay@uci.edu, [2]bhore@ics.uci.edu, [3]sharad@ics.uci.edu

Abstract

This paper explores query processing in a *hybrid cloud* model where a user's local computing capability is exploited alongside public cloud services to deliver an efficient and secure data management solution. Hybrid clouds offer numerous economic advantages including the ability to better manage data privacy and confidentiality, as well as exerting control on monetary expenses of consuming cloud services by exploiting local resources. Nonetheless, query processing in hybrid clouds introduces numerous challenges, the foremost of which is, how to partition data and computation between the public and private components of the cloud. The solution must account for the characteristics of the workload that will be executed, the monetary costs associated with acquiring/operating cloud services as well as the risks affiliated with storing sensitive data on a public cloud. This paper proposes a principled framework for distributing data and processing in a hybrid cloud that meets the conflicting goals of performance, disclosure risk and resource allocation cost. The proposed solution is implemented as an add-on tool for a Hadoop and Hive based cloud computing infrastructure.

1 Introduction

The rise of cloud computing has created a revolution in the computing industry by giving end-users access to sophisticated computational infrastructures, platforms, and services using a pay-as-you-use model. Several new systems like HadoopDB and Hive that support database query processing on the cloud have emerged. Such systems investigate the potential benefits of using cloud-based systems instead of traditional relational databases. An emerging trend in cloud computing is that of *hybrid cloud*. Unlike traditional outsourcing where organizations push their data and data processing to the cloud, in hybrid clouds in-house capabilities/resources at the end-user site are seamlessly integrated with cloud services to create a powerful, yet cost-effective data processing solution. Hybrid cloud solutions offer similar benefits as traditional cloud solutions. Yet, they provide advantages in terms of disclosure control and minimizing cloud resources given that most organizations already have an infrastructure they can use. Exploiting such benefits, however, opens numerous questions such as how should one partition data and computation between the public and private side of the infrastructure? What are the implications of the different designs – from the perspectives of data disclosure? computational performance? overall costs/savings?

Let us illustrate some of the design choices and their implications using an example. Consider a scenario in which a university IT department is considering a cloud-based solution as a cost-effective approach to supporting increased data analyses needs. We consider a simplistic database schema consisting of the following tables:

```
Student(name, student_id, ssn, year_join, year_left, dept)
Catalog(c_id, title, descr, dept, units)
Instructors(i_id, i_name, ssn, i_dept, join_date)
Offering(instructor_id, course_id, quarter)
Enrollment(s_id, c_id, term, grade)
```

In the above student database, while data such as catalog information is not sensitive, information about students & instructors (e.g., ssn), student course enrollments, and their grades is deemed sensitive. In particular, information such as ssn and grades, if leaked could lead to misuse of data and/or loss of privacy and hence must be protected. In addition to supporting typical workload such as queries to prepare end-of-term grade reports, identify policy violations (e.g., low/high course enrollments, students with too low a GPA, too low term credits, etc.), the university administrators, now require the database to support significantly more complex analyses such as degree of reciprocity amongst different departments in terms of student enrollments in courses offered by the departments, longitudinal (e.g., over the past 10 years) study of students' relative performance (viz. grades) in courses offered by different departments grouped based on students' home department, etc.

On one extreme, the university may choose to outsource its entire data and workload to the public cloud (as is typical to outsourcing solutions). While simple to implement, such a solution, incurs the most monetary cost in terms of cloud service (both storage & computing), and is most vulnerable to data leakage[1]. In addition, the outsourcing strategy may not even be optimal in terms of performance since it wastes local resources which are now unused. An alternate strategy might be to replicate data at both private and public sides and to split the workload between the two. For instance, policy checks and student reports may be prepared locally, while periodic complex analyses queries may be computed on the cloud side. The above strategy exploits local resources, and thereby reduces cost of cloud services required.

Another possibility might be to partition data in a way that allows multiple workload queries to be evaluated partially on both sides. For instance, *Student* and *Enrollment* tables might be split horizontally on both sides whereas *Catalog* may be replicated completely on the public side (since it is expected to be a small table). Such an approach will allow for end-of-term grade report generation, policy violation checks, as well as analyses queries to be parallelized, thus gaining performance. In addition, this plan (like the previous) may also reduce the storage and potentially processing needs at the public cloud since now only part of the data processing is done on the public side while the remainder is done at the local side. The fourth possibility might be to only store the projection of the *Enrollment* table without the "grade" attribute on the public side. This will reduce the risk of disclosure (to the public cloud) of data that is potentially sensitive (viz., grades). Likewise, "ssn", "s_id", and "name" from the *Student* relation might only be exposed in encrypted form to the public side to curb disclosure.

The possibilities described above are just four of the multitude of design choices, each of which represents different tradeoffs in terms of performance, costs, and disclosure.

In this paper, we develop a principled framework that provides mechanisms/opportunities to end users to tune storage and workload execution over private and public clouds in a way that the resulting query processing strikes the right balance of performance, risks, and costs from the users' perspective. The key challenge in designing such a framework is deciding what should be stored at the public and local sites. The data partitioning and representation dictates not only the cloud storage requirement and level of disclosure of sensitive data, but also the query processing strategies used to process the workload. Query processing, in turn dictates the level of load generated on the public and private sides, the risks incurred, and the monetary cost of cloud infrastructures. We postulate data partitioning as a minimization problem where the goal is minimizing the performance cost while adhering to bounds on risks and monetary costs[2]. By controlling the degree of risks and monetary costs, an end user can ensure appropriate tradeoffs of the three factors in supporting query processing in hybrid clouds.

For a given data partitioning, the level of data exposure, as well as, workload distribution across public and private clouds depends upon the underlying data representation used. For instance, using non-deterministic encryption to store data on the public cloud will minimize disclosure risks, but will also limit query processing capabilities on the public cloud. Alternatively, storing data using full-homomorphic encryption [3] will allow processing any class of queries while minimizing the risks of sensitive data exposure. However, it increases computational overheads (and hence monetary costs) since it is practically infeasible as the data size increases. A *disclosure-limited indexing* technique such as *bucketization* [4] allows processing a large class of queries at reasonable monetary costs and performance, albeit with potentially higher data exposure risk compared to full-homomorphic encryption. A plain-text representation, of course, minimizes overheads but incurs the highest risks of disclosure. The framework

[1] Services such as S3 allow encrypted storage at no additional costs [1] ensuring protection for data at rest, however, the data will be in cleartext form when in memory and hence susceptible to memory attacks [2].

[2] Alternate definitions of the problem are equally feasible and our framework allows for them to be studied as well.

we develop in this paper to explore tradeoffs between costs, performance, and disclosure control is largely agnostic to the specificity of the underlying data representation. Of course, the specific cost models used to compute performance, costs, and risks depend upon the representation and need to be modeled and input into the framework appropriately. While our framework is general, we use bucketization to store sensitive data on the public side for concreteness. Bucketization is a powerful framework that allows a range of possibilities from clear text representation (when buckets are very granular, with each value corresponding to a bucket) to a completely secure representation (when all values are mapped to a single bucket). We note that encryption-based approaches (such as homomorphic techniques) can also be supported in our framework, if and when they become practical.

We note that the data partitioning problem is very well studied in the database literature in both the parallel and distributed context [5, 6, 7, 8]. We compare and contrast the partitioning problem that results in the context of hybrid clouds with the previously developed state-of-the-art approaches in the related work section. Like prior partitioning problems, our problem is also NP-Hard. The paper explores greedy heuristics that are extensively evaluated over standard TPC-H benchmark datasets under various parameter settings and cloud side data representations.

Our primary technical contributions are listed below:

- We formalize the optimal risk-aware data partitioning problem as a mechanism for query cost minimization. Our formalization allows us to plug in different disclosure risk models for various use cases. We provide an algorithm to derive good partitions for both, vertical as well as the horizontal cases. Specifically, we develop an algorithm that searches for an optimal horizontal/vertical partitioning scheme given a query workload and monetary cost constraints (resource allocation and sensitive data disclosure).
- We present a formal model for estimating the cost of SQL queries in a hybrid cloud setting and develop techniques for their execution.
- We conduct extensive evaluation on realistic datasets and experimentally validate the benefits of our algorithm.

The rest of the paper is organized as follows: Section 2 presents an overview of our Hadoop HDFS and Hive based architecture for creating a hybrid cloud. In section 3, we present details on the formalization of the data partitioning problem while in section 4 we present an iterative solution to the problem that covers the horizontal and vertical partitioning approaches. Section 4 also provides a summary of our query execution and estimation model that is used in solving the data partitioning problem. In section 5, we present results of our experimental evaluation of the horizontal and vertical partitioning strategies using the TPC-H benchmark. Section 6 reviews related work in the area of secure distributed data processing. Finally, we present our conclusions and future work in section 7.

2 System Architecture

We begin by presenting an overview of our proposed system architecture in Figure 1. The system consists mainly of two components: *data design* component responsible for optimal partitioning in the hybrid cloud, and *query processing engine* that, given a partitioning, decides on the query execution strategy. Our focus in this paper is on the data design component of the system, though, as will become clear, cost estimation required to determine optimal partitioning depends upon the query processing strategies implemented by the query processing engine.

For the data design component, a user begins by submitting a set of relations, $R = \{R_1, R_2, \cdots, R_m\}$, a query workload, $Q = \{Q_1, Q_2, \ldots, Q_k\}$, and a set of resource allocation and sensitive data disclosure constraints, C. The system initially performs the task of statistics collection over R and Q using the *statistics gathering module*. This module also generates a set of predicates, P, based on R, Q and a user-specified partitioning strategy (horizontal or vertical). The statistics SR are created as equi-width histograms and sent to the *data design layer*. The *data design layer* receives the set P and the statistics SR as well as the constraints C, and then systematically solves the data partitioning problem, *DPP*. In solving *DPP*, the *query cost estimation*, *QCEst* component of the system is used to estimate the execution costs of queries $Q_i \in Q$. The *QCEst*, in turn, estimates the execution costs of queries by determining the best plan for their execution by using the query processing engine. At the end, this layer outputs two

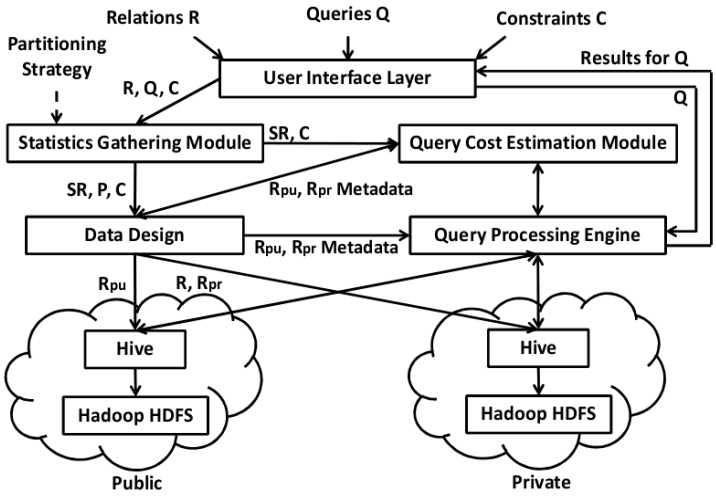


Figure 1: The Hybrid Cloud Architecture

partial replicas of R, $R_{pr} = \{R_{1_{pr}}, R_{2_{pr}}, \cdots, R_{m_{pr}}\}$ and $R_{pu} = \{R_{1_{pu}}, R_{2_{pu}}, \cdots, R_{m_{pu}}\}$ that correspond to the private and public replicas of R.

The private cloud stores R as well as its private side replica R_{pr}, while the public cloud stores only the public replica, R_{pu}. R_{pr} and R_{pu} are constructed by partitioning set P into private (P^{pr}) and public (P^{pu}) partitions respectively. Therefore, throughout the paper, a reference to "partitioning" denotes a partitioning of set P, while "replication" denotes the replication achieved by storing R_{pr} and R on the private side and moving R_{pu} to the public side. Finally, the data design layer also mandates how sensitive data in R_{pu} will be stored on the public cloud.

After the data design layer has determined the replicas (R_{pr} and R_{pu}) and the representation (cleartext or bucketization), the sensitive data in R_{pu} is stored on the public cloud in the decided representation. On the other hand, the non-sensitive data in R_{pu} and R_{pr} is stored in the cleartext format on the public and private clouds respectively.[3] The metadata about replicas R_{pr} and R_{pu} is also sent to the query processing engine. The details of constructing P as well as solving DPP will be described in Section 3, whereas information on how SR is created will be given in Section 5.

Once the system has stored the data based on the output of the data design component, the system can support query processing. This is achieved by the *query processing engine, QPE* which is responsible for executing queries based on the way data is stored in the hybrid clouds. Given a query Q, the QPE transforms Q into a execution plan using rewrite rules and estimates costs of different strategies for executing Q[4].

3 Data Partitioning Problem

In this section, we formalize the data partitioning problem in a hybrid cloud setting. The problem aims to minimize the execution cost of a query workload and is bounded by two separate constraints, the first of which limits the resources that can be rented on a public cloud, while the second captures the disclosure risk that a user is willing to accept when sensitive data on the public side is exposed. The solution to the problem results in a partitioning of data between the public and private sides. We model such partitioning using predicates as is discussed next.

[3]We use a Hadoop HDFS based infrastructure for implementing the storage schemas. Hadoop HDFS is a distributed file system designed to run on commodity hardware.

[4]We use a Hive based infrastructure for query processing. Hive is a data warehouse built on Hadoop that allows a user to define structure for files stored in the underlying HDFS.

3.1 Predicate Partitioning Model

In this paper, we use simple predicates as a foundation on which we can implement the horizontal and vertical partitioning strategies that form the basis of our solutions to the data partitioning problem. The use of predicates allows us to represent different data partitioning options (between the public and private clouds) within the same common framework. Predicates offer a general way to represent various partitioning strategies *viz.*, horizontal or vertical, using a common mechanism instead of developing separate notations for them. Additionally, in contrast to fixed strategies such as round-robin or hash partitioning, predicates offer us a fine-grained and adaptive strategy for controlling what sensitive data gets replicated on the public cloud. Previous techniques to policy specification for access control and privacy have also considered a similar approach, e.g., [9] used a predicate based approach for specifying access control policies in relational databases, [10] used predicated queries to specify confidentiality policies for structured data. In our data partitioning problem, the set of simple predicates P is extracted from relations R and workload Q. A simple predicate, P_i, is defined in one of the following three forms:

1. $P_i \leftarrow Attribute$
2. $P_i \leftarrow Attribute\ op\ Value$
3. $P_i \leftarrow (P_i \wedge P_i)|(P_i \vee P_i)|(\neg P_i)$

where *op* includes $\{=, <, >, \leq, \geq\}$. Predicates of type 1 above can be used to specify vertical partitioning, while predicates of types 2 and 3 can be used for horizontal partitioning. We illustrate how simple predicates can be extracted to specify horizontal and vertical partitionings using examples below. Consider the relations *Student* (R1), *Catalog* (R2), *Instructors* (R3) *Offering* (R4) and *Enrollment* (R5) discussed in the introduction and the following query (Q1) that retrieves names of all students who received at least one 'A' grade in Winter 2011:

```
Q1: SELECT DISTINCT name FROM Student JOIN Enrollment
      ON student_id = s_id
      WHERE term = "Winter 2011"
      AND grade = "A"
```

Vertical Partitioning: To specify a vertical partitioning policy, a simple predicate P_i is created for each attribute of relations R1-R5 using predicates of type 1. The set P will consist of predicates such as $P_1 \leftarrow name$, $P_2 \leftarrow title$, *etc.* that have been extracted from relations R1 and R2 respectively.

Horizontal Partitioning: In this case, the set P is extracted from queries (viz., Q1) instead of relation schemas (i.e., R1-R5). We first identify predicates that appear in the where clause of queries that are of types 2 or 3 and belong to the same relation. Then, we insert them into P. Additionally, if a predicate P_i is of type 2, then a predicate $\neg P_i$ is also inserted into P. On the other hand, for a predicate P_i of type 3, if P_i is either $P_{i_1} \wedge P_{i_2}$ or $P_{i_1} \vee P_{i_2}$, then predicates P_{i_1}, $\neg P_{i_1}$, P_{i_2} and $\neg P_{i_2}$ will also be added to P. Our strategy recursively performs the same operations for P_{i_1} and P_{i_2}, until both the predicates are simple (that is, of type 2). The set P for Q1 would consist of predicates such as: $P_1 \leftarrow term = "Winter\ 2011"$, $P_2 \leftarrow term \neq "Winter\ 2011"$, $P_3 \leftarrow grade = "A"$, $P_4 \leftarrow grade \neq "A"$, $P_5 \leftarrow term = "Winter\ 2011"\ AND\ grade = "A"$ and $P_6 \leftarrow term \neq "Winter\ 2011"\ OR\ grade \neq "A"$.

Above, we have specified how predicates can be used to represent both horizontal and vertical partitioning strategies. We note that a predicate based specification can capture a much larger set of hybrid strategies as well, that combine both horizontal and vertical partitionings. However, we restrict our solutions to only consider either horizontal or vertical partitioning leaving more complex policies (which significantly increase the complexity of the problem) to future work.

3.2 Data Partitioning Problem Formalization

Given the predicate based representation of partitioning, we can model the data partitioning problem (DPP) as an optimization problem whose aim is to partition a set of simple predicates, $P = \{P_1, P_2, \ldots, P_l\}$, over a hybrid cloud such that the total execution cost of workload Q is minimized.

DPP Problem Definition: The data partitioning problem (DPP) is constructed as an optimization problem that finds a simple predicate set P' where $P' \subseteq P$,

$$\begin{aligned}
\underset{P'}{\text{minimize}} \quad & \sum_{i=1}^{n} freq(Q_i) \times QPC_{Q_i}(P') \\
\text{subject to} \quad & store(P') + \sum_{i=1}^{n} freq(Q_i) \times (comm_{Q_i}(P') + proc_{Q_i}(P')) \\
& \leq PRA_COST \\
& sens(\bigcup_{p \in P'}) \times dis_cost \leq DIS_COST
\end{aligned}$$

where $freq(Q_i)$ denotes the access frequency of a query Q_i while $QPC_{Q_i}(P')$ denotes the query processing cost of Q_i given that set P' is stored on a public cloud, PRA_COST represents the maximum allowable public resource allocation cost, and DIS_COST represents the maximum allowable disclosure cost. The use of a predicate partitioning model in conjunction with constraints (public side monetary cost and sensitive data disclosure risk) allows us to capture several realistic scenarios within the same framework. A few examples of such scenarios are as follows: (i) Users that are extremely averse to storing sensitive data on a public cloud possibly due to laws/regulations. (ii) Users that want to achieve a speed-up in performance and are willing to pay a price for the risk of storing sensitive data on the public side. Furthermore, such a general framework also enables us to study different tradeoff's that exist within the problem domain in a systematic fashion.

Special Cases: The horizontal and vertical partitioning problems are specific cases of the general problem defined above. The set of predicates P is constructed differently for each of these cases as was presented previously.

Complexity of DPP: The DPP problem in general is NP-Hard as the 0-1 Knapsack Problem can be reduced to the DPP problem. A proof sketch is given in Appendix A.

3.3 Cost Metrics

The formalization of the DPP problem above refers to three different performance metrics – query processing cost (i.e., performance), monetary costs and disclosure risk. We now discuss these metrics in some detail:

Performance (QPC): Query performance depends upon the strategy used for query execution. Given a particular query plan, we can estimate the performance overheads using standard cost estimation techniques for distributed query processing. The cost comprises of the cost of total computation at both the public and private clouds as well as the network cost of data exchange between the private and public clouds. We discuss the query cost estimation technique in Section 4.3.

Monetary costs: All cloud providers typically support competitive pricing models and provide different service level agreements (SLA's) for data storage and processing services. For example, Amazon Web Services (AWS)[5] provides a tiered pricing model where, the amortized prices become cheaper as more data and processing services are used. AWS also provides SLA's for Elastic Compute Cloud (EC2) or Simple Storage Service (S3) that return a user between 10-25% of their monthly fee if Amazon fails to meet their commitment of at least 99% up time. Monetary costs can be controlled by limiting the data and processing outsourced to the public side, and therefore, we use this metric as a constraint in our problem, since public cloud services will usually be limited by an operational expenditure (OpEx). We compute the cost associated with a public side predicate set P' as a function of the following three components: a) $comm_{Q_i}(P')$: The **communication cost** for data items d satisfying some predicate $p \in P'$ where d is required to answer query Q_i. b) $store(P')$: The **storage cost** for data items satisfying any predicate $p \in P'$. c) $proc_{Q_i}(P')$: The **processing cost** for data items d satisfying some predicate $p \in P'$ where d is required to answer query Q_i. Furthermore, we ensure that the total monetary cost for P' is limited by a maximum public resource allocation cost, PRA_COST.

Disclosure risk: Disclosure risk is an important issue for organizations that deal with sensitive data, since in the event that they lose such sensitive information, they will be required to pay compliance fines as well as possible litigation expenses [11][6]. One way to measure disclosure risk is in monetary

[5] http://aws.amazon.com

[6] See Accenture's Technology Vision 2011 report, "Data Privacy Will Adopt a Risk-based Approach" section.

terms. According to separate reports, organizations may need to spend between 90-1000$ per lost record to cover various fines and expenses [12, 13]. An organization would necessarily want to limit these expenses, therefore, we model the risk as a constraint in the data partitioning problem. For our problem, we estimate the total disclosure cost over the set of predicates (P') in a public side partition as a product of the number of sensitive tuples over the predicate set P' (computed as $sens(\bigcup_{p \in P'})$ since there may be overlapping predicates) times the per-tuple unit disclosure cost (dis_cost). Additionally, the disclosure risk depends on the data representation used to store sensitive data, which is fixed in our data partitioning problem. Also, our problem ensures that the computed disclosure cost is bounded by a user-defined value, DIS_COST[7].

We observe that the above two constraints can be merged in a principled fashion into a single constraint, since both are modeled as monetary costs. This will, however, require us to normalize the two monetary costs into a single metric. We leave such an undertaking for the future. We would also like to stress that our model can incorporate other risk assessment techniques by modifying the disclosure cost for a predicate p_j.

4 Solution to the Data Partitioning Problem

Our solution approach to the data partitioning problem first partitions the set of predicates into initial public and private side partitions. Then, we iteratively create different partitions and estimate the execution cost of the query workload for each of these partitions and check whether any monetary and disclosure risk constraints are violated or not. The execution cost estimation process makes use of the query processing model discussed in subsequent subsections. Finally, we choose the best partition as the one that minimizes the execution cost without violating the given constraints.

4.1 Greedy Approach to Finding the Best Partition

Given the exponential number of possible partitions (both for vertical and horizontal partitioning), we use a hill climbing approach to finding the best partition. We present the iterative algorithm that solves DPP for horizontal and vertical partitioning next.

Algorithm 1 builds an initial solution (seed) in which the public side predicate set is empty (line 1), ensuring that we start with a feasible solution. This solution corresponds to storing the entire data and computing all queries on the private side. Then, an initial estimate of execution cost for workload Q is computed (line 2). Next, we iterate until we do not get any performance gain by moving predicates to the public side or a max number of iterations (equal to $P.size$) is reached. In each iteration, a loop determines the predicate that brings the most gain from amongst all the remaining private side predicates ($bestPredicate$). In this loop, a predicate ($p \in P$) is copied one at a time to the public side (line 8). If the transition of p to the public side does not violate the disclosure risk constraint, then we estimate the execution cost of Q ($queryCost$ in line 10) as well as the monetary cost associated with moving p to the public side ($totalMoneyCost$ in line 11). If the estimated query execution cost is lower than the current minimum execution cost ($curMinCost$) and the monetary cost constraint is also not violated, then we update $curMinCost$ with $queryCost$ (line 13) and we set $bestPredicate$ to p (line 14). Finally, after the loop terminates, if we have found a new $bestPredicate$ then we move that predicate to the public side (line 20) and proceed to the next iteration. On the other hand, if no such $bestPredicate$ is found, the algorithm exits and returns the set of predicates on the public side (P^{pu}).

Note that the above algorithm is general in the sense that it applies to both horizontal and vertical partitioning; the only difference being the construction of predicate set P during the statistics collection process. In vertical partitioning, P contains one predicate for each attribute of the relations, while in

[7]In this paper, we have restricted our attention to a framework that selects optimal data partitioning given a fixed approach for representing the data. Different choices of representation offer different levels of information disclosure -e.g., a cleartext representation reveals the sensitive data completely, while bucketization, based on an information hiding technique, offers a higher level of protection and encrypted representation offers the most protection. Furthermore, these different representations have implications on both, monetary costs and performance of queries. An extension of our framework that jointly optimizes the partitioning and selects the right representation for different data items is an interesting, and a non-trivial, direction of future work. One of the main complexities is the lack of concrete comparisons of different information hiding approaches in terms of the security they offer.

Algorithm 1 ITERATIVE()

Input: P, Q, PRA_COST, DIS_COST, dis_cost
Output: P^{pu}

1: $P^{pu} = \emptyset$ {Initial solution}
2: $curMinCost \leftarrow$ QWC-EST(Q, SR, P^{pu}) {Initial cost}
3: $isGain \leftarrow true; ctr \leftarrow 1$
4: **while** $isGain$ AND $ctr++ < P.size$ **do**
5: $isGain \leftarrow false$ {Checks if there is a gain at each pass}
6: $bestPredicate \leftarrow \emptyset$ {Keeps the predicate brings the most gain}
7: **for** $Predicate\ p \in P$ **do**
8: $P^{pu} \leftarrow P^{pu} \cup p$
9: **if** $sens(\bigcup_{p \in P^{pu}}) \times dis\_cost < DIS\_COST$ **then**
10: $queryCost \leftarrow$ QWC-EST(Q, SR, P^{pu})
11: $totalMoneyCost \leftarrow store(P^{pu}) + \sum_i^n freq(Q_i) \times (comm_{Q_i}(P^{pu}) + proc_{Q_i}(P^{pu}))$
12: **if** $queryCost < curMinCost$ AND $totalMoneyCost < PRA\_COST$ **then**
13: $curMinCost \leftarrow queryCost$
14: $bestPredicate \leftarrow \{p\}$
15: $isGain \leftarrow true$
16: **end if**
17: **end if**
18: $P^{pu} \leftarrow P^{pu} - p$
19: **end for**
20: $P^{pu} \leftarrow P^{pu} \cup bestPredicate$
21: $P \leftarrow P - bestPredicate$
22: **end while**
23: **return** P^{pu}

horizontal partitioning, P contains predicates derived from the queries in the query workload using the rules specified in Section 3.1.

The algorithm above requires us to determine various costs (viz., disclosure, monetary, and query execution) for a given workload of queries Q for a given partitioning P_i. We note that disclosure costs, in our model, depend entirely on the partitioning P_i and are independent of the query workload. Disclosure cost can simply be computed as $sens(\bigcup_{p \in P^{pu}}) \times dis\_cost$. Determining query execution costs and monetary costs, however, depends upon the query workload. They can both be estimated as the sum of costs of individual queries[8].

Thus, to fully specify our approach, we need to further describe the query costs (both performance, and monetary) for a given query Q_j over partition P_i. Such a cost estimation depends upon the query execution strategy, which, in turn, depends upon the underlying data representation. Therefore, we next discuss the query processing strategies over a hybrid cloud, followed by our mechanism to estimate costs.

4.2 Query Processing

In this subsection we briefly outline how query execution proceeds in our hybrid cloud setting. Before a given query Q_p is executed, we will already have a fixed partitioning of predicate set P (based on Algorithm 1 in Section 4.1). Then, using the methodology of our query cost estimation module (Section 4.3), we first generate private-only and public-most strategies for Q_p. Then, we select the best strategy out of them in terms of execution cost, and use this strategy to execute Q_p. We note that our query execution strategy is simplistic, however, the goal of this paper is data partitioning in a hybrid cloud setting, rather than building a complete query execution and optimization engine for a hybrid cloud. In addition, we note that the problem of query optimization has been addressed in the context of single-cluster systems [15], but not in the context of hybrid clouds. The query processing strategy for a given query Q_p in our framework depends on the underlying data representation used.

[8]Recent work has explored techniques such as shared scans in the context of executing queries over MapReduce frameworks [14] which can reduce costs of query workloads. We, however, do not consider such optimizations in developing our partitioning framework in this paper.

For concreteness, we use bucketization as a flexible security mechanism while allowing approximate query answering to be performed by a public cloud. This representation is therefore appropriate for users that want to limit their disclosure cost while performing some amount of query processing on a public cloud. In addition, varying the number of buckets for a predicate provides a tradeoff between performance *vs.* disclosure cost.

Note that, Encrypted and Cleartext representations can be simulated using bucketization in which the buckets for each attribute are set to 1 and number of unique values, respectively.

We illustrate our query processing strategy for a hybrid cloud through the following example. Consider the following relational schema and query q:

```
S(A string, B int, C string)
T(A date, B int, C int, D int)

Query q: SELECT S.A, T.A, T.C
         FROM S, T
         WHERE S.B = T.C AND T.A < 1995-03-15 AND S.C = "FRANCE"
```

$S.B$ and $T.C$ are primary keys in S and T, while $T.D$ is a foreign key in T. We assume that S and T are horizontally partitioned into S_{pr}, S_{pu}, T_{pr} and T_{pu}. Note that S_{pr} and T_{pr} are the private cloud replicas, while S_{pu} and T_{pu} are the public cloud replicas of S and T. The details on creation of S_{pr}, S_{pu}, T_{pr} and T_{pu} have already been discussed in Section 3. Also, both S and T are stored along with the replicas in the private cloud in case q is executed entirely on that side. Furthermore, we assume that attribute A of relation T is sensitive and is stored in a bucketized representation on the public side. Therefore, conditions involving $T.A$ need to be mapped according to the bucketization schema after which they can be applied onto the data. We now describe each of the three stages used in query execution:

a) **Private-only Plan Generation**: At this step, an initial query plan (*Private-only Plan*, τ_{pr_on}) which could completely run on the private side and doesn't involve any public side processing will be created. Figure 2(a) shows the private-only plan for q which is always a valid plan since the private cloud always contains the original dataset R.

b) **Public-most Plan Generation**: The plan (*Public-most Plan*, τ_{pu_mo}), that migrates the largest amount of processing to the public side amongst the candidate query plans, without allowing multi-round data transfer between the two sides (i.e. base data or generated intermediate data can only be transferred from public to private clouds) will be produced. τ_{pu_mo} is created by successively applying query rewrite rules to the initial Private-only Plan (Appendix B provides some details on query rewrite rules). Figure 2(b) shows the public-most plan for q produced through the private-only plan from Figure 2(a). In the case of bucketization, when results of the public query are transferred to the private side, additional decryption and filtering of false positives may be required before executing the combination query. The operator D in the figure denotes the combination of these tasks.

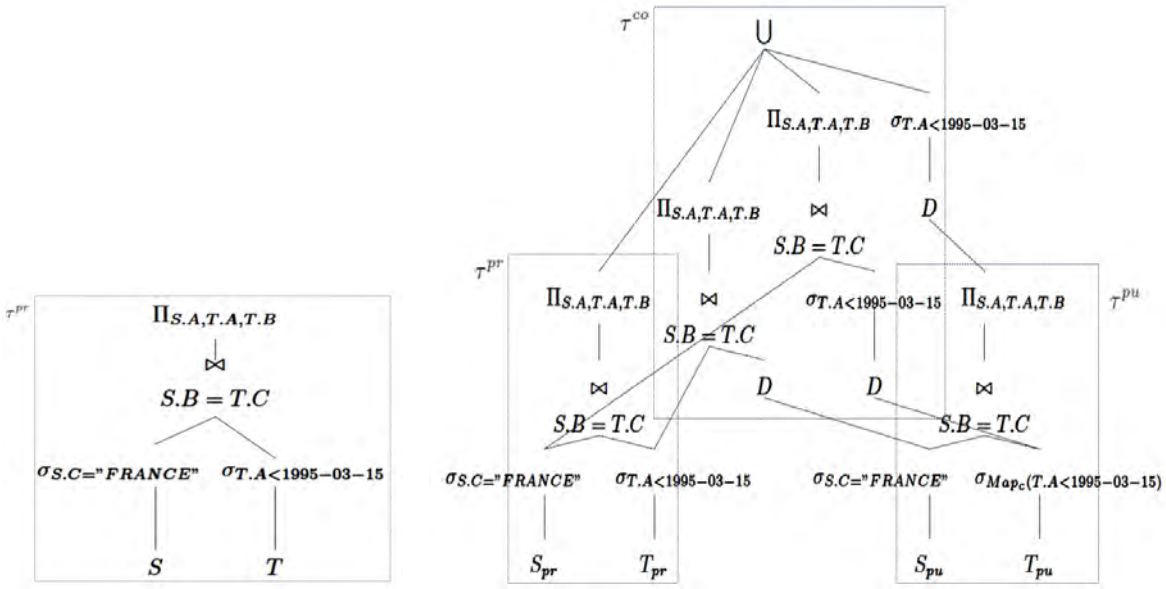
c) **Best Strategy Selection** : We now select the cheaper of the Private-only Plan and Public-most Plan to be used as our strategy for executing query q over the hybrid cloud. While these two plans might not be optimal, they represent two extremes in terms of monetary cost and disclosure risk. Namely, the private-only plan has the least monetary cost and disclosure risk, whereas the public-most plan is the one that has the largest monetary cost and disclosure risk. Therefore, they provide a reasonable estimation of execution costs for queries over a hybrid cloud, and moreover, they give acceptable solutions to the data partitioning problem[9].

4.3 Query Workload Cost Estimation

Estimating costs for a query workload requires us to estimate the cost of each individual query Q_i. Having discussed query processing in the previous section, we next discuss mechanisms to estimate query costs.

The execution cost of a query plan over a hybrid cloud can be measured in terms of the total running time required to execute that plan. However, in this paper we will use the I/O size of a query plan as a substitute for the running time in the estimation process. Previous approaches to query cost estimation in a cloud environment have also used a similar approach, e.g., Afrati *et al.* [16] use an I/O

[9]We leave the incorporation of complex plans and more accurate cost estimation techniques for the future.



(a) Private-only Plan (τ_{pr_on})

(b) Public-most Plan (τ_{pu_mo})

Figure 2: A join query execution

based data-volume cost model to evaluate different algorithms for executing query plans on a cluster. In another paper, Wu *et al.* [15] propose an I/O based cost model that is used to evaluate the performance of query plans in the MapReduce framework. In addition, they also observe that the cost model based on query response time does not improve the accuracy of estimation by much over a model that uses I/O size. The I/O size of a query plan τ in a hybrid cloud can be estimated as:

$$c_\tau = (max(S^{pu} + w_2 \times S^{tr}, w_1 \times S^{pr}) + w_1 \times S^{co}), \tag{1}$$

where S^{pu} and S^{pr} are the estimated I/O sizes on the public and private cloud respectively, S^{tr} is the estimated transfer size of the intermediate results from public to private cloud, and S^{co} is the I/O size required to combine intermediate results at the private side. The weights w_1 and w_2 account for differences in computational resources between a public and private cluster (details are given in Section 5).

In order to accurately calculate S^{pu}, S^{pr}, S^{tr} and S^{co} for any given plan τ, τ will be split into 3 subplans: *public cloud subplan* (τ^{pu}), *private cloud subplan* (τ^{pr}) and *combination subplan* (τ^{co}). The subplan τ^{pu} is that part of the plan τ that is executed over data in the public cloud, whereas τ^{pr} is executed directly over data in the private cloud. Also, τ^{pu} is executed in parallel with τ^{pr}. Further, τ^{co} is used to combine the intermediate results generated by τ^{pu} and τ^{pr} into the final result. Additionally, only τ^{pu} is executed within the public side while τ^{pr} and τ^{co} are executed at the private side. We also observe that τ^{pr} and τ^{pu} need to be executed before being able to process τ^{co}. Splitting the private-only plan τ_{pr_on} is quite simple; since the entire plan is only a private cloud subplan. (i.e. $\tau^{pr}_{pr\_on} = q$, whereas $\tau^{pu}_{pr\_on} = \tau^{co}_{pr\_on} = \emptyset$). On the other hand, dividing the public-most plan may be challenging. To overcome this challenge, we use the partitioning of set P as well as query rewrite rules to divide the public-most plan into its constituent elements. Figure 2(a) and Figure 2(b) illustrate how the private-only and public-most plans for query q are divided into a public cloud, private cloud and combination subplans.

The I/O sizes for the various components of the plan τ are computed using statistics SR and the partitioning of P. Unfortunately, Hive does not maintain attribute level statistics for a relation. Therefore, we implemented a statistics gathering module that analyzes the data for a relation once, and maintains statistics for that relation and its attributes using separate histograms for unencrypted (using partitions) and encrypted (using buckets) versions of attributes. Then, at run-time the I/O size for a query is estimated using histograms over partitions for non-sensitive attributes and histograms over buckets for sensitive attributes.

Algorithm 2 performs the strategies proposed earlier to compute the cost of each query Q_i by applying Equation 1 over the generated τ_{pr_on} and τ_{pu_mo} for Q_i. Further, Algorithm 2 sums up the individual

Algorithm 2 QWC-EST()

Input: Q, SR, P^{pu}
Output: Query Workload Cost, c

1: $c \leftarrow 0$
2: **for** $i \leftarrow 1$ **to** $Q.length$ **do**
3: $c_i \leftarrow \infty$
4: $plan\_list \leftarrow \emptyset$
5: Generate τ_{pr_on} and τ_{pu_mo} for Q_i
6: $plan\_list.add(\tau_{pr\_on}); plan\_list.add(\tau_{pu\_mo})$
7: **for** each candidate plan $\tau \in plan\_list$ **do**
8: $resSize \leftarrow 0$ {Output Size of plan τ}
9: $S^{pu} \leftarrow 0$ {I/O Size of public cloud subplan}
10: $S^{tr} \leftarrow 0$ {I/O Size of Transferred Results }
11: $S^{pr} \leftarrow 0$ {I/O Size of private cloud subplan}
12: $S^{pr}_{op} \leftarrow 0$ {Output Size of private cloud subplan}
13: Divide τ into τ^{pr}, τ^{pu} and τ^{co}
14: Estimate S^{pu}, S^{tr}, S^{pr}, S^{pr}_{op} and $resSize$ using SR and P^{pu}
15: $S^{co} \leftarrow |S^{tr} + S^{pr}_{op} + resSize|$
16: $c_{tau} \leftarrow max(S^{pu} + w_2 \times S^{tr}, w_1 \times S^{pr}) + w_1 \times S^{co}$ {Total execution cost for plan τ}
17: **if** $c_{tau} < c_i$ **then** $c_i = c_{tau}$ **end if**
18: **end for**
19: $c \leftarrow c + freq(Q_i) \times c_i$
20: **end for**
21: **return** c

query costs to return the overall execution cost of Q. We begin by creating candidate plans (private-only or public-most plan) for Q_i, and then we estimate the cost of each of these plans (lines 5-17). The algorithm first initializes various I/O cost variables for a plan τ to 0 (lines 8-12). Next, we divide the plan τ into τ^{pr}, τ^{pu} and τ^{co} (line 13) after which values for the variables are estimated using SR as well as the given partitioning P^{pu} (lines 14-15). Note that, for a private-only plan $S^{pu} = S^{co} = S^{tr} = 0$. The execution cost of τ is computed using Equation 1 (line 16) and is compared with the current minimum cost plan value for Q_i. Finally, the overall cost of Q is computed as a sum of each Q_i's minimum cost multiplied with $freq(Q_i)$ and is returned as the output (lines 19-21). Our current work supports simple SPJ queries while we leave the support of nested queries as future work.

Additionally, as we said earlier our architecture should be able to compute the monetary cost of the query workload for a fixed partitioning P'. Our system calculates this monetary cost as the summation of the monetary cost associated with storing P' on the public side (i.e. $store(P')$) and the total processing monetary cost ($TPMC$) for the workload over the partition P' (i.e. Monetary Cost of Q for $P' = store(P') + TPMC$). $TPMC$ is computed by adding the monetary cost for processing each individual query Q_i. The monetary cost for processing Q_i is equal to the sum of the monetary cost related to transferring results from a public to a private cloud during Q_i's execution, $comm_{Q_i}(P')$, and the actual monetary cost of processing Q_i over the public cloud, $proc_{Q_i}(P')$. As a result, $TPMC = \sum_{i=1}^{n} comm_{Q_i}(P') + proc_{Q_i}(P')$ and so monetary cost of Q for $P' = store(P') + \sum_{i=1}^{n} comm_{Q_i}(P') + proc_{Q_i}(P')$.

5 Experimental Results

This section presents results of experiments that we conducted to compare the performance of our partitioning algorithm for horizontal and vertical partitioning. We first present details of our setup followed by the experiments.

Experimental Setup: We conducted experiments on two local clusters setup on different subnetworks of the same intranet. The first cluster consists of 9 nodes (used as private cloud) while the second consists of 14 nodes (used as public cloud). Each node consists of a Pentium IV processor with 290GB-320GB disk space and 4GB of main memory. Both clusters are setup using Hadoop v0.20.2 and

Hive v0.6.0.

Statistics collection: The statistics gathering module analyzed the 100GB TPC-H dataset and generated equi-width histograms for every attribute of TPC-H relations. We used the datatypes, int, double and string in Hive to represent TPC-H data. We also created a datatype 'date' that allows us to represent various dates from the TPC-H schema. The number of partitions used in a histogram is dependent on the datatype; this number is fixed for a given datatype: (i) For integers and doubles, the number of partitions $= log_2(max - min)$, where min and max represent the min and max domain values mandated by TPC-H. (ii) For dates, since TPC-H only allows dates between '1992-01-01' and '1998-12-31', we created one partition for each year from 1992 through 1998. (iii) For strings, we created 95 partitions that cover alphabets ($a - z$ and $A - Z$), digits ($0 - 9$) and all special characters (!, @, #, *etc.*).

Bucketization parameters: In our experiments, the number of buckets, B, is a user-configurable parameter that can be varied from 1 to the number of unique values in the domain of an attribute. We also used AES in CTR mode to encrypt subsets of attributes in our experiments.

Query Workload: We have used the TPC-H benchmark with a scale factor 100 in our experiments. We used a query workload of 60 queries containing modified versions of TPC-H queries Q1, Q3, Q6 and Q11. In particular, we do not perform grouping and aggregate operations in any query. Further, we assumed that each query was equally likely in the workload. The predicates in each of the queries are randomly modified to vary the range (as mandated by TPC-H) of the data that is accessed.

Estimation of weights w_1 and w_2: The weight w_1 is calculated as a ratio between the relative processing time of a private cloud to that of a public cloud. We estimated the relative processing times of our public and private clusters by running all 22 TPC-H queries for a 300GB dataset on them. Then, w_1 is computed as the average ratio of the running times of queries on the private cluster to that on the public cluster. In this way, w_1 was estimated to be 1.624. Another interesting observation to be made here is that the value 1.624 is very close to the ratio of number of nodes used in our public cloud to that in our private cloud ($14/9 \approx 1.556$). A value greater than 1 indicates that our private cloud is slower than our public cloud. Additionally, w_1 is used for the private cloud as well as the combination components of a query since they both run on the same private cluster. The estimation of w_2 is done in a similar way. However, since S^{tr} is the estimated transfer size, we need to account for the network bandwidth when computing w_2. Therefore, w_2 is computed as a ratio of the inverse of the network bandwidth to the processing time on a public cloud. The previously computed processing time was reused. The network bandwidth was taken to be $\approx$ 672KB/sec since this is the bandwidth available on our clusters. In this way, w_2 was estimated to be 61.481. This clearly suggests that in the time a unit of data is transferred from our public to private cloud (a unit is 1 byte, 1MB and so on), the public cloud processes ≈ 61 times more data. This confirms the intuition that we should move processing to the public side rather than transferring relations and computation from public to private clouds.

Computation of resource allocation cost: The resource allocation cost was computed using unit prices from Amazon Web Services. We specifically used Amazon S3 pricing to determine storage (\$0.140/GB + PUT) and communication (\$0.120/GB + GET) costs, where the price for PUT and GET operations is \$0.01/1000 requests and \$0.01/10000 requests respectively. Also, we used Amazon EC2 and EMR pricing to calculate the processing cost (\$0.085 + \$0.015 = \$0.1/hour). Finally, the total public cloud resource allocation cost was computed as, $PRA_{pu} \approx 77K$, using the previously given values.

Computation of disclosure cost: The sensitive data disclosure cost can be computed as:

$$DIS\_COST^{P'} = sens(\bigcup_{p \in P'}) \times dis\_cost$$

$$dis\_cost = UNIT\_COST \times Pr(\text{data is disclosed}|\text{data repr.})$$

where $sens(\bigcup_{p \in P'})$ is the number of sensitive tuples released for a public side predicate set, which is approximated by our query estimation module. In addition, dis_cost represents the per-tuple unit disclosure cost which depends on the following factors: (i) A unit price per-tuple, $UNIT\_COST$, which we assume to be \$100. (ii) The probability that sensitive data is disclosed given that a particular data representation is used for that data. In our experiments, we considered four different data representations by bounding the buckets (B) used to store sensitive data such as names, phone numbers and addresses from TPC-H on the public side. Note that, we assume the values used are supplied to us by an oracle, since the task of determining them is beyond the scope of the current paper. We present each case along

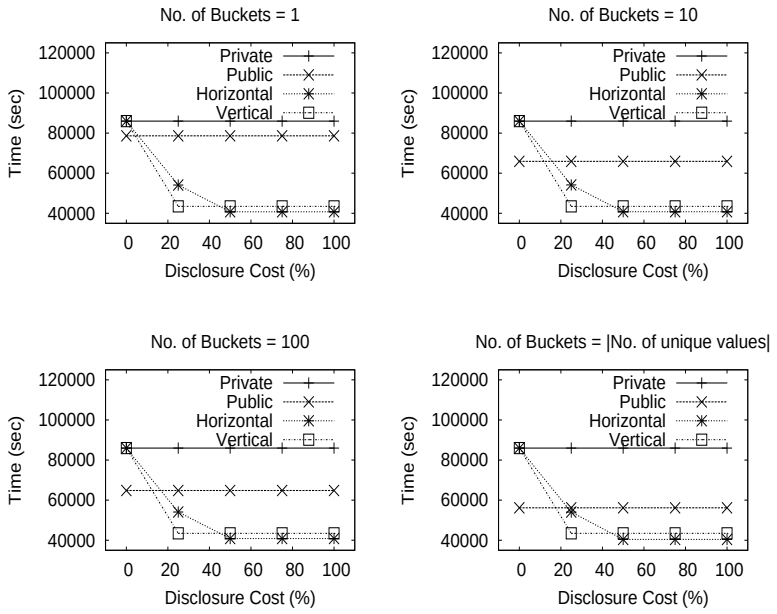


Figure 3: Comparison of partitioning strategies for a variable disclosure cost

with their probabilities (Pr): (a) Cleartext - B is set to the number of unique values for each sensitive attribute; $Pr = 10e^{-5}$; (b) Bucketized - For our experiments, this case was further divided into the following sub-cases: (1) $B = 100$ for each sensitive attribute; $Pr = 10e^{-6}$; (2) $B = 10$ for each sensitive attribute; $Pr = 10e^{-7}$; (c) Encrypted - B is set to 1 for each sensitive attribute; $Pr = 10e^{-8}$. Also, the choices of $B = 10$ and $B = 100$ were randomly made for the bucketized case. Finally, using the previously defined values, we calculated the various maximum possible public cloud disclosure costs (DIS_{pu}) as: (a) Cleartext - \$1.18M; (b) Bucketized - (1) $B = 100$: \$118K; (2) $B = 10$: \$12K; (c) Encrypted - \$1.2K.

Preliminary Experiments: For all our experiments, we first computed the running time of the query workload for the following cases: (i) All computation is performed on our private cloud (Private). (ii) All computation is performed over public cloud data using one of the four strategies presented earlier (Public). Figures 3 and 5 clearly show that the time taken to run the query workload on the public cloud is much faster than the private cloud. Moreover, there is an additional overhead involved for all cases due to decryption and filtering. This overhead increases as the number of buckets is reduced from $B =$ number of unique values to $B = 1$.

Experiments with a variable disclosure cost: The goal of these experiments is to compare for both, the horizontal and vertical partitioning strategies, the query workload performance as well as the monetary expenditure on public side resources when the acceptable disclosure cost increases (from 25% to 100% of DIS_{pu} costs) while the resource allocation cost is fixed randomly (at 50% of the PRA_{pu} cost). Figure 3 clearly shows that when a user is willing to take additional risks by storing more sensitive data on the public side, they can gain a considerable speed-up in overall execution time (almost 50% for both strategies). On the other hand, Figure 4 shows that the monetary expenditure on public side resources is substantially low even when a user takes additional risks by storing increasing amounts of sensitive data on the public cloud. Finally, from Figures 3 and 4, we also observe that for a relatively low resource allocation cost ($\approx$ 28K on average over both cases) we can gain a considerable amount of speed-up in performance ($\approx$ 50%).

Experiments with a variable resource allocation cost: These experiments measure the time to run the query workload as well as the monetary expenditure on sensitive data exposure when the acceptable resource allocation cost is increased (from 25% to 100% of PRA_{pu}) while the disclosure cost is fixed randomly (at 50% of the DIS_{pu} costs). As seen from Figure 5, when a user invests more capital towards resource allocation, a considerable gain in overall workload performance (almost 50% for both strategies) can be achieved. This is expected since when more resources are allocated on the public side,

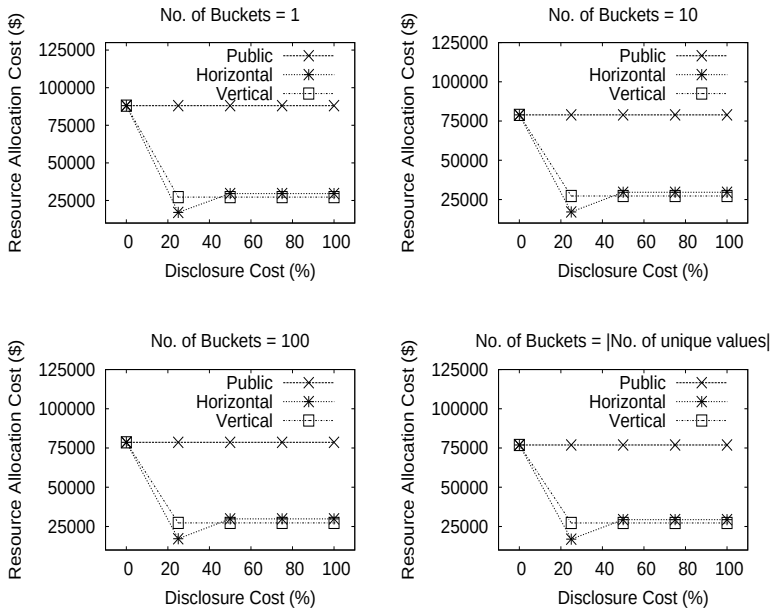


Figure 4: Comparison of partitioning strategies for a variable disclosure cost

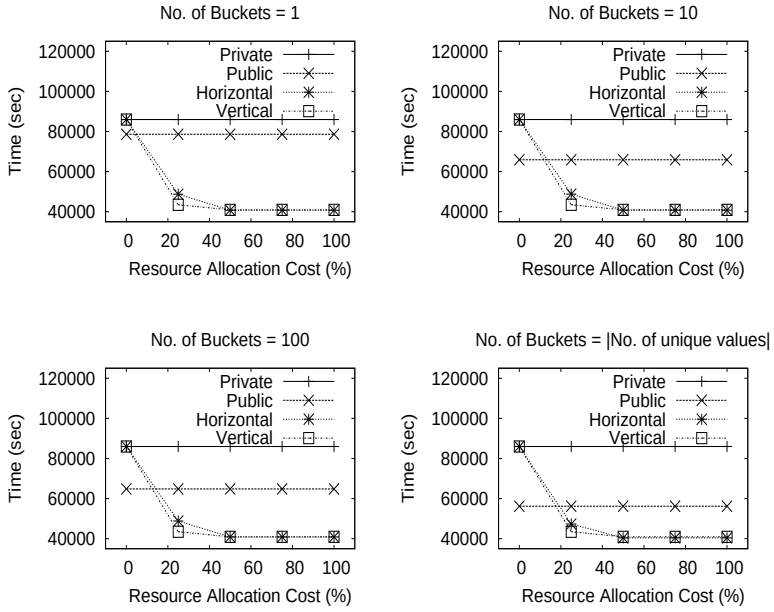


Figure 5: Comparison of partitioning strategies for a variable resource allocation cost

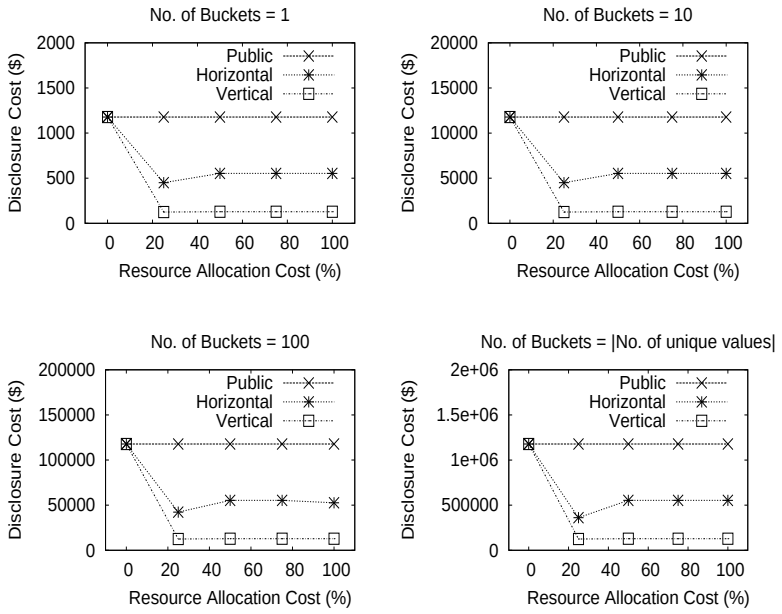


Figure 6: Comparison of partitioning strategies for a variable resource allocation cost

we are better able to exploit the parallelism that is afforded by a hybrid cloud. Thus, the intuition that a hybrid cloud improves performance due to greater use of inherent parallelism is justified. Additionally, Figure 6 shows that a hybrid cloud approach can considerably downgrade sensitive data exposure risks when compared with the outsourcing approach of moving everything to a cloud provider. Moreover, the sensitive data exposure risks continue to remain low even when a user rents more public side resources. Finally, from Figures 5 and 6, we also notice that irrespective of the data representation used, we can achieve a considerable improvement in query performance ($\approx 50\%$) for a relatively low risk ($\approx 30\%$).

Experiments with dynamic workloads: These experiments measure the robustness of our partitioning algorithm to variations in query workload characteristics. For these experiments, we initially constructed multiple query workloads (*Workload-1* to *Workload-5*) by randomly changing the predicates of all queries from the original workload. For example, a selection condition such as $l\_shipdate = 1992 - 03 - 01$ was randomly changed to $l\_shipdate = 1998 - 10 - 31$. The experiments were then conducted in the following two phases: (i) We first fixed the public and private clouds to the partitions obtained for the case, $B = 100$, $DIS_{pu} = 50\%$ and $PRA_{pu} = 50\%$. Then, the running times of the modified workloads were computed for this original partitioning (Original-Partitions). (ii) We next ran the partitioning algorithm using the modified workloads in place of the original workload to create partitions specific to these changed workloads. Next, we computed the running time of the modified workloads on these newly created partitions (Modified-Partitions). Figure 7 clearly shows that for both, horizontal and vertical partitioning, the running times for (i) and (ii) are nearly identical. This assures us of the robustness of our algorithm and further implies that partitions do not need to be changed each time a query workload is modified.

General Observations: We observe from Figures 3 and 5 that for both, horizontal and vertical partitioning, changing the number of buckets (B) has little effect on the performance time. This is because our partitioning algorithm leaves most of the sensitive data that is part of the query workload on the private side. Therefore, very little additional overhead (associated with decrypting and filtering false positives) is incurred even when B is reduced from number of unique values to 1, leading to a negligible change in query performance. Consequently, both approaches are suitable for users who place a higher weight on executing mission-critical tasks (operating on sensitive data) locally. On the other hand, Figures 4 and 6 show that for both, horizontal and vertical partitioning, there are significant monetary savings to be made even when the risks of data exposure as well as resource allocation costs increase. Therefore, both approaches are appropriate for users who want to balance the risks of sensitive

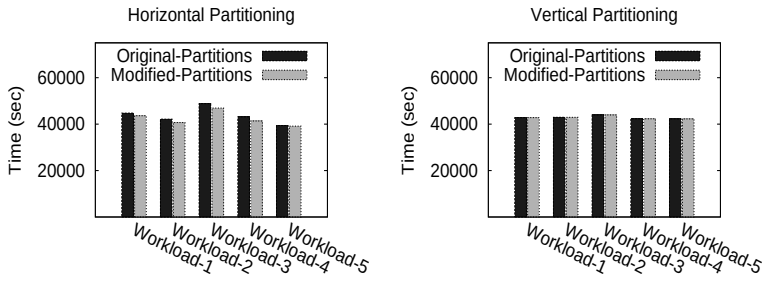


Figure 7: Comparison of partitioning strategies for dynamic workloads

data exposure and a reduced operational budget with optimal query performance.

6 Related Work

Our work builds upon a significant body of prior work on data partitioning (e.g., [7, 8, 5, 6]), distributed query processing (e.g., evolution from systems such as SDD-1 [17] to DISCO [18] that operates on heterogeneous data sources, to Internet-scale systems such as Astrolabe [19], and cloud systems [20]), and data privacy [21, 4, 22]. However, to our knowledge, this is the first paper that takes a risk-based approach to data security in the hybrid model. Also, the modeling of monetary constraints into the optimization problem at hand is a novel exploration in this domain. We summarize a few of the most relevant previous works below.

Data partitioning has been studied fairly extensively in distributed and parallel databases from a variety of perspectives, ranging from load balancing [5], efficient transaction processing [6] to physical database design [7, 8]. In [8], the authors consider the problem of workload driven horizontal data partitioning for parallel databases. They propose a two step approach wherein the first step involves candidate partition generation and the second step is partition evaluation which selects the best partition for each table with respect to the given workload. They generate partitioning "keys", which are basically sets of attributes to be considered together as keys to hash functions that a partitioner will use. They use columns that appear in join and grouping statements to generate candidate keys. In contrast, we use the predicates appearing in query statements (or predicates derived from them) as criteria for partitioning the tables across the private and public clouds which gives us greater control over deciding which particular records get replicated on the public side since disclosure risk is also a concern in our framework. Furthermore, having a predicate based partitioning makes our statistics more accurate and more efficient to compute than in [8] where they have to resort to sampling after the partitioning function is applied.

A more recent related paper [6] looks at the data partitioning and replication problem in distributed databases for supporting OLTP queries efficiently. The objective is to improve the throughput by reducing the time it takes to commit a transaction that needs to access multiple records distributed across multiple nodes of a cluster with a shared nothing architecture. The time taken to complete a transaction is dependent upon whether it accesses data on a single node or multiple nodes and therefore, reducing the number of such multi-node transactions can significantly increase the throughput. They propose a graph based data partitioning (including limited replication) approach based on a well known class of graph partitioning algorithms called METIS [23] which are known to generate balanced partitions in practice. The idea is to store all nodes within each partition at a single node and minimize the number of edge crossings between different partitions which in turn minimizes multi-node transactions. Previous approaches have used round-robin, range partitioning and hash partitioning based techniques [24] for distributed and parallel databases. However, a graph partitioning based approach proposed in [6] may not be very suitable for our setting, most importantly because of the poor scalability of graph partitioning algorithms with size of the graph. Since the graph size is proportional to the number of records in tables, such an approach is not amenable (as yet) for even medium sized databases. Also, since we are not considering typical OLTP workloads, the execution time of a query is not so dependent on it being

multi-site or single site (which is the focus in [6]) as it is on the size of the intermediate and final result which is what we consider.

The work most related to ours is Relational Cloud proposed in [21] which addresses a similar hybrid cloud problem. Relational Cloud uses the graph-based partitioning scheme described above to split data into private and public sides. The partitions are encrypted with multiple layers of encryption and stored on a server. A query is executed over the encrypted data with multiple rounds of communication between a client and server without considering the cost of decrypting intermediate relations. The difference between our work and Relational Cloud is that our data partitioning scheme explicitly considers the cost of queries over all components of a hybrid cloud: queries over data in a private cloud and queries over a mixed data representation (i.e., encrypted and plaintext) in a public cloud. To the best of our knowledge, ours is the first work to explicitly estimate the cost of querying over unencrypted and encrypted data in a distributed setting.

A paper by Aggarwal et al. [25] considered the problem of secure query processing using a distributed architecture. The authors propose a solution for secure outsourcing of relational data using two non-colluding servers. The privacy policies are stated in terms of subsets of attributes. The goal is to not give either of the two servers access to all attributes specified in a policy. The techniques employed are vertical partitioning of data along with selective encryption of some attributes. The data partitioning algorithm tries to partition the attributes across the two servers such that no set of attributes appear in plaintext on either server. While the two-server model can be mapped to our case, where the private cloud is both a server and the trusted client where the combination of partial query results take place, their model of disclosure risk is completely different from ours. While they do not allow all attributes specified in a confidentiality policy to be exposed to either one of the servers at any time, we are willing to do so in a controlled manner. This relaxation in our case, makes the fundamental solution approaches quite distinct from their solutions.

A recent paper by Ko *et al.* [26] proposes a "hybrid execution (HybrEx)" model that splits data and computation between public and private clouds for preserving privacy and confidentiality in a distributed MapReduce framework. In the paper by Chul Tak et al. [27] authors look at the economics of cloud computing and try to answer the important question of "when does it make economic sense to migrate to the cloud?" They identify a set of factors that may affect the choice of deployment (in-house, cloud, and hybrid). While, somewhat related to our work, these papers tackle different problems and we do not summarize them here due to lack of space.

7 Conclusions and Future Work

With the advent of cloud computing, a hybrid cloud is suitable for users who wish to balance data security risks with scalable processing. We have identified three challenges that must be overcome before this approach can be adopted.

The first challenge deals with data partitioning between a private cloud and a service provider when there are sensitive attributes in the data. We formalized this challenge as a risk-aware query optimization problem and presented an iterative approach that results in the construction of optimal partitions. The second challenge is how to store a user's data securely on a cloud provider. We presented three different solutions to this challenge that can be tailored to suit a user's data privacy needs. We specifically used the bucketization technique to store sensitive data as well as to push most of the query processing to the provider without the need of decrypting stored data. Finally, the last challenge addresses the problem of distributed query processing over data stored in a mixed representation (i.e., encrypted and plaintext). We proposed query rewrite rules that operate over a mixed representation for the construction of query plans over a partitioned database. In addition, we developed a cost model that estimates the cost of query execution over a mixed representation. Finally, we also presented a query processing engine that splits a user query into one or more public and private cloud queries, each of which can be executed at a site using the best available local query plan.

We are primarily exploring the following ideas for future research from amongst the various areas that we outlined throughout the paper: 1) We have considered horizontal and vertical partitioning of relations in this paper which can be extended to include hybrid partitioning schemes. 2) Our cost model only considers simple SQL queries. We plan to build a more sophisticated model with support for nested queries. 3) We used Hadoop and Hive as the underlying cloud computing technologies. We aim to extend

this work with more experiments into a generalized tool that will work with other existing public cloud services.

8 Acknowledgements

This material is based upon work supported by The Air Force Office of Scientific Research under Award No. FA-9550-08-1-0260. We thank Dr. Robert Herklotz for his support.

References

[1] Using Data Encryption. `http://docs.amazonwebservices.com/AmazonS3/latest/dev/index.html?UsingEncryption.html`.

[2] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 45–60. USENIX Association, 2008.

[3] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[4] H. Hacigümüş, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model, In SIGMOD'02. In *SIGMOD Conference*, pages 216–227, 2002.

[5] S. Ghandeharizadeh and D. J. DeWitt. Hybrid-Range Partitioning Strategy: A New Declustering Strategy for Multiprocessor Database Machines. In *VLDB*, 1990.

[6] C. Curino, E. Jones, Y. Zhang, and S. Madden. Schism: a Workload-Driven Approach to Database Replication and Partitioning. In *VLDB*, 2010.

[7] S. Agrawal, V. R. Narasayya, and B. Yang. Integrating Vertical and Horizontal Partitioning Into Automated Physical Database Design. In *SIGMOD Conference*, pages 359–370, 2004.

[8] J. Rao, C. Zhang, N. Megiddo, and G. M. Lohman. Automating physical database design in a parallel database, In SIGMOD'02. In *SIGMOD Conference*, pages 558–569, 2002.

[9] S. Chaudhuri, T. Dutta, and S. Sudarshan. Fine grained authorization through predicated grants. In *ICDE*, pages 1174–1183. IEEE, 2007.

[10] G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.*, 73(3):507–534, 2007.

[11] Accenture Technology Vision 2011 - The Technology Waves That Are Reshaping the Business Landscape. `http://www.accenture.com/us-en/technology/technology-labs/Pages/insight-accenture-technology-vision-2011.aspx`, 2011.

[12] S. Gaudin. Security Breaches Cost 90*To*305 Per Lost Record. `http://www.informationweek.com/news/199000222`, 2011.

[13] J. Vijayan. Defense Dept. hit with $4.9B lawsuit over data breach. `http://www.computerworld.com/s/article/9220874/Defense_Dept._hit_with_4.9B_lawsuit_over_data_breach?taxonomyId=13`, 2011.

[14] Tomasz Nykiel, Michalis Potamias, Chaitanya Mishra, George Kollios, and Nick Koudas. Mrshare: Sharing across multiple queries in mapreduce. *PVLDB*, 3(1):494–505, 2010.

[15] S. Wu, F. Li, S. Mehrotra, and B. C. Ooi. Query optimization for massively parallel data processing. In *SoCC*, 2011.

[16] F. N. Afrati, V. R. Borkar, M. J. Carey, N. Polyzotis, and J. D. Ullman. Map-reduce extensions and recursive queries. In *EDBT*, pages 1–8, 2011.

[17] J. B. Rothnie Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. L. Reeve, D. W. Shipman, and E. Wong. Introduction to a System for Distributed Databases (SDD-1). *ACM Trans. Database Syst.*, 5(1):1–17, 1980.

[18] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Heterogeneous Databases and the Design of Disco. In *ICDCS*, pages 449–457, 1996.

[19] R. van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[20] D. Logothetis and K. Yocum. Ad-hoc data processing in the cloud. *PVLDB*, 1(2):1472–1475, 2008.

[21] C. Curino, E. P. C. Jones, R. Ada Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database-as-a-Service for the Cloud. In *CIDR*, pages 235–241, 2011.

[22] R. Gennaro, C. Gentry, and B. Parno. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *CRYPTO*, pages 465–482, 2010.

[23] G. Karypis and V. Kumar. Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0. Technical report, 1995.

[24] D. J. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, 35(6):85–98, 1992.

[25] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *CIDR*, pages 186–199, 2005.

[26] Steven Y. Ko, Kyungho Jeon, and Ramses Morales. The hybrex model for confidentiality and privacy in cloud computing. In *HOTCLOUD*, 2011.

[27] B. C. Tak, B. Urgaonkar, and A. Sivasubramanian. To move or not to move: The economics of cloud computing. In *HOTCLOUD*, 2011.

A DATA PARTITIONING PROBLEM (DPP) IS NP-HARD

We begin by stating the assumptions we make in our proof: We eliminate the sensitive data disclosure cost constraint from the DPP problem (we assume DIS_COST is infinitely high) and we also assume that $\forall Q_i \in Q,\ freq(Q_i) = 1$. This simplified problem called Simplified-DPP $SDPP$, is a subset of our original problem, DPP. We now give a proof sketch showing that $SDPP$ is NP-Hard and since $SDPP \leq_P DPP$ can be trivially shown (fix DIS_COST to an arbitrarily large number and set the frequencies for all queries to 1), we conclude that DPP is NP-Hard.

To show that $SDPP$ is NP-Hard, we show that the 0-1 Knapsack Problem (KP) $\leq_P SDPP$. However, before proving 0-1 KP $\leq_P SDPP$, we convert the maximization function in 0-1 KP to a function that needs to be minimized and name this new problem as *MIN 0-1 KP*.

MIN 0-1 KP is defined as follows: Given a set of n items S and a *knapsack*, with $p_j = profit$ of item j, $w_j = weight$ of item j, $c = capacity$ of the *knapsack*, select a subset of S which *minimizes* z where $z = \sum_{j=1}^{m} p_j x_j$, under the following constraints:

$\sum_{j=1}^{m} w_j x_j \leq c$ and $x_j = 0$ *or* $1, j \in N = \{1, 2, \ldots, m\}$

As the only difference between 0-1 KP and MIN 0-1 KP is in the optimization function, 0-1 KP $\leq_P$ MIN 0-1 KP can be simply inferred through multiplying the given p_j values in 0-1 KP with -1

and assigning these new values as p_j values in MIN 0-1 KP, (i.e., $z = \sum_{j=1}^{m} p_j x_j$ in 0-1 KP equals to $z = \sum_{j=1}^{m} -p_j x_j$ in MIN 0-1 KP).

The reduction algorithm begins with an instance of MIN 0-1 KP. Let $S = \{s_1, s_2, \ldots, s_m\}$ be the set of items, each of which has a profit p_j and weight w_j. Also, let c be the capacity of the knapsack. We will construct an instance of $SDPP$ with a set of predicates P in which the predicates are defined as attributes over all relations R. Namely, the items in MIN 0-1 KP correspond to attributes in $SDPP$.

For every item $s_j \in S$, the instance of $SDPP$ has an attribute A_j in P (*i.e.*, $P = \{A_1, A_2, \ldots, A_m\}$). Further, the weight w_j of s_j corresponds to the overall monetary cost when $P' = \{A_j\}$, *i.e.*, $w_j = store(P') + \sum_{i=1}^{n} freq(Q_i) \times (comm_{Q_i}(P') + proc_{Q_i}(P'))$ where $P' = \{A_j\}$ and the capacity c in a MIN 0-1 KP instance will be matched with PRA_COST in the $SDPP$ instance. Then, profit of s_j can be computed as: $p_j = t_{in} - t_j$, in which $t_{in} = \sum_{i=1}^{n} freq(Q_i) \times QPC_{Q_i}(P')$ when $P' = \emptyset$ and $t_j = \sum_{i=1}^{n} freq(Q_i) \times QPC_{Q_i}(P')$ when $P' = \{A_j\}$. This instance of $SDPP$ can be easily computed from the instance of MIN 0-1 KP in polynomial time.

To conclude, the reduction that we have shown above provides a proof sketch that $SDPP$ is NP-Hard, and since $SDPP \leq_P DPP$, DPP is also NP-Hard.

B Query Rewriting Rules

We will briefly describe the query rewriting rules for the selection, projection and join operators, since we currently support only SPJ queries. Additionally, we also want to push as much processing to a public cloud as possible. Therefore, we neglect the trivial solution of bringing an entire relation from the public to the private side. We also omit the case of showing how these operators will be applied on a private cloud, since for this case the query processing will be similar to traditional database query processing.

In the context of rewriting rules, D denotes the combination of transferring results, decrypting the encrypted part of transferred results and removing the false positives within the transferred data. Moreover, $Map_{cond}(C)$ indicates the outcome of translating C by the rules studied in [4] if C involves any encrypted attribute. When C is not defined over any encrypted attribute, $Map_{cond}(C)$ will be equal to C.

To successfully perform any basic relational operator at any point in a query plan, we propose the following rewrite rule for each main operator in case of horizontal or vertical partitioning: *Rule 1* aims to perform operators over both, private and public side partitions, transfer the public side result of the processed operator and return the overall result by merging the results of both sides (through $\bigcup$ or $\bowtie$). Note that decrypting public side results and eliminating false positives within transferred data may be needed if the transferred data includes any encrypted information.

In the examples below, we define the rewrite rule for only one of the SPJ operators for each partitioning technique so as to understand the notion behind creating rewrite rules. The first example presents the *selection* rewrite rule for the horizontal partitioning case, while the second one details the *join* operator for the vertical partitioning case.

Example 1:

Selection Operator for Horizontal Partitioning ($\sigma_C(R)$)

- *Rule 1:* $\sigma_C(R_{priv}) \bigcup D\left(\sigma_{Map_{cond}(C)}(R_{pub})\right)$

Example 2:

Join Operator for Vertical Partitioning ($R \underset{R.A=S.B}{\bowtie} S$)

- *Rule 1:*

  $res1 \leftarrow R_{priv} \underset{R.A=S.B}{\bowtie} S_{priv}$

  $res2 \leftarrow R_{pub} \underset{Map_{cond}(R.A=S.B)}{\bowtie} S_{pub}$

  $res \leftarrow res1 \underset{R.A=S.B}{\bowtie} D(res2)$