2012 IEEE Fifth International Conference on Cloud Computing

Risk-Aware Workload Distribution in Hybrid Clouds

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

**University of California, Irvine*
[1]koktay@uci.edu, [2]bhore@ics.uci.edu, [3]sharad@ics.uci.edu
#The University of Texas at Dallas
[1]vvk072000@utdallas.edu, [2]muratk@utdallas.edu, [3]bxt043000@utdallas.edu

***Abstract*—This paper explores an efficient and secure mechanism to partition computations across public and private machines in a hybrid cloud setting. We propose a principled framework for distributing data and processing in a hybrid cloud that meets the conflicting goals of performance, sensitive data disclosure risk and resource allocation costs. The proposed solution is implemented as an add-on tool for a Hadoop and Hive based cloud computing infrastructure. Our experiments demonstrate that the developed mechanism can lead to a major performance gain by exploiting both the hybrid cloud components without violating any pre-determined public cloud usage constraints.**

I. 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, the foremost of which is how should one split the data and computation between the public and private sides of the infrastructure? Different choices have different implications from the perspectives of sensitive data disclosure, computational performance and resource allocation costs.

On one extreme, one may choose to outsource the entire data and workload to the public cloud (as is typical to outsourcing solutions). While simple to implement, such a solution, incurs the highest resource allocation cost in terms of cloud service (both storage and 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, the private and public sides, and to split the workload between the two sides. While simple queries may be computed on the private side, the complex ones can be performed over the public infrastructure. The above strategy exploits local resources, and thereby reduces the cost of the required cloud services. However, the resource allocation cost and the amount of sensitive data that is exposed to the public cloud will be maximum in this case. Another possibility could be to only replicate some part of the data to the public side so as to enable the distribution of the computation while limiting the disclosure risks and resource allocation costs to the desired thresholds.

The possibilities described above are just three of the multitude of computation partitioning choices. The third option seems to be the best one in terms of various end-user requirements such as performance, costs, and sensitive data exposure. An observation to be made here is that as different variants of the computation partitioning problem are formulated, a myriad of design choices present themselves. These choices are based on various data and workload formats (dynamic queries or batch jobs), as well as different query execution techniques over hybrid clouds.

This paper formalizes the computation (and the implied data) partitioning problem for hybrid clouds and develops a framework for splitting data processing tasks such that the desired goals of performance, disclosure risk and monetary expenses are achieved. In particular, given a workload of jobs (specifically SQL style HIVE queries), the underlying dataset (assumed to be relational) and the machine characteristics of private and public clouds, we propose a dynamic programming approach to solve the computation partitioning

[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].

problem. Our proposed approach is also extensively evaluated over the standard TPC-H benchmark dataset under various parameter settings.

Our primary technical contributions are listed below:

- We formalize the optimal risk-aware workload distribution problem as a mechanism for workload response time minimization. Our formalization allows us to plug in different levels of restrictions for public cloud usage. We develop an algorithm that searches for an optimal computation and data partitioning scheme given a query workload and public cloud usage boundaries (resource allocation cost and sensitive data disclosure).
- We present a formal model for estimating the monetary cost of SQL queries in a hybrid cloud setting.
- 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 II gives an overview of our Hadoop HDFS and Hive based architecture for creating a hybrid cloud. While Section III presents the details about the formalization of computation partitioning problem, Section IV covers our dynamic programming solution to this problem. In Section V, we present the results of our experimental evaluation for the dynamic programming strategy using the TPC-H benchmark. Section VI reviews the related work in the area of secure distributed data processing. Finally, we describe our conclusions and future work in Section VII.

II. System Architecture

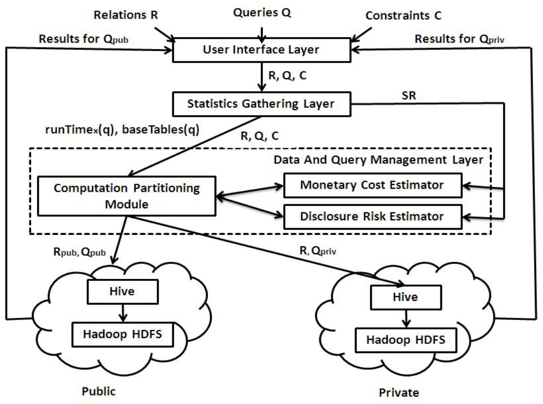

Figure 1. The Hybrid Cloud Architecture

We begin by presenting an overview of our proposed system architecture in Figure 1. The system mainly consists of two components: The *Statistics Gathering* layer performs the task of statistics collection over the dataset and query jobs, while the *Data and Query Management* layer decides on the data and workload partitioning for the given set of queries. Our focus in this paper is on the Data and Query Management layer of the system, though, as will become clear, statistics gathering is essential to determine optimal query workload and data distribution.

A user starts by submitting a set of relations, $R = \{R_1, R_2, \cdots, R_m\}$, a query workload, $Q = \{q_1, q_2, \ldots, q_n\}$, 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 estimates the minimum set of required data items and the I/O sizes (alternatively running time) of base relations required to answer each query in Q. Additionally, the statistics SR are created as equi-width histograms and sent to the estimator modules. The *computation partitioning module* receives R, Q, C as well as the estimated I/O sizes and the minimum required set of data items for each query in Q, and then systematically solves the computation partitioning problem, *CPP*. In solving *CPP*, the *monetary cost estimator* is used by our algorithm to estimate the monetary costs of processing public cloud queries as well as storing intermediate public-side data partitions, whereas the *disclosure risk estimator* is used to compute the amount of sensitivity that a solution candidate includes. On solving CPP, this layer produces two outputs: $R_{pub} = \{R_1^{pu}, R_2^{pu}, \cdots, R_m^{pu}\}$ where $R_i^{pu} \subseteq R_i$, the public cloud portion of R; and furthermore, $Q_{pub} = \{q_1^{pu}, q_2^{pu}, \cdots, q_t^{pu}\}$, the set of queries that will be executed over the public cloud.

The private cloud stores the entire dataset R, whereas the public cloud only maintains the public-side data partition, R_{pub}. The non-sensitive and sensitive data in R_{pub} and R are stored using an appropriate representation technique on the public and private clouds respectively.[2]

Once the system has stored the data based on the solution to CPP, the system is now ready to support query processing. This is achieved by assigning the public and private side queries to the corresponding Hive query engine. [3]

III. Computation Partitioning Problem

In this section, we formalize the computation partitioning problem, CPP, in a hybrid cloud setting. The problem aims to minimize the execution time of a given query workload and is bounded by two separate constraints, the first of which limits the total amount of processing that can be executed on the public cloud by restricting the monetary expenses incurred from the usage of public cloud resources, while the

[2]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.

[3]We use a Hive based infrastructure for query processing where Hive is a data warehouse built on Hadoop that allows a user to define structure for files stored in the underlying HDFS. The system only allows inter-query parallelism while executing the given set of queries. Nonetheless, techniques exploiting intra-query parallelism can be developed to find better solutions for CPP. However, this task is outside the scope of the current paper.

second captures the disclosure risk, the amount of sensitive data that a user is willing to maintain on the public side. The solution to the problem results in a partitioning of data and computation between the public and private sides.

A. Notation

Before defining CPP, we explicitly introduce the notations that will be used throughout the paper to express CPP as well as a solution for CPP.

- R: The dataset which will be partitioned over the hybrid cloud.
- Q: The query workload defined over R across the hybrid cloud. Q is represented as a set of queries, $Q = \{q_1, q_2, \dots, q_n\}$.
- $sens(R^{'})$: The estimated number of sensitive cells contained in $R^{'}$ where $R^{'} \subseteq R$.
- $baseTables(q)$: The estimated minimum set of data items required to answer a query $q \in Q$. Observe that $\forall i\ s.t.\ 1 \leq i \leq n\ baseTables(q_i) \subseteq R$.
- $freq(q)$: The access frequency of the query q.
- $runT_x(q)$: The estimated running time of query $q \in Q$ at site x (either public or private).
- $ORunT(Q^{'}, Q^{''})$: The overall execution time of the queries in $Q^{'}$ over the hybrid cloud, given that the queries in $Q^{''}$ are executed on the public cloud. Note that $Q^{''} \subseteq Q^{'}$, otherwise it will be undefined. In other words,

$$ORunT(Q^{'}, Q^{''}) = \max \begin{cases} \sum\limits_{q \in Q^{''}} freq(q) \times runT_{pub}(q), \\ \sum\limits_{q \in Q^{'} - Q^{''}} freq(q) \times runT_{priv}(q) \end{cases}$$

$baseTables$ for each query are constructed by using a parser for SQL-like queries. Our parser first parses the given query string and identifies the tables (relations) involved in that query. Then, for each associated table T, it scans the query string to find its associated attributes and predicates.

Additional details about the computation of $sens(R^{'})$, and $runT_x(q)$ will be provided in subsequent sections.

B. CPP Formalization

Given the query workload and their associated base tables, we can model the computation partitioning problem (CPP) as an optimization problem whose goal is to split the query workload over a hybrid cloud such that the overall execution time of workload Q (or performance) is minimized.

CPP Problem Definition: CPP is constructed as an optimization problem that tries to find a subset of the query workload, $Q_{pub} \subseteq Q$ and a subset of the dataset $R_{pub} \subseteq R$.

$$\begin{aligned} \text{minimize} \quad & ORunT(Q, Q_{pub}) \\ \text{subject to} \quad & (1) store(R_{pub}) + \sum_{q \in Q_{pub}} freq(q) \times proc(q) \\ & \leq PRA\_CONST \\ & (2) sens(R_{pub}) \leq DISC\_CONST \\ & (3) \forall q \in Q_{pub}\ baseTables(q) \subseteq R_{pub} \end{aligned} \tag{1}$$

where PRA_CONST represents the maximum admissible public resource allocation cost, and $DISC\_CONST$ refers to the maximum amount of sensitive data that can be disclosed to the public cloud. The use of a query workload splitting model in conjunction with constraints (public side monetary cost and sensitive data disclosure) 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 trade off's that exist within the problem domain in a systematic fashion.

C. Cost Metrics

The formalization of CPP above refers to three different metrics – query execution time (i.e., performance), monetary costs and disclosure risk. We now discuss these metrics in some detail:

Query Execution Time ($runTime_x(q)$): Computing the exact execution time of a query q on either the public or private sides depends on multiple factors. Besides the technical specifications of the underlying infrastructure such as its processor speed and the memory available in each machine, the selected query execution plan is an significant factor that impacts the query response time. However, in this paper we will use the I/O size of a query plan as a substitute for the execution 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.* [3] use an I/O based data-volume cost model to evaluate different algorithms for executing query plans on a cluster. In another paper, Wu *et al.* [4] 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 running time of a query plan τ in a hybrid cloud can be estimated as:

$$runTime_x(\tau) = \frac{\sum\limits_{\forall\ op.\ \rho \in \tau} inpSize(\rho) + outSize(\rho)}{w_x}, \tag{2}$$

where $inpSize(\rho)$ and $outSize(\rho)$ reflect the estimated input and output sizes of the operator ρ. The weight w_x accounts for how many I/O operations can be performed per unit time at site x (public or private). It also captures the difference in computational resources between a public and private cluster. Additional details about the computation of w_x are given in Section V.

The I/O sizes for the various portions of the plan τ are computed using statistics SR. Unfortunately, Hive does not maintain attribute level statistics for a relation. Therefore, we implemented a statistics gathering module that analyzes each relation in the dataset once, and maintains statistics for the relations and its attributes using separate histograms for non-sensitive (using partitions) and sensitive (using buckets) parts of the relations. Then, the I/O sizes for a query are estimated using histograms over the necessary attributes.

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)[4] 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 total monetary cost for public side computation as a sum of the following two components: a) $store(R_{pub})$: The **storage monetary cost** of the public cloud partition, R_{pub}. b) $proc(q)$: The **processing monetary cost** of a public side query q. Furthermore, the CPP problem ensures that the overall monetary cost for storing the public-side partition and performing the public-side queries (i.e. Q_{pub} and R_{pub}) is limited by a maximum public resource allocation constraint, PRA_CONST.

Sensitive Data Disclosure Risk: Sensitive data exposure is a significant 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 [5][5]. An organization would necessarily want to limit this risk, therefore, we model the risk as a constraint in the computation partitioning problem. For our problem, we estimate the total disclosure risk as the number of sensitive cells contained within the data partition for the public cloud (computed as $sens(R_{pub})$ or $sens(\bigcup_{q \in Q_{pub}} (baseTables(q)))$). Additionally, the disclosure risk depends on the data representation used to store sensitive data, which is fixed in our problem. Also, our problem ensures that the computed disclosure cost is bounded by a user-defined value, $DISC\_CONST$ [6].

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

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

[6]In this paper, we have restricted our attention to a framework that selects optimal partitioning of computation given a fixed approach to representing the data. Different choices of representation offer different levels of information disclosure e.g., a clear text representation reveals the sensitive data completely, while bucketization based on information hiding techniques offers a higher level of protection and encrypted representations offer the most protection.

We observe that the above two constraints can be merged in a principled fashion into a single constraint. This will, however, require us to find an accurate monetary cost metric for the sensitive data exposed to the public cloud. Even though sensitive data is very valuable to an organization, the probability of such data being disclosed when it is shipped to a public cloud should be taken into account. Therefore, we leave such an undertaking for the future. We would also like to stress that our model can incorporate other risk assessment techniques by assigning different disclosure costs to different cells in the dataset.

IV. Solution to CPP

As we stated earlier, the CPP problem tries to find a subset of the given dataset and query workload that can be shipped to the public cloud. The CPP problem tries to achieve this goal by aiming to minimize the total processing time of the query workload across the hybrid cloud under several constraints.

Nevertheless, CPP can be simplified to a more trivial version in which the problem only attempts to find Q_{pub}, since R_{pub} can be considered as being equivalent to $(\bigcup_{q \in Q_{pub}} baseTables(q))$. In other words, any other solution $R^{'}$ that minimizes the overall performance should be a superset of $\bigcup_{q \in Q_{pub}} baseTables(q)$ and, yet, the solution $\bigcup_{q \in Q_{pub}} baseTables(q)$ is the one with the least sensitive data exposure and monetary cost. As a result, CPP can be considered to be a problem that finds the subset of the query workload which minimizes the workload execution time without violating the given constraints.

To represent CPP along with its inputs and constraints, we use the following notation: $CPP(Q, PRA\_CONST, SENS\_CONST)$. We also assume that the query workload, Q, and the constraints, PRA_CONST and $DISC\_CONST$ are all given beforehand.

A. Dynamic Programming Approach to Solve CPP

Given the exponential number of query workload subsets, we use a dynamic programming approach to find the best one. We now present our dynamic programming algorithm that produces as its output a set of queries Q_{pub} which form a solution to $CPP(Q, PRA\_CONST, DISC\_CONST)$.

Algorithm 1 uses a data structure $pubQ$ and frequently calls a method labeled as $checkConstr$. The purpose of these constructs is as follows:

- $\mathbf{pubQ[i][j][k]}$: This data structure maintains the query solution set for $CPP(Q^i, j, k)$ where $Q^i = \{q_1, q_2, \ldots, q_i\}$. Given that the maximum admissible monetary cost and the maximum disclosure risk are equal to j and k respectively, this data structure stores queries from amongst the first i queries that are selected

Algorithm 1 DYNAMIC PROGRAMMING()

Input: Q, PRA_CONST, $DISC\_CONST$

Output: Q_{pub}

```
1: initialize $pubQ[][][]$
2: for $i = 1 \rightarrow Q.size$ do
3:     $procCost \leftarrow proc(q_i)$
4:     $totCost \leftarrow procCost + store(baseTables(q_i))$
5:     $disc \leftarrow sens(baseTables(q_i))$
6:     for $j = 0 \rightarrow PRA\_CONST$ do
7:         for $k = 0 \rightarrow DISC\_CONST$ do
8:             if $i = 1$ then
9:                 if $checkConstr(\{q_1\}, j, k)$
                   AND $ORunT(Q^1, Q^1) < ORunT(Q^1, \emptyset)$ then
10:                    $pubQ[i][j][k] \leftarrow \{q_1\}$
11:                else
12:                    $pubQ[i][j][k] \leftarrow \emptyset$
13:                end if
14:            else
15:                $pubCaseOT \leftarrow \infty$
16:                $(j', k') \leftarrow (NaN, NaN)$
17:                if $checkConstr(\{q_i\}, j, k)$ then
18:                    for all $j - totCost \leq iC \leq j - procCost$ do
19:                        for all $k - disc \leq iD \leq k$ do
20:                            $tmpSet \leftarrow pubQ[i][iC][iD] \cup q_i$
21:                            if $checkConst(tmpSet, iC, iD)$ AND
                               $ORunT(Q^i, tmpSet) < pubCaseOT$ then
22:                                $pubCaseOT \leftarrow ORunT(Q^i, tmpSet)$
23:                                $(j', k') \leftarrow (iC, iD)$
24:                            end if
25:                        end for
26:                    end for
27:                end if
28:                $privCaseOT \leftarrow ORunT(Q^i, pubQ[i-1][j][k])$
29:                if $privCaseOT \leq pubCaseOT$ then
30:                    $pubQ[i][j][k] \leftarrow pubQ[i-1][j][k]$
31:                else
32:                    $pubQ[i][j][k] \leftarrow pubQ[i-1][j'][k'] \cup \{q_i\}$
33:                end if
34:            end if
35:        end for
36:    end for
37: end for
38: return $pubQ[Q.Size - 1][PRA\_CONST][DISC\_CONST]$
```

to be processed over the public cloud so as to minimize the overall response time of the first i queries. Notice that $pubQ[i][j][k] \subseteq Q^i$.

- $\mathbf{checkConstr(Q', j', k')}$: This method returns whether monetary cost bound j' and the disclosure risk limit k' is satisfied when the queries in Q' are executed on the public side. In particular, the method checks if $store(\bigcup_{q \in Q'} baseTables(q)) + \sum_{q \in Q'} (freq(q) \times proc(q)) \leq j'$ and $sens(\bigcup_{q \in Q'} baseTables(q)) \leq k'$.

To make it easily understandable for readers, we present the notion behind our dynamic programming algorithm. Intuitively, $CPP(Q^n, PRA\_CONST, SENS\_CONST)$ can be generalized as $CPP(Q^i, j, k)$. As the solution to this general problem will be a subset of Q^i, there are two possible assignments for the last query q_i in Q^i. The query q_i is either in the solution to $CPP(Q^i, j, k)$ or is not. Therefore, both cases should be investigated carefully. Before expanding on both cases, let us illustrate how our algorithm works with an example. Assume that our query workload Q consists of 3 queries (i.e. $Q = q_1, q_2, q_3$) and $CPP(Q^3, j, k)$ needs to be solved. The detailed information about these 3 queries is given below.

q	$proc(q)$	$store(baseTables(q))$	$sens(baseTables(q))$
q_1	\$10	\$15	20
q_2	\$20	\$10	10
q_3	\$15	\$10	20

Before investigating the two different cases in further details, we need to check whether assigning q_3 to the public side violates any constraints (line 17). If we ship q_3 to the public side, then the overall monetary cost and the overall disclosure risk will be at least \$25 and 20 sensitive celss respectively (assume that $\forall 1 \leq i \leq 3 \; freq(i) = 1$). If $j < 25$ or $k < 20$, then any solution considering q_3 as a public side query will not be a feasible one, and in turn $CPP(Q^3, j, k) = CPP(Q^2, j, k)$ (line 30). Note that, since executing any query on the private side does not cause a violation of any constraints, this case essentially does not require a feasibility analysis. Now, we can go into the details of each case.

Case 1: If q_3 runs on the public side, then there will be more than 1 CPP subproblems that need to be investigated. This is due to the fact that the possible execution of q_3 on the public side will bring at least \$15 and at most \$25 into the overall monetary cost value. In terms of disclosure risk, the numbers will be between 0 and 20 sensitive cells. The reason is that a portion of (or the entire) $baseTables(q_3)$ could already be already partially included in the solution, Q_s, to some $CPP(Q^2, j', k')$, and in turn storing $baseTables(q_3)$ in addition to $\bigcup_{q \in Q_s} baseTables(q)$ may not bring as much monetary cost and disclosure risk as is represented in the table above. Consequently, $CPP(Q^2, j', k')$ where $j - 25 \leq j' \leq j - 15$ and $k - 20 \leq k' \leq k$ should be investigated in order to solve $CPP(Q^3, j, k)$ optimally (line 18-26). However, every candidate set of queries formed by taking the union of q_3 with the solution of $CPP(Q^2, j', k')$, should be tested to ensure that it does not violate any constraint and it is the best solution in terms of performance from amongst all the solutions obtained in *Case 1*(line 21). If it does produce the best solution, it will be one of the solution candidates with the one coming from *Case 2* (line 21-24).

Case 2: In case query q_3 runs on the private side, then $CPP(Q^3, i, j) = CPP(Q^2, i, j)$ (line 28).

After computing the best solution candidate for both cases, our algorithm compares the overall expected running times of both solutions and picks the minimum one as the solution to $CPP(Q^3, j, k)$ (line 29-33).

The algorithm above requires us to determine various costs (viz., disclosure, monetary, and query execution) for a given workload of queries Q and the arbitrary data partitions.

We note that the incurred disclosure risk, in our model, is dependent only on the public-side partition R_{pub}, which in turn is implicitly defined using the given query workload. Determining query execution times and monetary costs, however, depends upon the query workload. They can both be estimated as the sum of costs of the individual queries[7].

V. Experimental Results

This section presents the results of experiments that we conducted to validate the effectiveness of our dynamic programming algorithm that solves the CPP problem. We first present details of our setup followed by the experiments.

Experimental Setup: We conducted experiments on two clusters, one of which was located at UTD while the other one was located at UCI. The first cluster consists of 14 nodes and was used as the private cloud, while the second cluster consists of 38 nodes and was used as the public cloud. A node in the private cloud consists of a Pentium IV processor with $\approx$ 290GB-320GB disk space and 4GB of main memory, while a node in the public cloud consists of an AMD Dual-Core processor with $\approx$ 631GB disk space and 8GB of main memory. Both clusters are setup using Hadoop v0.20.2 and Hive v0.7.1.

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 data types, int, double and string in Hive to represent TPC-H data. We also created a data type '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 data type; this number is fixed for a given data type: (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.*).

Query Workload: We have used the TPC-H benchmark with a scale factor 100 in our experiments. We used a query workload of 40 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 because of the high complexity of estimating overall I/O size for these type of operators on HIVE. 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.

[7]Recent work has explored techniques such as shared scans in the context of executing queries over MapReduce frameworks [6] which can reduce costs of query workloads. We, however, do not consider such optimizations in developing our partitioning framework in this paper.

Estimation of weight w_x: The weight w_x is calculated as the number of I/O operations that can be performed per second on the public and private clouds. We estimated the weights w_{pub} and w_{priv} for our hybrid cloud infrastructure by running all 22 TPC-H queries for a 300GB dataset on them. Then, w_{pub} (resp. w_{priv}) was computed as the average ratio of the I/O operations required by the queries on the public side (resp. private side) to the total time required to run all queries on the public side (resp. private side). In this way, w_{pub} and w_{priv} were estimated to be 42306630.05 bytes/sec ($\approx$ 40MB/sec) and 8786423.69 bytes/sec ($\approx$ 8MB/sec). A larger value for w_{pub} indicates that our public cloud has a higher I/O throughput than our private cloud.

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, we estimated the total public cloud resource allocation cost, PUB_MAX_COST, by shipping the entire dataset and query workload computation to the public side, as $\approx 25K$, using the previously defined values. For our experiments, we then defined PRA_CONST as a fraction (25%, 50%, 75% and 100%) of PUB_MAX_COST.

Definition of sensitive data disclosure risk: We defined the sensitive data disclosure risk using the following two-part strategy: (i) We defined an overall sensitivity level for the dataset used in a given experiment. This sensitivity level assumes that all cells for the c_name, c_phone and $c\_address$ attributes of the $customer$ table are always sensitive while fractions of tuples in the $lineitem$ table are made sensitive ($\approx$ 1% or 5% or 10% of the $lineitem$ table is made sensitive). This gives us several different overall sensitivity levels that are then used in the various experiments. (ii) We then defined the $DISC\_CONST$ as being a fraction of the overall sensitivity level for a given dataset, i.e., $DISC\_CONST$ is varied between 0-100% of the overall sensitivity level defined earlier. For example, in the case when the sensitivity level for the $lineitem$ table is defined to be 10%, we now vary the $DISC\_CONST$ to be between 0% (none of the 10% sensitive tuples should be exposed) and 100% (all of the 10% sensitive tuples may be exposed).

Preliminary Experiments: For all our experiments, we first computed the running time of the query workload when all computations are performed on the private cloud (Private). The experiments subsequently use this case as a baseline to determine the performance of the dynamic programming approach that was proposed earlier to solve

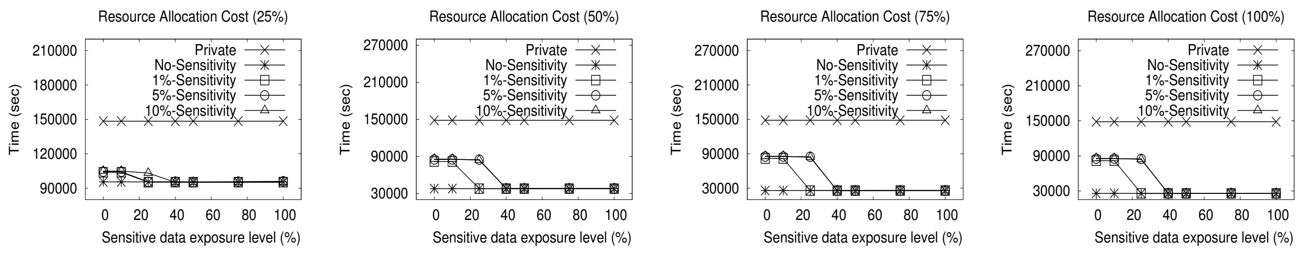

Figure 2. Performance of the Dynamic Programming approach towards solving the CPP problem

the CPP problem.

Experiments for Dynamic Programming approach: The goal of these experiments is to measure the performance of the dynamic programming approach that was proposed earlier for solving the CPP problem. To perform these experiments we varied all parameters under consideration in the following way: (i) Resource allocation cost: The resource allocation cost was varied between 25-100% of the PRA_CONST value that was defined earlier. (ii) We defined four different overall sensitivity levels as, No-Sensitivity (the entire dataset is non-sensitive), 1%-Sensitivity, 5%-Sensitivity and 10%-Sensitivity (1%, 5% and 10% of the tuples of the $lineitem$ table are made sensitive). (iii) We defined seven different sensitive data exposure levels as 0% (none of the sensitive data is exposed), 10%, 25%, 40%, 50%, 75% and 100% (all of the defined sensitive data may be exposed).

We then computed the overall performance of the query workload for different combinations of these three parameters, the results of which are presented in Figure 2. One of the first observations that can be made from Figure 2 is 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 (even greater than 50%). On the other hand, Figure 2 also 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 (graphs for 50%, 75% and 100% resource allocation cost show that even when more money is allowed to be spent on public side resources the overall performance is relatively the same for these cases suggesting that a budget of only about 50% of PRA_CONST is sufficient to boost the performance savings by upto 50%).

Figure 2 also shows that when a user invests more capital towards resource allocation, a considerable gain in overall workload performance (even greater than 50%) can be achieved. This is expected since when more resources are allocated on the public side, 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. Finally, from Figure 2, we also notice that we can achieve a considerable improvement in query performance ($\approx$ 50%) for a relatively low risk ($\approx$ 40%) and resource allocation cost ($\approx$ 50%).

VI. Related Work

Secure data processing in hybrid clouds has been explored in various contexts. While Sedic, introduced in [7], studies the problem only from the perspective of individual mapreduce jobs; Relational Cloud, proposed in [8], uses a graph-based partitioning scheme to split data and computation into public and private sides. The data partitions are encrypted with multiple layers of encryption and stored on a server, and will therefore possibly require multiple rounds of decryption and communication between the public and private sides while executing a query.

The distribution of query computation between trusted and untrusted servers has been previously studied in the context of database outsourcing [9], [10]. Such approaches have focused on partitioning individual queries to identify maximal portions that can be executed directly on encrypted representations over untrusted servers. This line of work has not considered a risk based approach to workload distribution as studied in this paper. Nonetheless, techniques for computing on encrypted data can be further exploited to support improved risk aware processing in hybrid clouds. We intend to explore such extensions as a part of the future work.

The distribution of data and queries across multiple servers has been considered in [11]. Their problem setting and design goals are different, since in their model both servers are assumed to be untrusted (but not colluding). 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 (risk based approach).

A risk based approach has been discussed in [12] for single machine architectures under memory attacks. It has not been studied for the cloud environment. To our knowledge, our paper is the first work that does a risk based data and workload partitioning for given relational database workloads in hybrid clouds.

Finally, previous research in the fields of data partitioning (e.g. [3], [4], [5], [6]) and distributed query processing (e.g., evolution from systems such as SDD-1 [13] to DISCO [14] that operates on heterogeneous data sources, to Internet-scale systems such as Astrolabe [15], and cloud systems [16]) are also related with our work. However, none of these approaches is directly applicable for hybrid clouds.

VII. Conclusions and Future Work

With the advent of cloud computing, a hybrid cloud is suitable for users who wish to balance data security risks while limiting the expenses for using public cloud services. We have identified three challenges that must be overcome before this approach can be adopted.

The first challenge deals with partitioning computations between a private cloud and a service provider when there is some sensitive information in the data. We formalized this challenge as a risk-aware performance optimization problem and presented a dynamic programming approach that results in an optimal distribution of the query workload. The second challenge relates to to keeping the amount of sensitive data that is exposed to the public machines under a given threshold. Finally, the last challenge addresses the issue of limiting the monetary costs that arise from public cloud usage.

We are primarily exploring the following ideas for future research from amongst the various areas that we outlined throughout the paper: 1) In this paper, we have only considered inter-query parallelism while partitioning the computation over a hybrid cloud. However, our current work can be extended to make use of intra-query parallelism as a possibility for executing the given query workload. 2) In this paper, we solved the computation partitioning problem for the case where the entire query workload is given to us *apriori*. This work can be enhanced to support computation distribution for dynamically changing (or arriving) workloads. 3) In our current implementation, 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. 4) Data sensitivity model adopted in this paper is restricted to whether a single attribute is sensitive or not. We plan to solve the same problem under a different sensitivity model in which an attribute might be insensitive by itself, but it can be sensitive when it's revealed together with another attribute.

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] 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.

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

[5] 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.

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

[7] Kehuan Zhang, Xiaoyong Zhou, Yangyi Chen, XiaoFeng Wang, and Yaoping Ruan. Sedic: privacy-aware data intensive computing on hybrid clouds. In *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, pages 515–526, New York, NY, USA, 2011. ACM.

[8] 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.

[9] Hakan Hacigümüs, Bijit Hore, and Sharad Mehrotra. Privacy of outsourced data. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 965–969. 2011.

[10] 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.

[11] 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.

[12] Mustafa Canim, Murat Kantarcioglu, Bijit Hore, and Sharad Mehrotra. Building disclosure risk aware query optimizers for relational databases. *PVLDB*, 3(1):13–24, 2010.

[13] 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.

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

[15] 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.

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