Exploiting Soft and Hard Correlations in Big Data Query Optimization

Hai Liu, Dongqing Xiao, Pankaj Didwania, Mohamed Y. Eltabakh

Worcester Polytechnic Institute (WPI), Computer Science Department, MA, USA
{hliu2, dxiao, pdidwania, meltabakh}@cs.wpi.edu

ABSTRACT

Big data infrastructures are increasingly supporting datasets that are relatively structured. These datasets are full of correlations among their attributes, which if managed in systematic ways would enable optimization opportunities that otherwise will be missed. Unlike relational databases in which discovering and exploiting the correlations in query optimization have been extensively studied, in big data infrastructures, such important data properties and their utilization have been mostly ignored. The key reason is that domain experts may know many correlations but with a degree of uncertainty (fuzziness or softness). Since the data is big, it is very challenging to validate such correlations, judge their worthiness, and put strategies for utilizing them in query optimization. Existing techniques for exploiting soft correlations in RDBMSs, e.g., BHUNT, CORDS, and CM, are heavily tailored toward optimizing factors inherent in relational databases, e.g., predicate selectivity and random I/O accesses of secondary indexes, which are not applicable to big data infrastructures, e.g., Hadoop.

In this paper, we propose the EXORD system to fill in this gap by exploiting the data’s correlations in big data query optimization. EXORD supports two types of correlations; hard correlations—which are guaranteed to hold for all data records, and soft correlations—which are expected to hold for most, but not all, data records. We introduce a new three-phase approach for (1) Validating and judging the worthiness of soft correlations, (2) Selecting and preparing the soft correlations for deployment by specially handling the violating data records, and (3) Deploying and exploiting the correlations in query optimization. We propose a novel cost-benefit model for adaptively selecting the most beneficial soft correlations w.r.t a given query workload while minimizing the introduced overhead. We show the complexity of this problem (NP-Hard), and propose a heuristic to efficiently solve it in a polynomial time. EXORD can be integrated with various state-of-art big data query optimization techniques, e.g., indexing and partitioning. EXORD prototype is implemented as an extension to the Hive engine on top of Hadoop. The experimental evaluation shows the potential of EXORD in achieving more than 10x speedup while introducing minimal storage overheads.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.

Proceedings of the VLDB Endowment, Vol. 9, No. 12
Copyright 2016 VLDB Endowment 2150-8097/16/08.
3 to 10 days from the shipping date. Assume the dataset already has an index on the shipping date to efficiently answer queries involving a selection predicate on that attribute, e.g., \([4, 6, 8]\). However, the crucial limitation of these techniques is that queries involving selection predicates on the delivery date cannot be optimized, and would require a full scan over the data (unless another index is created on the delivery date). Moreover, given the correlation’s uncertainty, a query issuer cannot manually translate the predicate to a corresponding one on the shipping date (by going backward 3 to 10 days), and then filtering the results. This blind re-writing may miss some tuples satisfying the original query but not conforming with the soft correlation.

Motivation Scenario 2 – Airline Analytics and Usage of Data Partitioning: Airline analytics companies manage very large data about customer requests, flight status, seat availability, and airport traffics. Most of the major airports worldwide have a unique three-character code, called IATA, which identifies the airport, and hence identifies its city and country. However, many small airports—usually with very limited traffic—do not have IATA code (denoted as “*”“*”“*”), and thus this “*”“*”“*” code neither identifies the city nor the country. Therefore, there is a soft correlation from the airport code to the country, i.e., for most (but not all) records the airport code uniquely determines the country. Assume the data is already partitioned on the country attribute to optimize certain queries \([9, 15]\). The limitation now is that queries involving predicates on the airport code would require a full scan over the data without making any use of the available partitioning. The challenge is that in the general case, the values violating the correlation may not be known in advance, e.g., each country may assign random code for these small airports of infrequent traffic, and thus manual re-writing to optimize the query is not feasible.

Clearly, the soft correlations mentioned above open big opportunities for query optimization—especially if we can systematically leverage the special organization that is already present in the data, e.g., indexing \([4, 6, 8]\) or partitioning \([9, 15]\), without the need to create additional ones. This is crucial because building auxiliary structures over big data is a very expensive process w.r.t both time and storage \([6, 8, 9]\), and thus should be kept to minimal whenever possible. Our experimental evaluation also confirms the substantial overheads involved in creating additional partitioning or indexes, which makes it almost impractical to create many of these over a single dataset (Results will be discussed in Figure 6). Therefore, by leveraging the possible correlations among the data attributes, the usage and benefits of the existing techniques (on few targeted attributes) can be extended beyond these attributes without paying these significant costs, and eventually a broader class of queries can be optimized.

Exploiting soft correlations in query optimization is not a new problem and it has been previously studied in the context of relational databases, e.g., \([2, 14, 16, 17]\). However, most of these techniques have objectives that are specific only to RDBMSs and not applicable to the new emerging Hadoop-like infrastructures. For example, CORDS \([14]\) exploits a correlation between a pair of attributes \(A_1\) and \(A_2\) to provide more accurate selectivity estimation for their conjunctive predicates at query time, and enable generating better query plans. On the other hand, Correlation Maps (CM) \([16]\) and CORADD \([17]\) aim at avoiding the random I/O accesses that may result from a blind usage of secondary indexes on un-clustered attributes, and instead they try to leverage any existing correlation that may link these un-clustered attributes to the clustered attribute on which the relation is sorted.

These issues of predicate selectivity and random vs. sequential access patterns are fundamental in RDBMSs. However, they are not applicable to the highly-distributed big data infrastructures, e.g., Hadoop. First, the physical execution plan in Hadoop is fixed—either map-only job or map-reduce job—and predicate selectivity does not play any role in optimizing this execution plan. Second, with the distributed nature of big data both in storage and retrieval, the data is usually not sorted on a specific attribute, and there is no notion of clustered vs. un-clustered attributes. Third, since the emerging infrastructures are highly distributed, there is no notion of random vs. sequential accesses, instead execution tasks, e.g., map tasks in Hadoop, are assigned to where the data resides, which is called data locality. And Fourth since there is no notion of random accesses, it is always safe in Hadoop-like systems to use indexes to answer selection queries regardless of their predicate selectivity. These fundamental differences warrant the need for new correlation exploitation tools that target the new big data distributed infrastructures (More detailed discussion is provided in Section 6).

In this paper, we propose the “EXORD” system for Exploiting soft and hard correlations in big data query optimization (Refer to Figure 1). EXORD is distinct from the existing techniques mentioned above in three key aspects, which are:

1) Infrastructure Type: EXORD is the first correlation exploitation tool targeting the emerging Hadoop-like infrastructures, which are fundamentally different from RDBMSs in their distribution nature, query processing, data retrieval, and index access patterns. Our objective is to avoid full scan plans whenever possible.

2) Domain Knowledge: All existing techniques are discovery-based, where they put the extreme assumption of having no knowledge about the possible correlations that may exist in a given dataset. This assumption puts various restrictions on the correlations that can be captured, e.g., BHUNT \([2]\) is only limited to numerical attributes and the correlations must be in the form of a simple algebraic expression of one operator \(\{+,-,*/,\}\). In contrast, EXORD puts the more practical assumption that domain experts have some knowledge on the possible correlations that may exist in the dataset. And hence, they can define “hard correlations” (with certainty) and “soft correlations” (with fuzziness). Correlations in EXORD can be between any pair of attributes either numerical or categorical, and the relationship logic may range from complex expressions to general look-up functions.

3) Optimized Resource Management: Preparing the soft correlations to be usable by the query optimizer involves a cost for handling the violating records. Therefore, given a set of candidate correlations and limited system resources, deciding on which ones to select and be more beneficial to a given query workload turns out to be a complex NP-Hard optimization problem. EXORD introduces a novel cost-benefit model for soft correlations augmented with a heuristics-based algorithm under which it adaptively and dynamically selects the most beneficial correlations in a practical polynomial time.

As illustrated in Figure 1, the core features of EXORD are infrastructure-independent, and hence they are applicable to big data query optimization in general. As a proof of concept, EXORD prototype is built on top of the Hadoop infrastructure—as one of the most popular big data platforms—and it uses Hive as its high-level query engine. We opt for Hive since it assumes a known structure for the data, which facilitates defining the correlations among the data attributes.

The key contributions of this paper are summarized as follows:

- Proposing “EXORD” as the first system for exploiting the data’s correlations in big data query optimization. EXORD realistically put the assumption that domain experts may not guarantee with certainty many of the correlations, and thus EXORD introduces and supports the two types of hard and soft correlations.
• Introducing a new multi-phase approach for managing soft correlations, which consists of a Validation Phase, followed by a Selection & Optimization Phase, and then a Deployment & Exploitation Phase. Throughout these phases EXORD decides whether or not a given soft correlation is worthy of being used, and what is the best strategy to prepare it for deployment. We propose two different strategies, namely Exclusion, and Materialization, for handling the violation records, each is suited for specific cases. (Section 3)

• Proposing a novel cost-benefit model for soft correlations, and mapping the selection of the most beneficial ones for optimizing a given query workload to the well-known submodular knapsack optimization problem. As an NP-Hard problem, we propose a sound heuristic to efficiently solve the selection problem in a practical polynomial time. (Section 3)

• Proposing an algorithm for exploiting the soft and hard correlations in query optimization. We experimentally utilize EXORD on top of two well-known optimization strategies in Hadoop, which are the indexing, and partitioning strategies. EXORD extends the benefits of these strategies beyond their targeted attributes to optimize a broader class of queries. (Section 4)

• Implementing EXORD as an extension to Hive query engine on top of Hadoop. The extensions do not require any changes to Hadoop, and hence they are easily portable across versions. The system’s empirical evaluation shows that EXORD enables optimization opportunities that the state-of-art techniques fail to discover. These optimizations lead to up to 12x speedup compared to the naive full-scan evaluation plans. (Section 5)

DEFINITION 2.1 (TARGET QUERIES). A target query Q in EXORD is a query involving an equality predicate on an attribute A₁ that has no associated access methods—other than a full scan—to check its predicate. The primary objective of EXORD is to leverage any available correlations related to A₁ to re-write Q in terms of other attributes that have more efficient access methods, e.g., indexing or partitioning, to evaluate their predicates.¹

DEFINITION 2.2 (CORRELATION). A correlation C over a given dataset is a directed relationship from one attribute, called “source”, to another attribute called “destination”. The correlation defines how the source’s values can be mapped to the destination’s values. C is defined as a five-ary vector ⟨Src, Dest, Type, Granularity, F()⟩, where, Src and Dest, are the source, and destination attributes of the correlation, respectively. Type is the correlation’s type as either “Hard” or “Soft”, and Granularity is the granularity of mapping from a given source’s value to the destination value(s), and it takes either of the values “Singleton” (one-to-one mapping), “Range” (one-to-range mapping), or “List” (one-to-list mapping). Finally, Function F() is the mapping function that takes a value s ∈ Src and returns its corresponding mapping in Dest. Depending on the granularity, F(s) returns either of a single value (for “Singleton”), a list of values (for “List”), or a record (lower, upper) (for “Range”).

Example 1: Referring to our motivation scenarios, the correlation in Scenario 1 can be formulated as: ⟨deliveryDate, shippingDate, “Soft”, “Range”, F()⟩, where for a given delivery date d, F(d) = (d - 10 days, d - 3 days). In contrast, the correlation in Scenario 2 can be formulated as: ⟨airportCode, country, “Soft”, “Singleton”, F()⟩, where for a given airport code a, Function F(airport) looks up a small table having the list of distinct airport codes and returns the corresponding country (For the special code “***”, the function returns Null).

In general, EXORD treats Function F()—which is provided by the system admin—as a black box without the need to know its internal logic. The only requirement is that F() should be a low-cost light-weight function that can be executed with a very little cost per record. This requirement does not limit the applicability of the system since the Src and Dest attributes are not restricted to any specific data type or domain and they can be numerical or categorical. Moreover, F() may range from any mathematical or algebraic expression, to general lookup functions that search some auxiliary structures and perform the mappings—as long as this auxiliary structure is relatively small (few MBs) and can be distributed to the main memory of each machine in the distributed system. Most correlation examples studied in literature, e.g., [2, 16, 17], fall under this scope.

DEFINITION 2.3 (HARD CORRELATION). A hard correlation C is a correlation having C.Type = “Hard” and it is guaranteed to hold for all records in the dataset D.

Unlike hard correlations, soft correlations have a degree of uncertainty and not all of them are useful, e.g., the violations for a given correlation can be too many. Therefore, depending on the degree of violations, we categorize soft correlations into Valid (useful) and Invalid (useless) as follows.

DEFINITION 2.4 (VALID SOFT CORRELATION). For a soft correlation C of C.Type = “Soft”, let Φ(C) denotes the set of distinct violating values in C.Src with cardinality |Φ(C)|, and Γ(C) denotes the set of violating records in the dataset (with cardinality |Γ(C)|). Given two user-defined thresholds MaxVioDistinct > 1 and MaxVioRec > 1, C is called a “valid” soft correlation iff either (or both) of the following two conditions is met:

1. |Φ(C)| ≤ MaxVioDistinct,
2. |Γ(C)| ≤ MaxVioRec

DEFINITION 2.5 (INVALID SOFT CORRELATION). A soft correlation C of C.Type = “Soft”, is called “invalid” iff C is not a valid soft correlation.
According to Def. 2.4, EXORD considers a soft correlation to be **valid** (useful) under two cases. The first case is where the number of distinct violating values in the correlation’s source attribute is less than a given threshold $MaxVioDistinct$ regardless of the number of violating records. This case captures the scenarios where there can be many violating records, but possibly because of few distinct values that are repeating in these records. The second case is where the set of violating records is smaller than a given threshold $MaxVioRec$ even if the number of distinct violating values is large. Typically, $MaxVioDistinct < MaxVioRec$. As we will present in Section 3.2, EXORD offers two different strategies for managing these two cases.

### 3. EXORD: BASIC FRAMEWORK

We propose a three-phase approach to manage the soft correlations as depicted in Figure 1. In the 1st phase (The Validation Phase), EXORD collects few statistics to identify the valid soft correlations. In the 2nd phase (The Selection & Optimization Phase), the system selects and prepares a possibly subset of the valid soft correlations that are the most beneficial for a given query workload. This selection is based on a novel cost-benefit that works under limited system resources. The 3rd phase is the Deployment & Exploitation Phase in which the correlations are used at runtime for query re-writing and optimization. In the following, we present the first two phases, while the 3rd phase is presented in Section 4.

#### 3.1 Validation Phase

Assume a dataset $D$ and a set of soft correlations in their validation phase $V = \{C_1, C_2, ..., C_n\}$. Since scanning the dataset can be an expensive operation over big data, EXORD will not perform a dedicated job on $D$ to collect the needed statistics. Instead, the system waits for the first user’s job that is going to scan the data anyway, and then piggybacks the statistics collection task over this job. More specifically, the user’s map task will be encapsulated within a larger system-created map task, and before the execution of the user’s code on an input record $r$, $r$ is tested against each soft correlation $C_i \in V$ according to its granularity as follows:

<table>
<thead>
<tr>
<th>Correlation Granularity</th>
<th>Test Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singleton</td>
<td>$Dest \sim C_i \cdot F(Src)$</td>
</tr>
<tr>
<td>Range</td>
<td>$Dest \geq C_i \cdot F(Src).lower \ And \ Dest \leq C_i \cdot F(Src).upper$</td>
</tr>
<tr>
<td>List</td>
<td>$Dest \in C_i \cdot F(Src)$</td>
</tr>
</tbody>
</table>

For each correlation, each mapper reports two types of statistics: (1) The number of records violating $C_i$ (without reporting the actual records), and (2) Either the distinct violating values $\Phi(C_i)$ if their number is less $MaxVioDistinct$, or a flag indicating that the number has exceeded the allowed threshold. Notice that the goal is not to enumerate all distinct violating values otherwise it becomes an expensive map-reduce job by itself. Instead, each mapper keeps maintaining the seen-so-far distinct violating values in its memory until the given threshold is exceeded (if happened). And as we will demonstrate in the experiment section, $MaxVioDistinct$ is typically set to a few thousands, e.g., 10,000 or less, and in this case the mapper’s consumed memory is very small, e.g., less than 1MB.

After all mappers are completed, a centralized task aggregates the results to compute $\Gamma(C_i)$, and the final duplicate-free set of $\Phi(C_i)$ (or a flag indicating that its size exceeded $MaxVioDistinct$). Based on these statistics and according to Def. 2.4, each correlation is marked as either $Valid$ (and advances to subsequent phases) or $Invalid$ (and get eliminated from any further consideration). The statistics and the status of each correlation are maintained in the system’s Metadata Repository, which will be highlighted in Section 5.1.

#### 3.2 Selection & Optimization Phase

Unlike hard correlations that are ready for exploitation by the query optimizer, the valid soft correlations need to be first prepared by specially handling the violations and putting strategies for guaranteeing correct query execution. This handling involves a storage cost, which may vary significantly from one correlation to another. Moreover, not all correlations have the same usefulness degree w.r.t query optimization. For example, the system may pay the cost of preparing many valid soft correlations while they are of little or no actual benefit to the current query workload, which may lead to a significant waste in system resources.

In this section, we propose a novel cost-benefit model to automatically and adaptively select the correlations based on their costs and benefits w.r.t to a given query workload and under limited system resources. We show that this optimization problem is very complex, and can be formulated as a submodular knapsack problem, which is known to be an NP-Hard problem [21]. And then, we propose a heuristic to efficiently solve it in polynomial time.

##### 3.2.1 Correlations Cost Model

EXORD offers two different strategies—each comes with an associated cost—for preparing a valid soft correlation, namely “Exclusion” and “Materialization”. Each strategy is applicable to a given soft correlation according to the following definitions:

**Definition 3.1 (Exclusion Strategy).** For a given valid soft correlation $C$, the “Exclusion Strategy” is applicable to $C$ iff Condition (1) in Def. 2.4 is True, and it involves copying the $\Phi(C)$ set to EXORD’s Metadata Repository.

**Definition 3.2 (Materialization Strategy).** For a given valid soft correlation $C$, the “Materialization Strategy” is applicable to $C$ iff Condition (2) in Def. 2.4 is True, and it involves copying the $\Gamma(C)$ set to a separate file, called exception bucket, in the file system.

As will be discussed in detail in Section 4, the Exclusion strategy relies on logically excluding the violating values from being re-written or optimized for during the compilation and optimization time. And since the decision of such exclusion needs to be taken very fast (typically in few milli-seconds), the distinct violating values are not intended to be written to (and read from) the slow-storage of the main file system, e.g., HDFS. Instead, they are kept in EXORD’s Metadata Repository—which is a lightweight relational DBMS. The Exclusion strategy is well suited for the cases where the number of distinct violating values is small even if the violating records are many (As in Motivation Scenario 2). On the other hand, the Materialization strategy relies on physically copying the violating records into separate files (referred to as “exception buckets”)—recall that in HDFS, the deletion or update of the original files is not a possible operation. These exception buckets can be relatively large in size, and thus kept in the file system, e.g., HDFS. Nevertheless, their access will be only required during query execution not query optimization.

Given that the storage resources are not infinite, the maximum resources allotted to EXORD, which are referred to as the Resource Pool, are defined as follows:

**Definition 3.3 (Resource Pool).** The Resource Pool is the maximum allocated storage that valid soft correlations can compete for and consume. It is denoted as $RPool = \{MPool, HPool\}$, where $MPool$ and $HPool$ are the maximum sizes in the Metadata Repository, and the file system (HDFS), respectively.

Recall that $MPool$ is the storage space to be used in the case of the Exclusion strategy, whereas the $HPool$ is the storage space to be used in the case of the Materialization strategy.
Referring to Figure 2, it is evident that correlations may have different priorities w.r.t. each of the four factors mentioned above. For example, if we consider the 1st factor (based on coverage), then C1 and C2 will have higher (and equal) priorities over C3, whereas, if we consider the 2nd factor (based on exclusive coverage), then C3 gets higher priority. Moreover, according to the 4th factor (W’s coverage with minimal cost), if C1 is selected by the system, then C2’s priority should be significantly lowered since it does not cover any new queries beyond those covered by C1.

Clearly, including all four factors into the benefit metric makes it very complicated especially because estimating the execution savings (The 3rd factor) requires execution statistics, which may not be available for all correlation-query pairs. To simplify the metric, we make a reasonable and practical assumption that no matter how a query is optimized, e.g., through indexing or partitioning, the savings from executing an optimized version will be significant compared to executing the un-optimized version (full scan). This implies that, it does not matter which of the two correlations C1 or C2 to use for optimizing Q2, what matters is to have Q2 covered. It also implies that EXORD always favors the use of these special structures over a full scan regardless of the query selectivity. This is a valid assumption in Hadoop-like infrastructures because under the way the indexes are built [6, 8] and the data blocks are accessed in a distributed manner, there is no notion of secondary indexes that may lead to random data accesses and higher overheads compared simple sequential scans (this is unlike the case in RDBMSs).

As confirmed by our experiments, even under the worst case where selectivity is close to 100%, both partition-based and index-based techniques are safe to choose as they would perform very similar to a full scan.

Based on this simplifying assumption, the 3rd factor concerning the actual execution savings can be ignored because all applicable correlations are now assumed to bring “enough” and “acceptable” benefit. Now, for the remaining three factors, the correlation’s benefit can be formally defined as follows:

**Definition 3.5 (Static Correlation Benefit).** For a given workload W of size n queries, and a soft correlation Ci, the static benefit of C is computed as the percentage of queries for which Ci is either applicable or exclusively applicable. That is:

\[ S\text{Benefit}(C_i, W) = \frac{|\text{Coverage}(C_i)| + |\text{ExclusiveCoverage}(C_i)|}{n}. \]

**Definition 3.6 (Dynamic Correlation Benefit).** For a given workload W, and a soft correlation C, the dynamic benefit of C is re-computed after the selection of any other correlation as follows:

Initial State:

\[ D\text{Benefit}(C_i, W) = S\text{Benefit}(C_i, W), \quad \forall C_i. \]

Next State after the selection (and removal) of correlation C:

\[ W = W - \text{Coverage}(C_i), \quad D\text{Benefit}(C_i, W) = S\text{Benefit}(C_i, W), \quad \forall C_i. \]

The static correlation benefit (Def. 3.5) basically takes into account the 1st and 2nd factors, while ignoring the 4th factor. Yet, its computations are easier since the benefits do not depend on the previous decisions taken by the system. In contrast, the dynamic correlation benefit (Def. 3.6) takes the 4th factor into account, and thus whenever a specific correlation is selected, the benefits of the remaining correlations need to be re-calculated.

### 3.2.3 The Optimization Problem

The optimization problem is now to select a subset of valid soft correlations C with the objective of maximizing the total benefit.
the available HDFS storage. Given the computational complexity

\[ \text{Cost}(C_i) \leq \text{Cost}(C_j) \quad \text{IFF} \]

\[ \Phi(C_i).\text{size} \leq \Phi(C_j).\text{size} \quad \text{And} \]

\[ (\Gamma(C_i).\text{size} \leq \Gamma(C_j).\text{size}) \]

\[ C_i \rightarrow C_j \]

\[ C_i \rightarrow \Gamma C_j \]

\[ C_i \rightarrow \Phi C_j \]

Table 1: Dominance Notations w.r.t Cost() Comparison.

<table>
<thead>
<tr>
<th>Dominance Type</th>
<th>Cost(Ci) ≤ Cost(Cj)</th>
<th>IFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_i → Φ C_j</td>
<td>(Φ(C_i).size ≤ Φ(C_j).size) And</td>
<td></td>
</tr>
<tr>
<td>(Γ(C_i).size ≤ Γ(C_j).size)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_i → C_j</td>
<td>(Φ(C_i).size ≤ Φ(C_j).size)</td>
<td></td>
</tr>
<tr>
<td>(Γ(C_i).size ≤ Γ(C_j).size)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C_i → Γ C_j</td>
<td>(Φ(C_i).size ≤ Φ(C_j).size)</td>
<td></td>
</tr>
<tr>
<td>(Γ(C_i).size ≤ Γ(C_j).size)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A correlation C_i (soft or hard) is said to dominate another correlation C_j (soft), denoted as C_i → C_j, if:

\[ \text{Cost}(C_i) \leq \text{Cost}(C_j) \]

and

\[ \frac{\text{ExclusiveCoverage}(C_j)}{\text{Coverage}(C_i)} \leq \epsilon, \text{ where } \epsilon \text{ is a small threshold value} \in [0, 1). \]

The heuristic relies on that before solving the optimization problem, we apply a filtering step that eliminates correlations that are dominated (or nearly dominated) by other correlations. In other words, instead of aiming for the optimal solution according to Def. 3.6 (which is very expensive), we aim for avoiding the worst-case scenario that Def. 3.5 may generate. According to Def. 3.7, a soft correlation can be dominated by either a soft or a hard correlation (but not vice versa). Moreover, a correlation that has a non-empty ExclusiveCoverage() set cannot be dominated by other correlations. It is worth highlighting that if the \( \epsilon \) threshold is set to zero, then the dominance becomes a “complete dominance”. However, the heuristic allows for a “near dominance” to be more effective in filtering out correlations that add little contributions. For example, if \( \epsilon = 0.15 \) and one correlation C_i covers 10 queries, while another correlation C_j covers seven of these queries plus one additional new query, then

\[ \frac{\text{ExclusiveCoverage}(C_j)}{\text{Coverage}(C_i)} = 0.14, \text{ and thus } C_i \rightarrow C_j, \text{ and } C_2 \text{ can be eliminated if the other conditions are met}. \]

In Figure 3, we sketch the heuristic-based algorithm for solving the correlation-selection optimization problem. The algorithm takes as input a set of n valid soft correlations \( \mathcal{P} \), and an observed query workload \( W \). The output is a subset of selected correlations \( \mathcal{O} \) to deploy along with the deployment strategy assigned to each one. The algorithm is divided into three main phases: Phase 0 eliminates correlations that are dominated by others, Phase 1 solves the optimization problem for resource RPool.MPool, i.e., correlations will compete for the available metadata repository storage, and Phase 2 solves the optimization problem for resource RPool.HPool, i.e., the remaining correlations will compete for the available HDFS storage. Given the computational complexity of [13], which is \( O(n \log n) \), the proposed heuristic-based algorithm also has computational complexity of \( O(n \log n) \), where \( n \) is the number of correlations in \( \mathcal{P} \).

In Phase 0, we permanently eliminate any correlation \( C_j \) that is dominated by another correlation \( C_i \), where the cost function comparison takes both components (\( \Phi() \).size and \( \Gamma() \).size) into account as summarized in Table 1. This means that \( C_i \) cannot compete against \( C_j \) for either of RPool.MPool or RPool.HPool (denoted as \( C_i \rightarrow \Phi C_j \) and \( C_i \rightarrow \Gamma C_j \)), and thus it is permanently eliminated.

In Phase 1, the remaining correlations will compete for the RPool.MPool resource, i.e., compete for the “Exclusion Strategy”. For that purpose the cost function comparison will take only the \( \Phi() \).size component into account. The dominance heuristic will be applied again to “temporarily” eliminate any dominated correlations given this new cost function comparisons denoted as \( C_i \rightarrow \Phi C_j \) (Refer to Table 1). After that, the optimization problem is solved approximately using the “FPTAS” technique in [13]. The selected correlations will be added to the output set \( \mathcal{O} \), and will be assigned the “Exclusion” strategy.

Phase 2 has the same idea of Phase 1 but with two differences. First, Phase 2 needs to re-examine each correlation, say \( C_j \), that is dominated by others in Phase 1 (and hence skipped from the competition), and checks whether or not any of \( C_j \)’s dominating correlations is actually in the output set \( \mathcal{O} \). If not, then \( C_j \) is admitted back to the candidate pool \( \mathcal{P} \), and is given a second chance to compete for RPool.HPool only if \( \Gamma(C_j).\text{size} \neq \infty \). Second, correlations in Phase 2 will now compete for the RPool.HPool resource, i.e., compete for the “Materialization” strategy. Therefore, the cost function comparison for the dominance relationship will
4. DEPLOYMENT & EXPLOITATION PHASE

The ultimate goal of EXORD is to exploit the available correlations to re-write queries and enable more efficient access plans. In Figure 4, we present the flowchart of the exploitation procedure. The procedure takes as input a set of correlations in their deployment phase \( \mathcal{Y} = \{C_1, C_2, \ldots, C_n\} \), a dataset \( D \) to be queried, and a target query \( Q \) as defined in Def. 2.1 consisting of a set of conjunctive predicates \( p_1 \land p_2 \land \ldots \land p_k \). Set \( \mathcal{Y} \) includes a mix of hard correlations, and soft correlations along with their deployment strategies as either Exclusion, or Materialization.

As the first step, the system checks whether any of \( Q \)'s predicates, say \( p_k \), can enable an access plan other than a full scan, e.g., by leveraging indexing or partitioning strategies. If that is the case, then \( Q \) is returned without correlation-driven re-writing. Otherwise, EXORD tries to re-write any of the predicates using the available correlations in \( \mathcal{Y} \). While searching \( \mathcal{Y} \), the priority is given first to the hard correlations, followed by the Exclusion-based soft correlations, followed by the Materialization-based soft correlations (The 2\textsuperscript{nd}, 3\textsuperscript{rd}, and 4\textsuperscript{th} conditions in the flowchart, respectively). The intuition is that hard correlations apply to the entire dataset \( D \) without any restrictions or exceptions, and thus they are given the highest priority. On the other hand, the Exclusion-based correlations are given higher priority over the Materialization-based correlations because, as will be explained next, the processing of queries under the latter strategy involves more overhead due to the need for scanning the corresponding exception bucket.

Assume the selected correlation is \( C_i \in \mathcal{Y} \), and it will be used to re-write a specific predicate in \( Q \), say \( p_k \), which is in the form of \( p_k: A_k \equiv s_k \), where \( A_k \) is one of the data’s attributes, and \( s_k \) is a constant value. Therefore, \( C_i.Src = \{A_i\}' \), and we assume \( C_i.Dest = \{B_i\}' \), which is another attribute in \( D \). The re-writing procedure, which generates a new query \( Q' \) is the same regardless of the \( C_i \)'s type. That is, the three different correlation-driven re-writing cases illustrated in Figure 4 (The 2\textsuperscript{nd}, 3\textsuperscript{rd}, and 4\textsuperscript{th} conditions) execute the same \( \text{Re-Write}(Q, C_i, p_k: A_k \equiv s_k) \) procedure. \( \text{Re-Write} \) augments an additional predicate in (a conjunctive form) to \( p_k \), where the format of the new predicate depends on \( C_i \)'s granularity as illustrated in Figure 4 (bottom table).

After generating the new query \( Q' \), the execution plan to generate the correct results depends on the correlation’s type and the adopted deployment strategy. That is, in the cases where \( C_i \) is a hard correlation or an Exclusion-based soft correlation, only the new query \( Q' \) needs to execute on the original dataset \( D \) (The 2\textsuperscript{nd} and 3\textsuperscript{rd} cases in the flowchart). Whereas, in the case where \( C_i \) is a Materialization-based correlation, executing \( Q' \) on \( D \) may not be enough as it may miss some data records that satisfy \( Q \) but in violation of \( C_i \), i.e., may miss records in \( \Gamma(C_i) \). Therefore, the new query \( Q' \) executes on \( D \), and also the original query \( Q \) executes on the execution bucket file \( D.C_i.\text{-ExpBucket} \) and then the results union together to generate the final correct results. Since the results from the two queries are guaranteed to be disjoint, there is no special processing needed to eliminate duplicates when constructing the final answer.

5. EXPERIMENTS

5.1 Implementation & Setup Details

Implementation Details: EXORD is implemented as an extension to Apache Hive 1.2.0. We used MySQL DBMS as the metadata repository engine. We extended Hive SQL interface and added a new command ‘’Create Correlation ...’’ through which the database admins can define the soft and hard correlations along with their parameters—As introduced in Def. 2.2. The metadata repository stores the defined correlations, their status, the statistics collected from the validation phase, and the assigned strategy. In addition, for the Exclusion-based soft correlations, the distinct violating values are also stored in the metadata repository.

Cluster Setup: We used Apache Hadoop infrastructure (version 1.1.2). All experiments are conducted on a dedicated local shared-nothing cluster consisting of 20 compute nodes. Each node consists of 32-core AMD 3.0GHz CPUs, 128GB of memory, and
Table: Source and Destination Columns

<table>
<thead>
<tr>
<th>ID</th>
<th>Source Column</th>
<th>Dest Column</th>
<th>Short Desc.</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>StartAirportIATA</td>
<td>StartCountry</td>
<td>In most cases, the airport code determines the country (Refer to Motivation Scenario 2)</td>
<td>Exclusion</td>
</tr>
<tr>
<td>C₂</td>
<td>ConfTimestamp</td>
<td>RequestTimestamp</td>
<td>In most cases, the ConfTimestamp is within 1 to 5 mins from the RequestTimestamp</td>
<td>Materialization</td>
</tr>
<tr>
<td>C₃</td>
<td>TicketPrice</td>
<td>TicketClass</td>
<td>The price gives an indication on the ticket class code.</td>
<td>Materialization</td>
</tr>
</tbody>
</table>

Figure 5: Correlations in the Working Dataset.

<table>
<thead>
<tr>
<th>DataSet Size</th>
<th>Partitioning (map-reduce Job)</th>
<th>Indexing (map-reduce Job)</th>
</tr>
</thead>
<tbody>
<tr>
<td>500 GBs</td>
<td>Storage: 962, Time (sec): 192 GBs</td>
<td>Storage: 2605, Time (sec):</td>
</tr>
<tr>
<td>1 TB</td>
<td>2173, 413 GBs</td>
<td>6643</td>
</tr>
<tr>
<td>2.3 TBs</td>
<td>~ 7 TBs, 4669, 1.12TBs</td>
<td>16369</td>
</tr>
</tbody>
</table>

Figure 6: The storage (including 3-way replica) and time overheads of creating an additional data organization (Partitioning or Indexing) for Current-Auxiliary (Aux) technique.

2TBs of disk storage, and they are interconnected with 1Gbps Ethernet. We used one server as the Hadoop’s master node, while the other 19 servers are slave nodes. Each slave node is configured to run up to 20 mappers and 12 reducers concurrently. The following Hadoop’s configuration parameters are used: sort buffer size was set to 512MB, JVM’s are reused, speculative execution is turned off, and a maximum of 4GB JVM heap space is used per task. The HDFS block size is set to 128MB with a replication factor of 3.

Datasets (Application & Synthetic): Most of our experimental evaluation uses an application dataset from the airline analytics domain. In addition, we generate a synthetic dataset to stress test some extreme cases and broader ranges of configuration parameters. The application dataset contains airline traffic logs from 100s of airline companies and consists of customers’ ticket reservation records. Each record has more than 80 attributes, however the key ones of interest to us include: StartCountry, StartCity, StartAirportIATA, which define the starting point of a flight (and similar attributes exist for the destination point), RequestTimestamp, ConfTimestamp, which define the timestamp of a user requesting a reservation, and the timestamp of confirming the reservation, and the TicketClass, TicketPrice attributes, which define the seat class and the corresponding price. The total size of the dataset is around 2.3 TBs, and with the 3-way replication the total size is around 7 TBs in HDFS. For experimental purposes, we create three versions of the dataset with different sizes, which are Small (500GBs), Mid (1.0 TB), and Large (2.3 TBs).

The dataset has several interesting soft correlations. In Figure 5, we summarize few correlations that will be our focus. Correlation C₁ is explained in detail in Motivation Scenario 2 in Section 1. Correlation C₂ indicates that in the majority of the cases, the confirmation timestamp is within 1 to 5 mins after the request timestamp; however there can certainly be exceptions to this correlation. Moreover, there is a correlation between the ticket price and the ticket class (Correlation C₃), and we are going to synthetically control such correlation in different ways as will be explained in the experiments.

5.2 Performance Evaluation

5.2.1 Query Execution Gain

We start by studying the gain from using EXORD in query execution. We assume the soft correlations described above are already processed and they are in their deployment phase. As indicated in Figure 5, Correlation C₁ uses the Exclusion strategy, while the remaining correlations use the Materialization strategy. In the following, we demonstrate the effectiveness of EXORD in the context of both selection and aggregation queries on top of two special access methods, namely partitioning and indexing.

EXORD is compared against two baseline techniques, which are referred to as “Current” and “Current-Auxiliary” (or “Aux”) for short. For a given query involving a selection predicate over an attribute A, Current would perform a full scan over the data since A—by default—has no special organization associated with it. In contrast, in Aux, we manually build an additional special organization over A to efficiently support its queries. In Figure 6, we report the storage and time overheads involved in building such additional organizations. The key observation is that building these additional organizations involves significant overheads that make it almost impractical to build several of them over a single dataset. As the experimental evaluation will show, EXORD can provide very similar performance to Aux without paying this huge cost upfront.

Correlation C₁: To study the benefits of using Correlation C₁, we first partition the data on the StartCountry attribute. Nevertheless, our equality-based selection query (C₁-selection) involves a selection predicate on the StartAirportIATA attribute instead. In the query, we experiment with 10 different airport codes (excluding “***”) from different countries to have diverse selectivity, and then average their results. The performance of the C₁-selection query is presented in Figure 7(a). The current technique [15] (labeled as “Current”) have to perform a full scan over the data. EXORD makes use of Correlation C₁ and adds an additional predicate over the StartCountry which enables leveraging the existing partitioning, and “Aux” makes use of the additional partitioning created based on StartAirportIATA attribute. As the results show, both EXORD and Aux achieve a factor of 12x speedup, and their performance is identical as they both touch only the same relevant partitions. The key difference between them is that EXORD does not pay the partitioning overhead reported in Figure 6.

In Figure 7(b), we study a more complex variation of Query C₁-selection, i.e., C₁-aggr, in which the query involves an equality-based selection followed by an aggregation. In this case, the aggregation overhead—more specifically, the shuffling/sorting and the reduce phase overheads—are the same to all techniques including EXORD. Yet, the benefits from EXORD are still significant as it saves around 65% of the job’s execution time, which is mostly due to the savings in the map phase. Again, EXORD has identical performance to Aux.

Correlation C₂: In Figure 8, we demonstrate the usage of Correlation C₂ under the presence of indexes. In these experiments, we build a local Hadoop++ index over the RequestTimestamp attribute in each partition and use a customized InputFormat to access the data as introduced in [6]. Our experimental queries C₂-selection and C₂-aggr involve a selection predicate on the ConfTimestamp attribute instead. Correlation C₂ uses the Materialization strategy for storing the violating records in a separate exception bucket. The exception bucket sizes are 273MBs, 695MBs, and 1.17GBs for the datasets of sizes 500GBs, 1TB, and 2.3TBs, respectively. We repeat each experiment with 5 different partitions, and for each timestamp we execute three experiments with matching granularities of a Second, Minute, and Hour to have different selectivities.

In Figure 8(a), we present the results of the selection query C₂-selection. The state-of-art technique [6] (“Current”) have to perform a full scan over the data without the use of the index. EXORD triggers two jobs; one over the entire dataset and leverages the index, and another one to scan the exception bucket, and “Aux” leverages the additional index built on the ConfTimestamp attribute. As the results show, EXORD achieves up to 82% reduction in the
query time compared to Current, and the only additional overhead compared to Aux is the processing of the exception bucket—which is relatively very small compared to the data.

In Figure 8(b), we present the performance of the aggregation Query $C_2$-aggr in which we aggregate over the StartAirportIATA attribute and calculate the count of the entries for Correlation $C_3$. Both the Current and Aux techniques execute the query in a single map-reduce job, whereas EXORD executes three jobs (one selection over the entire dataset, one selection over the execution bucket, and then one aggregation on the output of the first two jobs). Again, EXORD achieves around 50% reduction in the query time compared to Current and has a slight overhead compared to Aux. Yet, this overhead is negligible compared to the pre-processing overhead that Aux pays to build the index, which would require around 150 of such $C_2$-aggr queries to just redeem the index-building overhead.

**Correlation $C_3$:** For Correlation $C_3$, we use the largest dataset of size 2.3TBs which contains 763 distinct codes in the TicketClass attribute, and we partition the data based on that attribute. We synthetically assign price ranges for the class codes such that some ranges are unique to specific classes, i.e., given a price in that range it maps to a single ticket class, while other ranges map to several (or all) classes. The purpose of such assignment is to have various selectivities as will be described next. In these experiments, we vary the percentage of the records violating the price class correlation over the values of {0.1%, 0.5%, 1%, 5%}, which lead to the exception bucket sizes of {1.9GBs, 11.4GBs, 22GBs, 97.3GBs}, respectively.

In Figures 9(a) and 9(b), we illustrate the performance of the selection query ($C_3$-selection), and its extended aggregation query ($C_3$-aggr), respectively. The selection predicate is on the TicketPrice attribute. On the x-axis of the figures, we vary the TicketClass selectivity of the price predicate such that the price maps to either 1, 10, 50, 500, or 763 (All) classes. The overall record selectivity from the predicate is kept the same of approximately 10% of the input data. For both experiments, the Current technique would perform a single job (either a map-only for $C_3$-selection, or a map-reduce for $C_3$-aggr) that scans the entire dataset. In contrast, EXORD would perform two map-only jobs for $C_3$-selection, and an additional third map-reduce job for $C_3$-aggr.

On the other hand, Aux makes use of the additional partitioning created on the TicketPrice attribute and can answer both queries in a single job as in the Current technique.

As the results show, both EXORD and Aux are very efficient compared to Current and their performance increases relative to the number of relevant partitions that need to be touched. In the worst case, where all partitions need to be touched, Aux performs identical to Current, and EXORD has a slight overhead due to the processing of the exception bucket (and triggering three jobs instead of one in the case of $C_3$-aggr in Figure 9(b)).

**Synthetic Dataset.** We generate a synthetic dataset of size 1TB (3 TBs with replication) to stress test the query selectivity and violation percentage parameters. The dataset is generated as follows. Each record consists of four attributes; namely Id, Field1, Field2, and Tail. The first three attributes are integers, and the Tail attribute is a large text field of size 3KBs. Our focus is on Field1 (which has an index by default), and Field2 which has a soft correlation (based on a simple mathematical formula) to Field1. Field1 has the domain range between 1 and 10^6. The tested query is a selection query over Field2 with a range predicate that varies the record selectivity from 1% to 100% as indicated in Figure 10, and within each selectivity degree, the correlation’s violation percentage varies from 0% (Hard correlation) to 40%. In Figure 10, we compare Current (which performs a full scan), Aux (which builds and leverages an additional index over Field2), and EXORD. The key insights from the figure are that: (1) Allowing large exception buckets may hurt the performance and may diminish the savings from exploit the correlation. Typically, it is recommended to only accept correlations, i.e., consider them as valid according to Def. 2.4, only if the violation’s percentage is around or below 10% of the base data. A more concrete recommendation is to have the upper bound on the exception bucket size, i.e., MaxViolRec in Def. 2.4, equal to the amount of data that can be processed by a single wave (or at most two waves) of mappers. For example, in our cluster setup, we have 400 concurrent mappers and each processes 128MBs of data, which leads to 51GBs that can be processed in a single wave of mappers. In Figure 10, the 1% and 5% violations fit in one wave, while 10% fits in two waves (and it doubles for 20%
and 40%). And (2) Even with bigger selectivity percentage, e.g., 80% and 100%, the overheads from the index processing and the exception bucket are relatively small compared to Current—if we exclude the un-recommended settings of 20% and 40% violations. These overheads are between 5% for Aux and EXORD-0% violation, and 11% for the EXORD-10% violation. The index overhead is due to increasing the dataset size (by around 10%), and hence increasing the I/O cost.

5.2.2 Validation and Preparation Overheads

We now focus on evaluating the overheads involved in the validation and preparation of soft correlations.

Validation Overheads: In Figure 11(a), we show the validation overhead if it were to be performed as a separate system-triggered map-only job without piggybacking. This experiment verifies the importance of piggybacking to avoid the high encountered overhead. The figure shows the execution time for each of the three dataset sizes while validating the the three correlations $C_1$, $C_2$, and $C_3$. The small stacked bars illustrate the additional time overhead in the cases of having 6, 12, and 24 correlations to be validated instead of just 3 correlations (these correlations are replicas of the 3 correlations with slight variations). The key observation is that a dedicated scan over the data just to validate the correlations can be expensive, especially in the case of large datasets.

In Figures 11(b) and 11(c), we illustrate the piggybacking performance that EXORD applies over a map-only, or map-reduce job, respectively. As expected, if the validation task is piggybacked over a map-reduce job, then the overhead is negligible, and many correlations can be verified at the same time compared to the additional CPU cost for checking more correlations is almost entirely masked by the job’s own execution time. In the case of the map-only job, the additional overhead percentage ranges from 3% to 13% (See Figure 11(b)). It is worth highlighting that as the dataset gets larger, the scanning overhead of reading the data from HDFS and starting more map tasks dominates the overall cost, and thus the additional CPU time becomes less apparent.

Figure 11: Validation Overhead of Soft Correlations.

Figure 12: Preparation Overhead of Soft Correlations.

Preparation Overheads: For the Exclusion strategy, the preparation overhead involves the insertion of the violating values into EXORD’s metadata repository, which is a MySQL DB. The overhead of this task is tiny and negligible as it takes few seconds for the values to be inserted into the database. For completeness, we report in Figure 12(a) the time overhead to insert a number of values varying from 1 (which is the case for correlation $C_1$) to 10,000 into MySQL database. On average, each value is 20 bytes, and the database table has a B+-tree index built before inserting the values.

For the Materialization strategy, the preparation overhead involves a piggybacked job to collect the violating records and copy them to an exception bucket. In Figure 12(b), we report this overhead under the two cases of piggybacking the preparation job over a user’s map-only job (scanning and reporting 10% of the data) and a map-reduce job (aggregating over the entire dataset). On the x-axis, we consider the creation of the exception buckets corresponding to the Correlations $C_2$ and $C_3$ studied in Section 5.2.1, and the y-axis shows the overhead percentage to the user’s job. As expected, the overhead depends on the exception bucket’s size, and it is clearly much smaller if the user’s job is map-reduce job. However, in EXORD, we perform the piggybacking over the first user’s job regardless of whether it is a map-only or map-reduce. This is because the savings in query execution are significant and they redeem the paid overhead in just one job.

5.2.3 Optimizations of Correlation Selection

In this section, we evaluate the proposed heuristic-based algorithm for correlation selection under limited resources. We enumerated 16 different correlations that the domain experts believe to exist in our working dataset. For example, in addition to the three correlations presented in Figure 5, other similar correlation include: StartAirportIATA → StartCity, DestAirportIATA → DestCountry, DestAirportIATA → DestCity, and RequestTimestamp → ConfTimestamp. Other interesting correlations include ToFlightDays → Purpose, and ToFlightDays → MapReduce. Capturing these correlations is not only important with query optimization, but also important with the business logic and providing better pricing models. In addition
Figure 13: Heuristic-Based Selection Correlation.

to these correlations, we replicated the TicketPrice attribute in correlation C3 three more times, and replicated C2 along with these new attributes. The purpose of this replication is to have the four violation percentages studied in Figure 9(a) all present in the dataset at once. Therefore, the selection pool has 19 soft correlations.

We then execute a validation task to collect statistics on these correlations. In Figure 13(a), we illustrate the exception bucket sizes for the 19 correlations (ordered by the size). Only 4 out of the 19 correlations qualify for the Exclusion strategy. Therefore, to simplify our experimental setup, we assume that these 4 qualified correlations will all fit in the metadata allowed pool $M_{pool}$ since the total combined sizes of their violating values are less than 1MB. We then focus on the optimization selection problem of the remaining 16 correlations competing for the $H_{pool}$ resource.

To setup the experiment, we vary the available resource pool $H_{pool}$ over the values 100GBs, 150GBs, and 200GBs, and the $\Gamma()$.size costs of the 16 correlations are set according to the results in Figure 13(a). The benefit model of the correlations is created as follows. We build a workload of 50 queries, and distribute these queries over the ExclusiveCoverage(), and Coverage() sets according two configuration parameters, namely ExFactor, and CovFactor. The ExFactor, which varies over the values {0, 2, 4}, defines the number of queries that are exclusively assigned to correlations (random assignment). And then, for the remaining correlations, each will be replicated between 2 to CovFactor times over the Coverage() set of some different correlations (random assignment). The CovFactor varies over the values {2, 3, 4}. For example, if CovFactor is set to 3, then each query is replicated either 2 or 3 times (random selection within this range), and the corresponding correlations are also chosen randomly.

As discussed in Section 3.2.3, the computational complexity of the proposed heuristic-based algorithm for correlation selection is $O(n \log n)$, where n is the number of correlations. In contrast, the optimal brute-force algorithm is exponential as it entails enumerating all permutations of the 16 correlations of all sizes from 1 to 16 (no repetition, but the order matters) and selecting the highest-benefit feasible solution.

We omit the detailed results of the heuristic-based algorithm under each configuration of the three changing variables $H_{pool}$, ExFactor, and CovFactor, due to space limitations. In summary, the execution time is in the order of milliseconds ranging from 34 to 52 milliseconds. In these experiments, the dominance threshold $\epsilon$ (in Def. 3.7) is set to 0.15. In contrast, the brute-force algorithm is prohibitively expensive and a single execution takes around 7.8 Days (187.2 Hours). We executed the brute-force algorithm under three selected configurations to get their ground-truth optimal solution, which are: {150GBs, 0, 2}, {150GBs, 2, 3}, and {150GBs, 4, 4}, where the numbers correspond to the $H_{pool}$, ExFactor, and CovFactor, respectively. Clearly, the brute-force algorithm is not a practical solution.

Regarding the accuracy of the heuristic-based algorithm, we focus on the three configurations for which we obtained the optimal highest-benefit selection, and test the heuristic-based algorithm under three values for the dominance threshold $\epsilon$ as depicted in Figure 13(b). In addition, we suppress the heurisitic and thus the algorithm now maps to the static definition of the correlations’ benefits (Def.3.5). The y-axis of Figure 13(b) shows the error percentage computed as $100 \ast (opt - approx)/opt$, where opt and approx, are the optimal and the approximated values, respectively.

As the results in Figure 13(b) show, the static definition yields a relatively high error rate. This is due to unnecessary selection of correlations having high overlap in their coverage. When $\epsilon$ is set to zero, the error rate is also high because in this case the heuristic-based algorithm is filtering out only the correlations that are completely dominated by others, which are very few. The 0.15 and 0.3 threshold values give much better results as they are more flexible in their dominance definition, and they can filter out correlations that offer minor contributions even if they are not completely dominated by others. As a conclusion, the proposed heuristic-based algorithm is a practical algorithm compared to the non-practical brute-force search algorithm. The accuracy of the results is within an acceptable range, especially since the optimization effort is typically a best effort task in the first place.

6. RELATED WORK

Query Optimization in Big Data. Query optimization in big data is a fundamentally important problem, especially because (1) the datasets to be processed are getting very large, (2) the analytical queries are increasing in complexity and may take hours to execute if not carefully optimized, and (3) the pay-as-you-go cost model for cloud computing adds additional urgency for optimized processing. Because of these reasons, various aspects of query optimization have been studied on the emerging highly-scalable infrastructures, e.g., Hadoop [22]. These optimizations include techniques such as indexing [4, 6, 8], pre-partitioning [15], reorganization and colocation [9], materialization and re-usability of intermediate results [3, 7, 19], among many others.

Although the aforementioned optimizations have shown to be very effective in saving system’s resources and execution time, they do not come for free. Instead, they usually encompass significant overheads w.r.t time and storage [4, 7, 8, 9] (Refer also to our reported results in Figure 6). The proposed EXORD system is complementary to and can work in conjunction with most of the existing techniques, e.g., indexing either local indexes [4, 6] or global indexes [8], pre-partitioning [9, 15], and materialization [7]. The key advantage is that EXORD would enable optimizing a broader class of queries (beyond those on a single indexed or partitioned attribute) with minimal additional cost. Without EXORD, existing systems either perform a full scan, which is up to 10x slower, or pay the high cost of building more auxiliary structures, and still get almost the same performance as in EXORD.

Data Correlations in Relational DBs. Correlations represent important features of the data, which if effectively captured and leveraged would lead to significant improvement in query processing [2, 14, 16, 17]. That is why discovering and exploiting correlations have been extensively studied in RDBMSs including functional dependencies [12, 18, 24], conditional functional dependencies [1, 10], soft correlations [2, 14, 16, 17], and denial constraints [5]. The closest techniques to EXORD (in spirit) include BHUNT [2], CORDS [14], Correlation Maps (CM) [16], and CORADD [17], which all try to discover and exploit soft correlations in query optimization. However, these techniques are either heavily tailored towards the processing mechanism of relational DB—which is fundamentally different from Hadoop-like infrastructures—and thus they are not applicable in our context [14, 16, 17], or very restricted in their correlation definition compared to EXORD [2].
In more details, CORDS [14] tries to discover only the presence of correlations between pairs of attributes (say $A_1$ and $A_2$) and estimate their strength without keeping track of the detailed mappings from one attribute to the other. This is because CORDS objective is not query re-writing but instead providing better estimation for predicate selectivity for queries involving conjunctive predicates, e.g., over $A_1$ and $A_2$. This is crucial in RDBMSs because under the typical assumption of independence the estimated selectivity can be way off, which leads to bad query plans and significant un-necessary overheads. However, in Hadoop-like infrastructures there is no notion of conjunctive predicate selectivity or different query plans; it is always a single plan (map-only for certain types of jobs or map-reduce for other types). Even more, predicate selectivity in general is not critical in Hadoop-like systems because as we discussed in Section 1 (and confirmed by the experimental evaluation), it is safe to always leverage an index (or partitioning) if exist instead of a full scan regardless of the selectivity.

The CM [16] and CORADD [17] techniques have the observation that secondary indexes (on un-clustered attributes) usually have poor performance due to the random access of disk pages. However, if such un-clustered attributes are correlated with the clustered attribute (on which the relation is sorted), then by re-writing the query and adding predicates on that clustered attribute a significant performance gain can be achieved. The two systems have focused on capturing such mappings using new compressed data structures, called Correlation Maps (CMs), instead of the traditional secondary indexes [16], and providing better design for the database in the form of materialized views and recommendations for their clustered attributes [17]. Again, these issues although fundamental in RDBMSs, they are not applicable to Hadoop-like infrastructures. First, big datasets typically do not have a clustered attribute on which the entire dataset is sorted. Second, as we highlighted in Section 3.2.2, there is no notion of a random vs. sequential accesses in Hadoop-like systems.

On the other hand, BHUNT [2] has the same objective as EXORD, which is capturing the soft correlations and the mapping mechanism from one attribute to another, and then using that for query re-writing and adding additional predicates. However, as an automatic discovery tool, BHUNT puts strong restrictions on the correlations that can be captured. First, the attributes $A_1$ and $A_2$ have to be numerical (numbers or dates), and second their mapping has to be an algebraic expression in the form of $(A_1 \oplus A_2)$, where $\oplus$ is one of {+, -, *, /}. This significantly limits the applicability of the system in big data applications. In contrast, EXORD is a validation tool, which enables defining correlations on attributes of any data type, and domain experts can provide more complex and broader ranges of mappings, e.g., complex expressions or even look-up functions searching auxiliary metadata information.

7. CONCLUSION

We proposed the EXORD system for exploiting the data’s correlations in the context of big data query optimization. EXORD supports two types of correlations; the hard correlations—which are guaranteed to hold for all data records, and the soft correlations—which hold for most of the data records with a degree of uncertainty. We introduced a multi-phase approach for the validation, selection, and deployment of the soft correlations. We proposed a novel cost-benefit model that maps the adaptive selection of the most beneficial soft correlations w.r.t a given query workload to the well-known submodular knapsack optimization problem. And as an NP-Hard problem, we proposed a heuristic-based algorithm to efficiently solve it in a polynomial time. EXORD is an infrastructure-independent system that can be integrated with various big data query optimization techniques, e.g., indexing, partitioning, and materialization. Our prototype is implemented as an extension to the Hive engine on top of the Hadoop infrastructure. The experimental evaluation has shown that, with minimal overheads, EXORD can optimize a broader class of queries and speeds them up by more than 10x compared to the state-of-art techniques.

8. REFERENCES