Noname manuscript No.
(will be inserted by the editor)

Adaptive Correlation Exploitation in Big Data Query Optimization

Yuchen Liu, Hai Liu, Dongqing Xiao, Mohamed Y. Eltabakh
Worcester Polytechnic Institute (WPI), Computer Science Department, MA, USA
{yliu4, hliu2, dxiao, meltabakh}@cs.wpi.edu

the date of receipt and acceptance should be inserted later

Abstract Correlations among the data attributes are abundant and inherent in most application domains. These correlations, if managed in systematic and efficient ways, would enable various optimization opportunities. Unfortunately, the state-of-art techniques are all heavily tailored towards optimizing factors intrinsic to relational databases, e.g., predicate selectivity, random I/O accesses, and secondary indexes, which are mostly not applicable to the modern big data infrastructures, e.g., Hadoop and Spark. In this paper, we propose the *EXORD*$^+$ system for exploiting the data's correlations in big data query optimization. EXORD$^+$ supports two types of correlations; *hard* (which does not allow for exceptions) and *soft* (which allows for exceptions). We introduce a three-phase approach for managing soft correlations including: (1) Validating and judging the worthiness of soft correlations, (2) Selecting and preparing the soft correlations for deployment, and (3) Exploiting the correlations in query optimization. EXORD$^+$ introduces a novel cost-benefit model for adaptively selecting the most beneficial soft correlations given a query workload. We show the complexity of this problem (NP-Hard), and propose a heuristic to efficiently solve it in a polynomial time. Moreover, we present incremental maintenance algorithms for efficiently updating the system's state under data appends and workload changes. 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 project is partially supported by NSF-CRI 1305258 grant.

Authors' address:
Worcester Polytechnic Institute
100 Institute Rd., Worcester, MA, USA, 01609.
Tel.: +1-508-831-6421
Fax.:+1-508-831-5000

1 Introduction

Most big data applications involve numerous correlations and relationships among their data attributes. These correlations range from *"hard correlations"* that must be satisfied by all data tuples, to *"soft correlations"* that are satisfied by most (but probably not all) data tuples. For example, in transaction log applications, a zip code attribute may imply the location attributes, e.g., city and state (hard correlation), whereas in online marketing applications, a delivery date can be within 3 to 10 days of the shipping date in most cases (soft correlation). In general, a *"correlation"* from one attribute A_1 to another attribute A_2 means that their values are not independent. Instead, a value in A_1 may determine a unique value, a possible range, or a list of possible values for the corresponding A_2 attribute.

In traditional DBMSs, defining these correlations has an immense advantage in both data integrity [11], and query optimization [3,15,17,18]. Unfortunately, no similar technology for capturing and exploiting the data's correlations has been investigated in big data infrastructures such as Hadoop and Spark systems. As a result, such important data properties have been abandoned by the state-of-art big data optimization techniques.

1.1 Benefits in Big Data Query Optimization

A key challenge in exploiting the data's correlation in query optimization is that domain experts may only be able to provide their expectations on the possible correlations without guarantees, i.e., most correlations are *soft correlations*. There can be many of such candidate correlations with no clear evidence on which ones are truly useful. In the following motivation scenarios, we demonstrate that despite the uncertainties inherent in these correlations, they still carry big opportunities for query optimization.

Motivation Scenario 1– Online Marketing and Usage of Data Indexing: *Analytics over transaction logs generated from online marketing is a typical big data application. An example of soft correlations that may exist among the data attributes is that the delivery date in most cases (but not necessarily all cases) is within 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., [5,7,9]. 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. 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 generate wrong results by missing the 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. Assume the data is already partitioned on the country attribute to optimize certain queries [10,16]. 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 leverage any available indexing [5,7,9] or partitioning [10, 16] on the data, without the need to create additional ones. This is crucial because building auxiliary structures over big data is very expensive in both time and storage [7,9,10], and thus should be kept to minimal whenever possible (Refer also to our analysis in Section 7). For example, referring to Motivation Scenario 1, creating an additional index on the *delivery date* field to remedy the problem may not be a justified solution because the index may not be used with enough frequency to redeem its cost.

1.2 Limitations of Existing Techniques

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., [3,15,17,18]. However, as we will discuss in Section 8, most of these techniques have objectives that are specific only to RDBMSs, e.g., enhancing the selectivity of conjunctive predicates, avoiding random I/O accesses, and enhancing the usage of secondary indexes. These issues, although fundamental in RDBMSs, are not applicable to the highly-distributed big data infrastructures, e.g., Hadoop. For example, Hadoop has two rigid execution plans; either map-only or map-reduce, and predicate selectively plays no role on deciding the execution plan. Moreover, given the distributed retrieval of the data in Hadoop clusters, there is no notion of random I/Os. Finally, given the batch nature of big data queries, in which milliseconds are not a big deal (unlike RDBMS queries), it is always safe in Hadoop-like systems to use indexes to answer selection queries regardless of their predicate selectivity (we will show that the worst case has little overhead over full scans).

1.3 EXORD$^+$and its Contributions

In this paper, we propose the *"EXORD$^+$"* system for Exploiting soft and hard correlations in big data query optimization (Refer to Figure 1), which is an extended version to our previous system [19]. EXORD$^+$ targets the emerging Hadoop-like batch processing infrastructures, which are fundamentally different from RDBMSs in their distribution nature, query processing, data retrieval, and index access patterns. EXORD$^+$'s main objective is to perform correlation-based query re-writing to trigger the usage of any available indexing or partitioning over the dataset, and thus avoid full scan plans. EXORD$^+$ introduces the following contributions:

(1) Balancing between System-Discovered and User-Defined Approaches: All existing techniques go one extreme, which is a *full discovery-based* approach [3,15,17, 18]. Although flexible, this approach puts major restrictions on the discovered correlations, e.g., BHUNT [3] is only limited to numerical attributes and the correlations must be in the form of a simple algebraic expression of one operator {+, -, *, or /}. The other extreme that fully relies on domain experts to precisely define the correlations present in the data is also a non-practical approach.

EXORD$^+$ puts a more practical assumption that domain experts have some (possibly uncertain) knowledge on the candidate correlations that may exist in the dataset. Therefore, EXORD$^+$ introduces and enables defining two types of correlations; *hard* and *soft* correlations. Then, the system takes the liability of automatically validating, assessing, and

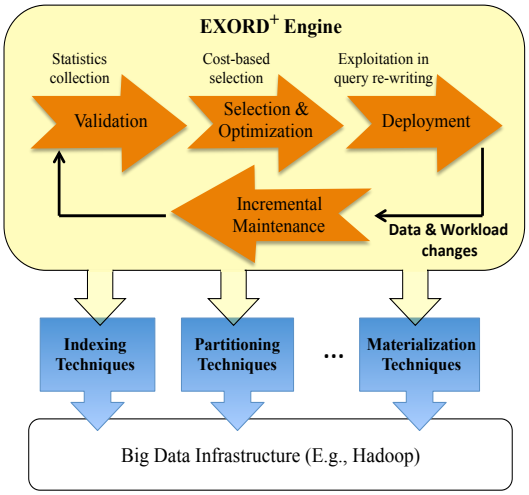


Fig. 1 EXORD$^+$ Architecture.

deploying the useful ones. 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.

(2) Multi-Phase Management of Soft Correlations: EXORD$^+$ introduces a three-phase pipeline for managing soft correlations (See Figure 1). The first phase is for collecting statistics and gathering information (*Validation Phase*), followed by a phase in which the system decides on the best strategy for handling the violating records (*Selection and Optimization Phase*). We propose two different violation-handling strategies, namely *Exclusion* and *Materialization*, which are suitable under different scenarios. Finally, the correlations worthy of being used are prepared for the query optimizer (*Deployment Phase*).

(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.

(4) Adaptivity to Incremental Data Appends and Evolving Workloads: Data appends and evolving query workloads are practical issues in most applications. Adaptive correlation maintenance is overlooked by all existing systems in literature. EXORD$^+$ introduces an incremental maintenance approach integrated into its cost-benefit model to efficiently update the system's state without the re-computing the multi-phase pipeline from scratch (See 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 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.

We utilized EXORD$^+$ on top of two well-known optimization strategies in Hadoop, i.e., indexing, and partitioning. EXORD$^+$ extends the benefits of these strategies beyond their targeted attributes to optimize a broader class of queries. 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 in some queries.

The rest of the paper is organized as follows. Section 2 introduces the formal definitions of EXORD$^+$ correlations. In Section 3, we present the first two phases of EXORD$^+$'s framework under a static environment, namely the *Validation* and *Selection and Optimization* phases. In Section 4, we propose enhancements and extensions to these two phases, and in Section 5, we propose the adaptive and incremental maintenance mechanisms under a dynamic environment. The deployment phase is presented in Section 6. In Sections 7 and 8, we present the experimental evaluation results, and the related work, respectively. Finally, the conclusion remarks are included in Section 9.

2 Preliminaries

2.1 EXORD$^+$ Correlations

In this section, we define the target queries to be optimized by EXORD$^+$, and formally introduce our definition of correlations.

Definition 1 (Target Equality-Predicate Query) *A target equality-predicate query is a query involving an equality predicate ($Src = a$), where Src is a data attribute of any type, a is a constant value, and Src has no associated access method to evaluate its predicate other than a full scan.*

EXORD$^+$'s goal is to leverage any available correlation related to Src to re-write the query in terms of another attribute that has more efficient access methods, e.g., indexing or partitioning, to evaluate its predicates. For the ease of presentation, we focus now on queries involving equality predicates as defined in Def. 1, and in Section 6.2, we relax this definition to include range-predicate queries under certain restrictions.

Definition 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 $\langle Src,\ Dest,\ Type,\ Granularity,\ F()\rangle$*, 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(s) is the mapping function that takes a value* $s \in Src$ *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").*

For a given correlation C and a data record r, we use the the notations "$C \rhd r$" (or "$C \not\rhd r$") for indicating that the correlation holds (or does not hold) over r, respectively.

Example 1: *Referring to our motivation scenarios, the correlation in Scenario 1 can be formulated as:* $\langle deliveryDate, shippingDate, \text{“}Soft\text{”}, \text{“}Range\text{”}, F()\rangle$*, where for a given delivery date d,* $F(d).lower$ *= (d -* $10\ days$*), and* $F(d).upper$ *= (d -* $3\ days$*). In contrast, the correlation in Scenario 2 can be formulated as:* $\langle airportCode,\ country,\ \text{“}Soft\text{”},\ \text{“}Singelton\text{”},\ F()\rangle$*, where for a given airport code a, Function* $F()$ *lookups 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 is neither restrictive nor limiting the applicability of the system because: 1) The Src and $Dest$ attributes in a correlation can be of any data type or domain and they can be numerical or categorical, and 2) The mapping function $F()$ may range from any mathematical or algebraic expression to general light-weight lookup functions that search some auxiliary structures and perform the mappings.

Definition 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, i.e.,* $C \rhd r\ \ \forall r \in D$.

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.

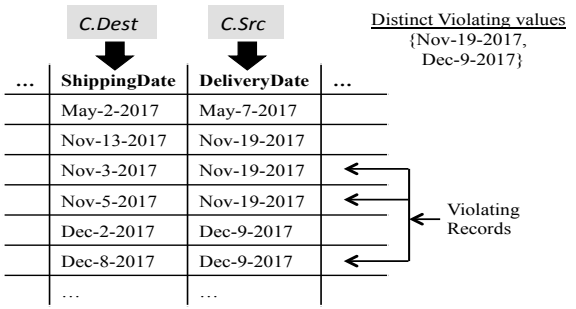


Fig. 2 Example of Violating Values and Violating Records (Refer to Example 1).

Definition 4 (Violating Value) *Given a dataset* D *and a soft correlation* C*, a value* $v \in C.Src$ *is called a violating value iff* $\exists\ r \in D,\ r.Src = v \mid C \not\rhd r$.

Example 2: *Continuing with Example 1, and referring to Figure 2, a value in the* $DeliveryDate$ *column (which is the correlation's* Src *column) is considered a violating value if it has at least one record in violation, e.g., Dates* $\{$*Nov-19-2017, Dec-9-2017*$\}$*. This indicates that it is not safe to blindly use the correlation on the transactions having these delivery dates.*

Definition 5 (Valid Soft Correlation) *For a soft correlation C of C.Type = "Soft", let* $\Phi(C)$ *denotes the set of distinct violating values in* $C.Src$ *(with cardinality* $|\Phi(C)|$*), and* $\Gamma(C)$ *denotes the set of violating records in the dataset (with cardinality* $|\Gamma(C)|$*). Given two user-defined thresholds MaxVioDistinct* > 1 *and MaxVioRec* > 1 *(Typically* $MaxVioDistinct << MaxVioRec$*), C is called "valid" iff either (or both) of the following two conditions is met, otherwise C is called "invalid".*

(1) $|\Phi(C)| \leq$ *MaxVioDistinct,*
(2) $|\Gamma(C)| \leq$ *MaxVioRec*

According to Def. 5, EXORD$^+$ considers a soft correlation to be *valid* (useful) when it satisfies at least one of the two given conditions. The following example illustrates the intuition behind having these two conditions.

Example 3: *Continuing with Example 1, assume a large transaction log dataset containing all the transactions delivered in Year 2017: R(..., ShippingDate, DeliveryDate, ...). The dataset consists of* 10^9 *records, and the two date fields are at the granularity of a day (See Figure 2). The soft correlation presented in Example 1 is defined on the dataset (referred to as C).*

Case 1: When testing C over R's records, a large number of transactions is found to be in violation (say 5×10^7*), which might not be feasible to keep track of all of them. However, all these violating records are happening around the Christmas season, i.e., in the two months of November and December. In this case, according to Condition 2 in*

Def. 5, C is useless because there are too many violating records. However, Condition 1 enables EXORD$^+$ to consider C useful by not tracking the violating records themselves, but by only memorizing that the 61 days of November and December are violating values. Transactions in any other date can make use of the correlation.

Case 2: When testing C over R's records, only 10^5 records are found to be in violation, which represents 0.01% of the original dataset size. However, these records are scattered across all days of 2017, e.g., in each day some transactions are in violation. In this case, according to Condition 1 in Def. 5, C is useless because all days have to be excluded from using the correlation. Nevertheless, Condition 2 still enables EXORD$^+$ to consider the correlation useful since the number of violating records is small.

2.2 MapReduce Overview

EXORD$^+$ targets the emerging batch processing highly-distributed infrastructures that are based on the MapReduce architecture. In this section, we briefly overview the building blocks of this architecture.

MapReduce Architecture: MapReduce is a popular widely-used architecture for managing big data. It is a batch-processing distributed architecture, where the data is uploaded to the system in big batches, e.g., large files of 100s of MBs or GBs of data. Then, users submit offline queries (A.K.A jobs) for processing the data. Common infrastructures that support this architecture are Hadoop [24] and Spark [27]. Other higher-level engines have been developed on top of these infrastructures to support structured data processing and SQL-like capabilities, e.g., Hive (for Hadoop) and SparkSQL (for Spark).

Job Tracker: This is the master node in a Hadoop's cluster. It receives a user's job, decides how many tasks to initiate, and where each task will execute (on which machine). Once tasks (map or reduce) are initiated, they work in total isolation of each other until completion—with the exception of heartbeat signals to report their progress and indicate that they are still alive.

HDFS File System: Hadoop operates on a distributed file system, called HDFS. Typically, applications upload large files into HDFS. Each file gets automatically partitioned into blocks, called HDFS blocks, of size 64MBs, which is the default setting. However, applications can change such configurations as desired. Then, each block is replicated in HDFS three times (again the replication factor can be configured by applications).

Map Phase: Any Hadoop job must involve a map phase in which Hadoop creates a number of *map tasks*, also called *mappers*, where each mapper is assigned one of the input file's blocks. A mapper reads the assigned block record-by-record, and for each input record, it executes the plugged-in application logic, and produces zero, one, or more output records. The output records must follow the format of <*key*, *value*> pairs. If the Hadoop's job involves only a map phase, then the output is written back to HDFS. Otherwise, the output is automatically fed to the following phases.

Shuffle/Sort Phase (Internal Phase): If Hadoop's job involves map-reduce phases, then the output from the mappers goes first through the internal *shuffle/sort* phase. In this phase, the data is automatically re-partitioned based on the *key* value produced from the mappers such that all records having the same key form one group in the form of <*key*, $\{v_1, v_2, ..., v_n\}$ >. This phase involves shuffling the data across machines—based on a hashing function, and then sorting the records in each machine to form the groups.

Reduce Phase: This is the last phase of a Hadoop's job in which Hadoop executes a set of *reduce tasks*, also called *reducers*. The number of these reducers typically depends on the cluster configuration (independent of the data size). Each reducer consumes as input a group-at-a-time from the set of assigned groups, executes the plugged-in application logic, and produces zero, one, or more output records. The output records must follow the format of <*key*, *value*> pairs, and they are written to HDFS.

Hive Engine: Apache Hive [25] is a SQL-like engine implemented on top of Hadoop to seamlessly support structured data. Hive allows users to create a relational schema (set of tables) to store their data. In our EXORD$^+$ system, a dataset corresponds to one Hive table. All metadata information related to a dataset, e.g., column names, and partitioning and ordering information, is defined while creating the table. Ultimately, a Hive table maps to a directory in HDFS, and any loaded files to that tables are stored as HDFS files under that directory. Hive maintains all the metadata information in a light-weight repository, called Hive Metastore. EXORD$^+$ makes use of this Metastore to store various types of correlation-related information.

3 EXORD$^+$ under Static Environment

As illustrated in Figure 1, the management of soft correlations is carried out using a multi-phase approach. In this section, we focus on the first two phases; the *Validation Phase*, and the *Selection & Optimization Phase*, under the assumption of a static environment, i.e., static dataset and query workload. We present various extensions to these two phases in Section 4, and then consider the adaptivity under a dynamic environment in Section 5.

3.1 Validation Phase

Assume a dataset D and a set of soft correlations in their validation phase $\mathcal{V} = \{C_1, C_2, ..., C_n\}$. EXORD$^+$ executes

Table 1 Testing correlation C_i over a given record.

Correlation Granularity	Test Format
Singleton	`Dest = C`$_i$`.F(Src)`
Range	`Dest` $\geq$ `C`$_i$`.F(Src).lower And Dest` $\leq$ `C`$_i$`.F(Src).upper`
List	`Dest in C`$_i$`.F(Src)`

a statistics collection task over D, called *StatsCollection*, to gather the needed statistics over each correlation. This task is a map-only job that checks each input record $r \in D$ against each soft correlation $C_i \in \mathcal{V}$ according to C_i's granularity as highlighted in Table 1.

EXORD$^+$ allows the *StatsCollection* task to be either manually triggered by users as a stand-alone task (through a high-level ``ASSESS CORRELATIONS ...'' command), or automatically triggered and piggybacked on the first user's job scanning the data. A Boolean system-level configuration parameter (`PiggybackEnabled`) controls this behavior. In the cases where the system might be idle for extended periods of time, the stand-alone execution is preferred to avoid adding additional overheads to users' queries and have them subject to optimizations immediately. Otherwise, the piggybacked option can be a good choice if the system cannot afford running separate background jobs. Additional implementation details are highlighted in Section 7.1.

For each correlation, each mapper in the *StatsCollection* task 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). Typically, $MaxVioDistinct$ is set to few thousands, e.g., $10,000$, 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. 5, 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.

In practice, collecting 100% accurate statistics from the entire dataset is not mandatory. In Section 4.1, we propose a *sampling-based* validation strategy that collects the needed statistics from a relatively small percentage of the dataset, e.g., 10% or 20%. We show—both experimentally and theoretically—that such strategy scales very well under large number of tested correlations while having a tiny and negligible error rate.

3.2 Selection & Optimization Phase

Unlike hard correlations that are ready for exploitation by the query optimizer, valid soft correlations require some preparation 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 for 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 cost-benefit model to automatically and adaptively select the correlations based on their costs and benefits given a 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 [23]. 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 6 (Exclusion Strategy) *For a given valid soft correlation C, the "Exclusion Strategy" is applicable to C iff Condition (1) in Def. 5 is True, and it involves copying the $\Phi(C)$ set to EXORD$^+$'s Metadata Repository.*

Definition 7 (Materialization Strategy) *For a given valid soft correlation C, the "Materialization Strategy" is applicable to C iff Condition (2) in Def. 5 is True, and it involves copying the $\Gamma(C)$ set to a separate file, called exception bucket, in the file system.*

Example 4: *To explain the rationale behind the two different strategies, we build on the two cases of Example 3:*

Exclusion Strategy: This strategy is designed to handle Case 1 in Example 3. More specifically, the Exclusion strategy relies on keeping track of only the distinct violating values (even if the number of violating records is very large). For example, the 61 days of November and December 2017 are extracted as illustrated in Figure 3(a). And then, whenever EXORD$^+$ sees these values in a given predicate, that

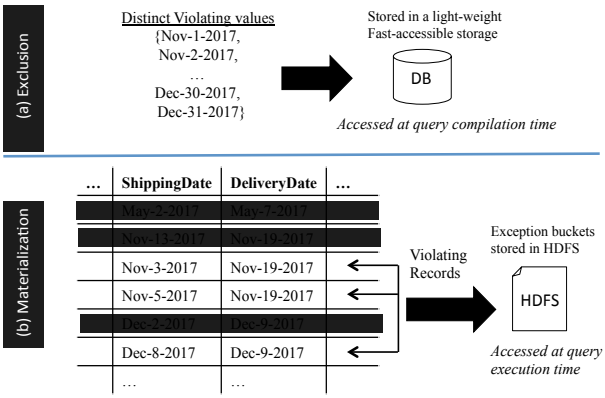


Fig. 3 Example of the Exclusion and Materialization Strategies.

predicate is excluded from being re-written or optimized for during the compilation and optimization time. Since the decision of such exclusion takes place at compile time, the distinct violating values are kept in EXORD$^+$'s Metadata Repository—which is a light-weight relational DBMS.

Materialization Strategy: This strategy is designed to handle Case 2 in Example 3. More specifically, the Materialization strategy relies on physically copying the violating records into separate files (referred to as "exception buckets"*)—recall that in HDFS, the update of the original file is not a possible operation. These exception buckets can be relatively large in size, e.g., the size of the* 10^5 *violating records in Example 3 can be in 100s of MBs. Therefore, these records are stored in the main HDFS file system (See Figure 3(b)). Nevertheless, as we will explain in Section 6, the exception buckets access is 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 8 (Resource Pool) *The Resource Pool is the maximum allowed storage that valid soft correlations can compete for and consume. It is denoted as* $RPool = < M_{Pool}, H_{Pool} >$*, where* M_{Pool}*, and* H_{Pool} *are the maximum sizes in the Metadata Repository, and the file system (HDFS), respectively.*

Recall that M_{Pool} is the storage space to be used for the *Exclusion* strategy, whereas H_{Pool} is the storage space to be used for the *Materialization* strategy.

Now, the formal cost model of soft correlations is defined as follows (Hard correlations always has zero deployment cost).

Definition 9 (Correlation Cost) *Let* $\Phi(C).size$ *and* $\Gamma(C).size$ *represent the size in bytes of the corresponding sets* $\Phi(C)$ *and* $\Gamma(C)$*. The deployment cost of a valid soft correlation* C *is defined as follows:*

$$Cost(C) = \begin{cases} < \Phi(C).size,\ \infty >, & \textit{if only Def. 5.(1) = True} \\ < \infty,\ \Gamma(C).size >, & \textit{if only Def. 5.(2) = True} \\ < \Phi(C).size, \Gamma(C).size >, & Otherwise \end{cases}$$

According to Def. 9, if only Condition (1) in Def. 5 is True, then the correlation is allowed to compete for a space in M_{Pool}, whereas if only Condition (2) is True, then it is allowed to compete for a space in H_{Pool}. And if both conditions are True, then the correlation is allowed to compete for both resources (although a higher priority is given to M_{Pool}). Notice that in the cost model, $\Phi(C).size$ can be exactly computed since the $\Phi(C)$ set is already available, whereas $\Gamma(C).size$ can be only estimated depending on the known $|\Gamma(C)|$ and its relative size to the base dataset.

3.2.2 Correlations Benefit Model

Given a workload of n queries $W = \{Q_1, Q_2, Q_3, ..., Q_n\}$, the benefit (reward) metric of a valid soft correlation C_i depends on several factors including: (1) The percentage of queries in W for which C_i is applicable, i.e., the queries that involve an equality predicate on $C_i.Src$ column. We refer to these queries as the $Coverage(C_i)$. (2) The percentage of queries in W for which only C_i is applicable, i.e., if C_i is not selected for deployment, then these queries will have no other correlations to optimize them. We refer to these queries as the $ExclusiveCoverage(C_i)$. (3) The actual execution savings, e.g., the wall clock time, achieved by C_i from executing a re-written optimized query Q compared to executing Q without re-writing. And (4) In addition to these factors, maximizing the coverage of W while minimizing the overall cost is also a desirable objective.

Example 5: *As an illustrative example, we present in Figure 4 a set of correlations* C_1*,* C_2*, and* C_3*, and a query workload consisting of four queries. An edge between a correlation* C_i *and a query* Q_j *indicates that* C_i *is applicable to* Q_j *and can be used to re-write and optimize this query. The edge labels, e.g., "*$X \rightarrow Y$*", indicate that the correlation can be used to re-write a predicate on its source column* X *in terms of a predicate on its destination column* Y *(and* Y *is assumed to have an associated efficient access method, e.g., an index). In the figure, we include the* $Coverage()$ *and* $ExclusiveCoverage()$ *sets of each correlation. For example, both* C_1 *and* C_2 *can be used to optimize the execution of* Q_2 *by targeting different predicates on different columns. That is why* Q_2 *is in their* $Coverage()$ *sets. In contrast, neither of* C_1 *nor* C_2 *has queries in their* $ExclusiveCoverage()$ *sets, but* C_3 *has* Q_4 *in its* $ExclusiveCoverage()$*.*

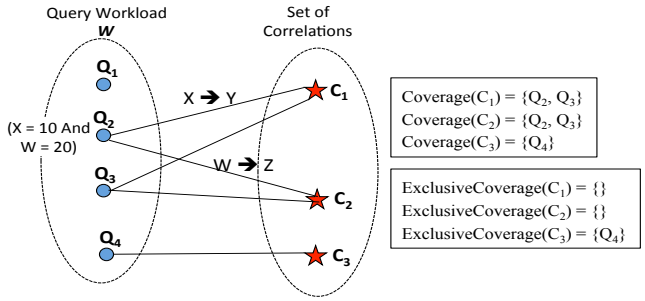


Fig. 4 Benefit Factors of Soft Correlations.

Referring to Figure 4, it is evident that correlations may have different priorities depending on each of the four factors mentioned above. For example, if we consider the 1^{st} factor (based on coverage), then C_1 and C_2 have higher (and equal) priorities over C_3, whereas, if we consider the 2^{nd} factor (based on exclusive coverage), then C_3 gets higher priority. Moreover, according to the 4^{th} factor (W's coverage with minimal cost), if C_1 is selected by the system, then C_2's priority should be significantly lowered since it does not cover any new queries beyond those covered by C_1.

Clearly, including all four factors into the benefit metric makes it very complicated especially because estimating the execution savings (The 3^{rd} 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 are significant compared to executing the un-optimized version (full scan).

This assumption implies that, it does not matter which of the two correlations C_1 or C_2 to use for optimizing Q_2, what matters is to have Q_2 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 [7,9] 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 3^{rd} 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 10 (Static Correlation Benefit) *For a given workload W of size n queries, and a soft correlation C_i, the static benefit of C_i is computed as the percentage of queries for which C_i is either applicable or exclusively applicable. That is:*

$$SBenefit(C_i, W) = \frac{|Coverage(C_i)|}{n}$$

Definition 11 (Dynamic Correlation Benefit) *For a given workload W, and a soft correlation C_i, the dynamic benefit of C_i is re-computed after the selection of any other correlation as follows:*

Initial State:

$$DBenefit(C_i, W) = SBenefit(C_i, W), \quad \forall\, C_i,$$

subsequent State after the selection (and removal) of correlation C_j:

$$W = W - Coverage(C_j)$$
$$DBenefit(C_i, W) = SBenefit(C_i, W), \quad \forall\, C_i.$$

The static correlation benefit (Def. 10) basically takes into account the 1^{st} and 2^{nd} factors, while ignoring the 4^{th} 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. 11) takes the 4^{th} factor into account, and thus whenever a specific correlation is selected, the benefits of the remaining correlations are re-calculated.

3.2.3 The Optimization Problem

The optimization problem is now to select a subset of valid soft correlations $\mathcal{C} = \mathcal{C}_\mathcal{X} \cup \mathcal{C}_\mathcal{M}$, where $\mathcal{C}_\mathcal{X}$ is the set of correlations assigned the Exclusion strategy, $\mathcal{C}_\mathcal{M}$ is the set of correlations assigned the Materialization strategy, and $\mathcal{C}_\mathcal{X}$ and $\mathcal{C}_\mathcal{M}$ are disjoint sets. The objective function is to maximizing the total benefit ($\sum_{\forall C_i \in \mathcal{C}} Benefit(C_i)$) subject to not exceeding the allowed resources by satisfying the following conditions:

$$\begin{cases} \sum \Phi(C_i).size \leq RPool.M_{Pool} & \forall C_i \in \mathcal{C}_\mathcal{X} \\ \quad \text{and} \\ \sum \Gamma(C_j).size \leq RPool.H_{Pool} & \forall C_j \in \mathcal{C}_\mathcal{M} \end{cases}$$

Clearly, if the benefit function follows Def. 10 (Static benefit), then the optimization problem maps to the classic "0/1 knapsack" problem, which is NP-complete, but efficient approximation algorithms exist with computable error bound [14]. On the other hand, if the benefit function follows Def. 11 (Dynamic benefit), which is semantically stronger, then the optimization problem maps to the "submodular knapsack" problem, which is even harder to solve or approximate than "0/1 knapsack" [23].

Because of that, we propose an algorithm that combines and retains the pros of both definitions. This is achieved by

Table 2 Cost()-Based Dominance Notation.

Dominance Type	Cost(C_i) ≤ Cost(C_j) IFF
$C_i \mapsto_{\Phi\Gamma} C_j$ (All-Cost Dominance)[1*]	$(\Phi(C_i).size \leq \Phi(C_j).size)$ And $(\Gamma(C_i).size \leq \Gamma(C_j).size)$
$C_i \mapsto_{\Phi} C_j$ (Exclusion Dominance)[2*]	$(\Phi(C_i).size \leq \Phi(C_j).size)$
$C_i \mapsto_{\Gamma} C_j$ (Materialization Dominance)[3*]	$(\Gamma(C_i).size \leq \Gamma(C_j).size)$

[1] C_i must satisfy Conditions (1) & (2) in Def. 5.

[2] C_i must satisfy at least Condition (1) in Def. 5.

[3] C_i must satisfy at least Condition (2) in Def. 5.

[*] C_j must satisfy Condition (1) or Condition (2) or both in Def. 5.

Id	Cost <Φ.size, Γ.size>	Coverage	Exclusize Coverage
C_1	<10, 200>	{Q_1, Q_2, Q_3, Q_4}	{}
C_2	<30, 700>	{Q_2, Q_3}	{}
C_3	<50, ∞>	{Q_5}	{Q_{10}}
C_4	<40, 100>	{Q_1, Q_2, Q_3, Q_4, Q_5}	{}
…	…	…	…

(a) Metadata on example correlations

ε = 0 (Total Dominance)
$C_1 \mapsto_{\Phi\Gamma} C_2$
$C_4 \mapsto_{\Gamma} C_2$
$C_4 \mapsto_{\Gamma} C_1$

(b) Dominance relationships (ε = 0)

ε = 0.2 (Partial Dominance)
$C_1 \mapsto_{\Phi\Gamma} C_2$
$C_4 \mapsto_{\Gamma} C_2$
$C_4 \mapsto_{\Gamma} C_1$
$C_1 \mapsto_{\Phi} C_4$

(c) Dominance relationships (ε = 0.2)

Fig. 5 Example on Dominance Relationships.

combining the static definition of the correlations' benefit (Def. 10) with a heuristic that captures the essence of the dynamic definition. More specifically, the heuristic prevents selecting correlations that adds no (or minimal) value to the already selected ones. The heuristic relies on the following definition of correlations' dominance.

Definition 12 (Correlations Dominance) *Given a dominance-relaxation threshold $\epsilon \in [0, 1)$, a correlation C_i (soft or hard) is said to dominate another correlation C_j (soft), denoted as $C_i \mapsto C_j$, iff $Cost(C_i) \leq Cost(C_j)$ according to Table 2, $ExclusiveCoverage(C_j) = \phi$, and $\frac{|Coverage(C_j) - Coverage(C_i)|}{|Coverage(C_j)|} \leq \epsilon$. The dominance type is either "All-Cost", "Exclusion", or "Materialization" according to Table 2.*

Definition 13 (Total and Partial Dominance) *Given two correlations C_i and C_j, where $C_i \mapsto C_j$, if $\epsilon = 0$, the dominance is called "total dominance", otherwise it is called "partial dominance".*

The main idea of the heuristic is that before solving the optimization problem, we apply a filtering step to eliminate correlations that are dominated (or partially dominated) by other correlations. In other words, instead of aiming for the optimal solution according to Def. 11 (which is very expensive), we aim for avoiding the worst-case scenario that Def. 10 may generate.

Example 6: *Figure 5 gives an example of few correlations and their dominance relationships based on Defs. 12 and 13. Figure 5(a) includes the metadata information for each correlation. The key observations from Figure 5(b) include: (1) C_3 cannot be dominated by any other correlation because its ExclusiveCoverage is not empty, (2) C_1 has an "All-Cost Dominance" over C_2, and it is also a "Total dominance", i.e., C_1's coverage is equal or superset of C_2's coverage, (3) C_4 has a "Materialization Dominance" over both C_1 and C_2, and it is also a "Total dominance", which indicates that if the three correlations are to compete for the H_{pool}, then C_1 and C_2 can be eliminated, And (4) If $\epsilon > 0$, i.e., allowing partial dominance (as in Figure 5(c)), then more dominance relationships can be leveraged. For example, for $\epsilon = 0.2$, C_1 is found have an "Exclusion Dominance" over C_4 (cost 10 is less than 40), and C_5 covers only one additional query (Q_5) out of its coverage set beyond what C_1 is already covering, i.e., only 20% of its coverage is useful compared to C_1.*

In Figure 6, 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 outcome 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 the dominated correlations based on All-Cost dominance, *Phase 1* solves the optimization problem for the M_{Pool} resource, and *Phase 2* solves the optimization problem for the H_{Pool} resource. Given the computational complexity of [14], which is *O($n\ log\ n$)*, the proposed heuristic-based algorithm has the same 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 *All-Cost* dominated by another correlation C_i ($C_i \mapsto_{\Phi\Gamma} C_j$). This is because C_j cannot compete against C_i for either of M_{Pool} or H_{Pool}.

In *Phase 1*, the remaining correlations compete for the M_{Pool} resource, i.e., compete for the *"Exclusion Strategy"*. The dominance heuristic is applied again to *"temporarily"* eliminate any dominated correlations based on the *Exclusion Dominance* type. After that, the optimization problem is solved approximately using the "FPTAS" technique in [14].

Workload-Driven Correlation Selection

Inputs:
- Set of valid soft correlations $\mathcal{P}$ = {C_1, C_2, …, C_n}
- Query workload $\mathcal{W}$
- Dominance-relaxation threshold ε

Output:
- subset of correlations to deploy $\mathcal{O}$ (initially empty) along with the deployment strategy for each one.

Phase 0: Heuristic-Based Filtering based on All-Cost Dominance
1- **For** (each soft correlation C_j in $\mathcal{P}$) **Loop**
2- **- If** ($\exists$ $C_i \mapsto_{\Phi\Gamma} C_j$) **Then** Delete C_j from $\mathcal{P}$
3- **End For**

Phase 1: 0/1 knapsack optimization for resource *RPool.M*$_{pool}$
//Heuristic-Based Filtering based on Exclusion Dominance
4- **For** (each soft correlation C_j in $\mathcal{P}$) **Loop**
5- **- If** ($\exists$ $C_i \mapsto_{\Phi} C_j$) **Then**
6- - Add C_j to *TempList*
7- - Delete C_j from $\mathcal{P}$
8 - **End If**
9- **End For**
10- *Result* = Solve the optimization problem [14]
11- Delete *Result* from $\mathcal{P}$
12- $\mathcal{O}$ ← (*Result* with assigned "*Exclusion*" strategy)

Phase 2: 0/1 knapsack optimization for resource *RPool.H*$_{pool}$
//Check correlations in TempList and may return some back
13- **For** (each correlation C_j in *TempList*) **Loop**
14- **- If** (none of C_j's dominating correlations is in $\mathcal{O}$) & ($\Gamma(C_j).size \neq \infty$) **Then** Admit back C_j to $\mathcal{P}$
15- **End For**

//Heuristic-Based Filtering based on Materialization Dominance
16- **For** (each soft correlation C_j in $\mathcal{P}$) **Loop**
17- **- If** ($\exists$ $C_i \mapsto_{\Gamma} C_j$) **Then** Delete C_j from $\mathcal{P}$
18- **End For**

19- *Result* = Solve the optimization problem [14]
20- $\mathcal{O}$ ← $\mathcal{O}$ + (*Result* with assign "*Materialization*" strategy)

Fig. 6 Workload-Driven Selection for Correlations.

The selected correlations are added to the output set $\mathcal{O}$, and 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 it is given a second chance to compete for H_{Pool} only if $\Gamma(C_j).size \neq \infty$. Second, the heuristic is applied based on the *Materialization Dominance* type. Finally, the selected correlations are added to the output set $\mathcal{O}$ and assigned the *"Materialization"* strategy.

• **Execution of the Selected Strategy:** For a given correlation C, if the assigned strategy is *Exclusion*, then no further preparation is needed since the $\Phi(C)$ set is already collected during the *Validation* phase, and the correlation advances to the deployment phase. In contrast, if the assigned strategy is *Materialization*, then a full scan map-only task, called *Prep4Deployment*[1], executes over the dataset D for reporting the violating records and materializing them into an exception bucket file uniquely identified for the $\langle D,\ C\rangle$ pair. After that C advances to the deployment phase.

4 EXORD$^+$ Design Extensions

4.1 Sampling-Based Validation Phase

The purpose of the *Validation* phase is to collect statistics on the violations of a given soft correlation C_i, and decide on whether or not C_i is valid according to Def. 5. In Section 3.1, we presented the *StatsCollection* task as a full-scan task. In this section, we present a less expensive sampling-based strategy for the *StatsCollection* task.

Granularity and Applicability: We adopt random sampling to sample from the input dataset (i.e., HDFS files) at the HDFS block level (see the validation phase below). We highlight below (and present more theoretical details in [20]) that random sampling is statistically sufficient and highly accurate for the types of statistics we are collecting assuming that the data values are randomly distributed across the HDFS blocks, i.e., no specific sorting or partitioning. Random sampling is also used in several papers to collect similar correlation-related statistics [3, 15, 17].

To ensure that the system applies the sampling-based strategy only on columns that have no specific partitioning or sorting, EXORD$^+$ applies the sampling at the granularity of a single correlation. That is, given a set of correlations to validate, say $\mathcal{V} = \{C_1, C_2, ..., C_n\}$, EXORD$^+$ identifies the correlations whose Src or $Dest$ columns have specific partitioning or sorting (say set $\mathcal{V}_{ORD}$). This information can be easily obtained from the system's metadata information (which is a Hive Metastore). Then, the *StatsCollection* tasks automatically adjusts its execution such that all mappers collect statistics for the correlations in $\mathcal{V}_{ORD}$, while only a small subset of them collect statistics for the rest of the correlations (say set $\mathcal{V}_{RAND}$). Given that in typical applications, the data might be ordered or partitioned only on one or two fields (if any), then the sampling strategy remain applicable for most of the correlations.

Modifications in Validation Phase: Instead of making all *StatsCollection* map tasks collect the violation statistics for all candidate correlations in $\mathcal{V}$, the system randomly selects a small percentage of these mappers (say $x\%$) to collect the statistics. Each map task randomly and independently decides with a probability $x/100$ on whether or not to test the corrections in $\mathcal{V}_{RAND}$ on its local data block[2]. Each of these mappers reports for each soft correlation $C_i \in \mathcal{V}_{RAND}$ an estimation for the two statistics presented

[1] Depending on the `PiggybackEnabled` system-level configuration parameter (Section 3.1), the *Prep4Deployment* task is either piggybacked on the next user's query or manually triggered as part of the ``ASSESS CORRELATIONS ...'' command (Section 7.1).

[2] If $\mathcal{V}_{ORD}$ is not empty, then each mapper needs to examine these correlations.

in Section 3.1, i.e., an estimation for the number of violating records $|\Gamma(\hat{C}_i)|$, and an estimated set for the distinct violating values $\Phi(\hat{C}_i)$ (or a special flag if the set size exceeds the $MaxVioDistinct$ threshold).

Validity of $|\Gamma(\hat{C}_i)|$ Estimator: The estimation of the total number of violating records $|\Gamma(\hat{C}_i)|$ is computed as follows $|\Gamma(\hat{C}_i)| = v * 100/x$, where v is the number of observed violations in the sample of size $x\%$. Since the values of $C_i.Src$ are in a random order, $|\Gamma(\hat{C}_i)|$ is an unbiased estimator of $|\Gamma(C_i)|$, where E($|\Gamma(\hat{C}_i)|$) = $|\Gamma(C_i)|$.

Validity of $\Phi(\hat{C}_i)$ Estimator: For the distinct violating values (the $\Phi()$ set), it is important to emphasize that our objective is not to estimate the total number of distinct violating values. This task is shown to be hard to estimate using a sample [12]. Fortunately, our problem is simpler due the presence of the upper bound threshold $MaxVioDistinct$, which is relatively very small compared to the number of checked records even in a small sample. Our problem requires listing the distinct violating values only if their number does not exceed $MaxVioDistinct$, otherwise reporting a special *Max-Exceeded* flag indicating that the *Exclusion* strategy is not applicable for the tested correlation. For this constraint reporting of the distinct violating values, we can theoretically prove that $\Phi(\hat{C}_i) = \Phi(C_i)$ with a tiny probability of error. The detailed analysis is presented in [20], and in the following, we give the main intuition.

In a nutshell, if any of the sampled mappers reports the *Max-Exceeded* flag, then with certainty the *Max-Exceeded* flag is the final correct output. Moreover, if all mappers provide their list of violating values, but the centralized process that unions and de-duplicates the values reports the *Max-Exceeded* flag, then again with certainty the *Max-Exceeded* flag is the final correct output. The only chance of error is when both all sampled mappers and the centralized merging process observe less than or equal to $MaxVioDistinct$ violating values while in reality the number is much larger. The theoretical analysis in [20] shows that if the number of randomly tested records is much bigger than $MaxVioDistinct$, e.g., two orders of magnitudes or more, then the error probability is negligible.

Modifications in Selection Phase: The proposed cost-benefit model and the proposed solution presented in Section 3.2 remain unchanged. The only extension is in the *preparation* step after selecting the subset of the soft correlations to be deployed and assigning either an *Exclusion* or a *Materialization* strategy to each one (The last paragraph of Section 3.2.3). Since the statistics are now collected based on a sample, the complete set of $\Phi(C)$ is not yet available. Therefore, the *Prep4Deployment* task needs—in addition to building the exception buckets for the Materialization-assigned correlations—to collect the exact and complete set of distinct violating values for the Exclusion-assigned correlations. Recall that the exact computations of $\Phi()$ and $\Gamma()$ sets are essential to ensure exact query results at runtime.

4.2 Weighted Query Workloads

In Section 3.2.2, we assumed for simplicity that all queries in the workload W have the same weight. In this section, we relax this assumption. In fact, EXORD$^+$ allows different queries to have different integer-value weights ≥ 1, e.g., weights that reflect their execution frequency or business priority. These weights can be integrated into the benefit model by replacing Def. 10 with the following definition:

Definition 14 (Static Correlation Benefit) *Given a workload $W = \{Q_1, Q_2, Q_3, ..., Q_n\}$, where each query Q_i has an associated positive integer weight $Q_i.w_i$. The static benefit of a soft correlation C_i is computed as follows:*

$$SBenefit(C_i, W) = \frac{\sum Q_k.w_k,\ \forall Q_k \in Coverage(C_i)}{\sum Q_j.w_j,\ \forall Q_j \in W}$$

This extension automatically propagates to the dynamic benefit (Def. 11) since it is an iterative function over the static benefit.

Now, the correlation dominance definition (Def. 12) needs to be modified. One criterion of the original definition that measures the extra contribution of correlation C_j over another correlation C_i uses a set difference operation between their coverage sets, which is correct if all queries have the same weight. Yet, under the different weights, the definition is as follows:

Definition 15 (Correlations Dominance) *A correlation C_i (soft or hard) is said to dominate another correlation C_j (soft), denoted as $C_i \mapsto C_j$, iff $Cost(C_i) \leq Cost(C_j)$, $ExclusiveCoverage(C_j) = \phi$, $\frac{|DiffCoverage(C_j,C_i)|}{|Coverage(C_j)|} \leq \epsilon_1$, and $\frac{\sum Q_k.w_k,\ \forall Q_k \in DiffCoverage(C_j)}{\sum Q_j.w_j,\ \forall Q_j \in Coverage(C_j)} \leq \epsilon_2$, where ϵ_1 and ϵ_2 are small threshold values $\in [0, 1)$.*

where *DiffCoverage(C_j, C_i) = Coverage(C_j) - Coverage(C_i)*. Basically, Def. 15 considers C_j to have a little contribution over C_i if both the number of the new queries covered by C_j is small relative to its coverage set (the ϵ_1 threshold), and the total weights of these new queries is small relative to the overall weights in its coverage set (the ϵ_2 threshold).

5 EXORD$^+$ under Dynamic Environment

5.1 Evolving Datasets

In big data applications, datasets are continuously updated by appending new batches of files. A naive inefficient approach to update EXORD$^+$'s state, e.g., the correlations'

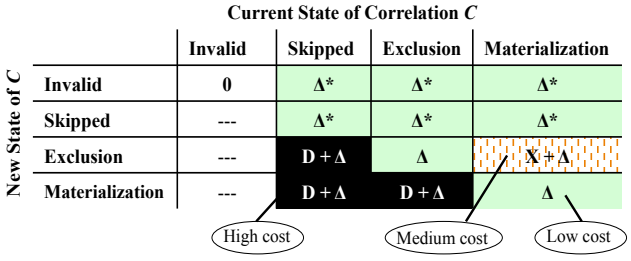


Fig. 7 State Transitions & Scanning Cost under Data Appends.

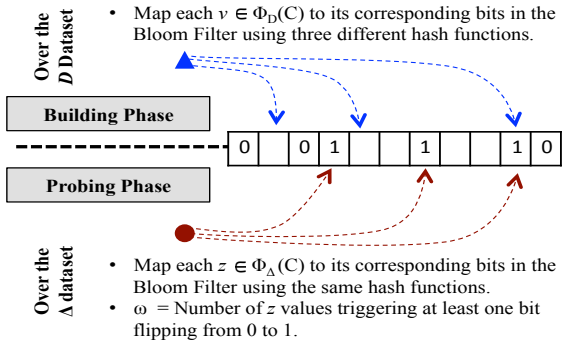


Fig. 8 Bloom Filter for $\Phi(C)$ values under Skipped and Materialization states.

costs, selections, and deployment strategies, is to redo the whole process from scratch over the entire dataset including the newly appended files. In this section, we present an efficient incremental mechanism for updating EXORD$^+$'s state under new dataset appends. In the rest of this section, we use the notations D and Δ to refer to the original dataset on which EXORD$^+$state is currently built, and the newly appended dataset, respectively.

The basic assumption is that hard correlations always hold and they remain satisfied in the appended dataset. On the other hand, a soft correlation C may change its state as summarized in Figure 7. These states are:

- ***Invalid***: If C is *invalid* according to Def. 5, then it remains *invalid* under any append since the *MaxVioDistinct* and *MaxVioRec* thresholds are both absolute values.
- ***Skipped***: A skipped correlation C indicates that C is valid, and it had competed for either of the M_{pool} or the H_{pool} resources (or both), but it had not been selected for deployment. Nevertheless, C's statistics are still maintained to enable incremental update. This includes $|\Gamma_D(C)|$ iff $|\Gamma_D(C)| \leq MaxVioRec$, and $|\Phi_D(C)|$ iff $|\Phi_D(C)| \leq MaxVioDistinct$.
- ***Exclusion***: Correlation C is in this state if it is deployed under the *Exclusion-Based* strategy. In this case, the exact $\Phi_D(C)$ set is already present in M_{pool}. In addition, we also maintain the $|\Gamma_D(C)|$ iff $|\Gamma_D(C)| \leq MaxVioRec$.
- ***Materialization***: Correlation C is in this state if it is deployed under the *Materialization-Based* strategy. In this case, the exact $\Gamma_D(C)$ records are already present in an exception bucket in H_{pool}. In addition, we also maintain the $|\Phi_D(C)|$ iff $|\Phi_D(C)| \leq MaxVioDistinct$.

The steps of the incremental update are as follow:

Step 1: Statistics and Cost Updates: Similar to collecting statistics on the base dataset, the *StatsCollection* task is executed over the new Δ dataset. Depending on the system's configuration parameters, *StatsCollection* executes either on the entire Δ file (and in this case, $|\Gamma_\Delta(C)|$ and $\Phi_\Delta(C)$ are computed for each correlation C), or on a sample of Δ (and in this case, $|\hat{\Gamma_\Delta}(C)|$ and $\hat{\Phi_\Delta}(C)$ are computed). Given that the sample-based estimates are good representatives to the true values (Section 4.1), we will use the true value notations ($|\Gamma_\Delta(C)|$ and $\Phi_\Delta(C)$) throughout this section.

The final cost of $\Gamma(C)$, i.e., the total number of violating records, is computed as follows:

$$|\Gamma_{Final}(C)| = |\Gamma_\Delta(C)| + |\Gamma_D(C)| \quad (5)$$

Nevertheless, the final cost of $\Phi(C)$—assuming that neither of $|\Phi_D(C)|$ nor $|\Phi_\Delta(C)|$ exceeds *MaxVioDistinct*—is not as straightforward. This is because the distinct values in $\Phi_\Delta(C)$ and $\Phi_D(C)$ may overlap, and thus their union could be much smaller than the sum of their sizes. If C is already in the *Exclusion* state, then $|\Phi_{Final}(C)|$ can be precisely computed since the $\Phi_D(C)$ set is maintained. Nevertheless, if C is in the *Skipped* or *Materialization* state, then the individual values of $\Phi_D(C)$ are not maintained and only set size is available. In the following, we present the straightforward approach for estimating $|\Phi_{Final}(C)|$ followed by a more efficient approach.

Loose Count-Based Estimation for $|\Phi_{Final}(C)|$: A straightforward, but loose, estimation relies on maintaining only the count of values in $\Phi_D(C)$. Therefore, the lower and upper bounds of $|\Phi_{Final}(C)|$ follow Eq. 6.

$$Max(|\Phi_D(C)|,\ |\Phi_\Delta(C)|) \leq |\Phi_{Final}(C)| \leq |\Phi_\Delta(C)| + |\Phi_D(C)| \quad (6)$$

And then $|\Phi_{Final}(C)|$ (the total number of distinct violating values across D and Δ) is estimated as the average of these two bounds. However, this is clearly a loose estimation that can be way off the true value.

Tight BF-Based Estimation for $|\Phi_{Final}(C)|$: To get a tighter estimation for $|\Phi_{Final}(C)|$ under the the *Skipped* and *Materialization* states, we maintain an additional bloom filter (BF) structure for C that acts as a replacement signature for the individual values of $\Phi_D(C)$. The main idea of using the bloom filter is illustrated in Figure 8. While col-

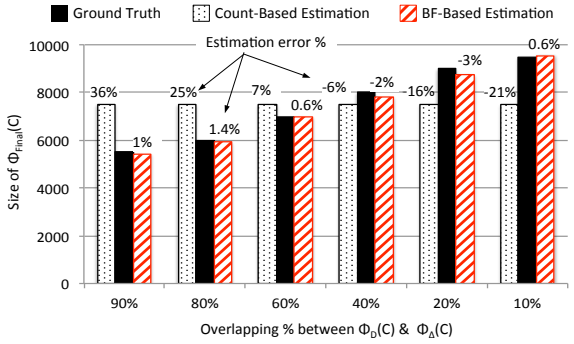


Fig. 9 Comparison between Count-Based and BF-Based Estimations for $|\Phi_{Final}(C)|$. Estimation error % = ((estimation - ground truth)*100/ground truth).

lecting C's statistics over the original dataset D, each value $v \in \Phi_D(C)$ is mapped to bits in the bloom filter using multiple hash functions (three functions is a typical number). These bits are switched from 0 to 1. If C ends up in the *Skipped* or *Materialization* state, then the individual violating values in $\Phi_D(C)$ are discarded and only C's bloom filter is kept (Top half of Figure 8).

When the new Δ dataset arrives, each distinct violating value $z \in \Phi_\Delta(C)$ probes the bloom filter using the same hash functions—without modifying the filter's content. If any of the probed bits is 0, then z is guaranteed to be a new violating value, and the count of these values is maintained, say ω (Bottom half of Figure 8).

The big advantage of bloom filters is that they have a theoretically proven error rate [2]. This error rate represents the number of new violating values that the bloom filter may not detect (false positives), i.e., they do not hit bits of value 0 although they are new values. Assume the bloom filter is constructed to have an error rate e, e.g., 10%, then $|\Phi_{Final}(C)|$ has the lower and upper bounds indicated in Eq. 7.

$$|\Phi_D(C)| + \omega \;\; \leq |\Phi_{Final}(C)| \leq \;\; |\Phi_D(C)| + (\tfrac{1}{1-e}) * \omega \qquad (7)$$

The lower bound reflects what has been actually observed, while the upper bound adds up the expected bloom filter's error rate. $|\Phi_{Final}(C)|$ is then estimated as the average of these two bounds.

The following example demonstrates the effectiveness of the BF-Based estimation.

Example 7: *Assume the "MaxVioDistinct" threshold is set to 10,000. Also assume that we use three distinct hashing functions for the bloom filter, and our target error rate is 5%. According to [2], the size of the bloom filter should be $\approx 8KB$. We performed an experiment where each of the $\Phi_D(C)$ and $\Phi_\Delta(C)$ sets is randomly populated with 5,000 distinct values. We varied the overlap between the two sets between 10% to 90% as indicated in Figure 9 (x-axis). Given that the ground truth of $|\Phi_{Final}(C)|$ is known (the black bars), we calculated the estimates generated from the Count-Based and BF-Based approaches (y-axis). Figure 9 also shows the estimation error % from each approach. The Count-Based approach always generates the same estimation because the lower and upper bounds are fixed (Eq. 6), and hence its estimation error can be very high (36% in our experiment). In contrast, BF-Based estimation is dynamic since the lower and upper bounds depend on the observed new values (Eq. 7), and hence it provides a very good estimation.*

In summary, Step 1 updates the cost of each soft correlation according to Eq. 5 ($|\Gamma_{Final}(C)|$) and Eq. 7 (for $|\Phi_{Final}(C)|$), and then Defs. 5 and 5 are applied to decide on which of the correlations remain valid, and which ones make the transition to the *Invalid* state (The 1^{st} row in Figure 7).

Step 2: Re-Evaluation of Extended Correlation Dominance: Under the new costs, the optimization problem presented in Section 3.2 need to be re-solved. However, the *Correlation Dominance* relationship introduced in Def. 12 requires a modification because correlations now have states, and transitioning from one state to another involves different costs, which affects the dominance relationships.

To illustrate this point, we first refer to the scanning costs indicated in Figure 7 for each possible transition. All transitions require scanning the Δ dataset, which is performed in Step 1 above. In addition, some transitions involve extensive high cost (marked in Black), e.g., a transition from *Skipped* to *Exclusion*. These high-cost transitions require going back and scanning the original dataset D to either collect the exact $\Phi_D(C)$ or extract the $\Gamma(C)$ records. Some other transitions involve medium cost (marked in Dotted Red), e.g., a transition from *Materialization* to *Exclusion*, which requires scanning the exception bucket in H_{pool} to retrieve the exact set of $\Phi_D(C)$.

Having these different transition costs in mind, it is wrong to assume according to Def. 12 and Def. 15, that for example, Correlation C_1 dominates C_2 without taking their current states into account. Therefore, under the incremental appends, the dominance definition is extended as follows (Def. 16 and Table 3) [3]:

Definition 16 (Correlations Dominance Under Incremental Append) *A soft correlation C_i is said to dominate another soft correlation C_j, denoted as $C_i \mapsto C_j$, iff $Cost(C_i) \leq Cost(C_j)$ according to Table 3, $ExclusiveCoverage(C_j) = \phi$, and $\frac{|Coverage(C_j) - Coverage(C_i)|}{|Coverage(C_j)|} \leq \epsilon$, where ϵ is a small threshold value $\in [0, 1)$.*

[3] Def. 16 assumes un-weighted query workload and extends Def. 12. It is straightforward to extend Def. 15 for weighted query workload in the same manner.

Dominance Type	Cost(C_i) ≤ Cost(C_j) IFF
$C_i \mapsto_{\Phi\Gamma} C_j$	(($\Phi(C_i).size \leq \Phi(C_j).size$) And ($\Gamma(C_i).size \leq \Gamma(C_j).size$)) And (($C_i$.state = C_j.state) Or (C_j.state = Skipped))
$C_i \mapsto_{\Phi} C_j$	($\Phi(C_i).size \leq \Phi(C_j).size$) And (($C_i$.state = C_j.state) Or (C_i.state = Exclusion))
$C_i \mapsto_{\Gamma} C_j$	($\Gamma(C_i).size \leq \Gamma(C_j).size$) And (($C_i$.state = C_j.state) Or (C_j.state != Materialization))

Table 3 Cost()-Based Dominance Notation under Data Appends.

Basically, the cost comparison presented in Table 3 has been extended to include the correlations' current state. In the three dominance types presented in the table, one possible condition that makes the cost comparison valid is that both correlations have the same state, i.e., (C_i.state = C_j.state). Furthermore, in the case of global dominance ($C_i \mapsto_{\Phi\Gamma} C_j$), it is possible to have the correlation to-be-eliminated (C_j) in a *Skipped* state regardless of C_i's state. This is because according to the transition costs indicated in Figure 7, the *Exclusion* and *Materialization* states always have lower (or at most equal) transition cost compared to the *Skipped* state. Therefore, it is safe to eliminate C_j if all other conditions included in Def. 16 are satisfied.

Following the same logic, the conditions of the other two dominance types have been extended. The M_{pool} dominance ($C_i \mapsto_{\Phi} C_j$) considers only the transition costs to an *Exclusion* state—which is the 3^{rd} row in Figure 7. Whereases, The H_{pool} dominance ($C_i \mapsto_{\Gamma} C_j$) considers only the transition costs to a *Materialization* state—which is the 4^{th} row in Figure 7.

Step 3: Re-Solving the Optimization Problem: The heuristic-based algorithm presented in Figure 6 still executes the same three phases. The extended definition of the correlations dominance takes effect in Lines 2, 5, and 17 of the algorithm. Finally, we incorporate the correlations' states while solving the knapsack problem (Lines 10 and 19). Notice that from Figure 7 all transitions must perform a scan over the Δ dataset, which is already performed in Step 1. Any additional scanning overhead translates to reducing the benefit of the correlation by a decaying factor (the more data to scan, the bigger the decaying factor). We do not add this scanning overhead to a correlation's cost because the cost and the total budget of the knapsack problem are in terms of storage units, which is not compatible with the processing overhead.

More specifically in Line 10, while competing for the M_{pool} resource based of the $|\Phi_{Final}(C)|$ cost estimated by Eq. 8—which reflects the storage cost— any additional scanning cost is taken into account. Recall that competing for M_{pool} means aiming for a transition to a new *Exclusion* state, which is the 3^{rd} row in Figure 7. The same logic applies in Line 19 when competing for the H_{pool} resource

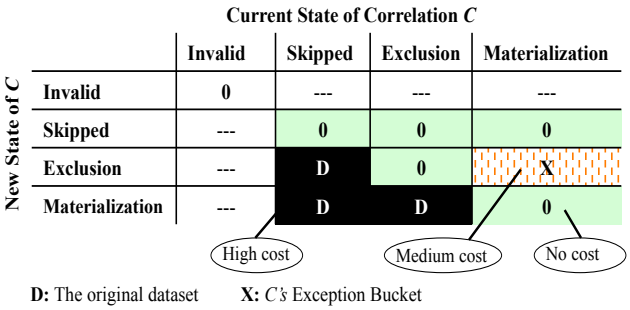


Fig. 10 State Transitions & Scanning Cost under Workload Changes.

based of the $|\Gamma_{Final}(C)|$ cost computed by Eq. 5 along with the corresponding scanning costs presented in the 4^{th} row in Figure 7.

Step 4: Execution of the Selected Strategies: The final step is to execute the *Prep4Deployment* task to handle the violations of the selected correlations. Figure 7 summarizes the datasets that are scanned to perform a transition depending on the current and the new state. For example, if a correlation switches from *Materialization* to *Exclusion*, then Δ is scanned (which is Step 1 above), and the correlation's exception bucket, denoted as X, is also scanned to collect the exact distinct violating values from D and combine them with those obtained from the Δ's scan.

Estimation Errors Reseting: It is important to highlight that despite any approximations performed during the statistics collection phase (the *StatsCollection* task), the final scans performed in this step (by the *Prep4Deployment* task) guarantee that the $\Gamma(C)$ or $\Phi(C)$ sets are collected in an exact and precise way. This ensures exact query answer, and it also resets any sampling-based or other approximation types. Hence, no error accumulation takes place as more future datasets are appends.

5.2 Evolving Query Workload

Incrementally updating EXORD$^+$'s state under evolving query workloads builds on top of the extensions presented in Section 5.1. Assume that the current workload is denoted as W, and the newly observed workload is denoted as W'.

The first step is to re-compute the benefit of each valid soft correlation. Since only the benefits are changing while costs remain the same, the possible state transitions are slightly different from the case of data appends. For example, any valid correlation is guaranteed to remain valid under the new workload W'. The possible state transitions along with their involved scanning cost are summarized in Figure 10.

After re-computing the benefits (according to Def. 10 or Def. 14), Steps 2, 3, and 4 presented in Section 5.1 are executed to decide on the correlations' dominance, re-solve the optimization problem and select the best correlations, and finally deploy them. Notice that the extended dominance definition in Def. 16 along with the cost comparisons in Table 3 still apply under the case of evolving query workload. This is because the relative cost comparison (equality, greater than, or less than) among the possible transitions in Figures 7 and 10 are the same.

6 Deployment & Exploitation Phase

The ultimate goal of EXORD$^+$ is to exploit the available correlations—more specifically the ones in the deployment phase—to re-write queries and enable more efficient access plans. In Section 6.1, we present the exploitation procedure for equality predicates, and in Section 6.2, we extend the support to range predicates.

6.1 Optimizing Equality-Predicate Queries

In Figure 11, we present the flowchart of the exploitation procedure. The procedure takes as input a set of correlations in their deployment phase $\mathcal{Y} = \{C_1, C_2, ..., C_n\}$, a dataset D to be queried, and a target query Q as defined in Def. 1 consisting of a set of conjunctive equality predicates $p_1 \wedge p_2 \wedge ...p_k$. 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^{nd}, 3^{rd}, and 4^{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.

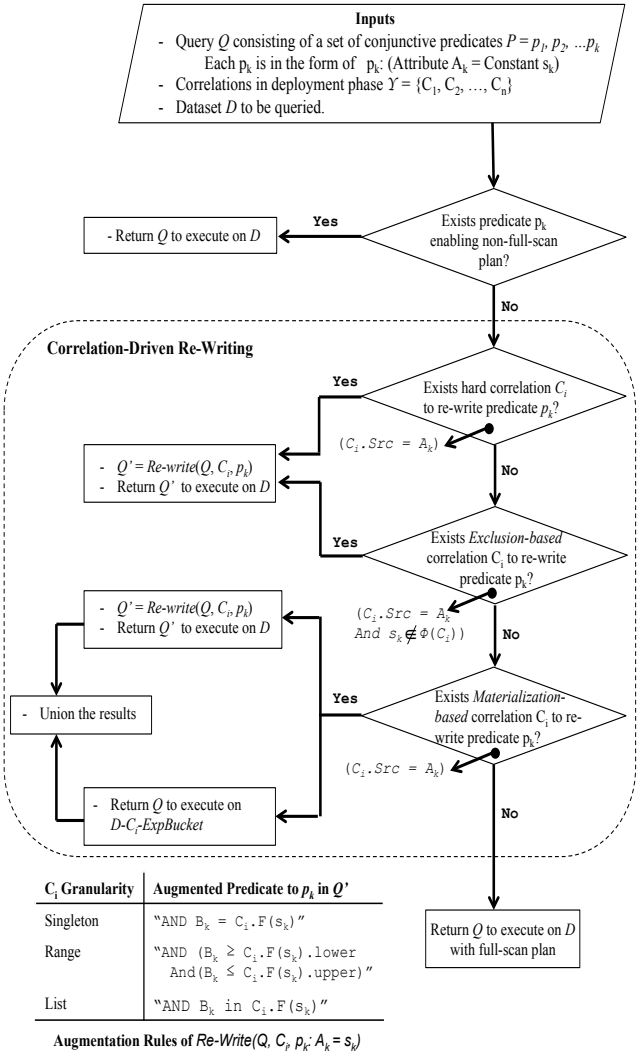


Fig. 11 Exploitation in Query Optimization Flowchart.

Assume the selected correlation is $C_i \in \mathcal{Y}$, and it is used to re-write a specific predicate in Q, say p_k, which is in the form of p_k: $A_k = s_k$, where A_k is one of the data's attributes, and s_k is a constant value. Therefore, $C_i.Src =$ "A_k", and we assume $C_i.Dest =$ "B_k", 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 11 (The 2^{nd}, 3^{rd}, and 4^{th} conditions) execute the same $Re\text{-}Write(Q,\ C_i,\ p_k\text{: } A_k = s_k)$ procedure. $Re\text{-}Write$ 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 11 (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^{nd}, and 3^{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-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.

Recall that Q' contains the newly added predicates on $C_i.Dest$, which has an associated efficient access method. And thus, Q' execution on the big dataset D is expected to be efficient as it avoids the expensive full-scan plans. On the other hand, the execution of Q on the exception bucket is also expected to be efficient—although it uses a full-scan plan—because exception buckets are relatively very small compared to the original dataset.

6.2 Optimizing Range-Predicate Queries

EXORD$^+$ enables optimizing range-predicate queries under certain restrictions. These restrictions come from the fact that a correlation's mapping function is blackbox to the system. And thus, without the mandatory properties specified in the following definition (Def. 17), EXORD$^+$ cannot guarantee correct mapping of the range predicates at query time.

Definition 17 (Target Range-Predicate Query) *Assume a range-predicate query Q involving predicate $p = (l\ op_1\ Src\ op_2\ u)$, where Src is a data attribute of* `numerical` *or* `date` *types, l and u are the lower and upper constant values, respectively, $op_1, op_2 \in \{<, \leq\}$, and Src has no associated access method to evaluate its predicate other than a full scan. Q is applicable for optimization by a correlation C having the same source attribute Src iff $C.Dest$ is also of* `numerical` *or* `date` *types, $C.granularity \neq List$, and $\forall(l, u)$ the following mapping of p holds:*

$$\begin{cases} (C.F(l)\ op_1\ C.Dest\ op_2\ C.F(u)), & \text{if "Singleton"} \\ (C.F(l).lower\ op_1\ C.Dest\ op_2\ C.F(u).upper), & \text{if "Range"} \end{cases}$$

In other words, in range-predicate queries, the source and destination attributes are restricted to numerical or date types because categorical attributes do not have a notion of continuous range. The data type condition alone is not sufficient since numerical types can be used, for example, to encode categorical attributes. Therefore, the inclusion of the additional mapping condition is mandatory for correct query re-writing at runtime. These mapping rules are to be applied within the `Re-Write()` function in the exploitation algorithm in Figure 11.

Given that the mapping function is a blackbox, the domain expert must inform the system whether or not the conditions in Def. 17 are satisfied. This is realized in EXORD$^+$ by extending Def. 2 to include the *"RangeOpt"* Boolean flag: $\langle Src,\ Dest,\ Type,\ Granularity,\ F(),\ RangeOpt\rangle$. The flag is set when defining the correlation. If set to True, then the range-predicate optimization is enabled, otherwise it is disabled.

Example 8: *Following on Example 1 in Section 2.1, the correlation in Motivation Scenario 1 is now formulated as: $\langle deliveryDate, shippingDate, \text{"}Soft\text{"}, \text{"}Range\text{"}, F(), True\rangle$, where for a given delivery date d, $F(d).lower$ = (d - 10 $days$), and $F(d).upper$ = (d - 3 $days$). Now, since $RangeOpt$ = True, given a query Q involving a range predicate:*

`March-14-2016` $<$ `deliveryDate` $\leq$ `April-20-2016`

Q can be re-written based on the correlation to augment the following predicate:

`March-4-2016` $<$ `shippingDate` $\leq$ `April-17-2016`

And then the rest of the algorithm presented in Figure 11 applies.

7 Experiments

7.1 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 ON <tableName> ...''
```

The command enables the database admins to define the soft and hard correlations along with their parameters, e.g., the basic parameters introduced in Def. 2, and the *RangeOpt* flag introduced in Section 6.2. The metadata repository stores the 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 stored in the metadata repository.

We introduced another command to allow the system admin to manually trigger the validation and selection phases on the correlations (if desired). The command takes the query workload defined in a specific JSON format, and offers a set of flags for flexible configurations as presented

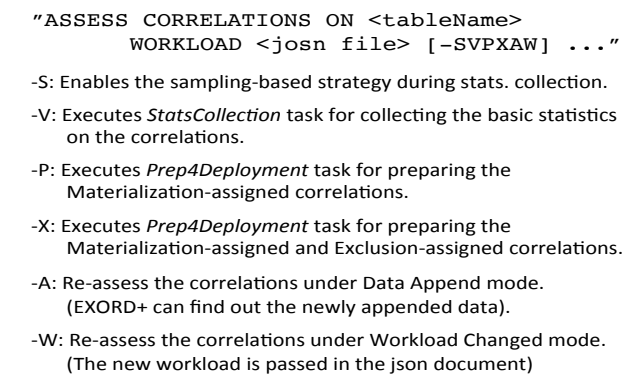


Fig. 12 *ASSESS CORRELATIONS* Command.

in Figure 12. For example, flags "−SV" indicate executing only the *StatsCollection* task (without the preparation step that materializes the execution buckets and the distinct violating values), and the sampling strategy is enabled. Flags "−VP" indicate running the cascaded tasks *StatsCollection* followed by *Prep4Deployment* using a full scan over the data. In contrast, flags "−SVPA" indicates that the system needs to re-assess the correlations on the specified table because a new dataset (Δ) is appended. The *StatsCollection* and *Prep4Deployment* tasks execute on Δ in order, and *StatsCollection* uses the sampling-based strategy.

In addition to these commands, a system-level Boolean configuration parameter `PiggybackEnabled` is defined[4]. If set to False, then the correlations' validation and preparation tasks are triggered only through the ''ASSESS CORRELATIONS ..'' command. Otherwise, they are triggered either by executing the command or by receiving a user's job (whatever comes first).

For detecting workload changes and triggering the incremental maintenance (Section 5.2), EXORD$^+$ has a configurable behavior through a set of configuration parameters and pluggable modules to accommodate the different needs of different applications. The initial workload (W) is either provided by the system's admin, or observed by the system over a given interval. The interval is either count-based, i.e., observing `NumQueries` queries, or time-based, i.e., observing the queries submitted over the last `TimeWindow`, where `NumQueries` and `TimeWindow` are configuration parameters. As presented in Section 5.2, the system keeps track of each distinct query Q_i along with its weight $Q_i.w_i$ representing the frequency of its occurrence. Then, the system optimizes its correlation selection based on W.

As more queries are submitted, EXORD$^+$ observes and collects a new workload (W') based on the `NumQueries` or `TimeWindow` configuration parameters. Then, the distance between the two workloads is calculated `Dist(W, W')`, where `Dist`$() \in [0, 1]$ can be a standard distance function, e.g., Jaccard index, or a pluggable function provided by the system's admin (in the form of a jar file). If the distance exceeds a threshold, then the adaptive workload incremental maintenance is triggered[5].

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 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 a real-world 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—the time difference includes the seat confirmation with the airline and the payment confirmation with the bank, 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 13, we summarize few correlations that are of our focus. Correlation C_1 is explained in detail in Motivation Scenario 2 in Section 1. Correlation C_2 indicates that in the majority of the cases, the confirmation timestamp is within 1 to 5 mins after the request timestamp, however there can

[4] The system maintains few other configuration parameters that are omitted from the discussion.

[5] The system dumps continuous reports on the queries that are optimized by correlation-based re-writing, the queries that are not optimized, the number of times a correlation is used to optimized queries. These reports enable system admins to tune the system as desired, or even manually trigger re-assessment using the ''ASSESS CORRELATIONS ..'' command.

ID	Source Column	Dest Column	Short Desc.	Strategy
C_1	StartAirportIATA	StartCountry	In most cases, the airport code determines the country (Refer to Motivation Scenario 2)	Exclusion
C_2	ConfTimestamp	RequestTimestamp	In most cases, the *ConfTimestamp* is within 1 to 5 mins from the *Requesttimestamp*	Materialization
C_3	TicketPrice	TicketClass	The price givens an indication on the ticket class code.	Materialization

Fig. 13 Correlations in the Working Dataset.

DataSet Size	Partitioning (map-reduce Job)		Indexing (map-reduce Job)	
	Storage	Time (sec)	Storage	Time (sec)
500 GBs	1.5 TBs	962	192GBs	2605
1 TB	3 TBs	2173	413GBs	6643
2.3 TBs	~ 7 TBs	4669	1.12TBs	16369

Fig. 14 The storage (including 3-way replica) and time overheads of creating an additional data organization (Partitioning or Indexing) for *Current-Auxiliary (Aux)* technique.

certainly be exceptions to this correlation. Moreover, there is a correlation between the ticket price and the ticket class (Correlation C_3), and we are going to synthetically control such correlation in different ways as explained in the experiments.

7.2 Performance Evaluation

7.2.1 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 13, Correlation C_1 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 14, 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 C_1: To study the benefits of using Correlation C_1, we first partition the data on the `StartCountry` attribute. Nevertheless, our equality-based selection query (C_1-*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 C_1-*selection* query is presented in Figure 15(a). The current technique [16] (labeled as *"Current"*) have to perform a full scan over the data, EXORD$^+$ makes use of Correlation C_1 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 14.

In Figure 15(b), we study a more complex variation of Query C_1-*selection*, i.e., C_1-*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 C_2: In Figure 16, we demonstrate the usage of Correlation C_2 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 [7]. Our experimental queries C_2-*selection* and C_2-*aggr* involve a selection predicate on the `ConfTimestamp` attribute instead. Correlation C_2 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 timestamps, and for each timestamp we execute three experiments with matching granularities of a Second, Minute, and Hour to have different selectivities. The selectivity varied from few 100s of records (in the case of a Second granularity) to several 100s of thousands of records (in the case of an Hour granularity).

In Figure 16(a), we present the results of the selection query C_2-*selection*. The state-of-art technique [7] (*"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

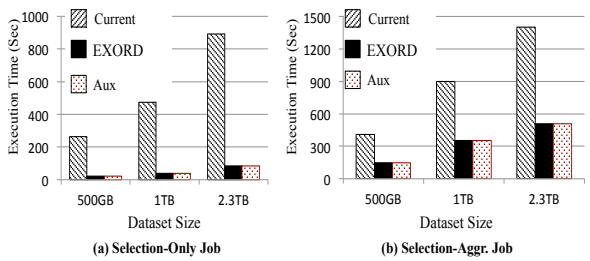


Fig. 15 Study of Correlation C_1 (Use of Partitioning).

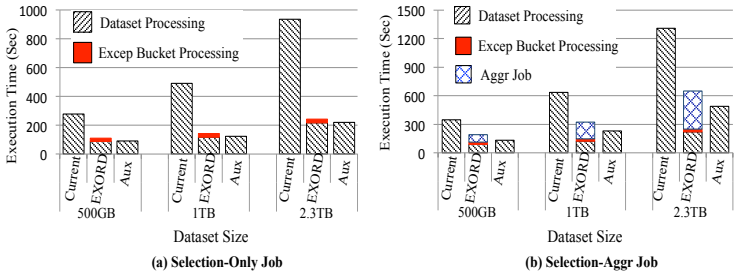


Fig. 16 Study of Correlation C_2 (Use of Indexing).

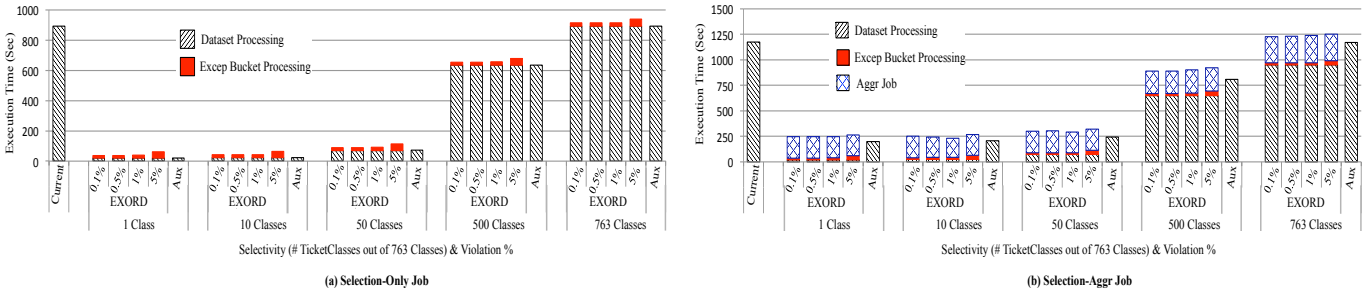


Fig. 17 Study of Correlation C_3 (Use of Partitioning over the 2.3TBs Dataset).

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 16(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 after applying the selection predicate. 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*, which is mostly due to the scanning of the exception bucket and the overheads of starting three jobs instead of one. 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 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 17(a) and 17(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 touched partitions. In the worst case, where all partitions are 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 17(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

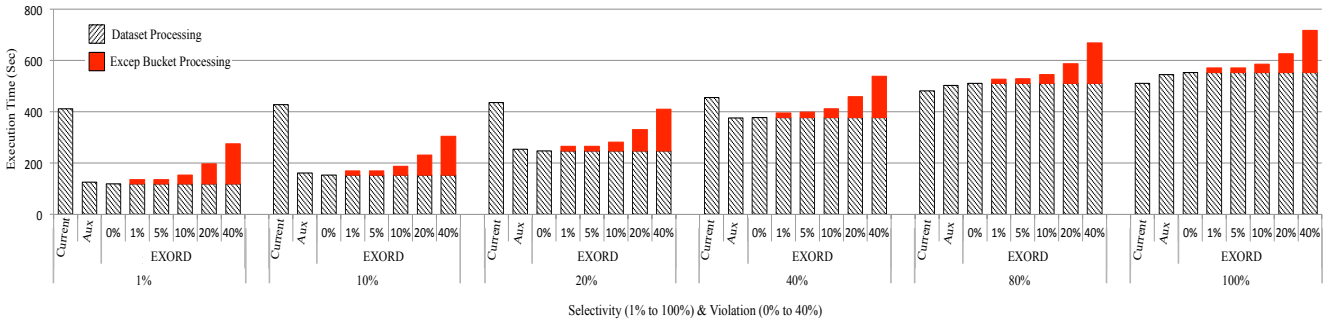


Fig. 18 Synthetic datasets under various selectivity and violation percentages.

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 18, and within each selectivity degree, the correlation's violation percentage varies from 0% (Hard correlation) to 40%.

In Figure 18, we compare *Current* (which performs a full scan), *Aux* (which builds and leverages an additional index over `Field2`), and EXORD$^+$. The first key insight from the figure is that allowing large exception buckets may hurt the performance and may diminish the savings from exploit the correlation. Typically is it recommended to only accept correlations, i.e., consider them as valid according to Def. 5, 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., *MaxVioRec* in Def. 5, 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 18, the 1% and 5% violations fit in one wave, while 10% fits in two waves (and it doubles for 20% and 40%).

The second key insight from Figure 18 is that 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.

7.2.2 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.

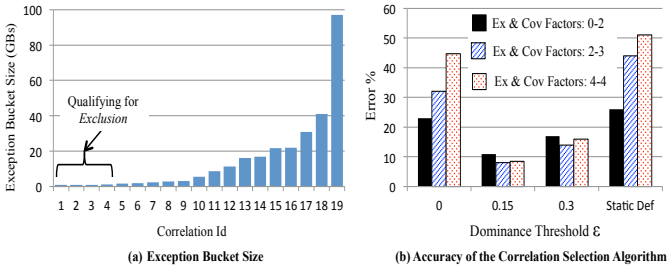


Fig. 19 Heuristic-Based Correlation Selection.

For example, in addition to the three correlations presented in Figure 13, other similar correlation include: `StartAirportIATA` → `StartCity`, `DestAirportIATA` → `DestCountry`, `DestAirportIATA` → `DestCity`, and `RequestTimestamp` → `ConfTimestamp`. Other interesting correlations include `ToFlightDays` → `Purpose`, and `ToFlightDays` → `Duration`. Capturing these correlations is not only important for query optimization, but also important for the business logic and providing better pricing models. In addition to these correlations, we replicated the `TicketPrice` attribute in correlation C_3 three more times, and replicated C_3 along with these new attributes. The purpose of this replication is to have the four violation percentages studied in Figure 17(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 19(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 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 15 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 15 correlations are set according to the results in Figure 19(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 is replicated between 2 to $CovFactor$ times over the $Coverage()$ set of some different correlations (random assignment). The $CovFactor$ varies over the values of $\{2, 3, 4\}$. 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 15 correlations of all sizes from 1 to 15 (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 its negligible tiny overhead. In summary, the execution time is in the order of milliseconds ranging from 34 to 52 milliseconds. In these experiments, the dominance threshold ϵ (in Def. 12) 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 ϵ as depicted in Figure 19(b). In addition, we suppress the heuristic and thus the algorithm now maps to the static definition of the correlations' benefits (Def.10). The y-axis of Figure 19(b) shows the error percentage computed as $100 * (opt - approx)/opt$, where opt and $approx$, are the optimal and the approximated values, respectively. As the results in Figure 19(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.

7.2.3 Validation and Preparation Overheads

We now focus on evaluating the overheads involved in managing the soft correlations, i.e., validation (the *StatsCollection* task) and preparation (the *Pre4Deployment* task).

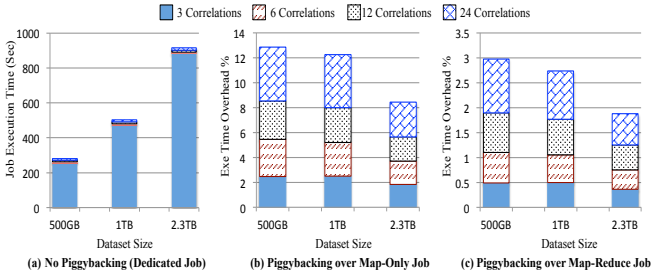


Fig. 20 Validation Overhead of Soft Correlations.

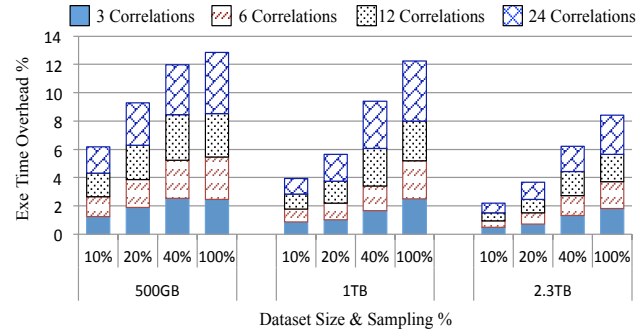
Full-Scan Validation Overheads: In Figure 20(a), we show the validation overhead without piggybacking. The figure shows the execution time for each of the three dataset sizes while validating the the three correlations C_1, C_2, and C_3. The small stacked bars illustrate the additional time overhead 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 full-scan *StatsCollection* task can be expensive, especially in the case of large datasets. Nevertheless, the advantage is that the system becomes ready to immediately optimize users' queries.

In Figures 20(b) and 20(c), we illustrate the piggybacking performance over a map-only, or a 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 since 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 20(b)).

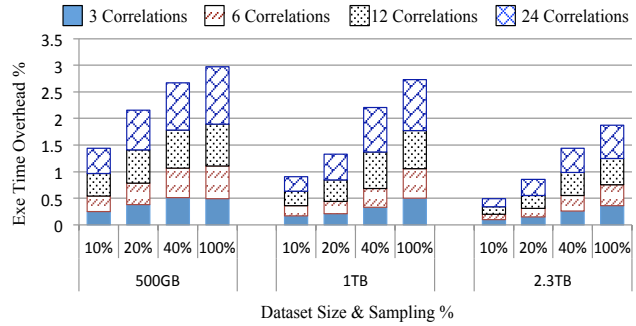
Sample-Based Validation Overheads: As discussed in Section 4.1, the validation phase can be executed over a sample of the data instead of the entire dataset. In Figure 21, we extend our previous experiment (reported in Figure 20) by studying the effect of sampling on the validation performance. We use the real-world dataset along with its three distinct subsets of sizes $500GBs$, $1TB$, and $2.3TBs$. For each dataset, we consider executing the validation job over either the entire dataset (100% size) or a sample of size 40%, 20%, or 10% (the x-axis). Figure 21(a) illustrates the job overhead without piggybacking (a dedicated job for validation), whereas Figures 21(b) and 21(c) illustrate the additional overhead percentage in the case of piggybacking over another map-only or map-reduce job, respectively. As expected, compared to the full dataset validation, the sampling-based validation can significantly reduce the overheads by two to four folds.

Accuracy of Sample-Based Validation: To support the theoretical analysis presented in Section 4.1 for the estimation errors, we experimentally measure the accuracy of the sample-based validation strategy. The exact differences between the estimated and actual values of $\Gamma()$ and $\Phi()$ are

(a) No Piggybacking (Dedicated Job)



(b) Piggybacking over Map-Only Job



(c) Piggybacking over Map-Reduce Job

Fig. 21 Sample-Based Validation Overheads of Soft Correlations.

Phase	Set	Sampling %		
		0.1%	0.2%, 1%, 2%	5%, 10%, 40%, 80%
Before Deployment	Valid Set	15/17	15/16	1
	Invalid Set	2/4	3/4	1
After Deployment	Exclusion Set	1	1	1
	Materialization Set	11/12	1	1

Fig. 22 Sample-Based Validation: Accuracy using Jaccard Coefficient J Compared to a Full-Scan Validation ($0 \leq J \leq 1$). $J = 1$ indicates exact matching.

not critically important, but what is more important is their effect on the status of the soft correlations, i.e., being *valid* vs. *invalid*, and the final deployment status. Therefore, in our experiments, we measure the accuracy by comparing the correlations at two phases (Refer to Figure 22): First comparing the *Valid* and *Invalid* sets from the full scan with their counterpart sets from the sample-based strategy. Second, comparing the deployment sets, namely *Exclusion* and *Materialization*, from the full scan with their counterpart sets from the sample-based strategy.

In this experiment, we only consider the largest real-world dataset of size 2.3TBs, and vary the sampling percentage between 0.1% to 80% as indicated in Figure 22. We use the set of the 19 correlations described in Section 7.2.2 as the testing set. We set the *MaxVioRec* and *MaxVioDistinct* parameters such that in the baseline case, which is a full-scan validation, three of these correlations are invalid while the rest are valid ones. Moreover, four of the valid correlations belong to the *Exclusion* strategy, eleven belong to the *Materialization* strategy, and one is skipped.

Under each sampling percentage, we execute one experiment to collect the violation statistics, categorize the correlations into *valid* and *invalid* sets, and then solve the optimization problem to label the valid ones as either *Exclusion*, *Materialization*, or *Skipped*. We use Jaccard coefficient as the similarity measure between the output sets from the sampling strategy and the corresponding sets produced from the full scan.

As the results in Figure 22 show, the sampling-based validation is highly accurate. With a very small sample of 5% (or larger) an exact matching is found compared to the full-scan output. Even with a smaller sample size between 0.2% and 2%, a tiny mismatch is found in the valid and invalid sets compared to the full-scan sets (the *Before Deployment* phase). However, this mismatch did not affect the final decision regarding the ones chosen for actual deployment. Only when the sample rate is set to 0.1%, we noticed a minor mismatch where the sample-based method added an additional false-positive correlation to the H_{pool}.

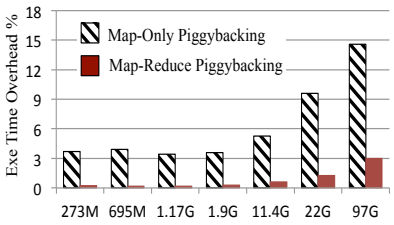
Preparation Overheads: For the *Exclusion* strategy, If the validation phase is performed over the entire dataset without sampling, then the violating values are already collected. Thus, the preparation overhead involves the insertion of these values into EXORD$^+$'s metadata repository, which is a MySQL 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 23(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.

In the case the validation phase is performed over a sample, then the preparation phase (the *Prep4Deployment* task) needs to collect the complete set of violating values. The performance of this job is similar to that reported in Figure 20.

For the *Materialization* strategy, the preparation overhead involves collecting the violating records and copy them to an exception bucket. In Figure 23(b), we report this overhead under the two cases of piggybacking the preparation

# Values	Insertion Time
1	12.6 msec
10	22.9 msec
100	43 msec
1000	0.97 sec
10,000	8.32 sec

(a) Exclusion Strategy: Insertion into the Metadata Repository



(b) Materialization Strategy: Overhead to create Exception Buckets

Fig. 23 Preparation Overhead of Soft Correlations.

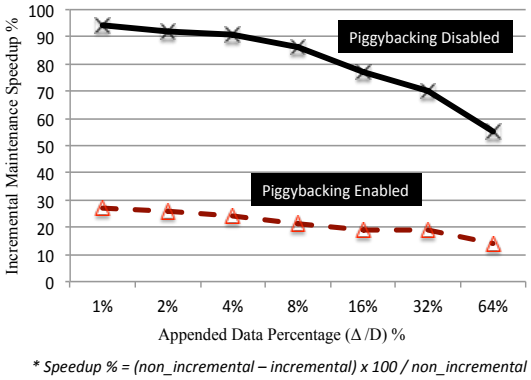


Fig. 24 Incremental Maintenance Speedup under Data Appends. *"Piggybacking Enabled"* approach has the disadvantage that the first user's job after each append operation is bypassed from query optimization and executed with a full scan.

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 7.2.1, 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 significantly smaller if the user's job is map-reduce job.

7.2.4 Evaluation of Incremental Maintenance

Data Appends: In Figure 24, we study the combined validation and preparation performance (the sum of there overheads) under data appends. We consider the real-world dataset size of 2.3TBs as the base dataset (D). The size of the appended dataset (Δ) varies from 1% to 64% of D (the x-axis of Figure 24). We assume the 19 candidate correlations discussed in Section 7.2.2. The state of these correlations under D is {3 *Invalid*, 1 *Skipped*, 4 *Exclusion*, 11 *Materialization*}. The statistics are collected based on 10% of the data, i.e., 10% of Δ for the incremental maintenance and 10% of $(D + \Delta)$ for the non-incremental maintenance.

When piggybacking over user's jobs is disabled, the maintenance of the system state is triggered either manually through the `ASSESS CORRELATIONS ...` command or automatically when a new dataset Δ is appended. The results in Figure 24 show the big savings that can be achieved

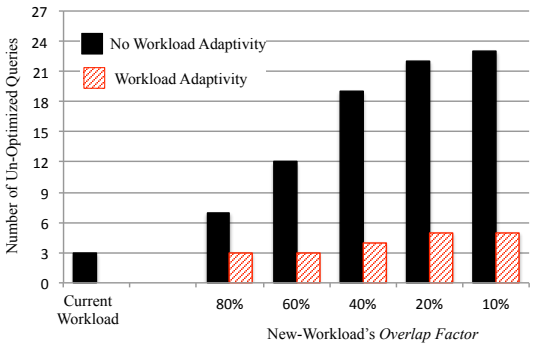


Fig. 25 Number of Un-Optimized Queries under Workload Changes. "Current Workload" consists of 50 queries. The "*Overlap Factor*" parameter controls the transition from the current workload to the new workload, which also consists of 50 queries.

under the incremental maintenance approach. The big advantage of disabling the piggybacking under data appends is that the system becomes ready to immediate optimize the next user's query without interruption.

When piggybacking over user's jobs is enabled, the maintenance of the system state is triggered by the first user's job after the data append operation. The main disadvantage here is that this job is bypassed from optimizations in order to collect the needed statistics. The non-incremental approach starts from scratch and sample from the entire dataset $(D + \Delta)$, whereas the incremental approach only samples from the appended dataset (Δ). The difference in the overhead in this case is shown to be around 15% to 30% because the user's job dominates the processing anyway.

Workload Changes: To study the adaptivity feature under workload changes, we use the largest dataset size (2.3TBs) and the 19 correlations $\{C_1, C_2, ..., C_{19}\}$ discussed in Section 7.2.2. Similar to our setup in Section 7.2.2, we generate a query workload W (referred to as the *current workload*) of 50 queries controlled by the two configuration parameters $ExFactor$ and $CovFactor$. The only difference here is that the queries in W can only reference 10 out of the 19 correlations, say $\{C_1, C_2, ..., C_{10}\}$, and the remaining 9 correlations are of no benefit to W. This implies that EXORD$^+$ selects and deploys correlations only from the referenced ones—depending on the available resources—because the others have zero benefit.

We then generate a new workload W' consisting also of 50 queries. We use a new parameter *OverlapFactor* to control the percentage of queries that are common between W and W'. As illustrated in Figure 25, *OverlapFactor* varies from 80% to 10%. The rest of the queries in W' (the new queries) are generated according to the same two configuration parameters $ExFactor$ and $CovFactor$ except that they can reference any of the 19 correlations. The smaller the value of the *OverlapFactor* parameter, the more new queries are introduced and the more new correlations become relevant to W' compared to the old workload W.

In Figure 25, we study how the system responds to the workload changes. The y-axis shows the number of queries that are not covered by any deployed correlation (un-optimized queries), mainly due to the restricted resources in the optimization problem. Under the current workload W, three queries are not covered. When the transition occurs to the new workload W', the non-adaptive approach fails to adjust the selected correlations, and thus more queries become un-covered by the previously chosen correlations. In contrast, EXORD$^+$ incrementally re-evaluates the benefit measure of the correlations and adjust the selected and deployed correlations accordingly.

8 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 [24]. These optimizations include techniques such as indexing [5,7,9], pre-partitioning [16], re-organization and colocation [10], materialization and reusability of intermediate results [4,8,22], 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 in time and storage [5, 8,9,10] (Refer also to our reported results in Figure 14). The proposed EXORD$^+$ system is complementary to and can work in conjunction with most of the existing techniques, e.g., indexing either local indexes [5,7] or global indexes [9], pre-partitioning [10,16], and materialization [8]. 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 [3,15,17,18]. That is why discovering and exploiting correlations have been extensively studied in RDBMSs including functional dependencies [13,21,26], conditional functional dependencies [1,11], soft correlations [3,15,17,18], and denial constraints [6]. The closest techniques to EXORD$^+$ (in sprit) include BHUNT [3], CORDS [15], Correlation Maps (CM) [17], and CORADD [18], 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 [15,17,18], or very restricted in their correlation definition compared to EXORD$^+$ [3].

In more details, CORDS [15] 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 unnecessary 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 [17] and CORADD [18] 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 [17], and providing better design for the database in the form of materialized views and recommendations for their clustered attributes [18]. 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 [3] 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.

9 Conclusion

We presented the *EXORD*$^+$ system for exploiting the data's correlations in the context of big data query optimization. EXORD$^+$ supports both *hard* and *soft* correlations. 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 for a given query workload to the well-known submodular knapsack optimization problem. We then, proposed a heuristic-based algorithm to efficiently solve the problem in a polynomial time. We introduced incremental maintenance strategies for efficiently updating the system's state under data appends and workload changes. EXORD$^+$ is can be applied on top of various big data query optimization techniques, e.g., indexing, partitioning, and materialization. Our prototype implementation demonstrates the significant speedup that can be achieved at query time compared to the state-of-art techniques.

References

1. P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *IEEE ICDE*, pages 746–755, 2007.
2. F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting bloom filters. In *14th Conference on Annual European Symposium*, pages 684–695, 2006.
3. P. Brown and P. J. Haas. BHUNT: automatic discovery of fuzzy algebraic constraints in relational data. In *VLDB*, pages 668–679, 2003.
4. Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. Haloop: efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, 2010.
5. S. Chen. Cheetah: a high performance, custom data warehouse on top of mapreduce. *Proc. VLDB Endow.*, pages 1459–1468, 2010.
6. X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *PVLDB*, 6(13):1498–1509, 2013.
7. J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, Y. Kargin, V. Setty, and J. Schad. Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing). In *VLDB*, volume 3, pages 518–529, 2010.
8. I. Elghandour and A. Aboulnaga. Restore: reusing results of mapreduce jobs. *Proc. VLDB Endow.*, 5(6):586–597, 2012.
9. M. Y. Eltabakh, F. Özcan, Y. Sismanis, P. Haas, H. Pirahesh, and J. Vondrak. Eagle-Eyed Elephant: Split-Oriented Indexing in Hadoop. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT)*, pages 89–100, 2013.
10. M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson. Cohadoop: Flexible data placement and its exploitation in hadoop. *PVLDB*, 4(9):575–585, 2011.
11. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.*, 33(2):6:1–6:48, 2008.
12. P. J. Haas, J. F. Naughton, S. Seshadri, and L. Stokes. Sampling-based estimation of the number of distinct values of an attribute. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 311–322, 1995.
13. Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.
14. O. Ibarra and C. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22:463–468, 1975.
15. I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *In SIGMOD*, pages 647–658, 2004.
16. D. Jiang, B. C. Ooi, L. Shi, and S. Wu. The performance of mapreduce: an in-depth study. *Proc. VLDB Endow.*, pages 472–483, 2010.
17. H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. Correlation Maps: A Compressed Access Method for Exploiting Soft Functional Dependencies. *PVLDB*, 2(1):1222–1233, 2009.
18. H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik. CORADD: correlation aware database designer for materialized views and indexes. *PVLDB*, 3(1):1103–1113, 2010.
19. H. Liu, D. Xiao, P. Didwania, and M. Y. Eltabakh. Exploiting soft and hard correlations in big data query optimization. *Proc. VLDB Endow.*, 9(12):1005–1016, 2016.
20. Y. Liu, H. Liu, D. Xiao, and M. Y. Eltabakh. Adaptive Correlation Exploitation in Big Data Query Optimization. Technical Report: `http://web.cs.wpi.edu/~meltabakh/WPITR1803.pdf`.
21. H. V. Nguyen, E. Müller, P. Andritsos, and K. Böhm. Detecting correlated columns in relational databases with mixed data types. In *SSDBM*, pages 30:1–30:12, 2014.
22. T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas. Mrshare: sharing across multiple queries in mapreduce. *Proc. VLDB Endow.*, pages 494–505, 2010.
23. Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM J. Comput.*, 40(6):1715–1737, 2011.
24. The Apache Software Foundation. Hadoop. http://hadoop.apache.org.
25. A. Thusoo, R. Murthy, J. S. Sarma, Z. Shao, N. Jain, P. Chakka, S. Anthony, H. Liu, and N. Zhang. Hive - a petabyte scale data warehousing using hadoop. In *ICDE*, 2010.
26. J. Ullman. Principles of database and knowledge-base systems. volume 1, 1988.
27. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *USENIX Conference*, pages 10–10, 2010.