Minimizing Data Movement through Query Transformation

Patrick Leyshock, David Maier, Kristin Tufte
Department of Computer Science
Portland State University
Portland, OR, U.S.A.
leyshock, maier, tufte@pdx.edu

Abstract— **Reducing data movement between desktop analytic systems and server-based data management systems is an important resource-management challenge. The system presented in this paper automatically minimizes data movement in these "hybrid" analytic systems through rewrite-based query-transformation techniques pioneered in relational database query optimization. We evaluate different classes of transformations both in terms of query improvement and optimization time.**

Keywords- Big-data analytics; hybrid analytic systems; query optimization; R; SciDB

I. Introduction

Hybrid systems for analyzing Big Data are receiving increased attention in scientific research and industry. These systems integrate sophisticated analysis tools with server-based database systems. Data can be stored at both components of a hybrid system, and operations can execute at both places; this design feature means that input data and intermediate results must move between the two components.

Data movement between hybrid components comes with costs. Recent work in high-performance computing shows that time spent moving data between computing nodes can dominate the time spent computing with it [1]. Similarly, researchers in energy-efficient computing expect that inter-machine data movement costs will soon rival computation costs for some scientific analyses [2-3]. Though there are other factors that affect the cost of performing analysis with hybrid systems – computation times at each component, obviously – the costs of data movement are worthy of a dedicated investigation.

A number of factors contribute to data-movement costs, including "time on the wire," the overhead of setting up and maintaining communication connections, competition with other systems for network bandwidth, and the formatting and restructuring required to map one system's storage model to another. Given the potential price, we should reduce the cost of data movement between hybrid components. There are several strategies for lowering costs: i) reduce the cost of switching storage formats between systems, ii) increase network communication speed and capacity, and iii) reduce the amount of data moved.

Existing literature on hybrid systems acknowledges the cost of data movement, and admits the validity of strategy (iii). To date, however, little effort has been spent on designing solutions founded upon the strategy [4-5]. Instead, most authors addressing the matter give data scientists the chore of reasoning about, and deciding on, the best ways to reduce data movement between hybrid components. Data scientists should not shoulder this burden, not only because it is outside their job description (they signed up to answer research questions, after all, not struggle with improving their research tools), but also because it is a challenging task best left out of human hands.

To investigate data movement costs we designed and implemented our own hybrid system. Our system is named Agrios, and it integrates the analytic system R with the array-modeled database management system SciDB. Agrios automatically minimizes data movement between these two components, operating on queries over array-modeled data such as vectors and matrices.

Previous publications illustrated the need for automatically minimizing data movement in hybrid systems [6], and demonstrated Agrios' utility in doing so [7]. This prior work, however, did not provide insight into the data-movement minimization process. This paper makes three main contributions: i) articulation of the conceptual details of Agrios' query-rewriting process, ii) experimental comparison of different data-movement minimization methods, and iii) analysis and discussion of the tradeoffs required by data-movement minimization.

II. Agrios

The Agrios middleware integrates R and SciDB. Agrios operates on array-modeled data, approximating R's data model. Agrios has four main components: *parser*, *accumulator*, *stager*, and *executor*. The inputs to Agrios are *queries*, or *expressions*; we use the two interchangeably here. Queries input to Agrios are "normal" R expressions enclosed by a call to an Agrios wrapper function. This paper focuses on Agrios' stager component.

A. *Key Components: R and SciDB*

R is a language and computing environment designed for data manipulation and analysis [8]. The vector is the primary data type in R, though it additionally supports complex operations on matrices and multidimensional arrays. In its "vanilla" form, R has several limitations: it can operate only on in-memory data, and it is implemented as a single-threaded process.

SciDB is a scalable database system built explicitly to handle extremely large array-modeled datasets [9]. The array is the fundamental data type of SciDB, and all of SciDB's components – including its optimizer, query processor and storage manager – are designed to perform array operations efficiently and at scale. SciDB scales through the addition of computing nodes, intended to be simple off-the-shelf commercial systems. One of SciDB's chief limitations is its languages: AFL and AQL. These new languages are unfamiliar to data scientists, and are a barrier to SciDB's adoption.

Agrios' approach combines the best parts of both of these systems: the scalability of SciDB with the familiar data model and interface of R.

B. Definitions

To better understand Agrios we need to examine a handful of terms and concepts: *query*, *data object*, *plan*, *placement*, *shape*, *size*, *location,* and *staging*. A query takes the form of an R script written by the user. It performs an analytic task, typically involving multiple operations on multiple data objects. Let A and B be two-dimensional arrays of floating-point values. The following R script is a single query, and identifies the top three average scores for a calculated value:

$$\text{result} \leftarrow \text{order (apply (A + B, 1, mean),} \\ \text{decreasing = TRUE) [1:3];}$$

Operators in queries are all *logical*; i.e. operators do not specify on which hybrid component the operator should be executed. While queries use exclusively logical operators, *plans* use exclusively *physical* operators. A physical operator in Agrios specifies at which hybrid component the operation is to be performed. In the case of Agrios, the two alternatives are R and SciDB. Physical operators and logical operators can be distinguished by the presence or absence of an execution location; physical operators subscripted with the operator's execution location, while logical operators are not. For example, in a script the logical elementwise addition operator is identified as "+", while its two physical counterparts are identified as "$+_R$" and "$+_{SciDB}$".

For our purposes, we assume that: i) for every operation, all of a query's inputs must be colocated at a processing site, and ii) all operations can be performed at either R or SciDB. These are reasonable assumptions, given both the computational model of R and SciDB, and the expressive power and extensibility of both R and SciDB's query languages. There are operations, such as plotting data, that might be available only in R, but such operations tend not to be the critical ones for optimizing data movement.

A *data object* is a unit of information that can serve as the input to an operator. We assume that any data object at the leaf level of a query or plan has a fixed location – R or SciDB. In contrast, data objects that are intermediate results between operations are not *a priori* constrained to a particular location. In the query above, the leaf-level data objects are A and B, and there will be intermediate data objects corresponding to the outputs of apply, order, and subscript.

Data objects have a number of logical and physical properties. Important logical properties include shape and size. The number of dimensions an array has, together with the relative lengths of its dimensions, determines the array's *shape.* For our purposes, the *size* of an array is the count of its data elements, which is the product of the lengths of each dimension.

The *location* of a data object is a physical property of the object, and its value is the component of the hybrid system on which it is stored or created. In Agrios, leaf-level data objects are stored once, either at R or at SciDB. If an operation creates an intermediate result, the location of that data object is the location at which the operation was performed. A *placement* is a complete assignment of locations to all leaf-level data objects in a query. If a query has *n* distinct leaf-level data objects, there are 2^n possible placements. Our sample query has four possible placements: i) A at R and B at SciDB, ii) A at SciDB and B at R, iii) both A and B at R, and iv) both A and B at SciDB.

The final concept to define is *staging*. A staging is a complete assignment of execution locations to a query's operators. The following is one staging for our sample query:

$$\text{result} \leftarrow \text{order}_{\text{SciDB}} \text{ (apply}_{\text{SciDB}} \text{ (A } +_{\text{R}} \text{ B, 1, mean),} \\ \text{decreasing = TRUE) [1:3]}_{\text{R}}$$

Here is another staging:

$$\text{result} \leftarrow \text{order}_{\text{SciDB}} \text{(apply}_{\text{SciDB}} \text{ (A } +_{\text{SciDB}} \text{ B, 1, mean),} \\ \text{decreasing = TRUE)[1:3]}_{\text{SciDB}}$$

These two stagings differ in the execution location of the plan's elementwise addition operation and its subscript operation. The process of staging – i.e., assigning execution locations to a query's operators – transforms queries into plans. The number of possible stagings is exponential in the number of the query's operators.

Stagings are important because they effectively determine data movement during query execution. *Together, a staging and a placement determine what data needs to be moved where for plan execution*. The placement states where the input data objects are, and the staging states where the plan's operations are to be performed. If the execution location of an operation differs from the storage or generation locations of its inputs, those inputs must be moved.

The amount of data moved by a plan is the plan's *cost. The cost of a plan is the total number of data elements required to be moved by the plan*. Stagings can vary in the amount of data they move, for a given placement. That is, two different stagings can have substantially different costs in terms of data movement. The example in Figure 1 illustrates such a cost difference. The plan with the lowest cost is the *movement-minimizing plan.*

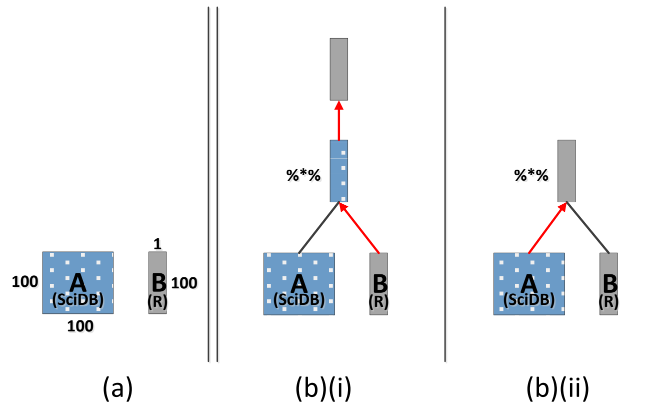

Figure 1. The amount of data moved during query processing depends on where the computation is performed. The initial state of affairs is shown in (a): array A is placed at SciDB (colored blue), and array B is placed at R (colored grey). The extents of both input arrays' dimensions are indicated. In (b)(i) the computation is performed at SciDB, and the result moved to R, for a total cost of 200. The movement of input B to SciDB and the intermediate result to R are indicated with red arrows. In (b)(ii) input A is moved to R and the computation is performed at R. The total cost of (b)(ii) is 10,000.

C. *Staging as Optimization*

Agrios' stager builds upon work in database query optimization. Optimizers in DBMSs improve database performance by automatically identifying a good implementation of a user-written declarative query.

In a similar manner, Agrios' stager component improves the performance of our hybrid system by automatically minimizing data movement between R and SciDB. This automation lets data scientists write analytic scripts while ignoring low-level properties of their data, in particular the shape, size, and storage locations of input data. After reoptimization by Agrios a script can even be reused, without modification, should these properties of the input data change.

During the optimization process, Agrios creates a *staging space* populated with queries and plans that are logically identical to the user-written query. Agrios explores this staging space through a top-down memoization algorithm that guarantees identification of the movement-minimizing plan, for a given set of transformation rules.

Agrios populates the staging space through two primary methods. First, Agrios applies rewrite rules to queries in the staging space, beginning with the user-written query. This step is *query rewriting,* and it generates alternative queries logically equivalent to the original query. Second, Agrios generates alternative stagings for all queries in the staging space, transforming each query into multiple plans.

Agrios can operate in two different modes, depending on whether or not query rewriting is enabled. If query rewriting is disabled, Agrios performs "simple staging": alternative stagings are generated only for the user-written query. If query rewriting is enabled, Agrios performs both query rewriting and staging. With query rewriting enabled, Agrios explores alternative stagings for not only the user-written query, but also for queries generated through the application of transformation rules to the user-written query.

Each plan in the search space is assigned an estimated data-movement cost, calculating according to Agrios' cost model. Based on these costs, the least-expensive plan in the staging space is then selected as the movement-minimizing plan. This plan is passed to Agrios' executor, which performs the plan's operations at the execution locations specified by the plan's staging.

D. *Query transformations*

Query rewriting generates alternative queries through the application of rewrite rules. Conceptually, rewrite rules can be conceived of as conditionals, and rule application described as a pattern-matching process. If the input query appropriately matches the antecedent, then Agrios adds to the staging space the query defined by the conditional's consequent. Thus, rule application increases the size of the staging space. Rules apply at the level of individual operations, so queries containing multiple operations may require multiple rule applications. A query created by a rewrite rule can sometimes generate lower-cost plans than plans generated by the query input to the rewrite rule. That is, applying a rewrite rule to a query may be an essential step in identifying the movement-minimizing plan.

If the application of a rewrite rule reduces data movement, it does so by *transforming* the input query. There are two main kinds of transformations: consolidating and reductive. These transformations decrease data movement by either lowering the *number* of transfers required by an expression or reducing the *amount* of data moved in a given transfer. A decrease in the number of transfers usually results from "consolidating" transformations that group operations and objects at the same location.

At present seven rewrite rules are implemented in Agrios. These rules can be classified into various types. One of the most salient type distinctions is between *consolidating rules* and *reductive rules*; these types are primarily responsible for consolidating transformations and reductive transformations, respectively. Implemented consolidating rules are: *commute, left-to-right associate*, and *right-to-left associate.* Implemented reductive rules are: *push subscript through addition, push subscript through matrix multiplication, push sum through addition,* and *push subscript through apply.* The changes effected by some of these rules are shown in Figure 2. Rules in Agrios can be switched on and off individually, or in groups.

III. EXPERIMENTAL RESULTS

Previous publications demonstrated the efficacy of Agrios in minimizing data movement. Beyond showing query rewriting to be beneficial to the staging process, this earlier work provided no visibility into the query rewriting process. Now equipped with the definitions and conceptual understanding provided in the previous section, we can take a

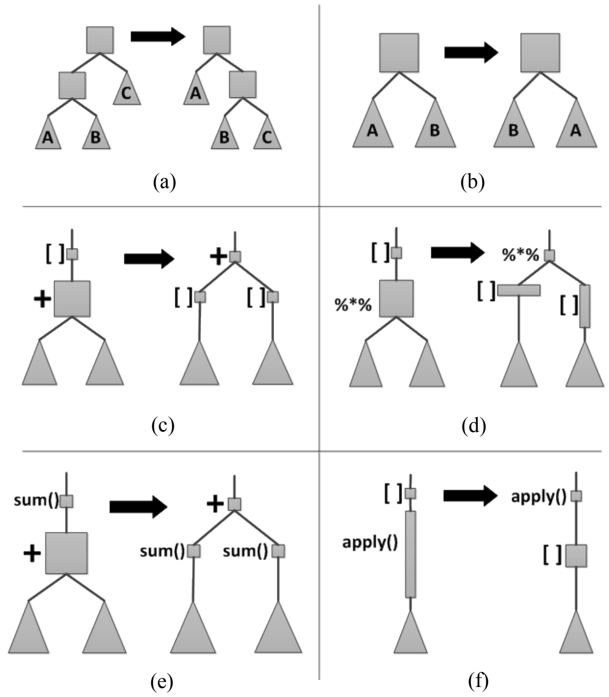

Figure 2. A number of transformation rules. Panel (a) shows left-to-right association. Agrios also contains a right-to-left association rule. The commute rule is shown in (b). This rule does not directly reduce data movement, but can do so when used in concert with other transformation rules. Subscript-through-binary addition is shown in (c), and subscript-through-matrix-multiplication shown in (d). Panel (e) shows the sum-through-binary addition rule, and panel (f) shows the subscript through apply rule.

closer look at aspects of the staging and query rewriting process.

This paper considers one particular aspect of the query rewriting process, viz., the roles different rule types play in staging. To explore these issues we ran several synthetic test queries through Agrios. Two representative test queries are discussed in this paper. Query 1 contains 10 objects and 12 operators, and Query 2 contains nine objects and 10 operators. Each query contains operators common in analytic scripts, and all operators can be executed at either hybrid component. These queries are sufficiently large and complex to permit application of transformation rules. They are also large enough to reflect the size of actual queries found in production use.

Each query was run through Agrios four times. For each run we calculated how many data elements (array cells) would have to move to execute the query; this quantity is the data-movement cost of the query. The first run of the query was staged without rewrites, the second staged using only consolidating rules, the third using only reductive rules, and the fourth using both rule types.

A. Effects of Rule Types

Results for Query 2 are shown in Figure 3. Results for Query 1 are similar, and summary data for both queries is shown in Table I. Each point in the plot shows the data-movement cost for a particular placement, for one of the four test runs. The results are sorted by plan cost, in decreasing order. Because Agrios guarantees identification of the movement-minimizing staging, all of the costs shown in Figure 3, are optimal costs, relative to the rewrite rules used.

The plot reveals a number of facts. Utilizing both rule types is sufficient for identifying the least-expensive query from the four runs. However, both rule types are not always necessary to identify the lowest cost: for some placements the optimal cost using only one rule type is identical to the optimal cost using both rule types. In some cases, the data movement reduction brought about by using both consolidating and reductive rule types together is greater than the sum of the reductions in data movement brought about by separately applying consolidating and reductive rules. As noted earlier the more rules used during query rewriting, the larger the search space for the query. A larger search space may contain a plan whose cost is less than the lowest-cost plan in a smaller search space. This result also shows that the union of the search spaces defined by consolidating rules and reductive rules is smaller than the search space defined by the union of both rule types. That is, when both rule types are used during query rewriting, the rules can interact and cause synergistic data movement reductions.

Another important feature of the results is the range of differences between optimal costs for different rule types. The differences between highest and lowest costs, for a given placement, range from many multiples of the lowest cost, to a small fraction above the lowest cost. That is, in some cases plans generated using one rule type move much less data than plans generated using another rule type. In other cases, the differences between plan costs generated from different rule types are small.

B. Rule Types and Staging Time

The results presented in Section 3.1 reveal some facts about how plan costs relate to different types of rewrite rules. Though informative, these tests provide no visibility into the time required to identify the movement-minimizing plan. Simple staging takes time; Agrios must consider and cost alternative stagings in order to identify the movement-minimizing plan. Staging with query rewriting take more time than simple staging alone.

Visibility into the tradeoff of time spent staging vs. data movement reductions is given by Figure 4; the figure compares measured staging time with data movement reductions gained through query rewriting. The plots shows summary statistics for each rule type, plotted as a function of staging time. The results reveal several important facts:

TABLE I. DATA MOVEMENT COSTS, BY RULE TYPE

	Query 1		Query 2	
	Median	Mean	Median	Mean
Neither type	30,000	26,940	18,100	17,600
Consolidating rules	22,000	24,880	13,700	15,930
Reductive rules	14,000	14,750	7,001	6,238
Both rule types	8,000	7,250	5,250	5,469

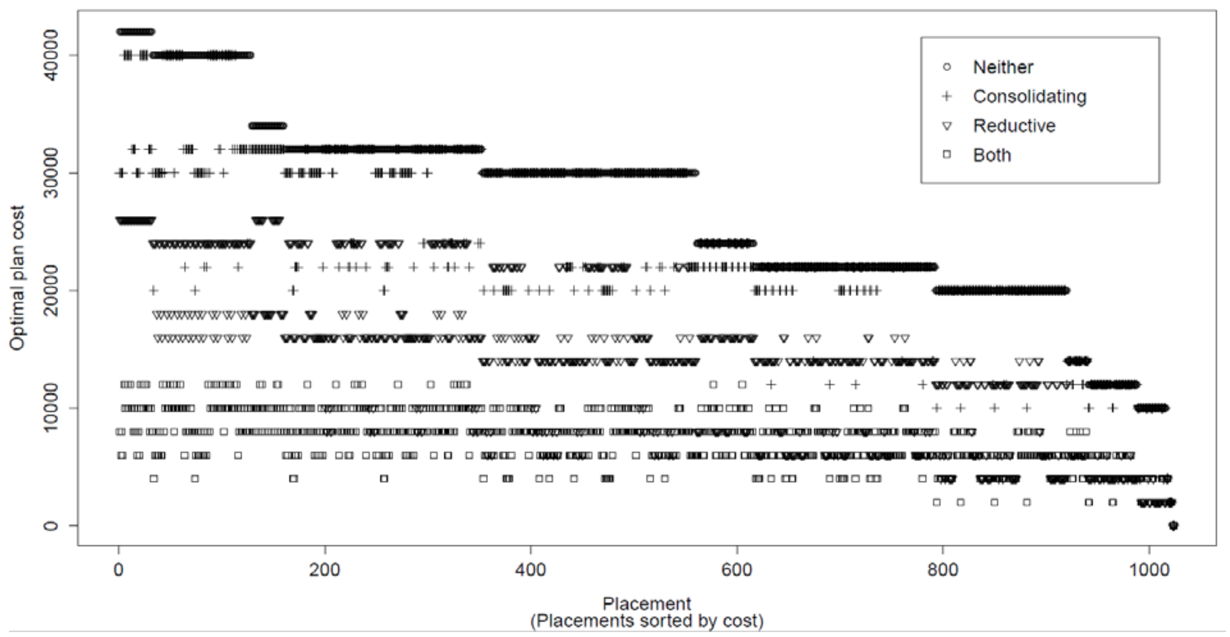

Figure 3. A comparison of plan costs by rule type for Query 2. The sort order for all placements was determined by optimal plan costs when no rewrite rules were used. The x-axis shows the query's 1024 placements. The y-axis shows the plan costs. For each placement, four costs are shown: i) the plan cost when no rewrite rules are applied (i.e. plan costs for simple staging), ii) plan costs using only consolidating rules, iii) plan costs using only reductive rules, and iv) plan costs using both consolidating and reductive rules.

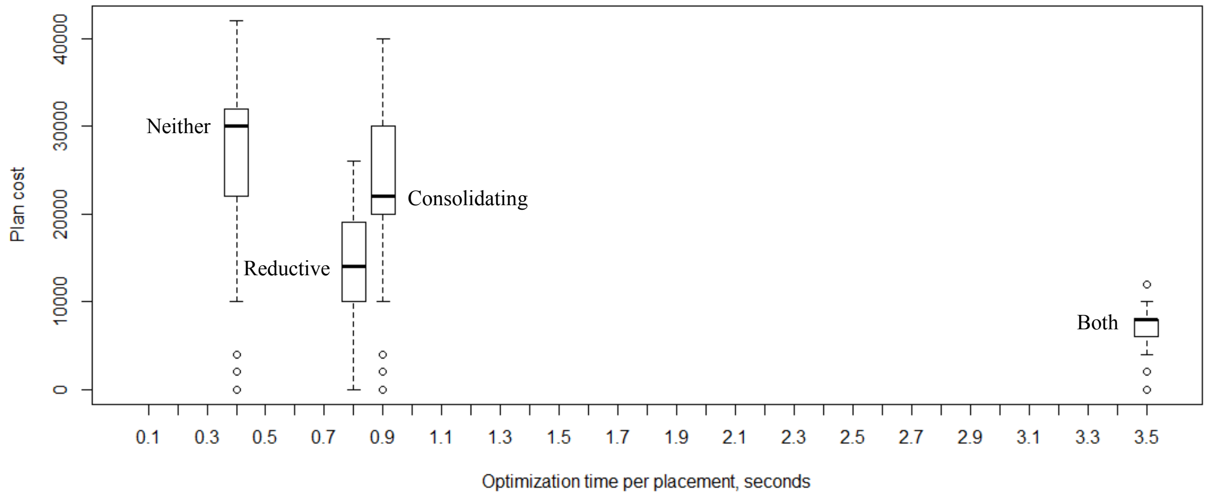

Figure 4. Summary statistics for plan data-movement costs, as a function of staging time, Query 2. Results for Query 1 are similar, though noteworthy in that staging using only consolidating rules takes less time than staging using only reductive rules.

1) Staging with transformation rules takes more time than staging without transformations. This finding aligns with our expectations; the plot in Figure 4 quantifies the overhead incurred by staging with transformations.
2) When only one rule type is used during staging, sometimes staging with consolidating rules is faster than staging with reductive rules, and sometimes *vice versa*. Use of both rule sets during staging takes substantially more time than using only one rule set. In addition, we see that the average time required to use both rule sets is greater than the sum of the average times required for using both rule sets individually.
3) On average, data-movement costs using only reductive rules are less than data movement costs using only consolidating rules. There is a notable amount of variability in costs, however, and in some cases a plan generated using only consolidating rules is less expensive than a plan using only reductive rules.
4) While plan costs using both rule sets are less than plan costs using individual rule sets, the plan costs using both sets are on average not much less than plan costs

seen with one rule set. Though we have evidence of synergistic interactions between rule types, the gains achieved through synergistic interactions typically appear to be additive and incremental, not multiplicative.

IV. RELATED WORK

A variety of optimization techniques have been applied to both database and workflow systems. Optimization techniques vary, and include both cost- and heuristic-based methods. Some systems use transformations [10-12], others focus on optimizing configuration parameters [13]. Some researchers even expressly optimize data movement reduction in scientific workflows. Guo et al., for example, use a particle-swarm optimization algorithm to do so [14]. Both of these systems differ significantly from ours in that they do not optimize at the query level.

Query-level transformation rules for array-modeled data were pioneered largely in the 1990s. Marathe and Salem's "Array Manipulation Language" (AML) included a number of transformation rules for exploring linearization orders [15]. The RAM system performs transformation-based query optimization, though only at the logical level [16].

Hybrid analytic systems are increasingly common, and chiefly differentiated by the components they integrate. R is often used as the analytic component of the hybrid: RICE integrates R with SAP's HANA database [4], RIOT-DB integrates R with a MySQL database [16], and Ricardo integrates R with Hadoop [5]. These hybrid systems acknowledge the cost of moving data between hybrid components, but none automatically minimizes data movement.

V. CONCLUSION

In a hybrid system, a desktop analytic tool is coupled with a server-based data management system. Hybrid systems show promise for the analysis of large disk-resident data, but their performance depends upon minimizing data movement between the two components. Agrios is a hybrid system integrating R and SciDB that automatically minimizes data movement between the two components. One technique used by Agrios to reduce data movement is query rewriting. Experimental results presented here provide insight into the query-rewriting process. The results also show the relative efficacy of different transformation rule types.

ACKNOWLEDGMENT

This work is supported by National Science Foundation Grant Number 1110917, and funded in part by the Intel Science and Technology Center for Big Data. Thanks to Jason Nelson, Portland State University's Datalab, the SciDB team, and Paradigm4. Remarks from our anonymous reviewers were appreciated and helpful.

REFERENCES

[1] Park, J., Bikshandi, G., Vaidyanathan, K., Tang, P., Dubey, P., and Kim, D. 2013. Tera-Scale 1D FFT with Low-Communication Algorithm and Intel Xeon Phi Coprocessors. *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*.

[2] Tiwari, D., Vazhkudai, S., Kim, Y., Ma, X., Boboila, S. and Desnoyers, P. 2012. Reducing data movement costs using energy-efficient, active computation on SSD. *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*. USENIX Association, 1-5.

[3] Wong, P., Shen, H., Johnson, C.R., Chen, C., and Ross, R. 2012. The top 10 challenges in extreme-scale visual analytics. *IEEE Computer Graphics and Applications*, Volume 32, Issue 4, 63-67.

[4] Grosse, P., Lehner, W., Weichert, T., Farber, F., and Li, W.S. 2011. Bridging two worlds with RICE. *Proceedings of the VLDB Endowment*, 1307-1317.

[5] Das, S., Simanis, Y., Beyer, K.S., Gemulla, R., Haas, P.J., and McPherson, J. 2011. Ricardo: Integrating R and Hadoop. *Proceedings of the 2010 International Conference on Management of Data,* 987-998.

[6] Leyshock, P., Maier, D., and Tufte, K. 2014. Data movement in hybrid analytic system: a case for automation. *SSDBM '14.*

[7] Leyshock, P., Maier, D., and Tufte, K. 2013. Agrios: A hybrid approach to big array analytics. *IEEE International Conference on Big Data*, 85–93.

[8] Ihaka, R., and Gentelman, R. 1996. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 299-314.

[9] Stonebraker, M., Brown, P., Poliakov, A., Raman, S. 2011. The Architecture of SciDB. *Proceedings of SSDBM 2011,* 1-10.

[10] Lim, H., Herodotou, H., and Babu, S. 2012. Stubby: A Transformation-based Optimizer for MapReduce Workflows. *VLDB 2012*, 1196-1207.

[11] Herodotus, H., Babu, S. 2011. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. *VLDB 2011*, 1111-1122.

[12] Hueske, F., Peters, M., Sax, M., Rheinlander, A., Bergmann, R., Krettek, A., and Tzoumas, K. 2012. Opening the Black Boxes in Data Flow Optimization. *VLDB 2012*, 1256-1267

[13] Guo, L., Zhao, S., Shen, S., and Jiang, C. 2012. Task scheduling optimization in cloud computing based on heuristic algorithm. *Journal of Networks*, 547-553.

[14] Marath, A. and Salem, K. 1999. Query processing techniques for arrays. *SIGMOD 1999*, 323-334.

[15] Cornacchia, R., van Ballegooij, A., and de Vries, A.P. 2004. A case study on array query optimization. *Proceedings of the 1st International Workshop on Computer Vision Meets Databases*, 3-10.

[16] Yi, Z., Herodotou, H., and Yang, J. 2009. RIOT: I/O-efficient numerical computing without SQL. *CIDR 2009*, 1-11.