Adaptive Query Processing: Technology in Evolution

Joseph M. Hellerstein Michael J. Franklin Sirish Chandrasekaran Amol Deshpande
Kris Hildrum Sam Madden Vijayshankar Raman Mehul A. Shah

Abstract

As query engines are scaled and federated, they must cope with highly unpredictable and changeable environments. In the Telegraph project, we are attempting to architect and implement a continuously adaptive query engine suitable for global-area systems, massive parallelism, and sensor networks. To set the stage for our research, we present a survey of prior work on adaptive query processing, focusing on three characterizations of adaptivity: the frequency of adaptivity, the effects of adaptivity, and the extent of adaptivity. Given this survey, we sketch directions for research in the Telegraph project.

1 Introduction

Adaptivity has been an inherent – though largely latent – aspect of database research for the last three decades. Codd's vision of data independence was predicated on the development of systems that could adapt gracefully and opaquely to changing data and data structures. Query optimization, with its attendant technologies for cost estimation, served as an early differentiator between DBMSs and other computer systems. This tradition of dynamic, statistically-driven system optimization remains one of the crown jewels of the database systems research community.

In the last few years, broad sectors of computer science have been exploring the design of systems that are adaptive to their environment. As computer systems scale up and federate, traditional techniques for system management and performance tuning must become more loosely structured and more aggressively adaptive. In the context of database systems, this stretches the traditional techniques for adaptive query processing to the breaking point, since very large-scale query engines operate in unpredictable and changeable environments. This unpredictability is endemic in large-scale systems, because of increased complexity in a number of dimensions [AH00]:

Hardware and Workload Complexity: In wide-area environments, variabilities are commonly observable in the bursty performance of servers and networks [UFA98]. These systems often serve large communities of users whose aggregate behavior can be hard to predict, and the hardware mix in the wide area is quite heterogeneous. Large clusters of "shared-nothing" computers can exhibit similar performance variations, due to a mix of user requests and heterogeneous hardware evolution. Even in totally homogeneous environments, hardware performance can be unpredictable: for example, the outer tracks of a disk can exhibit almost twice the bandwidth of inner tracks [Met97].

Data Complexity: Selectivity estimation for static alphanumeric data sets is fairly well understood, and there has been initial work on estimating statistical properties of static sets of data with complex types [Aok99] and methods [BO99]. But federated data often comes without any statistical summaries, and complex non-alphanumeric

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Frequency of Adaptivity				
Batch	Per Query	Inter-Operator	Intra-Operator	Per Tuple
System R [SAC+79], Late Binding [HP88, GW89] [INSS92, GC94] [ACPS96, LP97]	ASE [CR94], Sampling [BDF+97], Mariposa [SAP+96],	Query Scrambling [AFTU96, UFA98], Reoptimization [KD98], Tukwila [IFF+99]	Ingres [SWK76], RDB [AZ96] WHIRL [Coh98]	River [AAT+99], Eddies [AH00]

Table 1: Prior work on adaptive query processing, ordered by frequency of adaptivity. We omit specific adaptive operators from the table, including sorting and joins as they are inherently intra-operator in frequency. We discuss them further in Section 2.7.

data types are now widely in use both in object-relational databases and on the web. In these scenarios – and even in traditional static relational databases – selectivity estimates are often quite inaccurate.

User Interface Complexity: In large-scale systems, many queries can run for a very long time. As a result, there is interest in Online Aggregation and other techniques that allow users to "Control" properties of queries while they execute, based on refining approximate results [HAC+99].

1.1 Telegraph in Context

The goal of the Telegraph project at Berkeley is to architect and build a global-scale query engine that can execute complex queries over all the data available online[1]. In designing Telegraph, we are facing new challenges in wide-area systems, including the quickly shifting performance and availability of resources on the Internet. We also want Telegraph to serve as a plug-and-play shared-nothing parallel query engine, which must be trivial to manage and scale incrementally. Finally, we want to support sensor networks (e.g., [EGH99, KKP99, KRB99]), which are emerging as an increasingly important – and unpredictable – aspect of "ubiquitous" ("calm", "post-PC") computing.

Key to Telegraph is a continuously adaptive query processing engine that can gather and act upon feedback at a very high frequency. This high-frequency adaptivity opens up both new opportunities and challenges in our research agenda. In this paper we outline our view of earlier advances in adaptive query processing research, which set the stage for the challenges we are addressing in building Telegraph. Table 1 provides a brief overview of the work we survey.

1.2 Framework

Unfortunately, the phrase "adaptive system" is not canonical. These systems are sometimes refered to as "dynamic" or "self-tuning" systems, or systems that change their behavior via "learning", "introspection", and so on. There are many tie-ins to work in related fields, such as control theory [Son98] or machine learning [Mit97]. To avoid confusion in terminology, we present our own definition of adaptivity in this context. We will call a query processing system *adaptive* if it has three characteristics: (1) it receives information from its environment, (2) it uses this information to determine its behavior, and (3) this process iterates over time, generating a feedback loop between environment and behavior

We wish to stress a distinction between adaptive systems, and systems that do static optimization. Static optimization contains the first two of these characteristics, but not the third. The feedback involved in an adapative

[1] The name refers to Telegraph Avenue, the volatile and eclectic main street of Berkeley.

system is key to its efficacy: it allows the system to make – and observe the results of – multiple decisions. This allows it to consider its own effects on the environment in concert with external factors, and to observe the effects of "exploiting" previously beneficial behavior, and "exploring" alternative behavior [Mit97].

Based on this definition of adaptivity, we can isolate three distinguishing features of an adaptive system: (1) the *frequency* of adaptivity (how often it can receive information and change behavior), (2) the *effects* of adaptivity (what behavior it can change), and (3) the *extent* of adaptivity (how long the feedback loop is maintained). As we survey prior work on adaptive query processing, we attempt to address all three of these issues.

2 A Survey of Adaptive Query Processing

Without embracing historical relativism too broadly, we admit that our survey says as much about our research agenda as about the history we present. Indeed, our goal in this short paper is to place our research agenda for adaptive query processing in the context of our view of the promise and shortcomings of the research to date.

We organize our survey partly chronologically, and partly by increasing frequency of adaptivity. As is clear from Table 1, the two orderings are roughly correlated, with some interesting exceptions. The progression toward increasing frequency over time seems to be natural, as systems are designed for increasingly complex and changeable environments.

2.1 Early Relational Systems

From the very first prototypes, relational query processors have incorporated some minimal notion of adaptivity, typically to capture the changing distributions of data in the database, and use that to model subquery result sizes, and hence query operator costs.

The System R query optimizer [SAC$^+$79] (which inspired essentially all serious commercial DBMS implementations today) kept a catalog of statistics, including cardinalities of tables and coarse distributions of values within columns. By our definition, System R was not exactly adaptive: it did not have any explicit feedback within the system. On the other hand, the system administrator could manually direct System R to adapt its behavior to the data: on command, the system would scan the entire database and update its statistics, which upon completion would instantly affect all decisions made in query optimization. This heavyweight, periodic *batch adaptivity* approach remains in nearly all commercial DBMSs today, though of course the statistics gathered – and the means for gathering them – have become more sophisticated over time. While the frequency of adaptivity is quite low in these systems (statistics are typically updated once a day or once a week), the effects are broad and the extent far-reaching: as a result of new statistics, the optimizer may choose completely different access methods, join orders and join algorithms for all subsequent queries.

The Ingres "query decomposition" scheme [SWK76] was less effective but much more adaptive than System R. Ingres alternated between subplan selection and execution. For a query on n tables, it operated as a greedy but adaptive sequence of nested-loops joins. The smallest table[2] was chosen to be scanned first. For each tuple of the smallest table, the next-smallest table was probed for matches via the "one-variable query processor" (i.e. the table-scan or index lookup module), and a (duplicate-free) result of that probe was materialized. This greedy process was then recursively applied on the remaining $n-1$ tables – i.e., $n-2$ base tables and one materialized sub-result. After the recursion unwound, the process began again for the next tuple of the smallest table. Note that this does not correspond to a static "join order" in the sense of System R: the materialized result of each "tuple probe" could vary in size depending on the number of matches to each individual tuple, and hence the order of joins in the recursive call could change from tuple to tuple of the smallest table. Although this greedy optimization process often resulted in inefficient processing in traditional environments, it had a relatively high frequency of adaptivity, gathering feedback after each call to the one-variable query processor. The frequency of adaptivity

[2]Actually, the smallest table after single-table predicates were applied and results materialized ("one-variable detachment").

in Ingres was thus *intra-query*, and even *intra-operator*, in the sense that adaptation could happened from tuple to tuple of the outermost table in a sequence of nested loops joins. This remained one of the highest-frequency schemes in the literature until quite recently. The feedback gathered after each table-scan or index lookup had the effect of adapting the join *order* on subsequent iterations. Note that the effects of adaptivity in Ingres extended only across the lifetime of a single query.

2.2 Late Binding Schemes

The schemes we discuss next are in some sense no more adaptive than that of System R, gathering no more information than is available to a standard optimizer. However, these schemes have a flavor of dynamism, and are often discussed in the context of adaptive query processing, so for completeness and clarification we cover them here.

The focus of this body of work is to improve upon a particular feature of System R's optimizer for frequently re-executed queries. In addition to its other features, System R introduced the ability for queries to be optimized, compiled into machine code, and stored with the system for subsequent reuse. This technique, which is available in commercial RDBMSs today, allows the cost of query optimization to be amortized across multiple executions of the same query, even if the query's constants are left unbound until runtime.

In the late 1980's and early 90's, a number of papers addressed a weakness in this scheme: subsequent runs of the query occur under different easily-checked runtime parameters, including changes in user-specified constants that affect selectivity, changes in available memory, and so on [HP88, GW89, INSS92, GC94, ACPS96, LP97]. Query execution performance might be compromised under such changes, but the cost of complete reoptimization on each small perturbation of the environment could be wasteful. To strike a happier medium, these systems do some optimization in advance, and need to consider only a subset of all possible plans at runtime. The plan eventually chosen for execution is intended to be the one that would be achieved by running the full System R optimizer at runtime. In a typical example of this work, Graefe and Cole describe *dynamic query plans* [GC94]. Given constraints on possible changes in the runtime environment, their optimizer would discard only those query plans that were suboptimal in all configurations satisfying these constraints. The result of their optimizer was a *set* of possible query plans, which was searched at runtime based on easily checkable parameters of the environment.

These schemes focus on the problem of postponing a minimal decision until runtime, effectively doing "late binding" of unknown variables for frequently re-executed queries. But they do not take any special advantage of iterative feedback, and offer the same frequency, effects and extent of adaptivity that one gets by running a System R optimizer whenever a query is to be executed.

2.3 Per-Query Adaptivity: Making System R More Adaptive

System R's statistics-gathering scheme was coarse grained, running only periodically, requiring administrative oversight, and consuming significant resources. Chen and Roussopoulos proposed an *Adaptive Selectivity Estimation* (ASE) scheme to enhance a System R-style optimizer by piggybacking statistics-gathering on query processing [CR94]. When a query was executed in their scheme, the sizes of sub-results would be tracked by the system, and used to refine statistical metadata for future optimization decisions. This provided an organic feedback mechanism, leveraging the natural behavior of query processing to learn more and perform better on subsequent queries. ASE operated on a moderately coarse *per-query* frequency – still inter-query like System R, but finer grained and more naturally adaptive. The effects of feedback in ASE were potentially as significant as those of System R, affecting access method selection, as well as join order and join method selection, with a long-term, inter-query extent. Note however that while ASE exploited information available from queries once they were issued, it did not gather information on tables that had not yet been referenced in queries.

The Mariposa distributed DBMS enabled per-query adaptivity with federated administration [SAP$^+$96]. Mariposa used an economic model to let sites in a distributed environment autonomously model the changing costs of their query operations. During query optimization, Mariposa requested "bids" from each site for the cost of performing subplans. The bidding mechanism allowed sites to observe their environment from query to query, and autonomously restate their costs of operation for subsequent queries. In the initial experiments, Mariposa used this flexibility to allow the sites to incorporate load average and other transient parameters into their costs. More sophisticated "pricing agents" could be programmed into the system via a scripting language, and the commercial version of the system (Cohera [HSC99]) exposes an extensible API suitable for 3rd-party pricing agents (or "bots"). In terms of frequency, this is essentially like ASE: inter-query adaptivity at a per-query granularity. The novelty in Mariposa comes from the way that economics enables autonomous and extensible control of adaptive policies at sites in a federation. Mariposa used a "two-phase" optimization scheme, in which join orders and methods are chosen by a central optimizer, and bids only affect the choice of sites to do the work.

2.4 Competition and Sampling

One of the more unusual query optimizers to date is that of DEC (later Oracle) RDB [AZ96]. RDB was the first system to employ *competition* to help choose query plans. The RDB designers focused in particular on the challenge of choosing access methods for a given single-table predicate. They noted that the relative performance of the two access methods could be differentiated on-line by running both for only a short time. Based on this insight, they chose to simultaneously execute multiple access methods for a given table, and halt all but the most promising access method after a short time. RDB was the first system after Ingres to support adaptivity at an *intra-operator* frequency, though it only made one decision per table in a query, and only had effects on the choice of access method.

The RDB scheme is not unlike sampling-based schemes for selectivity estimation, which also perform partial query executions to learn more about the performance of a full run. A sequence of papers in the last decade has studied how to use sampling to estimate the selectivity of predicates (see Section 9 of [BDF+97] for a survey). Sampling has typically been proposed for use during query optimization, and used only to direct the initial choice of a query plan (with the exception of online query processing, Section 2.7.2). Thus like ASE or Mariposa, this is *per-query* frequency – a finer grain than System R, but not the intra-query frequency of RDB. The effects are like those of System R: changing statistics can affect all aspects of query processing. However the extent is much shorter than System R, lasting only for the run of a single query.

2.5 Inter-Operator Reoptimization and Query Scrambling

As outlined in the introduction, the need for adaptivity grows with the uncertainty inherent in the query processing environment. Highly unpredictable environments, such as the Internet, require adaptivity even during the exeuction of a single query. A number of projects focusing on uncertain environments have employed intra-query adaptivity. Inter-operator adaptivity was a natural first step in this agenda. One approach used in distributed systems was to send subqueries to remote sites and then to use the arrival of the subquery results to drive the scheduling for the remaining parts of the query that used them [TTC+90, ONK+96]. Such approaches deal with uncertainties in the execution costs of the remote subqueries, and if subquery results are materialized, can also cope with unexpected delays to some extent.

Query Scrambling [AFTU96] was developed specificaly to cope with unexpected delays that arise when processing distributed queries in a wide-area network. With Query Scrambling, a query is initially executed according to a plan generated by a System R-style query optimizer. If, however, a significant performance problem is detected during the execution, the query plan is modified on the fly. Query Scrambling uses two basic techniques to cope with unexpected delays: 1) it changes the execution order of operations in order to avoid idling, and 2) it synthesizes new operations to execute in the absence of other work to perform. As described in [UFA98], the scrambling process can be driven by a lightwieght, response-time based query optimizer.

Kabra and DeWitt proposed a *reoptimization* scheme [KD98] to address uncertainties in the sizes of subquery results. As in Query Scrambling, an initial query plan is chosen by a traditional System R-style optimizer. After every blocking operator in that plan, the remainder of the plan is reoptimized with the knowledge of the size of the intermediate result generated thus far. The Tukwila system proposed a very similar technique, with a rule language to specify logic for deciding when to reoptimize, and the preservation of optimizer state to reduce the cost of reoptimization [IFF+99].

In essence, inter-operator adaptivity is a long-postponed marriage of the Ingres and System R optimization schemes: like Ingres, it takes advantage of the cardinality information in materialized subresults; like System R it uses cost-based estimation of unknown work to be done. These schemes adapt at an inter-operator frequency, with arbitrary effects on the *remaining steps* after a block in a query plan; the extent does not go beyond the rest of the query.

2.6 Intra-Operator Adaptivity Revisited: WHIRL

WHIRL is a data integration logic designed at AT&T labs that focuses on textual similarity predicates, with an information-retrieval-style semantics of ranked results. The AT&T implementation of WHIRL attempts to return the top ranked answers quickly. The basic query processing operators in the WHIRL implementation are essentially scans ("exploding a literal") and ranked index lookups from inverted-file indexes ("constraining a literal"). At the end of each table-scan or index lookup, the implementation can choose among all subsequent possible table-scans or index lookups. The AT&T WHIRL implementation is remarkably similar to INGRES in its intra-operator adaptivity: it supports only nested loops joins, and adapts join order during a pipeline of these joins – potentially changing the order after each complete call to an access method.

2.7 Adaptive Query Operators

Up to this point, the effects of adaptivity that we have discussed have been at the level of query optimization: the choice of access methods, join methods, and join orders (and in the case of distributed queries, the choice of sites). Adaptivity can occur at the level of individual operators, which can adapt at an intra-operator frequency even within the context of a fixed query plan.

2.7.1 Memory-Adaptive Sorting and Hashing

Two of the basic operations in query processing are sorting and hashing. Both of these operations have costs that are a function of the amount of main memory available. Query optimizers estimate these costs under assumptions of memory availability. In some systems, adequate memory is reserved to guarantee these costs prior to execution; in others, memory allocation is allowed to proceed without regard to global consumption. The former technique is conservative, potentially resulting in under-utilization of resources and increases in query latency. The latter is aggressive, potentially resulting in paging and both decreased query throughput and increased latency.

A third approach is to modify sorting and hashing algorithms to adapt to changing amounts of available memory. Such schemes typically address both sudden losses and sudden gains in memory. Losses of memory are typically handled by spilling hash partitions or postponing the merger of sorted runs; gains in memory are exploited by reading in spilled hash partitions, or adding sorted runs to a merge phase. Exemplary work in this area was done by Pang, Carey and Livny, who studied both memory-adaptive variants of hash join [PCL93b] (based on earlier work including [NKT88, ZG90]), and memory-adaptive variants of out-of-core sorting [PCL93a] (followed by related work in [ZL97]).

2.7.2 Pipelining and Ripple Join Algorithms

The pipelining property of join algorithms has been exploited for parallelism; Wilschut and Apers proposed the symmetric hash join to maximize pipelined parallelism in their PRISMA/DB parallel, main-memory DBMS [WA91]. More recently, both the Tukwila and Query Scrambling groups extended the pipelining hash join in a natural fashion so that it would run out-of-core. The Query Scrambling group's *XJoin* algorithm [UF00] pushed the out-of-core idea further, to exploit delayed data feeds: when a particular tablescan is delayed, the XJoin expolits the idle time by pulling spilled partial partitions off of disk and joining them. Neither of these extensions is adaptive in the sense of our earlier definition, but both were used in the context of adaptive systems.

Recent work on *online query processing* in the Control project highlighted the ability for pipelining operators to provide continuous feedback and hence drive adaptivity [HAC$^+$99]. In the initial work on Online Aggregation [HHW97], pipelining query processing algorithms were cited as a necessary condition for allowing *user* feedback during query processing. In subsequent work, system feedback was used internally by *ripple join* algorithms to automatically adapt the relative rates at which they fetch data from different tables, subject to a statistical performance goal [HH99]. The access methods in online query processing have typically been assumed to do progressive sampling without replacement, putting this work in the tradition of sampling for cost estimation.

Haas and Hellerstein [HH99] broadly define the ripple joins as a family of pipelining join algorithms that sweep out the cartesian plane in ever-larger rectangles, with the opportunity to adaptively control or tolerate changing "aspect ratios" of those rectangles based on observed statistical and performance behavior. They include in their definition the earlier pipelining hash join and index nested-loops joins, along with new iterative and "block" ripple joins that they introduce as modifications of the traditional nested loops joins[3]. Ripple joins as a class provide *intra-operator* frequency of adaptivity: behavior is visible at a *per-tuple* frequency, though decisions are made in [HH99] at a slightly coarser granularity, at the end of each "rectangle" swept out in processing. This adaptivity was exploited only for controlling the relative rates of I/O from different tables, during the run of a single query.

2.8 Rivers: Adaptive Partitioning

In a shared-nothing parallel DBMS, intra-operator parallelism is achieved by partitioning data to be processed among the nodes in the system. Traditionally this partitioning is done statically, via hashing or round-robin schemes. *River* [AAT$^+$99] is a parallel dataflow infrastructure that was proposed for making this partitioning more adaptive. In developing the record-setting NOW-Sort implementation [AAC$^+$97], the River designers noted that large-scale clusters exhibited unpredictable performance heterogeneity, even across physically identical compute nodes. As a result, they designed two basic mechanisms in River to provide data partitioning that adapted to the relative, changing rates of the various nodes. A *Distributed Queue* mechanism balanced work among multiple consumers running at different (and potentially changing) rates; a *Graduated Declustering* scheme dynamically adjusted the load generated by multiple redundant producers of data. Both these mechanisms adapted at a per-tuple frequency, affecting the assignment of work to nodes, with effects lasting for the duration of the operator.

2.9 Eddies: Continuous Adaptivity

In a query plan (or subplan) composed of pipelining operators like ripple joins, feedback is available on a tuple-by-tuple basis. As a result, it should be possible for a pipelined query (sub)plan to adapt at that frequency as well. Ripple joins, rivers, and other schemes leverage the feedback within an operator to change the behavior of the operator; *Eddies* are a mechanism to get inter-operator effects with intra-operator frequency of adaptivity [AH00].

[3] The aspect ratio of index nested-loops join is actually not subject to modification: the indexed relation is fully "swept" upon each index probe. However it was included in the family since it can present a performance enhancement over the iterative ripple joins when an index is available

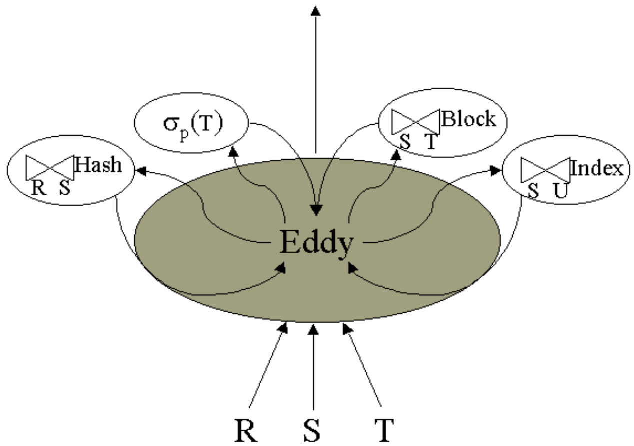

Figure 1: An eddy in a pipeline. Data flows into the eddy from input relations R, S and T. The eddy routes tuples to pipelining operators; the operators run as independent threads, returning tuples to the eddy. The eddy sends a tuple to the output only when it has been handled by all the operators. The eddy adaptively chooses an order to route each tuple through the operators.

An eddy is an encapsulated dataflow operator, akin to a join, sort, or access method, with an *iterator* interface. In the same way that the *exchange* operator of Volcano encapsulates parallelism, eddies are used to encapsulate adaptivity. An eddy is intended to be interposed between the tablescans in a plan (which produce input tuples to the eddy), and other "upstream" operators in the plan (which serve both to consume the eddy's output tuples and produce additional inputs).

As originally envisioned, the eddy – combined with pipelining operators like ripple joins – serves to adapt join order on a tuple-by-tuple basis. Every operator "upstream" of the eddy returns its results to the eddy, which passes them along to remaining operators for further processing. As a result, the eddy encapsulates the ordering of the operators by routing tuples through them dynamically (Figure 1). Because the eddy observes tuples entering and exiting the pipelined operators, it can adaptively change its routing to effect different operator orderings. It does so via a lottery scheduling scheme [WW94]. It can also control the rates of input from tablescans (if this is under the control of the system, as in the original online aggregation scenario.) In their original incarnation, the extent of adaptivity in eddies is restricted to a single query. In the next section we describe our agenda to extend the effects and extent of eddies.

3 Challenges in Adaptive Query Processing

While adaptivity has long been an issue in database research, we believe that the frequency of adaptivity in prior systems has been insufficient for many of the emerging large-scale environments. Work on continuous adaptivity has just begun, and we see a number of challenges and opportunities in this space.

In designing Telegraph, we are exploring new ideas in continuously adaptive query processing. The Telegraph query engine is a dataflow architecture based on rivers, eddies, and pipelining query processing operators like ripple joins and XJoins. As such, it is architected for fine-grained adaptivity and interactive user control. Rivers and eddies are quite general concepts, and we view them more broadly than the specific incarnations presented in the original papers. In particular, a major goal of Telegraph is to enhance both the extent and the effects of both schemes. Enhancing their extent is relatively simple: a metadata store can be used to save information at the end of each query, and this information consulted to set up initial conditions for subsequent queries (data partitioning, relative numbers of lottery tickets, etc.) Enhancing the set of effects from rivers and eddies is more complicated, and we are considering a number of challenges in this regard:

Spanning trees: As originally presented, eddies have no effect on the spanning tree for the query: i.e., the set of binary joins that connect the relations in the query. For cyclic join graphs, it is attractive to consider adapting the choice of spanning tree on the fly, by connecting the eddy to joins (or cross-products) between more pairs of tables than is strictly necessary. This is essentially a form of competition among spanning trees, and hence it can consume significant resources and produce duplicate results, two issues that we are currently investigating.

Data sources: River handles multiple potential data sources via graduated declustering, but requires a particular layout for those sources. We would like to extend this idea to a federated environment like the Internet, where data layout – and even data contents – are not under our control. This may produce duplicate or even inconsistent results, which need to be resolved.

Join and access methods: The choice of join and access methods can only be made on the fly by an eddy if they are run competitively, allowing the eddy to explore their behavior. In exploring this idea, we need to watch resource contention as well as the production of duplicates.

Moving state in rivers: The original River paper did not explicitly discuss how per-node state (e.g. hash partitions in a hash-join) could be migrated as the system adapts. We are investigating applying the ideas from memory-adaptive hash joins in this context.

In addition to extending rivers and eddies to have more general effects over a longer extent of time, we are considering a number of additional adaptivity questions:

Two-dimensional interactivity: Prior work on online query processing allowed "horizontal" user feedback on the *rows* to be favored for processing. We are also considering the "vertical" case in Telegraph: partially-joined tuples should be available at the output, and users should be able to express their relative desire for different *columns*, or even combinations of rows and columns. Partially-joined tuples may not necessarily have matches in all tables in the query, meaning that even though they are passed to the output, they may not be contained in the final answer. This has implications for user interfaces, client APIs, and of course the policies used in eddies.

Initial delays: Prior work on query scrambling addressed initial delays, but it is not clear how this work might dovetail with eddies and rivers. Challenges arise from the complete lack of feedback available in initial-delay scenarios.

Caching and prefetching: Caching is probably the most well-studied adaptive problem in computer systems, and is particularly relevant for a high-latency Internet environment. Prefetching is a similarly robust adaptive problem, dynamically predicting data fetches in the face of changing workloads. We are aggressively exploring both techniques in the context of Telegraph.

In addition to the constructive goals of Telegraph, many analytic questions remain as well. The first is to more carefully characterize the vagaries of all the application environments we are considering, including the wide area, clusters, and sensor networks. We have evidence that each of these environments merits a fine-grained frequency of adaptivity, and we believe that broad effects and extent of adaptivity will be appropriate as well. However the more we know about these environments, the better our adaptive systems should be able to perform. Second, we are actively engaged with our colleagues in theoretical computer science and machine learning in developing a formal framework for eddies, to try and characterize questions of convergence and stability, and to tune the policies used for adapting.

Acknowledgments

Thanks to Christos Papadimitriou, Stuart Russell and Alistair Sinclair for discussions on formal aspects of eddies and adaptivity. This work was supported by two grants from IBM Corporation, a grant from Siemens AG, a grant from ISX Corporation, NSF grant IIS-9802051, and DARPA contract N66001-99-2-8913. Hellerstein was supported by a Sloan Foundation Fellowship, Raman was supported by a Microsoft Graduate Fellowship.

Computing and network resources for this research were provided through NSF RI grant CDA-9401156.

References

[AAC+97] Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, David E. Culler, Joseph M. Hellerstein, and David A. Patterson. High-Performance Sorting on Networks of Workstations. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Tucson, May 1997.

[AAT+99] Remzi H. Arpaci-Dusseau, Eric Anderson, Noah Treuhaft, David E. Culler, Joseph M. Hellerstein, David A. Patterson, and Katherine Yelick. Cluster I/O with River: Making the Fast Case Common. In *Sixth Workshop on I/O in Parallel and Distributed Systems (IOPADS '99)*, pages 10–22, Atlanta, May 1999.

[ACPS96] S. Adali, K. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of the ACM SIGMOD Int. Conf.*, Montreal, Canada, 1996.

[AFTU96] Laurent Amsaleg, Michael J. Franklin, Anthony Tomasic, and Tolga Urhan. Scrambling Query Plans to Cope With Unexpected Delays. In *4th International Conference on Parallel and Distributed Information Systems (PDIS)*, Miami Beach, December 1996.

[AH00] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Dallas, 2000.

[Aok99] Paul M. Aoki. How to Avoid Building DataBlades(r) That Know the Value of Everything and the Cost of Nothing. In *11th International Conference on Scientific and Statistical Database Management*, Cleveland, July 1999.

[AZ96] Gennady Antoshenkov and Mohamed Ziauddin. Query Processing and Optimization in Oracle Rdb. *VLDB Journal*, 5(4):229–237, 1996.

[BDF+97] Daniel Barbara, William DuMouchel, Christos Faloutsos, Peter J. Haas, Joseph M. Hellerstein, Yannis E. Ioannidis, H. V. Jagadish, Theodore Johnson, Raymond T. Ng, Viswanath Poosala, Kenneth A. Ross, and Kenneth C. Sevcik. The New Jersey Data Reduction Report. *IEEE Data Engineering Bulletin*, 20(4), December 1997.

[BO99] J. Boulos and K. Ono. Cost Estimation of User-Defined Methods in Object-Relational Database Systems. *SIGMOD Record*, 28(3):22–28, September 1999.

[Coh98] William W. Cohen Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, Seattle, 1998*, pages 201–212.

[CR94] Chung-Min Chen and Nick Roussopoulos. Adaptive selectivity estimation using query feedback. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994*, pages 161–172. ACM Press, 1994.

[DH00] Amol Deshpande and Joseph M. Hellerstein. Decoupled query optimization in federated databases. Technical report, University of California, Berkeley, 2000.

[EGH99] Deborah Estrin, Ramesh Govindan, and John Heidemann. Scalable coordination in sensor networks. In *Proc. of ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*, Seattle, August 1999.

[GC94] G. Graefe and R. Cole. Optimization of Dynamic Query Evaluation Plans. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Minneapolis, 1994.

[GW89] Goetz Graefe and Karen Ward. Dynamic query evaluation plans. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989*, pages 358–366. ACM Press, 1989.

[HAC+99] Joseph M. Hellerstein, Ron Avnur, Andy Chou, Christian Hidber, Chris Olston, Vijayshankar Raman, and Peter J. Haas Tali Roth. Interactive Data Analysis: The Control Project. *IEEE Computer*, 32(8):51–59, August 1999.

[HH99] Peter J. Haas and Joseph M. Hellerstein. Ripple Joins for Online Aggregation. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 287–298, Philadelphia, 1999.

[HHW97] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online Aggregation. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Tucson, 1997.

[HP88] Waqar Hasan and Hamid Pirahesh. Query Rewrite Optimization in Starburst. Research Report RJ 6367 , IBM Almaden Research Center, August 1988.

[HSC99] Joseph M. Hellerstein, Michael Stonebraker, and Rick Caccia. Open, independent enterprise data integration. *IEEE Data Engineering Bulletin*, 22(1), March 1999.

[IFF+99] Zachary G. Ives, Daniela Florescu, Marc Fiedman, Alon Levy, and Daniel S. Weld. An adaptive query execution system for data integration. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Philadelphia, 1999.

[INSS92] Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. Parametric query optimization. In Li-Yan Yuan, editor, *18th International Conference on Very Large Data Bases, August 23-27, 1992, Vancouver, Canada, Proceedings*, pages 103–114. Morgan Kaufmann, 1992.

[KD98] Navin Kabra and David J. DeWitt. Efficient Mid-Query Reoptimization of Sub-Optimal Query Execution Plans. In *Proc. ACM-SIGMOD International Conference on Management of Data*, pages 106–117, Seattle, 1998.

[KKP99] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Mobile networking for smart dust. In *Proc. of ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*, Seattle, August 1999.

[KRB99] Joanna Kulik, Wendi Rabiner, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proc. of ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*, Seattle, August 1999.

[LP97] L. Liu and C. Pu. A dynamic query scheduling framework for distributed and evolving information systems. In *The IEEE Int. Conf. on Distributed Computing Systems (ICDCS-17)*, Baltimore, 1997.

[Met97] Rodney Van Meter. Observing the Effects of Multi-Zone Disks. In *Proceedings of the Usenix 1997 Technical Conference*, Anaheim, January 1997.

[Mit97] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[NKT88] Masaya Nakayama, Masaru Kitsuregawa, and Mikio Takagi. Hash-partitioned join method using dynamic destaging strategy. In François Bancilhon and David J. DeWitt, editors, *Fourteenth International Conference on Very Large Data Bases, August 29 - September 1, 1988, Los Angeles, California, USA, Proceedings*, pages 468–478. Morgan Kaufmann, 1988.

[ONK+96] F. Ozcan, S. Nural, P. Koksal, C. Evrendilek, and A. Dogac. Dynamic query optimization on a distributed object management platform. In *Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1996.

[PCL93a] HweeHwa Pang, Michael J. Carey, and Miron Livny. Memory-adaptive external sorting. In Rakesh Agrawal, Seán Baker, and David A. Bell, editors, *19th International Conference on Very Large Data Bases, August 24-27, 1993, Dublin, Ireland, Proceedings*, pages 618–629. Morgan Kaufmann, 1993.

[PCL93b] HweeHwa Pang, Michael J. Carey, and Miron Livny. Partially preemptive hash joins. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, pages 59–68. ACM Press, 1993.

[SAC+79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In Philip A. Bernstein, editor, *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, May 30 - June 1*, pages 23–34. ACM, 1979.

[SAP+96] Michael Stonebraker, Paul M. Aoki, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A Wide-Area Distributed Database System. *VLDB Journal*, 5(1):48–63, January 1996.

[Son98] Eduardo D. Sontag. *Mathematical Control Theory: Deterministic Finite-Dimensional Systems, Second Edition*. Number 6 in Texts in Applied Mathematics. Springer-Verlag, New York, 1998.

[SWK76] M.R. Stonebraker, E. Wong, and P. Kreps. The design and implementation of INGRES. *ACM Transactions on Database Systems*, 1(3):189–222, September 1976.

[TTC+90] G. Thomas, G. Thompson, C. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, and B. Hartman. Heterogeneous distributed database systems for product use. *ACM Computing Surveys*, 22(3), 1990.

[UF00] Tolga Urhan and Michael Franklin. XJoin: A Reactively-Scheduled Pipelined Join Operator. *IEEE Data Engineering Bulletin*, 2000. In this issue.

[UFA98] Tolga Urhan, Michael Franklin, and Laurent Amsaleg. Cost-Based Query Scrambling for Initial Delays. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Seattle, June 1998.

[WA91] A. N. Wilschut and P. M. G. Apers. Dataflow Query Execution in a Parallel Main-Memory Environment. In *Proc. First International Conference on Parallel and Distributed Info. Sys. (PDIS)*, pages 68–77, 1991.

[WW94] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proc. of the First Symposium on Operating Systems Design and Implementation (OSDI '94)*, pages 1–11, Monterey, CA, November 1994. USENIX Assoc.

[ZG90] Hansjörg Zeller and Jim Gray. An adaptive hash join algorithm for multiuser environments. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 186–197. Morgan Kaufmann, 1990.

[ZL97] Weiye Zhang and Per-Åke Larson. Dynamic memory adjustment for external mergesort. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 376–385. Morgan Kaufmann, 1997.