The Data Ring: Community Content Sharing

Serge Abiteboul*
INRIA-Futurs & Univ. Paris 11
firstname.lastname@inria.fr

Neoklis Polyzotis†
Univ. of California Santa Cruz
alkis@cs.ucsc.edu

ABSTRACT

Information ubiquity has created a large crowd of users (most notably scientists), who could employ DBMS technology to process and share their data more effectively. Still, this user base prefers to keep its data in files that can be easily managed by applications such as spreadsheets, rather than deal with the complexity and rigidity of modern database systems.

In this paper, we propose a vision for enabling non-experts, such as scientists, to build *content sharing communities* in a true database fashion: declaratively. The proposed infrastructure, called the *data ring*, enables users to manage and share their data with minimal effort; the user points to the data that should be shared, and the data ring becomes responsible for automatically indexing the data (to make it accessible), replicating it (for availability), and reorganizing its physical storage (for better query performance). We outline the salient features of our proposal, and outline research challenges that must be addressed in order to realize this vision.

1. INTRODUCTION

Imagine a community of scientists who collect experimental data. To infer interesting information out of the collected data, a scientist clearly needs the ability to query it. Moreover, it is common within such communities to share data, in order to promote more research and enable the quick dissemination of information. (SWISSPROT[1] and the Genome Browser[2] are characteristic examples of this scenario.) These two elements of querying and sharing information can be found in other contexts as well, especially with the advent of the Web and the creation of user communities, such as Flickr. This evidence indicates an emerging trend towards building what we name *content sharing communities*: groups of users that wish to share and query information in some specific domain.

A database expert would probably suggest the use of a distributed DBMS as the solution to this problem. In principle, the distributed DBMS would enable users to query their information declaratively (no more programming with Perl scripts!), while allowing them to share it transparently. In practice, however, the picture is completely different and users avoid using DBMS technology due to its complexity. Consider for instance the conceptually simple operation of loading data in a DBMS. This involves several tasks, such as, defining a schema, tuning the database, collecting statistics, etc., that are not trivial for non-experts. Of course, this operation is dwarfed in complexity by the next logical step, which is linking the local databases in a distributed or federated system. And even if a database expert is found to perform all these time consuming tasks, users may still find a conventional database system too rigid when it comes to its control over the data. Essentially, a DBMS has to import data before it can query it, and export it before the data can be processed with other software; users (and especially scientists), on the other hand, prefer to store their data in file systems, where it is easily accessible by 3rd party programs.

We believe that this user base forms a formidable challenge for database researchers and a chance to bring database systems to the "masses". In this direction, we propose the *data ring* that can be seen as a network analogue of a database or a content warehouse. The vision is to build a P2P middleware system that can be used by a community of non-experts, such as scientists, to build content sharing communities in a declarative fashion. Essentially, a peer joins a data ring by specifying which files (or services in general) are to be shared, without having to specify a schema for the data, load it in a store, create any indices on it, or specify anything complex regarding its distribution. The data ring enables users to perform declarative queries over the aggregated data, and becomes responsible for reorganizing the physical storage of data and for controlling its distribution. Thus a primary focus of the data ring is *simplicity of use*. The inherent diversity and complexity is hidden behind a simple and unified interface for publishing, querying, monitoring, integrating, and updating the available information. Besides clicking to select resources, the use of a ring resource should be no more complicated than the use of the same resource in a familiar local system.

*The work of S. Abiteboul was partially supported by the ANR Grant webContent.
†Part of the work was performed while the author was visiting INRIA, France.
[1] `http://www.expasy.org/sprot/`
[2] `http://genome.ucsc.edu`

3^{rd} *Biennial Conference on Innovative Data Systems Research (CIDR)* January 7-10, 2007, Asilomar, California, USA.

As previously stressed, data rings target non-expert users, implying that the deployment of the ring and its administration should be almost effort-less. The "pay-as-you-go" philosophy (promoted by "dataspaces" [21]) is replaced here by "pay-only-by-providing-content"; the physical organization is (almost) for free. A first essential facet of the data ring proposal is thus the use of fully automatic and distributed data administration, both for data residing in file systems (a most common situation) and for more controlled systems, such as databases. Access structures, e.g., materialized views or indexes, are automatically introduced by the system when needed. Also, the system may choose to "activate" some static file-systems data, e.g. a large collection of RSS seeds, to assign its management to a more efficient system such as a relational database. The relational system is then in charge of managing the collection and its efficient querying and monitoring.

To facilitate the exploitation of the ring by non-experts, we propose the use of declarative languages in the style of SQL/QBE, rather than languages such as Java or Ajax that require programming skills. More precisely, the data ring adopts the (reasonably well-accepted) standard XML for data exchange. Most content, from emails to relational data including multimedia and meta data, can easily be mapped to XML. This said, we see the vision of a data source as a set of XML documents waiting to be queried in a Xquery-style as way too limited. A data source (as in deductive databases) may include intensional data, e.g. mediate data from different external Web services. A data source (as in active databases) may provide active features, e.g., monitor changes in a data source or push answers of query subscriptions. In short, the data ring promotes a rich vision of content sharing based on combining extensional (stored) data with intensional, dynamic or active data.

The richness of these concepts may seem in contradiction with the original goal to make the data ring usable by non-experts. However, the data ring thesis is that a high-level declarative language will enable users to integrate their data effectively, and to make it readily accessible by other non-experts. More precisely:

(Thesis-logic) Non-experts, as well as application programmers, should use a logical language to declaratively combine information from push or pull Web services, databases, files, and all other resources managed by the data ring.

We propose to use ActiveXML [10, 3] for this logical language, namely a language based on XML documents with embedded Web service calls. Of course, a most important issue becomes query evaluation and optimization in this setting. For that, we need an algebraic optimization framework. More precisely, we believe:

(Thesis-algebraic) An algebraic language over distributed XML streams[3] should be used for describing distributed query plans interleaving query optimization, query evaluation, and possibly, error recovery and transaction processing, that can be exchanged between peers.

We stress that our goal is *not to market* ActiveXML specifically, but to argue, more generally, that further efforts should be devoted towards languages that do address what we believe are most important issues: the quest for a language for defining data distribution and data exchange, the quest for a language for describing distributed query plans, and the study of optimization techniques in this context.

In this paper, we discuss the salient features of the data ring proposal, insisting on its most novel and challenging aspects. The remainder of the paper is organized as follows. Section 2 presents a cursory review of related work. Section 3 presents a high-level overview of the data ring proposal, while the following sections focus on specific issues, namely, self-administration (Section 4), data activation for file sources (Section 5), query optimization/evaluation (Section 6), and logical and algebraic languages for data distribution (Section 7). Section 8 concludes the paper.

2. RELATED WORK

In principle, a data sharing community can be maintained as a distributed database system using existing commercial solutions. Under this model, each user runs a local DBMS on the data that he/she wishes to publish, and all the participating databases are connected in a loose federation that forms a virtual global repository. As mentioned earlier, however, the main issue is the increased complexity of setting up and tuning such a system. It should be noted that modern database systems are equipped with several "advisors" [13, 46, 19], that can assist users in the difficult task of system maintenance. Such tools, however, implicitly assume that the user has some experience with database systems and can thus make informed decisions based on the recommendations that the advisors generate. Thus, the target audience is primarily system administrators rather than non-expert users. Moreover, the proposed techniques focus primarily on the tuning of a centralized system, and it is not clear if they can be extended to the equally important problem of tuning the distributed system.

Peer-to-Peer (P2P) file sharing systems, such as, Gnutella or eDonkey, offer a light-weight alternative to the complexity of a full blown database system. The main shortcoming, however, is that their query functionality is limited to locating files based on name matching, whereas users in data sharing communities are primarily interested in locating data based on content. Moreover, P2P file-sharing systems typically rely on query flooding as the main query processing mechanism, which does not scale well for complex queries and high querying rates [23].

Several research projects in academia have investigated the development of P2P database systems as the platform for supporting data sharing communities. In particular, previous works have investigated issues related to data integration [45, 34], system design [17], and processing of complex queries [29, 28, 22, 23] over P2P networks. These aspects are undoubtedly important in the development of tools for the creation of data sharing communities. As noted earlier, however, an important issue is the development of mechanisms for self-administration that allow the P2P system to operate efficiently in an autonomous fashion. In this direction, earlier studies [24, 18] have investigated self-tuning techniques for the DHT [44, 40] substrate of P2P systems. The proposed techniques monitor the *get/put* requests submitted to the DHT and adaptively change its configuration in order to optimize for the exchange of network messages. These works are clearly an important step in the realization

[3]An XML stream may be used to produce the trees in an XML forest and different pieces of a large XML tree.

of self-managed distributed systems. It is equally important, of course, to explore self-tuning mechanisms for the upper layers of the system, that provide the support for complex, declarative queries over the distributed data.

The data ring proposal borrows ideas from several previous works on distributed data management. From P2P content warehouse [1], we adopt the idea of using semantic tools to enrich the data at our disposal, e.g. by discovering links between pieces of data. From [21], we borrow the vision of a *dataspace* where data is integrated. Optimization techniques can be used from the existing large body of works on distributed query optimization, such as, PIER [29, 28] and ActiveXML [10]. In particular, our proposal resembles the goals of PIER [29] in that it introduces a middleware system for running distributed query processing applications. The main difference, however, is that we focus on a system to be used by non-experts, insisting on declarative management wherever possible.

Technology already developed for other systems may prove useful for data rings: structured p2p networks such as Chord or Pastry, XML repositories such as Xyleme, DBMonet, or eXist, file sharing systems such as BitTorrent or Kazaa, distributed storage systems such as OceanStore or Google File System, content delivery network such as Coral or Akamai, multicast systems such as Bullet or Avalanche, Pub/Sub system such as Scribe or Hyper. Works on data integration (see e.g., [36, 35, 30]), warehousing [12] and distributed database systems [37] are also relevant. The data ring proposal treats all these works mostly as enabling technologies, and less as areas for innovation. (We prefer to leave this task to the respective experts.)

Finally, the present paper has been strongly influenced by on-going works with Ioana Manolescu and Tova Milo around ActiveXML, the KadoP P2P indexing system [6, 5] and the Edos content distribution system [20].

3. THE DATA RING

In this section, we present a high-level overview of the data ring and its conceptual architecture. Abstractly, a data ring is formed by a collection of peers, where each peer exports a set of resources (services and data). The collaborating peers are autonomous, heterogeneous, and their capabilities may greatly vary, e.g., from a sensor to a large database. Moreover, their exported resources may be quite diverse, e.g., some peers enable access to data resources, such as, a phone book or a genome bank, while others offer computational resources, such as, an ontology-based classifier, or a gene matching library.

We use P to denote a peer in the data ring, and $r@P$ to denote a resource r that peer P exports. A resource r can be either a data item or a service that P hosts. As an example, a scientist (with peer name "Joe") that joins a data ring may export data files "exper1.txt@Joe" and "exper2.xls@Joe", as well as a service "run_SQL()@Joe" that enables the execution of SQL queries over a locally hosted relational DBMS. The notion of exporting means that P is willing to share r with the other participants of the data ring[4]. Thus, other peers gain access to the contents of r if it is a data item, or are able to invoke r at P if r is a service.

As mentioned earlier, we advocate a data model for services and data that is based on ActiveXML. More specifically, we assume that each data item is represented with an XML view, while services are specified in the Web Services Description Language [15] (WSDL). Two observations are in order. First, we employ XML solely as a model for data integration, and do not insist that each peer uses XML as its native data model. We assume that the latter is determined on a per-peer basis, depending on the type of exported data. Second, we advocate an enriched XML model that can capture both extensional and intensional data. We discuss this point in Section 7, where we argue that the default XML model is too rigid and cannot describe effectively the information in a distributed data store.

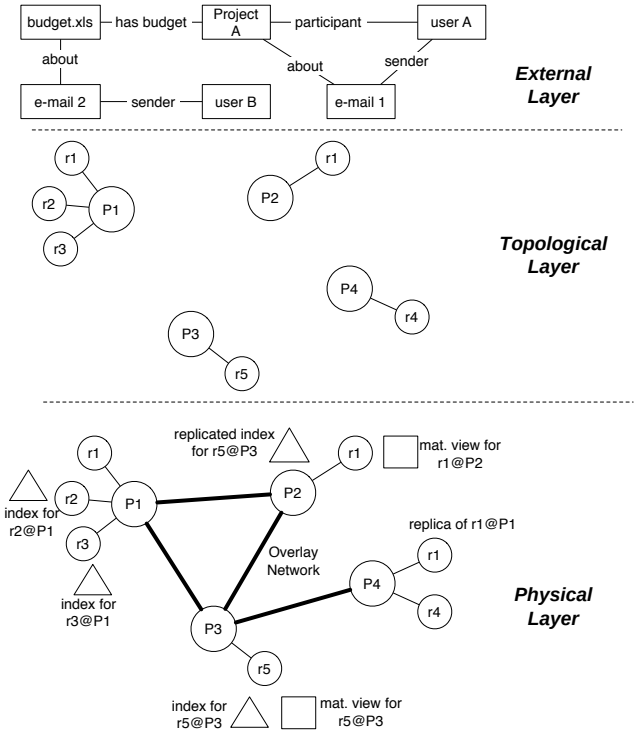

Figure 1: Information abstraction in a data ring.

Peers query the information in the data ring using a suitable XML query language. Given an XML query issued at some peer, the data ring translates it to a set of native queries over the published resources. Continuing with our previous example, a query issued over "exper1.txt@Joe" will be translated to a query over a suitable file algebra, while a query on "exper2.xls@Joe" will be translated to operations over the spreadsheet; similarly, a query over the relational database that "Joe" exports will be translated to an SQL query that will be pushed to the service "run_SQL@Joe". In a sense, therefore, the data ring borrows several features from mediation systems [32]. A key issue of course is that of efficiency. For instance, while a relational database can enable optimized access to its contents, file sources (e.g., the text file or the spreadsheet in our example) only provide limited access methods for query evaluation. We revisit this issue in Section 5, where we introduce the novel concept of self activation for files.

Following the established principles of database systems,

[4] P also provides the permissions for accessing r. We ignore this aspect for simplicity.

a data ring organizes information at three levels of abstraction: the external layer, the topological layer, and the physical layer. (See Figure 1). The following summarizes the functionality of each part.

Topological Layer The topological layer contains the logical description of the information managed by the data ring, i.e., the set of participating peers and the resources that they export. While the topological layer does not expose the overlay network that is used to realize the distributed system, it does include information on the location of resources, i.e., the $@P$ specification. (This is why we use "topological" instead of the more common term "logical" for the identification of the layer.) Hence, a valid query is "Retrieve Mike's phone number in David's PDA, e.g., using the service contacts@DavidPDA". The topological layer also supports generic resources that are offered by several peers in the ring. We denote such a resource as $r@any$, where *any* is a reserved location name. The idea is that the data ring may choose to replicate a resource depending on its querying patterns in order to increase its availability or ensure its efficient access. By using $r@any$, a query leaves it to the data ring to resolve the best method for accessing the particular resource.

The topological layer provides declarative query services, in the same way that the logical model of a relational database supports SQL queries. A peer can thus submit an ad-hoc query against the registered resources, monitor a resource for changes, or open a continuous query against a stream resource (e.g., sensor readings, or an RSS stream). We also assume that the topological layer records the lineage and uncertainty properties of the data, and supports their integration in the query language. Following our previous example, a valid query can thus be "Retrieve Mike's phone number along with the source of this information, the date it was acquired, the name of the server where it was found, and the estimated confidence in it". We discuss the specifics and requirements for the query language further.

External Layer As described previously, the topological layer manages different kinds of information: traditional data (as in relational systems), content (mails, letters, reports, etc.), metadata, as well as domain specific knowledge (e.g., ontologies) needed to interpret data and metadata. Since this level of detail can still be daunting for a large class of users, we enable the generation and maintenance of semantically richer data models in the *external layer*. For instance, the external layer can house the data models that are proposed in dataspaces. In this fashion, a user will observe concepts and relationships between concepts, and will pose semantically rich queries such as "what is the name of John Doe's company", typically using some simple syntax or some forms-based interface. We expect such queries to be realized by combining declarative queries over the topological layer, driven by the semantic information in the external data model.

Physical Layer The physical layer supports the evaluation of distributed query plans over the registered resources. It comprises a physical model and language for distributed query evaluation, as well as physical structures for managing the registered data, such as, DHTs, indexes, caches, and/or replicas of data and services. A key component of the physical layer is the data catalog that stores meta-data on the published resources. The meta-data includes the location of resources, information on their contents/capabilities, statistics on data and workload distribution, and in general any information that is pertinent to the evaluation of distributed queries. In general, we distinguish two key features of the physical layer: (a) it is completely distributed, and (b) it is self-administered. As we discuss in Section 4, these two elements combined introduce novel challenges in the design of autonomic (i.e., self-tuning) systems.

Overall, the aforementioned organization follows the basic principles of database systems, where the data is managed at three separate layers (external, logical, and physical) in order to ensure logical and physical independence. Our conceptual architecture is motivated by the same goals. Also similar to relational systems, we view the topological and physical layer as being absolutely necessary for realizing a data ring and we thus focus on these two layers for the remainder of this paper. This is not to say that we find the external layer to be less important; on the contrary! We hope to find some synergy here with works in dataspace support platforms [21], which examine data models similar to the ones that we envision for the external layer.

In the following sections, we discuss key issues towards the realization of our proposal, in particular with respect to the topological and physical layer of the data ring. We want to stress that our goal is to outline major research challenges involved in implementing a data ring, rather than to propose concrete solutions to specific problems. We are also well aware that our proposal is likely to keep several researchers busy for a while before it can be declared realized.

4. AUTONOMIC ADMINISTRATION

As mentioned from the beginning, our vision is to enable non-experts, such as scientists, to create content sharing communities with minimal overhead. In turn, this clearly implies that the users should be alleviated from any administration duties, and hence the ring should function without the intervention of an administrator or any central authority. Moreover, the distributed nature of data rings suggests that this autonomic operation must be completely decentralized.

> The administration of the data ring is fully autonomic and distributed.

Our goal of autonomic administration clearly targets the physical layer of the data ring. Similar to previous proposals on autonomic computing, we envision the following functionality:

Self monitoring The system must monitor itself, gathering statistics and logs. This information can be used to derive predictive models of system usage that can be used for tuning (next point). Also, the logs are useful for error diagnosis, billing, etc.

Self tuning The system must reorganize itself to achieve better performance by enriching the physical layer with the appropriate access structures (e.g., indices, caching, replication, materialized views). Observe that this may involve both some local decision (the addition of some local index) and possibly some cooperation between peers (e.g., the replication of a service). We note that there are two-levels of tuning: (a) a data resource may self-tune its organization at the owning peer, e.g., by building local indices or materialized views, (b) the data ring may self-tune its organization across peers, e.g., by replicating resources, caching them locally, or increasing the level of detail of its metadata.

Self healing The system must help the peers recover from errors, e.g., by suggesting a substitute for some failing Web service, or by taking corrective actions when some process fails. We revisit this issue in Section 6, where we discuss in more detail the integration of error recovery in the query evaluation process.

The notion of self-administration (and most importantly self-tuning) has been already explored in the context of relational databases [8, 11, 14]. Our proposal may thus initially sound as a mere rehashing of old ideas. However, the issue of autonomic computing acquires unique flavors in the context of data rings, and thus requires new approaches. More precisely, one can distinguish the following novel features:

System integration In the data ring, self-administration is not only a cool gadget that sits on the side of the system; it becomes an absolute necessity. It is thus necessary to rethink the architecture of the database system to incorporate self-administration as a core component, and not as an add-on.

Distribution Existing solutions on self-administration typically target a centralized scenario, e.g., the selection of physical access structures in a server based on the workload. In the case of data rings, the administration becomes totally distributed. For instance, physical structures can be created in several peers, and there is no centralized authority to coordinate the tuning.

On-line tuning Existing self-tuning solutions typically adopt an off-line approach, assuming that the tuning occurs separately from normal database operation. As a result, the tuning process is generally expensive and cannot be executed too often. This makes off-line tuning unattractive for a data ring, since the system may need a frequent re-tune due to unpredictable changes in the user generated workload. A more suitable option is *on-line tuning*, where the system monitors its performance continuously and re-organizes its configuration based on the latest traits of the query workload. This is a major paradigm shift that we started exploring in [42, 43].

Proactive tuning One can classify existing self-tuning techniques as reactive, since they are invoked after a change in the workload is detected. To deal with the highly volatile nature of a P2P system, however, a data ring has to be *proactive*. For instance, the system may decide (in a proactive manner) to replicate a file based on the prediction that it is going to be heavily used, before this actually happens.

One can argue that we set the bar too high. After all, the simpler problem of zero-administration for centralized relational systems still remains an elusive goal. Is there any reason to believe that the self-administration of a data ring is feasible? We believe that there is no alternative! We also believe that the power of parallelism that comes with a P2P system may compensate limitations of the technology. For instance, with many machines at our disposal, we can run in parallel more costly tuning algorithms that can meet some (or hopefully all) of the requirements that we have outlined.

5. DATA ACTIVATION FOR FILES

One of our basic assumptions is that a significant portion of the available data resides in file systems that do little effort to optimize their access or their updates. Still, we envision that the data ring should provide efficient declarative query capabilities over such sources, by means of *data self-activation*. To illustrate this idea, let us consider a music catalog that has been published as a file in the ring and is queried heavily. Again, we assume that the catalog is presented as an XML view to the users, and queries are translated to a suitable file algebra. Clearly, the efficiency and reliability of accessing this data is constrained by its physical organization in its owning peer. Since that physical organization cannot be changed, the data ring may instead decide, based on self-statistics, to reorganize the catalog in redundant physical structures that are more efficient to access. For instance, it may decide to replicate the catalog (or some part thereof) on a different peer. Another option is to replicate the catalog in a different physical system, e.g. an indexed relational database, in order to answer efficiently queries that access some specific attributes.

> The data ring monitors the querying patterns over files and automatically reorganizes them to optimize their access.

Typically, a specific file resource $r@P$ starts its life in the ring as "static", i.e., being manipulated through file system calls (essentially, block-based reads and updates). Based on the usage patterns, the data ring may decide to activate some portion of r in order to increase its access efficiency. There are a variety of means to do so from very light (just providing several replicas of a file), to very heavy (reorganizing part of the data in a relational database), to intermediate ones (e.g., indexing the strings in a plain text file). The key idea of course is that access to the activated portions is now more efficient, while inactivated portions are still accessed through file system calls. Clearly, this hints at the development of a suitable cost model that determines when activation can be beneficial, which in turn implies the existence of query algebras for file based sources. Previous studies [4, 16] have looked at the theoretical foundations of this problem, but have not examined the associated systems issues, i.e., optimization techniques, cost model for physical operators, and optimization statistics over files. Finally, an interesting research direction is the integration of self activation with the file system itself, e.g., by exploiting the functionality of active disks [41]. In this fashion, the data ring can optimize the physical file organization at the storage level in addition to materializing secondary access structures. We believe that this topic provides fertile ground for building collaborations with the data storage community [25].

As a last remark, we stress that the data ring should be extensible in terms of the supported file types. Here, we adopt the "pay-as-you-go" principle of dataspace systems and require some minimal amount of work from the user in order to incorporate a new file type in the topological and physical layers. More precisely, the user has to register a parser module (written in his/her language of choice) that enables serial access to the XML view describing the contents of a file of this type. This is a relatively easy task, since a sequential parser has a simple interface and we assume that the user is already familiar with the structure of the new files. This parsed content represents the static version of the file, which may then be activated by the data ring depending on the query patterns.

6. QUERY EVALUATION

In classical (distributed) data management systems, the query evaluation process proceeds in distinct stages: the query is first optimized; it is then handed over to a coordinator site that is in charge of orchestrating the sub-queries; sub-queries are finally executed at different sites and their results typically combined by the coordinator to generate the final answer. This clean separation, however, is not feasible in a data ring where there is no notion of global state and the environment is highly volatile. Consider, for instance, a query Q that accesses some resource $r@P$, and assume that P leaves the data ring when Q starts being evaluated. In a classical setting, this would trigger an error since P is no longer available (its sub-query failed). It might be the case, however, that $r@P$ has been replicated in another node P' and Q can thus be rerouted on-the-fly to P'. Even if $r@P'$ is an old replica of $r@P$, the user may still be satisfied with getting some results (with specific uncertainty and lineage properties) rather than no results at all. Of course, such decisions must be driven by the query optimizer which will evaluate the trade-offs (and the feasibility) in recovering from specific errors. Hence, we arrive at the following requirement for query evaluation in a data ring:

> Query optimization, query execution, and error recovery are unified.

Previous works [9, 31, 33] have explored the idea of mixing query optimization and execution in the context of relational database systems, while there has been some initial work on XML query evaluation over a distributed system [38]. The key difference here is the integration of error recovery in the query evaluation process, and the massively distributed setting that results from the P2P layer. In particular, the last point raises several interesting issues with respect to the optimization of declarative queries, as the search space of an optimizer grows tremendously and performance statistics are not readily available. Hence, it seems necessary to integrate optimization and execution in a continuous process: the system starts with a rough physical plan, and gradually refines it as more of the query is evaluated and more statistics are gathered. The optimization of declarative queries over massively distributed systems remains an open problem [28] and is thus an interesting area for future work.

7. STREAM CALCULUS & ALGEBRA

The success of the relational model essentially comes from the combination of a declarative language (relational calculus), an equivalent relational algebra, and optimization techniques based on rewrite rules. There have been a number of extensions, such as object databases, but the classical pattern (calculus, algebra, rewrite rules) has proved its robustness. It is only natural therefore to adopt it the data ring context. The situation however is fundamentally different (e.g., distributed vs. centralized, semi-structured vs. very structured, Web scale) which requires a complete overhauling of the languages. This is the topic that we discuss in this section. We argue for the need of a calculus and an algebra for distributed data management, and present a number of features that we believe are fundamental for such languages.

We make the following claim and explain it next.

> The logical model should be based on extensional, and intensional XML streams both in pull and push modes, i.e., in the style of ActiveXML.

We believe that, to support effectively the loose integration paradigm of the ring, one essential aspect is the seamless transition between explicit and intentional data. One should not have to distinguish between extensional data (e.g., XML or HTML pages) and intensional data (e.g., access to a relational database provided by a Web service). As an example, consider the query "give me the name and phone number of the CEO of the Gismo company". Answering this query may require first finding the name of that CEO in an XML collection of company synopses, finding the service that exports the phone book of Gismo Inc, and finally calling this service with the name of this CEO. The query can be answered only (a) because we have a logical description of the resources, and (b) because based on that, we have derived a distributed query plan.

ActiveXML was designed to capture such issues. An ActiveXML document is an XML document where certain elements denote embedded calls to Web services. For instance, the company document may contain the CEO phone number as a Web service call. The service calls embedded in the document thus provide intensional data in the sense of deductive databases. Suppose now that the phone number of the CEO changes. The next call to the service will return the updated phone number, which implies that the company document also becomes up-to-date. Thus the embedding of service calls is also capturing active data in the sense of active databases [39]. We note that the use of intensional information is quite standard on the Web, e.g. in Web sites that are realized with PHP-mySQL. It is also common in databases, as in object or deductive databases. The main novelty here is that the intensional data is provided by Web services. Thus the corresponding service calls may be activated by any peer and do not have to be evaluated prior to sending the document.

Henceforth, we assume that every peer exports its resources in the form of ActiveXML documents or Web services. The topological layer thus consists of a set of ActiveXML documents and services and their owning peers. The external layer will be dealing with the semantics of documents, but this aspect will be ignored in this section. A computation will consist in local processing and exchanging

such documents. Articles on ActiveXML as well as the open-source code of an ActiveXML peer may be found from [10].

Before turning to the algebra, we discuss two aspects that are essential to the framework and motivate basing it on XML streams (as in ActiveXML) and not simply on XML documents:

Push vs. Pull In pull mode, a query-service is called to obtain information. In push mode, on the other hand, the service provides a stream of answers, e.g., notifications of certain events of interest. A company document may include such a service to, for instance, obtain the news of the company. Essentially, the push mode captures the notion of subscriptions and is thus very relevant in the context of the Web. Such a subscription feature is also essential for supporting a number of functionalities ranging from monitoring the ring, to synchronization and reconciliation of replicas, or gathering ring statistics.

Recursion The embedded service calls may be seen as views in the spirit of those found at the core of deductive databases. In classical deductive databases, recursion comes from data relationships and recursive queries, such as *ancestor*. In our setting, recursion kicks in similarly and is also found in the query language (e.g., the XPATH // primitive). But more fundamentally, recursion comes from the graph nature of the Web which is also reflected on the processing of Web services in the ring: site1 calls site2 that calls site3 that calls site1, etc. Indeed, the use of recursive query processing techniques in P2P contexts has been recently highlighed in several works in topics as different as message rooting on the Web [27] and error diagnosis in telecom networks [2]. Now, recursive query processing clearly requires the use of streams.

Besides the logical level, our thesis is that a language in the style of ActiveXML should also serve as the basis for the physical model. In particular, the use of streams is unavoidable at this level. As a trivial example, observe how answers are returned by Google or try to send 100K in a standard Web service without obtaining a *timeout*. As shown in a recent work [7], distributed query evaluation and optimization can be naturally captured using ActiveXML algebraic expressions, based on the exchange of distributed query execution plans. The expressions include standard algebraic XML operations and send/receive operators, all over XML streams. Note that these may be seen as particular workflow descriptions of a strong database flavor.

The physical model is based on a distributed algebra over XML streams, i.e., in the style of ActiveXML algebra.

An example will best illustrate this principle. Consider the data described in Figure 2. We use here a visual representation of ActiveXML documents. Peer $P1$ and $P2$ have their own collections of music with metadata described in relations $r1$ and $r2$ respectively. Peer $P1$ knows about t(itles) and s(ingers), whereas $P2$ knows about a(lbum) t(itles) and s(ingers). Peer $P1$ also knows that $P2$ has some music; $P2$ knows that $P3$ (not shown here) has some; $P3$ knows $P4$, etc. The metadata of $P3, P4, P5$ are organized as that of $P1$. The actual texts underneath the tags s, t, at are not shown. Now suppose that $P1$ wants to get the titles of songs by Carla Bruni. Figure 3 shows three different query plans. Each box describes some peer computation. Query plan (a) is the one that would result from an evaluation of the query without optimization, i.e., from applying the pure semantics of ActiveXML. Query plan (b) results from pushing selections, while Query plan (c) is obtained by also optimizing data transfers (cutting some middle peers in data transmissions). One (particularly interesting) rewrite rule is illustrated in Figure 4 which shows the evaluation of a service call at peer $P1$. To perform the evaluation, the external service call at peer $P1$ is replaced by a "receive" node and a remote computation is activated at peer $P2$. This computation proceeds in parallel and sends the results asynchronously to the receive node.

We can make the following observations:

1. Peers $P1$ and $P2$ can already be producing answers, while $P3$ is still optimizing the request it receives, while $P5$ is still not even aware of the query. This illustrates the need for streaming: Peer $P2$ can send answers to $P1$ before obtaining results from other peers.

2. Each peer separately receives a request and is fully in charge of evaluating it. (Some optimization guidelines may be provided as well.) For instance $P2$ receives a query where she cannot really contribute and may decide to cut herself out of it and ask $P3$ to evaluate its part and send the result directly to $P1$.

3. We assumed so far that the peers cooperate to evaluate a query. Think now that the goal is to support a subscription. Then the same plans apply. Suppose a new song of Carla Bruni is entered in $P3$. It is then sent to $P1$ (with Query Plan (c)), and produced as a new answer unless this title has already been returned.

In all cases, a query or a subscription for the songs of Carla Bruni (at the topological layer) is translated to a distributed plan (at the physical layer). Observe that the physical plan is essentially a workflow of Web services (i.e., an ActiveXML document), where the services encapsulate the different plan operators and the respective locations encode the distribution of computation and the flow of data. The main idea therefore is that the complete plan itself (or a portion of it), along with its current state of execution, can be described as an ActiveXML document, which in turn can be exchanged between peers in order to support query optimization and error recovery in a distributed fashion.

Another important element in the Figure 3 is the distinction between local query evaluation (inside each box) that is the responsibility of a local system, perhaps a relational system, and global query evaluation that is the domain of the data ring. The functional architecture of a peer query processor is shown in Figure 5. Note that the processor comprises both a local query optimizer and a ring query optimizer that collaborates with other peers to perform global query optimization. Essentially, this separation leads to physical plans that combine local query processing with distributed evaluation. Clearly, a collaboration between the two systems (local and data ring) is preferable but is unlikely to be widespread in the near future. This implies that the data ring will have to view the local query optimizers

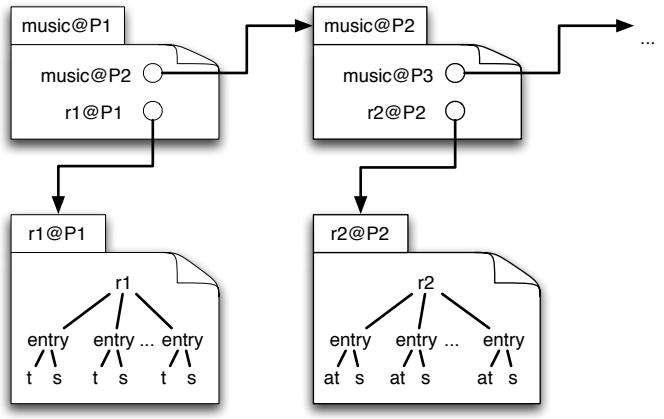

Figure 2: A graphical representation of ActiveXML data.

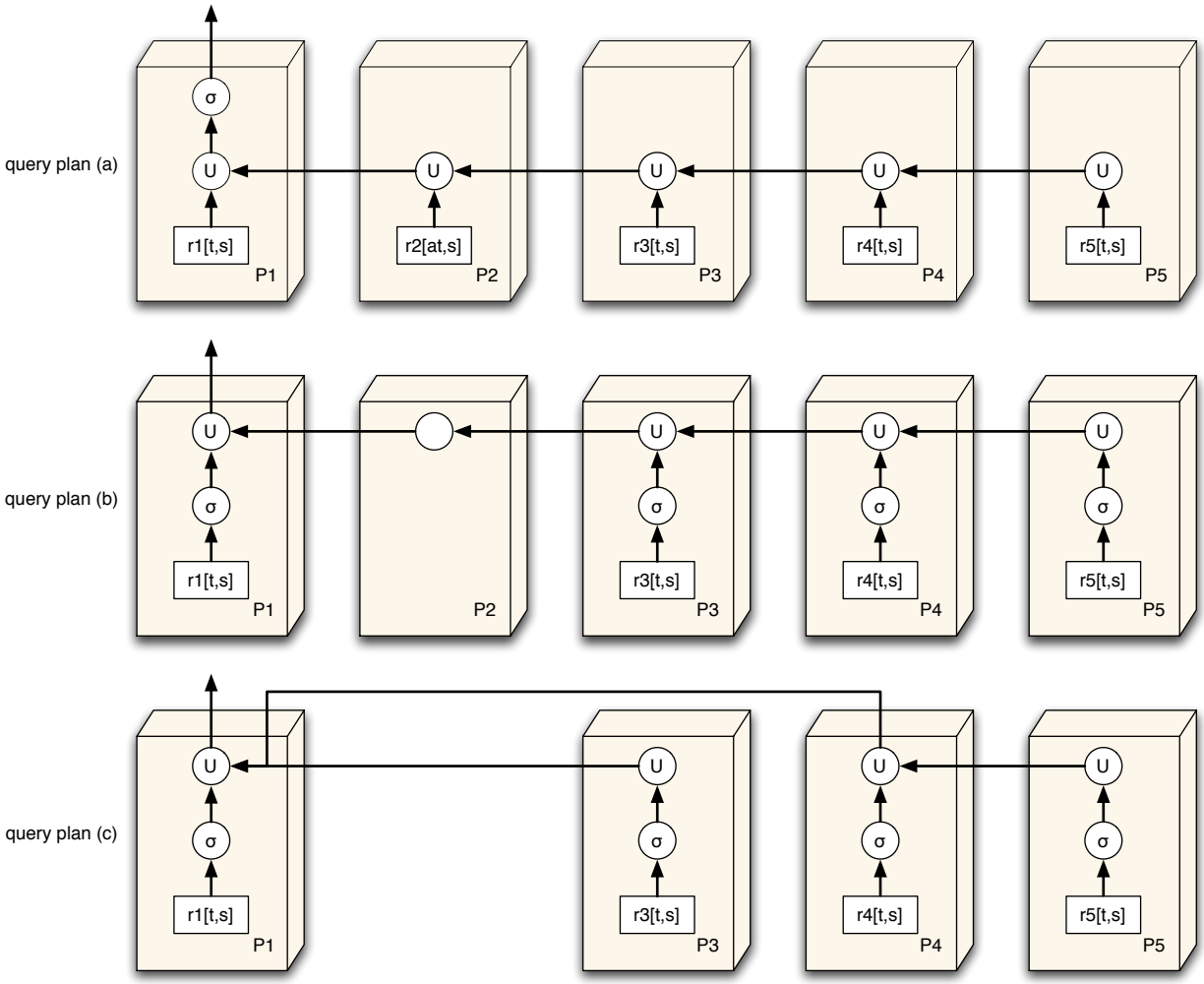

Figure 3: Three equivalent distributed query plans.

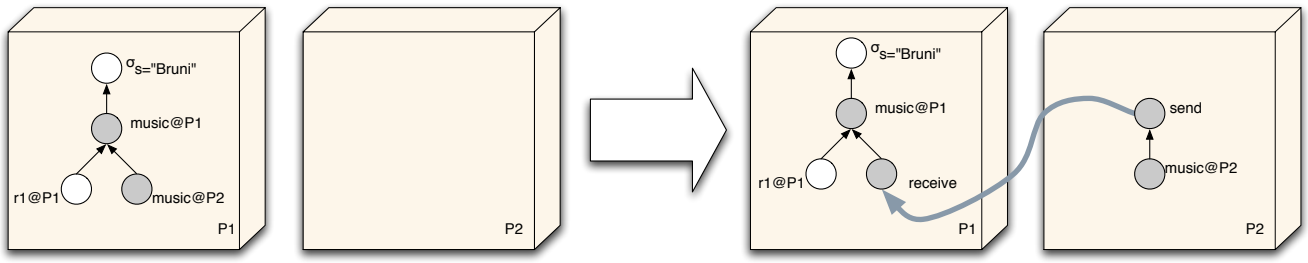

Figure 4: An algebraic rewriting.

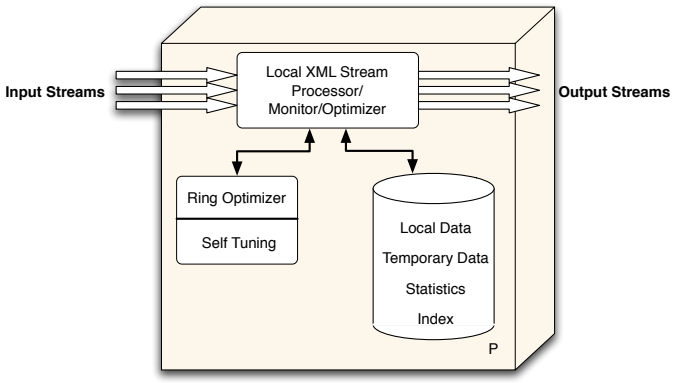

Figure 5: Functional architecture.

as boxes with possibly different querying capabilities, in the same vein as mediation systems [26].

To conclude this section, we want to note again that the goal was not to advertise particular languages but to stress the need for more works in this area. ActiveXML and ActiveXML algebra were used to illustrate aspects that a calculus and an algebra for the ring should emphasize.

8. CONCLUSION

The starting point of this paper is the observation that large communities of users are (and will) more and more share content over the Internet. We proposed the data rings, that are meant to bring the great benefits of database technology to these communities by enabling them to easily create, administer, and exploit shared content. We have outlined the general principles behind data rings, and discussed some research challenges met on the way to the realization of this vision. The realization of data rings will clearly require the collaboration of numerous researchers, and this is precisely one goal of this paper: to entice other researchers to join us in this endeavor.

9. REFERENCES

[1] S. Abiteboul. Managing an XML Warehouse in a P2P Context. In *CAiSE*, pages 4–13, 2003.

[2] S. Abiteboul, Z. Abrams, S. Haar, and T. Milo. Diagnosis of Asynchronous Discrete event systems. Datalog to the rescue! In *ACM Conference on Principle of Database Systems*, 2005.

[3] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. Lazy Query Evaluation for Active XML. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 227–238, 2004.

[4] S. Abiteboul, S. Cluet, and T. Milo. A logical view of structured files. *The VLDB Journal*, 7(2):96–114, 1998.

[5] S. Abiteboul, I. Manolescu, and N. Preda. Constructing and Querying Peer-to-Peer Warehouses of XML Resources. In *Proceedings of the 21st Intl. Conf. on Data Engineering*, pages 1122–1123, 2005.

[6] S. Abiteboul, I. Manolescu, and N. Preda. Sharing Content in Structured P2P Networks. In *BDA*, 2005.

[7] S. Abiteboul, I. Manolescu, and E. Taropa. A Framework for Distributed XML Data Management. In *10th International Conference on Extending Database Technology*, pages 1049–1058, 2006.

[8] S. Agrawal, S. Chaudhuri, and V.Narasayya. Automated Selection of Materialized Views and Indexes for SQL Databases. In *Proceedings of the 26th Intl. Conf. on Very Large Data Bases*, pages 496–505, 2000.

[9] R. Avnur and J.M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pages 261–272, 2000.

[10] ActiveXML Web Site. `http://activexml.net`.

[11] N. Bruno and S. Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005.

[12] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Rec.*, 26(1):65–74, 1997.

[13] S. Chaudhuri and V. R. Narasayya. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceedings of the 23rd Intl. Conf. on Very Large Data Bases*, pages 146–155, 1997.

[14] S. Chaudhuri and G. Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In *Proceedings of the 26th Intl. Conf. on Very Large Data Bases*, pages 1–10, 2000.

[15] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. (Available from

http://www.w3.org/TR/wsdl), 2001.

[16] M. P. Consens and T. Milo. Optimizing queries on files. In *Proceedings of the 1994 ACM SIGMOD Intl. Conf. on Management of Data*, pages 301–312, 1994.

[17] B. F. Cooper and H. Garcia-Molina. SIL: Modeling and Measuring Scalable Peer-to-Peer Search Networks. In *DBISP2P*, pages 2–16, 2003.

[18] B. F. Cooper and H. Garcia-Molina. Ad Hoc, self-supervising peer-to-peer search networks. *ACM Trans. Inf. Syst.*, 23(2):169–200, 2005.

[19] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zaït, and M. Ziauddin. Automatic SQL Tuning in Oracle 10g. In *Proceedings of the 30th Intl. Conf. on Very Large Data Bases*, pages 1098–1109, 2004.

[20] The EDOS project. http://www.edos-project.org.

[21] M. J. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, 34(4):27–33, 2005.

[22] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating Data Sources in Large Distributed Systems. In *Proceedings of the 29th Intl. Conf. on Very Large Data Bases*, 2003.

[23] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Processing Queries in a Large Peer-to-Peer System. In *CAiSE 2003 : international conference on advanced information systems engineering* , pages 273–288, 2003.

[24] P. Ganesan, Q. Sun, and H. Garcia-Molina. Adlib: A Self-Tuning Index for Dynamic Peer-to-Peer Systems. In *Proceedings of the 21st Intl. Conf. on Data Engineering*, pages 256–257. IEEE Computer Society, 2005.

[25] G.R. Ganger, J.D. Strunk, and A.J. Klosterman. Self-* Storage: Brick-based Storage with Automated Administration. Technical Report CMU-CS-03-178, Carnegie Mellon University Technical Report, 2003.

[26] L.M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing Queries Across Diverse Data Sources. In *Proceedings of the 23rd Intl. Conf. on Very Large Data Bases*, pages 276–285, 1997.

[27] M. Harren, J. Hellerstein, R. Huebsch, B. Thau Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *Peer-to-Peer Systems Int. Workshop*, 2002.

[28] R. Huebsch, B. Chun, J.M. Hellerstein, B.T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A.R. Yumerefendi. The architecture of pier: an internet-scale query processor. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, 2005.

[29] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proceedings of 19th International Conference on Very Large Databases*, 2003.

[30] Z. G. Ives, N. Khandelwal, A. Kapur, and M. Cakir. Orchestra: Rapid, collaborative sharing of dynamic data. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, 2005.

[31] Z.G. Ives, D. Florescu, M. Friedman, A. Levy, and D.S. Weld. An Adaptive Query Execution System for Data Integration. In *Proceedings of the 1999 ACM SIGMOD Intl. Conf. on Management of Data*, pages 299–310, 1999.

[32] V. Josifovski, P. Schwarz, L.M. Haas, and E. Lin. Garlic: a new flavor of federated query processing for db2. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 524–532, 2002.

[33] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the 1998 ACM SIGMOD Intl. Conf. on Management of Data*, pages 106–117, 1998.

[34] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: semantics and algorithmic issues. In *Proceedings of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, pages 325–336, 2003.

[35] P. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *pods05*, pages 61–75, 2005.

[36] M. Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–246, 2002.

[37] M. Tamer Ozsu and P. Valduriez. *Principles of distributed database systems*. Prentice-Hall, Inc., 1991.

[38] V. Papadimos and D. Maier. Distributed queries without distributed state. In *WebDB*, pages 95–100, 2002.

[39] N. W. Paton and O. Diaz. Active database systems. *ACM Comput. Surv.*, 31(1):63–103, 1999.

[40] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, 2001.

[41] R. Riedel, C. Faloutsos, G.A. Gibson, and D. Nagle. Active disks for large-scale data processing. *Computer*, 34(6):68–74, 2001.

[42] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. Colt: continuous on-line tuning. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 793–795, 2006.

[43] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. On-line Database Tuning. Technical Report UCSC-CRL-06-07, UC Santa Cruz, 2006.

[44] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, 2001.

[45] I. Tatarinov, Z. G. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The piazza peer data management project. *SIGMOD Rec.*, 32(3):47–52, 2003.

[46] D. C. Zilio, J. Rao, S. Lightstone, G.M. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *Proceedings of the 30th Intl. Conf. on Very Large Data Bases*, pages 1087–1097, 2004.