A Vision for Next Generation Query Processors and an Associated Research Agenda

Anastasios Gounaris

Aristotle University of Thessaloniki
Department of Informatics
541 24 Thessaloniki, Greece
gounaria@csd.auth.gr

Abstract. Query processing is one of the most important mechanisms for data management, and there exist mature techniques for effective query optimization and efficient query execution. The vast majority of these techniques assume workloads of rather small transactional tasks with strong requirements for ACID properties. However, the emergence of new computing paradigms, such as grid and cloud computing, the increasingly large volumes of data commonly processed, the need to support data driven research, intensive data analysis and new scenarios, such as processing data streams on the fly or querying web services, the fact that the metadata fed to optimizers are often missing at compile time, and the growing interest in novel optimization criteria, such as monetary cost or energy consumption, create a unique set of new requirements for query processing systems. These requirements cannot be met by modern techniques in their entirety, although interesting solutions and efficient tools have already been developed for some of them in isolation. Next generation query processors are expected to combine features addressing all of these issues, and, consequently, lie at the confluence of several research initiatives. This paper aims to present a vision for such processors, to explain their functionality requirements, and to discuss the open issues, along with their challenges.

1 Introduction

The success of databases and the reason they play such a key role in data management lies largely in their ability to allow users to specify their data retrieval and update tasks declaratively, relieving them from the burden of specifying how these tasks should be processed; the responsibility of the latter aspect rests with the query processor. To this end, query processing involves the phases of translation, optimization and execution, which have been the topic of investigation for many decades, resulting in a broad range of effective and efficient techniques (e.g., [8, 25, 30]). In conventional static query processors, the three phases of query processing occur sequentially, whereas adaptive query processing (AQP) solutions differ in that they interleave query optimization with query execution [12].

Since it is rather unusual for all the data of a single organization to reside at a single physical place, distributed query processing (DQP) has received considerable attention mainly with a view to supporting transactional tasks over geographically spread data resources. Distributed query execution involves the same three phases as centralized query processing, but considers more issues than in traditional centralized systems, such as dynamic data placement, replica management and communication cost [34, 43]. Consequently, optimization of distributed queries is inherently more complex than when applied to a single-node setting [56] and this is aggravated by the fact that different distributed environments may shape it towards different, and potentially contradicting goals [42]. For example, in some environments accuracy is not as important as returning the first results as early as possible (e.g., [31]), or the economic cost may be an issue (e.g., [53]). A typical optimization criterion is the minimization of a query's response time; this problem is often reduced to the problem of minimizing the communication cost under the assumption that communication is the dominant cost and there exist several proposals that try to address it [6, 17, 39]. This type of DQP is suitable for short transactional tasks requiring strong guarantees on the ACID properties [43].

Nevertheless, modern applications are often characterized by different needs. Conventional query processing techniques are tailored to settings where data reside on disk, appropriate indices have been built and metadata have been collected, whereas, nowadays, data sources, either in the form of sensors or not, may continuously produce data to be processed on the fly. There are several complications stemming from that, including the unavailability of the metadata that are typically fed to the optimizer and the applicability of one-pass algorithms only. In addition, more and more data management applications do not comprise transactional tasks. They rather focus on decision support, analysis and data driven research, and as such, they tend to comprise particularly data- and computation-intensive tasks requiring relatively high degrees of parallelism. The emergence of new computing paradigms, such as grid and cloud computing, has enabled the runtime pooling of hardware and software resources, whereas technologies such as web services facilitate the reuse and sharing of remote complex analysis tools called from within database queries (e.g., as in [3, 38]) or generic workflows (e.g., [50]). Finally, in modern applications, economic cost, energy consumption, network utilization, quality of analysis tools employed, accuracy of results, data quality and other QoS criteria may be equally if not more important than query response time, throughput and communication cost.

In this paper, we make a statement that next generation query processors should evolve so that they can efficiently and effectively support scenarios addressing all the afore-mentioned issues. Note that not all of these issues are new, however, until now, they have tended to be investigated in isolation. For instance, AQP techniques try to deal with the unavailability of statistics mostly when processing takes place in a centralized manner [12], and programming frameworks such as MapReduce [10, 11] and PigLatin [41] offer massive parallelism at the expense of less declarative task definition and lack of sophisticated optimization.

Also, there have been significant advances in stream data management (e.g., [19, 9]), and multi-objective optimization (e.g., [45]), but without considering wide-area heterogeneous computation platforms with arbitrarily high degrees of parallelism. The surveys in [27, 44, 55], which focus on data-management on grid computing infrastructures, and the discussion in [52] about the future of database are relevant to this work, too.

In the remainder of this paper, we present our vision for next generation query processors in more detail (Section 2). Next, in Section 3, we discuss three of the most important open research areas in order this vision to be fulfilled, namely the programming and execution model, the optimization process and the need for advanced autonomic techniques. Section 4 concludes the paper.

2 The vision and its requirements

Our vision is in line with the suggestion of the recent Claremont report that *"database researchers should take data-centric ideas (declarative programming, query optimization) outside their original context in storage and retrieval, and attack new areas of computing where a data-centric mindset can have major impact"* [2]. Recently, some scepticism has been expressed about the suitability of a single database engine for meeting the afore-mentioned modern needs [52]. This scepticism is mostly grounded on the diversity of characteristics and requirements of modern applications, although there have been some efforts in developing unifying systems (e.g., [14]). Orthogonally to any architectural choices, the query processors envisaged in this work have extended functionality, so that they become capable of supporting scenarios involving arbitrary data sources requiring arbitrary computational and analysis resources while benefitting from the significant advances in and the maturity of database technologies in the last decades. More specifically, the query processors envisaged are characterized by:

– *Declarative task statement:* Assuming that there are catalogues (either centralized or distributed) of (i) the computational, (ii) data, and (iii) analysis resources, users can define complex data analysis tasks in a both expressive (e.g., SQL-like as opposed to simple keyword searches) and declarative manner. These tasks may be mapped not simply to traditional query plans, but also to workflows comprising calls to web services (e.g., [50]), or combination of both, (e.g. [38]). Note that the corresponding language may be lower level than SQL in order to benefit from a richer type system and programming patterns such as iteration, which are necessary for several read only, data-intensive analysis tasks that are typical in data driven research (e.g., [41, 57]).
– *Declarative statement of optimization objective(s):* Nowadays, query processors typically operate on a best-effort basis, trying to minimize fixed system-wide metrics such as total work, throughput, query response time and time to deliver early results. These metrics still apply, but, in addition, metrics such as economic cost [53] and energy consumption [28, 36] are becoming

increasingly important. Clearly, there is a need for data management systems firstly to provide support for multiobjective query optimization [45], and secondly, to allow users to decide which are the preferred optimization criteria for a given task. In other words, apart from describing the task, users should also describe the aspects that they are more interested in (perhaps along with their weights), with a view to guiding the optimization process. The proposal of the ripple joins, which are initially proposed in [26] and enhanced later in [37], can be deemed as an early example of such an approach. This type of joins constitute a family of physical pipelining operators that maximize the flow of information during processing of statistical queries that involve aggregations and are capable of adapting their behavior during processing according to user preferences about the accuracy of the partial result and the time between updates of the running aggregate. More importantly, these preferences can be submitted and modified during execution. In [45], an efficient optimization algorithm is presented for the case where the system enables users to supply a function describing the desired trade-off between contradicting parameters. Similar functionality is of significant importance for modern query processing engines.

– *Massively parallel execution (and optimization):* Parallelism is now well understood in databases, both in terms of architectures, where the shared-nothing model seems to be the most attractive [51], and query processing; parallel query processing can be further divided in independent, pipeline and partitioned parallelism [13]. Although partitioned parallelism can lead to the most significant performance improvements, it is typically employed to a limited extent in wide area heterogeneous settings (e.g., [9, 40, 49]). Data analysis tasks may require exceptionally high degrees of parallelism [11, 14, 57] and next generation query processors should evolve to support this requirement, even in the case where the execution environment is remote, non-dedicated and potentially significantly different from fully controlled homogeneous clusters. In addition, since wide area optimization is computationally demanding, optimization may take place in a distributed and parallel manner, as well. Finally, massive parallelism should be applicable to both stream and stored data.
– *Autonomic execution:* This property is becoming increasingly important as tasks are becoming more complex and long running. Two aspects of autonomic computing [32] that are particularly relevant are self-optimization and self-healing. The former strongly relates to AQP, and the latter to fault-tolerance. Today, the vast majority of AQP techniques tackle the problem of unavailable, inaccurate and/or changing data statistics; however there is also a need to be capable of adapting to changing resource characteristics (e.g., [21]).

The characteristics above will render query processors more suitable for online, inbound [52], high performance data analysis in grid and cloud computing environments. In order to develop such a query engine, several features of existing research initiatives should be combined, and open research issues should be investigated, as outlined in the next section.

3 A research agenda

In order to efficiently and effectively support scenarios involving complex analysis of sheer data volumes in remote, non-dedicated environments with flexible optimization criteria, next generation query processors should be equipped with mechanisms to process tasks and optimization criteria submitted declaratively, benefit from massive parallelism and improve their autonomic behavior. Given that partial solutions to some of these requirements have been proposed, the development of the envisaged next generation query processors lies at the confluence of several paradigms and a question arises as to how these partial solutions can be combined and in which areas further research should be conducted. In this section, we deal with this problem focusing on three complementary aspects, namely the programming and execution model, the optimization process and the autonomic execution at runtime.

3.1 Unifying programming and execution model

The combination of stream, parallel and distributed databases yields database systems that can scale to thousands of nodes and numerous data sources, may employ both push and pull execution model, and exhibit good performance for lookup tasks that require reading part of the data (due to the presence of indices); also parallel and distributed databases can effectively employ pipeline parallelism [25], incorporate sophisticated optimization techniques, modify the execution plan at runtime and, in principle, support multiple optimization criteria. Developing efficient integrated systems combining features from stream, parallel and distributed databases is a promising research direction, and may require radical changes in the way DBMSs are designed and implemented [52].

However, integrated systems based solely on database technologies suffer from inherent limitations. They lack efficient fault tolerance mechanisms; typically queries are re-executed in case of failure, which is not desirable in long running tasks. Also, several analysis tasks cannot be easily expressed as user-defined functions so that they can be called from within queries. On the other hand, execution paradigms based on, or inspired by, MapReduce offer built-in fault-tolerance capabilities, a user-convenient way to express complex analysis tasks, and good performance when these tasks involve the reading of complete sets of either structured or unstructured data, even in heterogeneous settings [58].

Non-surprisingly, the development of hybrid engines combining the strong points of both approaches has already been suggested [1]. According to [1], the integration should be at both the language level and the system and engine level. At the language level, efforts such as PigLatin [41], and DryadLINQ [57] can be deemed as an important initial step to this direction. However, developing a unifying execution model seems to be a more challenging research issue.

Current query processors receive as input a query in a high level language and transform it to a query execution plan, which, in most of the cases, is in the form of a directed acyclic graph (DAG), and more specifically, of a binary tree. In the case of SQL, for example, the tree nodes are query operators, such as joins, scans, projections and so on. Interestingly, workflows consisting of calls to web services, which is a common means of expressing data analysis tasks, are typically represented as DAGs, as well [50]. MapReduce jobs can also be regarded as simple single-input two stage workflows. Although in both databases, and workflow management and MapReduce-like paradigms the tasks can be expressed as DAGs, the vertex semantics and the execution logic in each case are different. So, it is unclear how integrated and unifying engines can be developed. In database query plans, the order of operators may change without affecting the quality of results with a view to improving the efficiency of the execution, and there exist well established rules to determine when such re-orderings should take place. This is not the case in generic workflows. Also, database technologies may adopt an adaptive execution model, like Eddies [4], which, in essence, corresponds to different DAGs for each data item; this feature, again, is not supported by today's workflow management systems and MapReduce implementations. Finally, stream query processing, apart from specific primitives [52], requires effective pipelining.

Perhaps, an intermediate goal is to develop integrated systems that can compile and execute database queries, workflows and MapReduce-like tasks in a common environment, without actually developing a unifying execution engine. Clustera [14] is an early example of such initiatives.

3.2 Optimization

Optimization of parallel data-intensive query plans involves several important open research issues. In general, query optimization problems in distributed settings are intractable, even in the case where partitioned parallelism and scheduling issues are ignored [56]. Frameworks such as [14, 41, 57] either do not perform any optimization or apply a limited set of greedy heuristics. The degree of parallelism in these settings is configured in an ad-hoc manner, which may deviate from (near) optimal solutions significantly. A promising research direction is to develop more sophisticated, cost based optimizers in this context both for the execution plan and the order of operations, and the number of machines.

However, even when only traditional database queries are considered, solutions should be developed for the multi-objective cases and for the resource allocation and scheduling problem. For the later, efficient solutions have been proposed only for homogeneous settings [18]. In heterogeneous settings, not only the degree of parallelism must be decided, but also the exact resources participating in the execution must be selected (e.g., [20]); the challenge here is that any solution should be as close to optimal as possible without being too complex with respect to the number of candidate nodes, as, in grid and cloud settings, this number may be too high. Borrowing solutions from generic workflow scheduling does not fully help; typically, in DAG scheduling, partitioned parallelism is not considered [35]. Nevertheless, there exist interesting proposals that do consider economic cost [7] and heterogeneity [47].

When high level optimization criteria apply, e.g., energy consumption and economic cost, there is a need to establish how these criteria impact on the execution and what is their exact correlation with the decisions taken during optimization; i.e., there needs to be a mapping between high level objectives and lower level metrics directly manipulated by the system. Furthermore, in the case of multi-objective optimization further research is required for establishing a standard efficient way to describe the trade-offs (e.g., [45]). The optimization process may also differ according to whether the objectives are specified as utility functions or not [33], and until now, the exact consequences of such a decision are not well understood.

To summarize, the envisaged query processors require optimizers that can handle arbitrary combinations of objectives and provide optimal or near optimal parallelized execution plans in reasonable time. Note that the whole optimization process may be interactive: the users may submit some criteria the exact configuration of which can be negotiated, in order to reach a Service Level Agreement (SLA). This mode of optimization has not been adequately explored in database query processors.

3.3 Runtime Execution

Operating on data that may have not been preprocessed in a non-dedicated environment consisting of numerous nodes calls for more advanced AQP techniques. We have already mentioned the need to further emphasize on changing resources in combination with volatile data characteristics; e.g., adapting the degree of parallelism in heterogeneous environments on the fly or investigating the interplay between resource allocation and load balancing [54] are issues that have been largely overlooked. In addition, more attention should be placed on the theoretical analysis of adaptive solutions and investigation of properties such as stability, accuracy, and settling time. To this end, AQP can benefit from control theory techniques, which are well-established in engineering fields and are typically accompanied by theoretical investigations of such properties [29]. Interesting examples of applications of control theory to data management systems have been proposed in [15, 16, 22–24]. Another open research area is the development of AQP techniques in which actions are driven by utility functions (e.g., [46]).

Efficient fault tolerance is equally important to runtime re-optimization. Fault tolerance features are built-in in platforms such as MapReduce; also some proposals exist also in the database field, mainly for continuous queries and data streams (e.g., [5, 48]). However, all these solutions are based on different protocols and concepts, so there is a need to reach a consensus as to which approach is more appropriate in each case.

Finally, data analysis and data driven research may be processed in a way that early results are delivered as soon as possible, in case users choose to cancel their queries if, for example, realize that the results are not interesting or the task has not been specified appropriately. Pipelining execution addresses to an adequate extent this need. Pipelining may be employed by database engines in a straightforward manner, but this may not be the case for other computing paradigms, such as MapReduce.

4 Conclusions

This work aims at specifying the requirements of modern applications and presenting a vision for next generation query processors, so that they are more tailored to data analysis and data driven research, benefit from declarative task and objective statement and are capable of being highly parallelized while exhibiting advanced autonomic behavior to cope with the volatility of the environment. In addition, this work discusses important areas in which further research is required in order to fulfill the vision. These areas include the development of integrated execution engines, advanced wide-area optimization and autonomic runtime execution.

Acknowledgements. The author would like to thank his colleagues Apostolos Papadopoulos and Constantinos Tsichlas for their insightful comments and help during the preparation of this work.

References

1. D. J. Abadi. Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32(1):3–12, 2009.
2. R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. Garcia-Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. C. Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. S. Szalay, and G. Weikum. The claremont report on database research. *SIGMOD Record*, 37(3):9–19, 2008.
3. M. N. Alpdemir, A. Mukherjee, N. W. Paton, P. Watson, A. A. A. Fernandes, A. Gounaris, and J. Smith. Service-based distributed querying on the grid. In *Proceedings of 1st International Conference on Service Oriented Computing - ICSOC*, pages 467–482. Springer, 2003.
4. R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 261–272. ACM, 2000.
5. M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-tolerance in the borealis distributed stream processing system. *ACM Trans. Database Syst.*, 33(1), 2008.
6. P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. R. Jr. Query processing in a system for distributed databases (sdd-1). *ACM Trans. Database Syst.*, 6(4):602–625, 1981.
7. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

8. S. Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 34–43. ACM Press, 1998.
9. M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.
10. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
11. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
12. A. Deshpande, Z. G. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.
13. D. J. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, 35(6):85–98, 1992.
14. D. J. DeWitt, E. Paulson, E. Robinson, J. F. Naughton, J. Royalty, S. Shankar, and A. Krioukov. Clustera: an integrated computation and data management system. *PVLDB*, 1(1):28–41, 2008.
15. Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. S. Parekh, and C. Garcia-Arellano. Incorporating cost of control into the design of a load balancing controller. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 376–387, 2004.
16. Y. Diao, C. W. Wu, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chu, and J. Colaco. Comparative studies of load balancing with control and optimization techniques. In *Proceedings of the American Control Conference*, pages 1484–1490, 2005.
17. R. S. Epstein, M. Stonebraker, and E. Wong. Distributed query processing in a relational data base system. In E. I. Lowenthal and N. B. Dale, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 169–180. ACM, 1978.
18. M. N. Garofalakis and Y. E. Ioannidis. Parallel query scheduling and optimization with time- and space-shared resources. In *VLDB*, pages 296–305, 1997.
19. L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.
20. A. Gounaris, R. Sakellariou, N. W. Paton, and A. A. A. Fernandes. A novel approach to resource scheduling for parallel query processing on computational grids. *Distributed and Parallel Databases*, 19(2-3):87–106, 2006.
21. A. Gounaris, J. Smith, N. W. Paton, R. Sakellariou, A. A. Fernandes, and P. Watson. Adaptive workload allocation in query processing in autonomous heterogeneous environments. *Distrib. Parallel Databases*, 25(3):125–164, 2009.
22. A. Gounaris, C. Yfoulis, R. Sakellariou, and M. D. Dikaiakos. A control theoretical approach to self-optimizing block transfer in web service grids. *TAAS*, 3(2), 2008.
23. A. Gounaris, C. Yfoulis, R. Sakellariou, and M. D. Dikaiakos. Robust runtime optimization of data transfer in queries over web services. In *Proc. of ICDE*, pages 596–605, 2008.
24. A. Gounaris, C. A. Yfoulis, and N. W. Paton. An efficient load balancing LQR controller in parallel databases queries under random perturbations. In *3rd IEEE Multi-conference on Systems and Control (MSC 2009)*, 2009.
25. G. Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.
26. P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 287–298, 1999.

27. A. Hameurlain, F. Morvan, and M. El Samad. Large Scale Data management in Grid Systems: a Survey. In *IEEE International Conference on Information and Communication Technologies: from Theory to Applications (ICTTA)*, 2008.
28. S. Harizopoulos, M. Shah, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. In *CIDR*, 2009.
29. J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.
30. Y. E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1):121–123, 1996.
31. Z. G. Ives, D. Florescu, M. Friedman, A. Y. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA.*, pages 299–310. ACM Press.
32. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.
33. J. O. Kephart and R. Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, 2007.
34. D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.
35. Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.
36. W. Lang and J. M. Patel. Towards eco-friendly database management systems. In *CIDR*, 2009.
37. G. Luo, C. Ellmann, P. J. Haas, and J. F. Naughton. A scalable hash ripple join algorithm. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 252–262. ACM, 2002.
38. S. Lynden, A. Mukherjee, A. C. Hume, A. A. A. Fernandes, N. W. Paton, R. Sakellariou, and P. Watson. The design and implementation of ogsa-dqp: A service-based distributed query processor. *Future Generation Comp. Syst.*, 25(3):224–236, 2009.
39. L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *VLDB'86 Twelfth International Conference on Very Large Data Bases*, pages 149–159. Morgan Kaufmann, 1986.
40. K. W. Ng, Z. Wang, R. R. Muntz, and S. Nittel. Dynamic query re-optimization. In *SSDBM*, pages 264–273, 1999.
41. C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD Conference*, pages 1099–1110, 2008.
42. M. Ouzzani and A. Bouguettaya. Query processing and optimization on the web. *Distributed and Parallel Databases*, 15(3):187–218, 2004.
43. M. Ozsu and P. Valduriez, editors. *Principles of Distributed Database Systems (Second Edition)*. Prentice-Hall, 1999.
44. E. Pacitti, P. Valduriez, and M. Mattoso. Grid data management: Open problems and new issues. *J. Grid Comput.*, 5(3):273–281, 2007.
45. C. H. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM, 2001.
46. N. W. Paton, M. A. T. Aragão, K. Lee, A. A. A. Fernandes, and R. Sakellariou. Optimizing utility in cloud computing through autonomic workload execution. *IEEE Data Eng. Bull.*, 32(1):51–58, 2009.
47. R. Sakellariou and H. Zhao. A hybrid heuristic for DAG scheduling on heterogeneous systems. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*. IEEE Computer Society, 2004.

48. M. A. Shah, J. M. Hellerstein, and E. A. Brewer. Highly-available, fault-tolerant, parallel dataflows. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 827–838. ACM, 2004.
49. J. Smith, A. Gounaris, P. Watson, N. W. Paton, A. A. A. Fernandes, and R. Sakellariou. Distributed query processing on the grid. *International Journal of High Performance Computing Applications*, 17(4):353–367, 2003.
50. U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query optimization over web services. In *VLDB*, pages 355–366, 2006.
51. M. Stonebraker. The case for shared nothing. *IEEE Data Engineering Bulletin*, 9(1):4–9, 1986.
52. M. Stonebraker. Technical perspective - one size fits all: an idea whose time has come and gone. *Commun. ACM*, 51(12):76, 2008.
53. M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *VLDB J.*, 5(1):48–63, 1996.
54. F. Tian and D. J. DeWitt. Tuple routing strategies for distributed eddies. In *VLDB*, pages 333–344, 2003.
55. S. Venugopal, R. Buyya, and K. Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Comput. Surv.*, 38(1), 2006.
56. C. Wang and M.-S. Chen. On the complexity of distributed query optimization. *IEEE Trans. Knowl. Data Eng.*, 8(4):650–662, 1996.
57. Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In *OSDI*, pages 1–14, 2008.
58. M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *OSDI*, pages 29–42, 2008.