Resiliency-Aware Data Management

Matthias Boehm
TU Dresden
Database Technology Group
matthias.boehm@tu-dresden.de

Wolfgang Lehner
TU Dresden
Database Technology Group
wolfgang.lehner@tu-dresden.de

Christof Fetzer
TU Dresden
Systems Engineering Group
christof.fetzer@tu-dresden.de

ABSTRACT

Computing architectures change towards massively parallel environments with increasing numbers of heterogeneous components. The large scale in combination with decreasing feature sizes leads to dramatically increasing error rates. The heterogeneity further leads to new error types. Techniques for ensuring resiliency in terms of robustness regarding these errors are typically applied at hardware abstraction and operating system levels. However, as errors become the normal case, we observe increasing costs in terms of computation overhead for ensuring robustness. In this paper, we argue that ensuring resiliency on the data management level can reduce the required overhead by exploiting context knowledge of query processing and data storage. Apart from reacting on already detected errors, this was mostly neglected in database research so far. We therefore give a broad overview of the background of resilient computing and existing techniques from the database perspective. Based on the lack of existing techniques on data management level, we raise three fundamental challenges of resiliency-aware data management and present example use cases. Finally, our vision of resiliency-aware data management opens many directions of future work. Fundamental research, including the partial reuse of underlying mechanisms, would allow data management systems to cope with future hardware characteristics by effectively and efficiently ensuring resiliency.

Keywords

Resiliency, Fault-Tolerance, Data Management Challenge

1. INTRODUCTION

Physical limits of processor design such as minimal size of transistors and power-thermal constraints of high frequency processors [6, 19] caused a fundamental change of computing architectures towards large-scale parallel systems that will also continue in the future. This especially includes an increasing number of cores and an increasing heterogeneity of computing components and interconnects. Multi- and many-core systems have become standard nowadays and the development will lead to hundreds, thousands and more of cores per processor. Heterogeneity of cores and interconnects also increases substantially: future systems will therefore most likely consist of few high frequency cores, many low frequency cores and special purpose cores [19]. An example for this trend is the Intel Sandy Bridge ring architecture, which already connects GPU and CPU cores on the same die. In the long term, new technologies such as Memristors, Carbon Nanotubes, Silicon Nanowires as well as biological and chemical components might be included in such architectures that will further increase the heterogeneity.

The increasing number of components inherently leads to increasing system error rates. Assume a fix error probability per component (core or transistor). As the number of components increases, the total error probability of the overall system, i.e., at least one component fails, increases linearly with the number of components [25]. Furthermore, component error rates even increase due to decreasing feature sizes [6] or due to new technologies. Errors and error-prone computation will therefore become the normal case.

So far, the detection and correction of permanent and transient technical errors are mainly addressed at hardware abstraction layers and operating system level; technical errors therefore received little attention in database research. The major problem of ensuring a low overall error probability by these abstraction layers are dramatically increasing reliability costs (overhead) with increasing number of components [2]. From a data management perspective, we are able to reduce these costs because we "only" need to ensure correct outcomes in terms of query results. Hence, we argue that resiliency-awareness at data management level can provide higher effectiveness (higher detection rates) and efficiency (lower overhead) than general purpose techniques.

The primary contribution of this paper is to introduce the challenge of resilient (error-aware) data management in order to cope with changing hardware characteristics by exploiting context knowledge of query processing and data storage. Furthermore, we make the following more concrete contributions, which also reflect the structure of the paper:

- First, we explain the background of resiliency including error types and reasons in Subsection 2.1.
- Second, we discuss existing techniques from database research in Subsection 2.2.
- Third, we derive three fundamental challenges of resilient data management in Section 3.

Finally, we conclude the paper in Section 4 with a summary of our vision of resiliency-aware data management.

Proceedings of the VLDB Endowment, Vol. 4, No. 12

2. BACKGROUND OF RESILIENCY

To assist the reader in understanding the vision of resiliency-aware data management, we first describe the background of types and reasons of errors. Second, we also present existing techniques from database research.

2.1 Types, Reasons, and Rates of Errors

Taxonomies of dependable systems [3] distinguish faults (technical defects or variability), errors (system-internal misbehavior), and failures (system-external misbehavior). Due to hierarchically structured data management systems, i.e., a failure at a system level is a fault at the next level, we globally use the term error. We further distinguish *static* (hard errors, permanently corrupted bits) and *dynamic* errors (soft errors, transiently corrupted bits) [6, 25]:

- *Static Errors [6]:* For small transistor sizes, there is transistor variability due to different electrical characteristics of random dopant atoms fluctuation. In addition, the sub-wavelength lithography is a reason for line edge roughness and other effects in transistors.
- *Dynamic Errors:* Heat flux variations cause time-dependent, dynamic, supply voltage variations and thus, dynamic errors [6]. Furthermore, energetic alpha particles and neutrons (from cosmic ray) cause increasing charges that might dynamically invert the state of a logical device (e.g., SRAM cell, latch, gate) [25].

Finally, it is worth to note that component aging might lead to increasing static and dynamic error rates over time [22].

In order to prevent *silent* errors (silent computation/data corruption, mainly by dynamic errors) typically basic error detection techniques are employed to enable a fail-stop behavior [25]. Depending on the used error correction code (ECC) [8], the error is corrected automatically, or if the number of affected bits exceeds certain limits, it is reported as an uncorrectable error [22] or it even remains undetected. We use the terms of *implicit* (silent) and *explicit* (detected or corrected) errors. Note that error correction codes can cause significant storage and latency overhead [2, 25].

As an example for DRAM error rates, consider a study by Schroeder et al. in a Google environment [22]. They report rates of 25,000 to 70,000 errors per billion device hours per Mbit and 8% of DIMMs affected by errors per year. Furthermore, there are predictions of about 8% increase in soft-error rate per logic state bit each technology generation [6]. Similar observations have also been made for soft disk errors [4, 17]. For example, Bairavasundaram et al. reported 400,000 checksum mismatches in a field study of 1.53 million disk drives over a time period of 41 months [4]. Finally, the combination of increasing error rates (including silent errors) and associated increasing costs for general-purpose error detection and correction motivates to exploit context knowledge of data management for more efficient resiliency.

2.2 Existing Database Techniques

In contrast to existing work from systems research [3], resilient computing did not receive much attention in database research. Existing techniques can be mainly classified into (1) work that relies on error-aware frameworks, (2) work addressing explicit errors, and (3) work that addresses implicit errors for specific aspects of data management.

There are techniques that simply rely on error-aware processing frameworks such as MapReduce or Hadoop. By building a data management solution on top of such frameworks (e.g., Pig [12], Hive [24]) or by hybrid approaches (e.g., HadoopDB [1], Dremel [21]) enables resilient processing. However, these approaches use general-purpose resiliency mechanism of these frameworks and thus, do not exploit context knowledge from data management.

In addition, there is a long history of work concerning explicit (detected) errors that trigger recovery. Major research directions are efficient recovery processing [15, 26] and data replication techniques [7]. For example, fault-tolerance in (distributed) data streaming systems was addressed by checkpointing and recovery [5] as well as operator replicas and replication transparency [20]. Recent work also integrates the estimated recovery time into the optimization of ETL flows by considering the placement of recovery points [23]. All these techniques use a fail-stop-retry model [15], where recovery is triggered on explicit errors only.

Finally, recent work addressed also implicit errors. For example, Graefe and Stonecipher discussed efficient consistency verification (error detection) within and between B-tree indexes [14] in order to cope with silent soft errors. They even achieved performance that allows for online verification. Simitsis et al. presented so-called replicate operators (for redundant execution of ETL flow operators) in combination with specific voter policies for deciding on result correctness [23]. In addition, the Dremel system addressed the problem of stragglers (unfinished subtasks) in distributed environments [21]. While these proposals are important steps towards resilient data management, they address only specific aspects. In contrast, we argue for holistic solutions to achieve resilient data management. In conclusion, we see a lack of resiliency-aware data management techniques supporting query execution, data storage, and query optimization, especially, for implicit (silent) errors.

3. RESILIENT DATABASE CHALLENGES

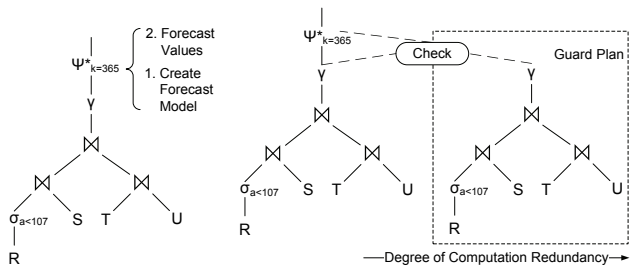
Future hardware characteristics in terms of increasing scale and decreasing feature sizes as well as the lack of existing techniques lead to the vision of resiliency-aware data management. We introduce the three major challenges of (1) resilient query processing, (2) resilient data storage, and (3) resiliency-aware optimization. In addition, we exemplify use cases, where resiliency-aware data management enables higher efficiency compared to general purpose techniques.

3.1 Query Processing

While explicit computation errors are traditionally addressed with recovery and restore techniques, higher error rates and implicit (silent) errors pose a fundamental novel requirement to query processing.

CHALLENGE 1. (Resilient Query Processing) *Implicit errors during query processing can lead to false negatives (missing tuples), false positives (tuples with invalid predicates) or inaccurate aggregates. Due to silent errors, there is no indicator for result correctness. Thus, the challenge is to ensure reliable query results (1) by efficient and effective detection and correction of inaccurate (sub)query results, or (2) by the use of error-tolerant algorithms.*

Consider as an application example advanced analytics such as clustering, classification, and forecasting. There is a trend of integrating such advanced analytics into data management systems [10, 11, 13]. The overall goal is to support



(a) Forecast QP (b) Resilient Forecast QP

Figure 1: Example Resilient Advanced Analytics.

complex statistics that go beyond traditional aggregation queries on huge data sets. In the following, we make a case for resilient advanced analytics, where it would be advantageous to ensure resiliency on query processing level.

EXAMPLE 1. (Resilient Advanced Analytics) *Forecast queries [11] are an example for advanced analytics. Figure 1(a) shows a query plan (QP) of such a query, where we compute a time series by an aggregation query and subsequently use a forecast operator ψ for forecasting* 365 *steps ahead of this time series. This operator includes the expensive parameter estimation (e.g., with iterative gradient descent algorithms), where the search space grows exponentially with the number of parameters. Thereafter, we use the trained model for prediction. These queries produce per se inaccurate results (predictions) such that we can trade accuracy, performance, and resiliency but also energy efficiency [18] that is tightly coupled with resiliency. We can leverage context knowledge from query processing as shown in Figure 1(b) in order to efficiently ensure (1) the correctness of the computed time series and (2) the correctness of parameter estimation. During blocking execution of parameter estimation, we asynchronously execute guard plans and specific check operators (e.g., with voting policies [23]) for ensuring correctness of the time series (subquery result). This can be done with small overhead, exploiting the available parallelism (number of cores). Furthermore, the iterative parameter estimators (often local optimization) allow for error-tolerant behavior—as long as we stay within the same local optima region—because a wrong computation is replaced by the next iteration result. Thus, we can tolerate certain rates of data computation errors or state corruptions. This enables us to disregard costly error detection and correction for parameter estimation.*

Thus, crucial operations should be guarded by redundant computation, while other are per se error-tolerant and require no additional resiliency efforts. Even if error rates are unbounded, we can give guarantees by exploiting redundancy, knowledge of the algorithm, or gathered statistics.

Major research directions of resilient query processing are (1) error-aware query processing with redundant (partial query/data) processing, (2) algorithm design for error-tolerant operators, as well as (3) probabilistic guarantees according to the underlying components and algorithms.

3.2 Data Storage

Resiliency-aware data management further requires resilient data storage in terms of data stability. Here, implicit errors have even higher negative impact because dynamic errors can cause permanent data corruption, i.e., all subsequent queries use the corrupted data. For this reason, almost all commercial DBMS employ data verification techniques such as simple checksum mechanisms [14]. With regard to effectiveness and efficiency, we formulate this as the second fundamental challenge.

CHALLENGE 2. (Resilient Data Storage) *Explicit (detected) and implicit (undetected) errors lead to data loss or data corruption over time. Due to the silence of implicit errors, there might be no trigger for recovery. The challenge is to ensure data stability with certain guarantees by error detection and correction on unreliable components (incl. disks, main memory) with different resiliency characteristics.*

Typically, data-level resiliency is ensured with partitioning and replication (e.g., RAID systems or transparent replication). In the following, we use an example to show how we could ensure resilient data storage more efficiently.

EXAMPLE 2. (Resilient Data Partitioning) *Assume a table R. In order to ensure resilient data storage, we maintain a replica R' of this table as shown in Figure 2. We may also exploit this redundancy for more efficient query processing. Similar to systems such as HYRISE [16] that computes the best vertical partitioning for a* single *copy of a table, the core idea is to compute a set of complementary partitioning schemes (vertical/horizontal) for* multiple *table replicas with regard to the query workload. Figure 2 shows two replicas, where we use row- (for point-queries) and column-oriented layouts (for aggregation queries) at the same time. Furthermore, we employ sample synopses [13] for efficient time-based or on-the-fly error detection with probabilistic guarantees. In case of detected errors, we subsequently use the replicas for error correction. Synopses and replicas might be located on different components with different resiliency.*

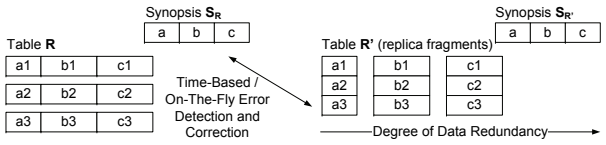


Figure 2: Example Resilient Data Partitioning.

Thus, resiliency-aware data storage can be beneficial due to consistency checks on query processing level and due to exploiting replicas for more efficient query processing.

Major research directions are (1) synopsis/replica design with probabilistic guarantees, (2) adaptive detection strategies, offline or online during query processing/data manipulation, (3) efficient correction strategies, (4) exploitation of different partitioning/compression schemes, (5) efficient update handling, and (6) re-organization/re-partitioning.

3.3 Query Optimization

Resilient query processing (operational) and resilient data storage (physical design) both span multi-objective optimization problems. This leads to the third fundamental challenge that completes the resilient database challenges.

CHALLENGE 3. (Resiliency-Aware Optimization) *The increasing number of cores as well as increasing heterogeneity of components and interconnects, pose the challenge of how and where to execute subqueries, on which physical design, in order to optimize result accuracy, performance, energy efficiency and resiliency. The integration of resiliency and energy efficiency into query optimization requires (to some extend) architecture-awareness of the underlying system.*

EXAMPLE 3. (Resiliency-Aware Optimization) *Recall the query from Example 1 and assume heterogeneous cores. Regarding performance, we assign compute-bound operators (such as the forecast operator ψ) to high frequency cores, while we assign memory-bound operators to low frequency cores. Furthermore, we trade accuracy, performance, energy consumption, and resiliency. The power consumption is computed by $P \approx C_S \cdot V^2 \cdot f$, where C_S is the switching capacity, V is the voltage, and f is the frequency [9]. Given the processor capability of dynamic frequency and voltage scaling [9, 18], we can reduce the energy consumption and error rates (due to thermal influences) by decreasing frequency or voltage. The required energy, i.e., the integral of used power over time, is a non-monotonic convex function [9] and below a minimum voltage, the error rates will increase again. In addition, decreased frequency/voltage decreases the performance of query processing (if compute-bound) but due to lower error rates, we might require fewer guard plans. This poses a challenging re-optimization problem.*

In general, heterogeneity of cores leads to hybrid query execution in terms of plan partitioning, where the different CPU, memory, and resiliency characteristics fundamentally strengthens the traditional query optimization problem.

Major research directions regarding resiliency-aware optimization are (1) scheduling redundant subqueries and verification checks, (2) placement of subqueries on reliable/unreliable components (plan partitioning), (3) parallelization of subqueries (data partitioning), (4) topology-awareness (e.g., NUMA), (5) adjustment of platform parameters (e.g., frequency/voltage), (6) error-aware runtime adaptation, and (7) multi-objective cost models and global optimization.

4. CONCLUSIONS

In conclusion, we would like to encourage the database community to reconsider many aspects of data management and query processing with regard to resilient computing in order to cope with future hardware characteristics in terms of increasing scale and decreasing feature sizes that lead to increasing error rates and the resulting high overhead of resilient computing. We argue that resiliency-aware data management can provide higher effectiveness and efficiency than general-purpose resiliency techniques by exploiting context knowledge of query processing and data storage. This vision includes the three fundamental challenges of (1) resilient query processing, (2) resilient data storage, and (3) resiliency-aware optimization. Furthermore, we presented examples for these challenges, where resilient data management would be beneficial. Finally, this inter-disciplinary research field exhibits many opportunities for future work and bridges the gap between systems and database research.

5. REFERENCES

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *PVLDB*, 2(1):922–933, 2009.

[2] T. M. Austin, V. Bertacco, S. A. Mahlke, and Y. Cao. Reliable Systems on Unreliable Fabrics. *Design and Test of Computers*, 25(4):322–332, 2008.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *TDSC*, 1(1):11–33, 2004.

[4] L. N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. An Analysis of Data Corruption in the Storage Stack. In *FAST*, pages 223–238, 2008.

[5] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-Tolerance in the Borealis Distributed Stream Processing System. In *SIGMOD*, pages 13–24, 2005.

[6] S. Y. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *Micro*, 25(6):10–16, 2005.

[7] E. Cecchet, G. Candea, and A. Ailamaki. Middleware-based Database Replication: The Gaps Between Theory and Practice. In *SIGMOD*, pages 739–752, 2008.

[8] C.-L. Chen and M. Y. B. Hsiao. Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review. *IBM J. Res. Dev.*, 28(2):124–134, 1984.

[9] J.-J. Chen and C.-F. Kuo. Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms. In *RTCSA*, pages 28–38, 2007.

[10] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton. MAD Skills: New Analysis Practices for Big Data. *PVLDB*, 2(2):1481–1492, 2009.

[11] S. Duan and S. Babu. Processing Forecasting Queries. In *VLDB*, pages 711–722, 2007.

[12] A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava. Building a HighLevel Dataflow System on top of MapReduce: The Pig Experience. *PVLDB*, 2(2):1414–1425, 2009.

[13] R. Gemulla, W. Lehner, and P. J. Haas. A Dip in the Reservoir: Maintaining Sample Synopses of Evolving Datasets. In *VLDB*, pages 595–606, 2006.

[14] G. Graefe and R. Stonecipher. Efficient Verification of B-tree Integrity. In *BTW*, pages 27–46, 2009.

[15] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. 1993.

[16] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudré-Mauroux, and S. Madden. HYRISE - A Main Memory Hybrid Storage Engine. *PVLDB*, 4(2):105–116, 2010.

[17] J. L. Hafner, V. Deenadhayalan, W. Belluomini, and K. Rao. Undetected Disk Errors in RAID Arrays. *IBM J. Res. Dev.*, 52(4-5):413–426, 2008.

[18] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy Efficiency: The New Holy Grail of Data Management Systems Research. In *CIDR*, pages 1–8, 2009.

[19] J. Held, J. Bautista, and S. Koehl. From a Few Cores to Many: A Tera-scale Computing Research Overview. White paper, Intel, 2006.

[20] J.-H. Hwang, U. Çetintemel, and S. B. Zdonik. Fast and Highly-Available Stream Processing over Wide Area Networks. In *ICDE*, pages 804–813, 2008.

[21] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive Analysis of Web-Scale Datasets. *PVLDB*, 3(1):330–339, 2010.

[22] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM Errors in the Wild: A Large-Scale Field Study. In *SIGMETRICS/Performance*, pages 193–204, 2009.

[23] A. Simitsis, K. Wilkinson, U. Dayal, and M. Castellanos. Optimizing ETL Workflows for Fault-Tolerance. In *ICDE*, pages 385–396, 2010.

[24] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive - A Warehousing Solution Over a Map-Reduce Framework. *PVLDB*, 2(2):1626–1629, 2009.

[25] C. T. Weaver, J. S. Emer, S. S. Mukherjee, and S. K. Reinhardt. Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor. In *ISCA*, pages 264–275, 2004.

[26] G. Weikum and G. Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. 2002.