Online Bit Flip Detection for In-Memory B-Trees Live!

Till Kolditz, Benjamin Schlegel*, Dirk Habich, Wolfgang Lehner

Database Technology Group
Technische Universitt Dresden
01062 Dresden, Germany
{first.last}@tu-dresden.de

Abstract: Hardware vendors constantly decrease the feature sizes of integrated circuits to obtain higher performance and energy efficiency. As a side-effect, integrated circuits – like CPUs and main memory – become more and more vulnerable to external influences and thus unreliable, which results in increasing numbers of (multi-) bit flips. From a database perspective bit flip errors in main memory will become a major challenge for modern in-memory database systems, which keep all their enterprise data in volatile, unreliable main memory. Existing hardware error control techniques like ECC-DRAM are able to detect and correct memory errors, but their detection and correction capabilities are limited and come along with several downsides. To underline this we heat up RAM live on-site to show possible error rates of future hardware. We previously presented various techniques for the B-Tree – as a wide-spread index structure – for online error detection and thus increase its overall reliability [Kea14b]. We also show live performance comparisons in terms of throughput and error detection rates between several bit flip detecting B-Tree variants. By that, we demonstrate the tradeoff between detection accuracy and index throughput. Furthermore, we show annotated structural information about the trees like corrupted nodes and inaccessible sub-trees.

1 Introduction

For the last 30 years disk-centric database systems have been state-of-the art, but in the last couple of years this approach has dramatically changed due to several reasons. Especially the significant developments in the hardware sector are the major driver for that change. Due to these hardware developments, servers with 2 TiB of main memory are available for a reasonable price. To efficiently leverage the provided main memory capacities, the database system architecture trend shifted from a disk-centric to a main memory-centric approach, where the entire data pool is kept completely in main memory. The performance gain is massive because database operations are able to benefit from its higher bandwidth and lower latency. However, due to the constant decrease of transistors' feature sizes cosmic rays [BSS08], heat dissipation or electric interaction [Kea14a] have increasing impacts on transistor state stability, which lead to arbitrary (possibly multi-) bit flips (MBFs). Studies show that current hardware is already subject to many bit flips, and even to a substantial portion of multi bit flips. As a first contribution, we will heat up RAM live on-site to show

*currently working at Oracle Labs

behavior of current hardware at extreme operational temperatures (about $130°C$), which gives an outlook on future hardware being error-prone at much lower temperatures. Applications must be hardened to detect and handle MBFs in memory, especially in the case of (main memory) databases, because users or customers typically regard data stability as an essential property of database systems. Data stability is, however, orthogonal to, e.g., the ACID properties concerning transactional behavior of databases. Current techniques like error correction code (ECC) RAM on the one hand introduce higher latencies and monetary costs, while on the other hand they will not suffice to deal with increasing MBF rates. Furthermore, such general purpose techniques cannot leverage application-specific knowledge about applied data structures as, e.g., in DBMSs. Therefore, we proposed several light-to-heavy-weight error (bit flip) detection adaptations to the still wide-spread B-tree [Kea14b], for whose many variants like B+ trees or B*-trees these error detection mechanisms can be adopted. The bit flip detection is integrated in an online fashion during tree traversal, whereas there is no error correction published yet. As a second contribution, in a live demo we visualize the benefits and tradeoffs of our proposed error-detecting B-Tree variants, especially the tradeoff between throughput and protection against main memory MBFs. Due to the size restrictions, please refer to [Kea14b] for any related work.

2 Online Bit Flip Detection for In-Memory B-Trees

As a first step towards database systems being resilient against arbitrary main memory MBFs we introduce our adaptations of the widely used index structure B-tree [Kea14b].

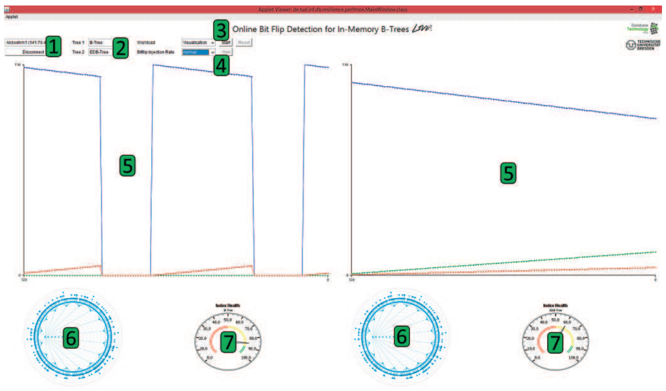
TMR B-Tree: Triple Modular Redundancy (TMR) is a common technique for tolerating arbitrary data corruption, where three copies – replicas – of all data are maintained and algorithms are executed on all three copies. The results are compared and finally a majority voting should lead to the correct result. Execution time and required amount of memory are tripled, which might not always be acceptable. We accordingly also demonstrate a TMR B-tree variant.

EDB-tree: Our first error-detecting B-tree adds pointer sanity checks. Therefore, we exploit the large virtual address space and allocate the whole tree in one contiguous memory area. By that, pointers pointing outwards of this region can be identified. Also, backlinks are added and validated against the originating node's address during tree traversal (parent-child relation). Finally, it is checked whether the address which is pointed at is aligned to the node size. These checks detect most if not all pointer corruptions.

EDB-Tree PB: Our second adaptation sets parity bits for each and every node member, by which also corruption in nodes' meta data, keys and values can be partially detected. Each node member's value domain is stripped by the most significant bit (MSB), which in return is set to obtain an even parity. During tree traversal whenever a node member is accessed the first time, its parity is first checked for evenness. Simple parities only detect any odd-numbered bit flip in a data word.

EDB-Tree CS: In our third variant, for detecting more bit flips we employ XOR checksums to groups of node members. On the one hand these can be computed easily and quickly by XOR-ing the concerning data elements, while on the other hand they will detect almost any bit flips. The exception is when exactly an even number of bit flips occurs at the very

(a) Heating RAM



(b) Live application GUI

Figure 1: Setup for the demonstration.

same bit position in an even number of data objects, so that these even out each other. We add four checksums to each node: one for a node's meta data, all keys, all values and all pointers, respectively. During tree traversal, a checksum is validated before one of the according node members (over which it was computed) is accessed.

3 Demo Description

With this demo we contribute two things. First, we show a video on how we heat up RAM and measure MBFs in main memory. Second, we visualize the benefits and tradeoffs of our proposed error-detecting B-Tree variants, especially the tradeoff between throughput and protection against main memory MBFs.

RAM on Fire Live on-site we heat up RAM to demonstrate possible error rates of future hardware (cf. Figure 1a). An application checks a portion of memory for MBFs, while virtually setting the RAM on fire. We therefore use a heat gun and raise the memory chips' surface temperature to $100\dots130°C$. A live diagram displays the bit flip distribution across the checked memory range.

Error Detecting B-Trees Live! For visualizing different aspects of our proposed error detecting B-trees we will present an application shown in Figure 1b. The demo setup and details highlighted in the figure as well as the demo experience are described in the following. For the demo an Intel server-grade machine will be available running an Intel i7-3960X 12-core CPU (including hyper-threading) and 32 GiB RAM. The GUI interaction starts with the user connecting to the machine (1 in Figure 1b). The available B-tree variants can then be selected from the according combo boxes labeled "Tree 1" and "Tree 2" (2 in Figure 1b). By that, the user can choose which B-tree variants she or he wants to compare against each other. Next, the user selects a workload (3 in Figure 1b) depending on which particular graphs are shown (see below). Afterwards, the user may start, pause and resume the workload and reset the trees, i.e. have them rebuilt (also 3 in Figure 1b). The user may choose between 3 different workloads: performance, visualization, and manual, as presented below.

Data corruption: Additionally, the user may select a bit flip injection rate to control how fast data is corrupted (4 in Figure 1b). To simulate future error rates, bit flips are synthetically induced into arbitrary data words in the indexes. While running a workload, at the selected bit flip rate a random 8-byte word is XORed against a bit pattern of 1 - 5 ones to simulate accordingly many bit flips.

Performance Workload: The performance workload compares the throughput between two tree variants (indexes). It runs constant point queries on the selected indexes while corrupting arbitrary nodes randomly. This workload only displays the 2 line graphs (5 in Figure 1b). It is intended to show the performance impact of the different error detection variants using 3 different lines: Blue represents the throughput of successful queries without any errors. Red represents undetected errors (cf. [Kea14b]), which means false positives, false negatives or segmentation faults, while green stands for the amount of errors detected by the according tree variant. As can be seen in Figure 1b trees might have to be rebuilt when encountering a segmentation fault – segmentation faults lead to program termination and are much harder to recover from in contrast to false positives or negatives. Therefore a complete rebuild of the tree is required.

Visualization Workload: The visualization workload additionally shows a graphical representation of the trees themselves – as multi-layered circled graphs – and health indicators – in the form of dials – (6 in Figure 1b). This allows the user to see the current structure of the tree and its evolving over time, i.e. which parts of the tree are corrupted, and which key ranges or nodes are not accessible anymore. Thereby, blue dots represent accessible nodes, black dots stand for corrupted nodes, and red nodes show inaccessible nodes. The health indicators represent the amount of queries which are successful.

Manual Workload: The manual workload also displays all graphs, and there is also a constant load of point queries. However, the user may directly select a node which she or he wants to corrupt. The corruption is done randomly and visual feedback immediately shows what happens due to the corruption. For instance, when a child pointer was corrupted, the according sub-tree is marked red.

Acknowledgments

This work has been supported by the state of Saxony under grant of ESF 100111037 (SREX) and the German Research Foundation within cfAED cluster of excellence.

References

[BSS08] L. Borucki, G. Schindlbeck, and C. Slayman. Comparison of accelerated DRAM soft error rates measured at component and system level. IRPS, 2008.

[Kea14a] Yoongu Kim et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. ISCA. IEEE Press, 2014.

[Kea14b] Till Kolditz et al. Online Bit Flip Detection for In-memory B-trees on Unreliable Hardware. DaMoN. ACM, 2014.