Exploiting the overlap between temporal redundancy and spatial redundancy in storage system

Pengju Shang, Saba Sehrish, Jun Wang
University of Central Florida

Recent years have seen ever increasing file systems and storage applications employ versioning or log-based techniques to protect data integrity and improve the overall system performance, in addition to traditional database systems. The basic idea behind these techniques is to add extra data redundancy before committing new updates on disks. More specifically, in database systems, the log record is viewed as one kind of temporal redundancy to store copies of different time stamps, which can support rollback or redo services. Similarly, in versioning file systems, previous versions of the data, which is another kind of temporal redundancy, are recorded persistently to keep track of the history changes.

On the other hand, different kinds of RAID are often employed as the underlying storage subsystem in the afore-mentioned cases to guarantee the system reliability and availability with high I/O performance at block-level. RAID system employs another kind of redundancy called spatial redundancy (mirroring or parity) to recover the data in the same time domain. There exist significant overlaps between temporal redundancy and spatial redundancy and redundant information is repeatedly copied at different levels. Existing systems do not consider such overlaps due to the information gap between every two storage layers. By wisely exploiting and eliminating the overlap between the temporal redundancy and the spatial redundancy, we can significantly reduce the disk I/O latency and improve the storage space efficiency. One main idea is to create new but compact data using coding techniques. Given the increasing popularity of multi-core/many-core systems, the excessive computing power is used to trade for high data-access speed.

We implement this idea and build new Transactional RAID systems (TRAID in brief) for Databases. Specifically, RAID5 (parity redundancy) and RAID10 (mirroring redundancy) are chosen as case studies in the experiments. TRAID is implemented as high-performance storage for transaction processing systems by reducing the log wait time and log size. It is reliable as we also demonstrate that how redo and undo operations in transactional processing are performed correctly, i.e. recovery correctness, along with the ACID semantics in traditional relational database systems. Our TRAID prototypes exploit the existing redundancy in two most commonly used RAID architectures — parity based (RAID5) redundancy and mirroring based (RAID10) respectively, as illustrated in Figure 1 for TRAID5 and TRAID10. Parts of the experimental results are shown in Figure2. Our extensive experimental results demonstrate that TRAID10 (use Berkeley DB as the database) is 43.23% better than RAID10; TRAID5 outperforms RAID5 by 56.89%.

In versioning file systems such as Ext3cow, Elephant, CVFS, etc, given the underlying RAID storage subsystem, we can exploit and eliminate the overlapped redundancy between the above mentioned two to largely reduce the versioning overhead and thereby boost the overall I/O performance. We will develop a new prototype that revisits main versioning file system operations, such as how to create, store and retrieve the old versions, how to implement full versioning mode and sparse (incremental) versioning mode, as well as what the retention polices will be. We will utilize the inode of the file (current version or old version) to tell which blocks belong to the file and implement our redundancy deduplication techniques at block-level rather than other granularity levels (e.g., record-level, page-level, even table-level in Database systems).

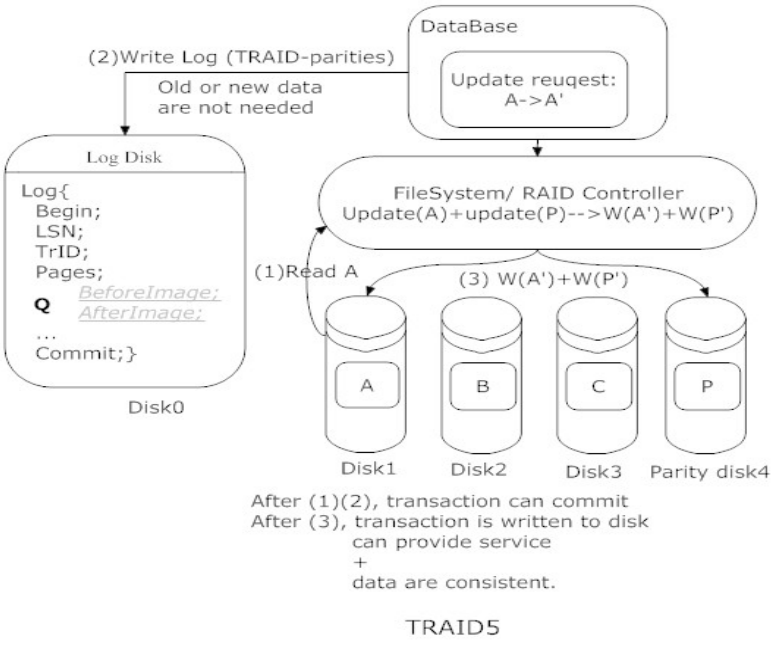

Figure 1. TRAID5 design

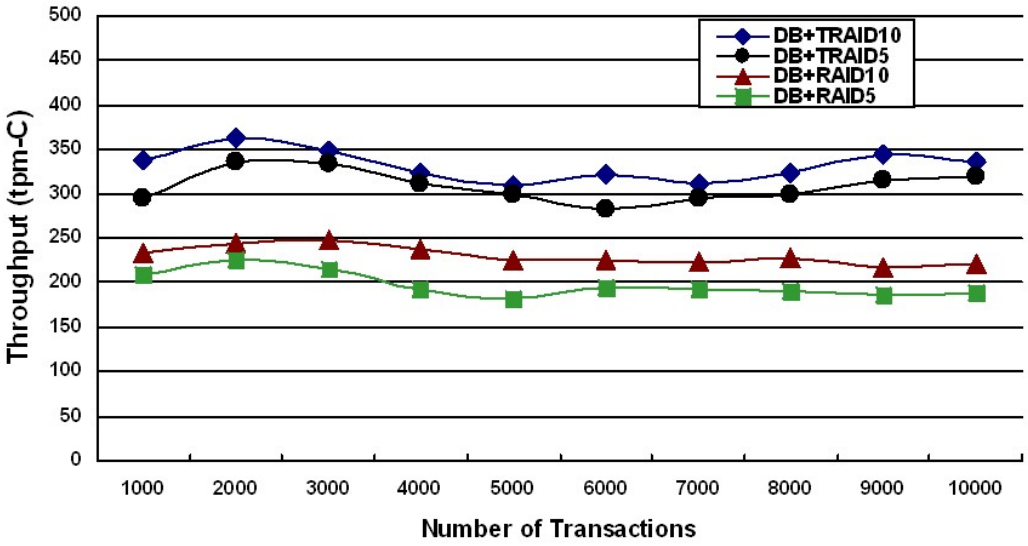

Figure 2. Throughput (TPC-c benchmark)