Profiling and Analyzing the I/O Performance of NoSQL DBs

Jiri Schindler
NetApp
jiri.schindler@netapp.com

ABSTRACT

The advent of the so-called NoSQL databases has brought about a new model of using storage systems. While traditional relational database systems took advantage of features offered by centrally-managed, enterprise-class storage arrays, the new generation of database systems with weaker data consistency models is content with using and managing locally attached individual storage devices and providing data reliability and availability through high-level software features and protocols. This tutorial aims to review the architecture of selected NoSQL DBs to lay the foundations for understanding how these new DB systems behave. In particular, it focuses on how (in)efficiently these new systems use I/O and other resources to accomplish their work. The tutorial examines the behavior of several NoSQL DBs with an emphasis on Cassandra - a popular NoSQL DB system. It uses I/O traces and resource utilization profiles captured in private cloud deployments that use both dedicated directly attached storage as well as shared networked storage.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Measurement, Performance

Keywords

Database systems, NoSQL, storage

1. INTRODUCTION

The performance of structured data management systems has always been determined to a large extent by the architecture and performance of the underlying storage system. The proliferation of the so-called NoSQL databases in the last few years as well as the adoption of Flash memory for latency-sensitive I/O operations has brought about a new model of using storage systems. Traditional relational database systems relied on high availability of centrally-managed, enterprise-class storage arrays and utilized their advanced features such as snapshots and transparent remote site replication. In contrast, the new generation of database systems with weaker data consistency models are content with using locally attached individual storage devices and providing high availability through their own software features and protocols instead.

The majority of existing scale-out clustered NoSQL systems do not manage storage devices directly nor do they worry about RAID configuration alignment to stripe-unit boundaries etc. Instead, they rely on the OS and file system services to do so and use POSIX files as logical containers that can grow to store their data. Since most of these systems run on commodity nodes, The node-local disk file system (e.g., Linux ext3 or xfs) determines the performance and the set of features available to the database system.

These new data management systems are designed to support different workloads compared to traditional relational database systems (RDBMS) optimized for transactional processing with frequent updates. The workloads of these new systems are dominated by high-throughput append-style inserts rather than frequent updates of multiple values in a single transaction and read accesses in support of mostly point queries i.e., queries for a single value. As a result, the data access patterns these new systems generate can be quite different from traditional DBMSes. This work examines whether these new workloads and systems also exhibit different I/O behavior.

2. TRADITIONAL RDBMS

Traditional relational databases with well-defined schema and row-major orientation favor efficient record updates typical for online transactional processing. A single transaction may update few values in a handful of tables, which can result in dirtying many different pages including pages of system-maintained structures such as indexes and materialized views.

Databases use write-ahead logging to record to stable store the changes caused by the executed transactions. Periodically, the log is checkpointed by writing out to stable media dirty pages affected by recent updates. Thus, a checkpoint operation can amortize the cost of writing out a page across many update operations. However, it still results in inefficient random disk I/Os as the dirty pages are written out logically in-place to the data store.

Traditional RDBMSes were built for in-memory data access through page-based cache and optimized for hard disk drive I/O. Their architecture is not optimized for flash memory with efficient random I/O and sub-millisecond access latencies, although than can take advantage of these types of devices for fast logging of every committed transaction.

SIGMETRICS'13, June 17–21, 2013, Pittsburgh, PA, USA.
ACM 978-1-4503-1900-3/13/06.

3. CLUSTERED NOSQL SYSTEMS

NoSQL DB systems run on a cluster of nodes, each running a separate OS instance. They were designed to use directly-attached storage for storing data on each node and to to replicate data across several nodes to prevent data loss when a node fails. Cluster services create a single system image, restore the data from the failed node, and redistribute it to balance load across the cluster.

There are three broad types of NoSQL DBs: key-value stores, document stores, and extensible large-scale columnar data stores. All three types have similar logical organization: a fixed key, typically generated by the system or derived from a user-provided key, followed by the value. In the most basic form, the value is a variable-length set of bytes opaque to the NoSQL DB system.

The internal data organization for NoSQL databases use a distributed hash table or variants of a partitioned B-tree that spread data across nodes. They perform random read I/Os as the system traverses the structure to locate the queried K/V pair(s). Some document stores also create secondary indexes on specific document elements for more efficient execution of range queries i.e., queries operating on a range of keys.

Generally, the I/O patterns of NoSQL DBs resemble those of traditional RDBMSes. They also use write-ahead logging even though they provide much weaker consistency compared to those of traditional RDBMS with ACID properties, Append-style log writes minimize the I/O cost by eliminating in-band B-tree update for systems that use it as their primary storage structure. Additionally, they perform a variant of a group commit, or more precisely, a periodic sync of the log to the disk. It is typically this relatively infrequent (ranging from 100ms up to 10s) sync of the log files that makes NoSQL DBs effective when handling high-throughput update or inserts. However, this comes at a cost of possible data loss in case of a multi-node failure.

NoSQL DB systems such as HBase and Cassandra vertically partition data into column families. They timestamp every value and store changes into an append-only log. Each column family is partitioned horizontally across different nodes with each node hosting partitions for all columns.

A single partition of a column family contains timestamped K/V pairs that include the name of the column for which the value was inserted and are appended at the tail-end of the file. The partition also includes an index, which sorts the pairs lexicographically and points to their current versions. Thus, the access patterns of the third type of NoSQL DBs are similar to those of document stores: Index scans are used to execute point queries. The system first locates the node responsible for the given key range and the node-local index locates the current version of the K/V pair. Table scans allow for serial I/O through the column segments; however, invalid K/V pairs must be skipped.

Since updates are appends to the data stores rather than in-place overwrites, old versions with stale data or deleted values in the column segments need to be periodically garbage-collected. The process called compaction is similar to wholesale rewrite of a column in columnar RDMBSes as it moves live data from their original locations into a new segment, updates the index and deletes the old segment. Even though compaction is similar to checkpointing, the two activities are typically decoupled; compaction can proceed in the background on previously-written segments, while new data is being checkpointed from the log by writing it into a new segment and updating the index structure on the given column partition. Thus, unlike in traditional RDBMS, NoSQL systems also exhibit substantial amount of background I/O activity, which is mostly sequential. While the NoSQL DB distinguishes between foreground I/O, which includes write-ahead logging and read queries, and background activity such as compaction, the storage system is not aware of this distinction although it would leverage it to better schedule individual I/O requests.

4. ANALYZING I/O ACCESS PATTERNS

With the exception of HBase, which uses a distributed file system (HDFS), most NoSQL DBs use locally-attached storage and file systems for storing both data and logs. As stated before, the most critical I/Os for achieving high throughput of inserts and updates are sequential writes to the log. Most systems use memory-mapped access, whereby they treat the log as a memory region that is backed by a file. Individual writes are thus simple writes (typically of a few tens of bytes) followed by a periodic syncing of the file to the disk via the `msync()` system calls. Given the relatively large period between individual calls to `msync()` the small updates are turned into large I/Os with many pages flushed out.

Through profiling and tracing MongoDB and Cassandra NoSQL DBs, we discovered that the log write I/Os are not as sequential nor as big as one would expect. That is because file system fragmentation due to aging and the interactions between the virtual memory subsystem and the file system of the Linux kernel. When the virtual memory is under duress, it may not walk all pages and pick them up in an ascending order to make large I/O possible, subject to fragmentation. Using flash memory, where random I/O is nearly as effective as sequential I/O, for logging is useful irrespective of the faster I/O response time.

NoSQL DBs rely on the services and capabilities of the underlying storage systems. Since those are in most cases Linux local file systems they do not include features like snapshots or transparent backup to a remote location. However, as NoSQL DBs are increasingly deployed for business-critical applications, these features will become more important for providing continuous operation in the face of whole data center unavailability. Similarly, as many NoSQL DBs can benefit from fast access to local flash memory, it is likely we will see an adoption of architectures that provide automatic tiering and movement of data between local storage and networked storage systems. In short, we envision a gradual introduction of features in NoSQL DBs traditionally associated with centrally-managed storage systems.

5. SUMMARY

Even though NoSQL DB systems offer different programming styles and approaches to managing data, their I/O profile does not differ greatly from those of traditional RDBMSes. hey include logging writes dominated by small append-style I/O as well as checkpointing, and index scans that exhibit random I/O. NoSQL DBs for semi-structured data with columnar organization also exhibit large I/O for table scan reads or column compaction. What differs most is their approach to managing data. However, as their role in the enterprise shifts, so will the deployment model and the reliance on advanced data management features.