I/O Characteristics of NoSQL Databases

Jiri Schindler
NetApp Inc.
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 work aims to review the architecture of several existing NoSQL DBs with an emphasis on how they organize and access data in the shared-nothing locally-attached storage model. It shows how these systems operate under typical workloads (new inserts and point and range queries), what access characteristics they exhibit to storage systems. Finally, it examines how several recently developed key/value stores, schema-free document storage systems, and extensible column stores organize data on local filesystems on top of directly-attached disks and what system features they must (re)implement in order to provide the expected data reliability.

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. Instead, they rely on the OS and file system services to do so and use POSIX files as logical containers to store their data. 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 row-major oriented RDBMS or vertically-partitioned columnar DBs. The workloads of these new systems are dominated by high-throughput append-style inserts and read accesses in support of point or range queries. 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 on-line transactional processing. A single user-visible transaction updating only a few values, 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.

Another class of systems place vertically partitioned data into individual columns that are normalized and sometimes compressed for efficient execution of table scans. In columnar DBs, checkpointing is more expensive compared to row-oriented RDBMS. Inserting new or updating existing values may cause the entire column to be updated. This entails reading the uncached pages from the storage system and writing them out in the new format with large and efficient disk I/O. Regardless of their internal data organization, 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.

3. CLUSTERED NOSQL SYSTEMS

NoSQL DB systems run on a cluster of nodes, each running a separate OS instance. They typically use directly-attached storage for storing data on each node. They 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

Proceedings of the VLDB Endowment, Vol. 5, No. 12
Copyright 2012 VLDB Endowment 2150-8097/12/08... $ 10.00.

a variable-length set of bytes opaque to the NoSQL DB system. The difference between the first type and the second type is that document stores understand the format of the "values", which can be XML documents or a JSON (java-script object notation) object. Examples of these include MongoDB or CouchDB.

The internal data organization for NoSQL databases reflects their prevalent use case: point queries with inserts and updates. They use a distributed hash table or variants of a partitioned B-tree that spread data across nodes. Even though these DBs do not use query plan operators like traditional RDBMSes, their access patterns resemble index scans. They perform random read I/Os as the system traverses the structure to locate the queried K/V pair(s). Some also create secondary indexes on specific document elements for more efficient execution of range queries.

NoSQL DBs 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. Asynchronous checkpointing amortizes the cost of updating the main data store structures. Some systems employ multi-version concurrency control (MVCC). In those systems, checkpointing removes older versions no longer needed for conflict resolution. For those reasons, checkpointing is generally expensive and I/O intensive operation as it may involve extensive restructuring of the main store. In that regard, it is similar in terms of cost and complexity to merging differences to the main store of RDBMSes with columnar organization.

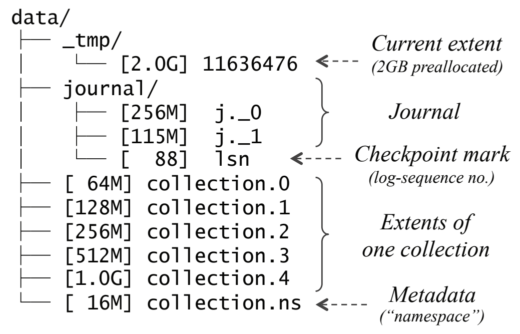
The third type of NoSQL DB systems such as HBase and Cassandra use vertically partition data into column families. They use MVCC and log changes to the data into an append-only log. Each column family is partitioned horizontally across different nodes with each node hosting partitions for all columns. There is no transactional support for atomic updates.

A single partition of a column family contains time-stamped K/V pairs that 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. This allows for in-order traversal of the data and efficient location of the most-recent version of the data. 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.

As with columnar DBs, checkpointing can be expensive. 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 whole-sale 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.

4. MANAGING STORAGE

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. Figure 1 illustrates a typical data organization on an example of a MongoDB. It shows the structure and file sizes for a single collection (or a database) with separate directories for logs and data. Variable-length extents comprise a collection with each extent stored in a separate file that can grow up to 2 GB in size (chosen to work on 32-bit Linux).



```
data/
├── _tmp/
│   └── [2.0G] 11636476
├── journal/
│   ├── [256M]  j._0
│   ├── [115M]  j._1
│   └── [  88]  lsn
├── [ 64M] collection.0
├── [128M] collection.1
├── [256M] collection.2
├── [512M] collection.3
├── [1.0G] collection.4
└── [ 16M] collection.ns
```

Figure 1: Example of MongoDB directory structure with logs and extents of data store.

MongoDB rotates through two logs; one log is being written to while the other is checkpointed. It caps the log size at 256 MB. In the default organization in Figure 1, the same local file system would share physical resources for both logs and data. It is possible to configure MongoDB such that logs use a different mount point/volume just as RDBMSes would typically do.

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 to provide operational continuity 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 with row-based or columnar organization. They 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 may 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.

BIOGRAPHICAL SKETCH

Jiri Schindler is a member of technical staff at the NetApp Advanced Technology Group where he works on storage architectures integrating flash memory and disk drives in support of applications for management of (semi)structured data. He earned his PhD from Carnegie Mellon University and M.Eng. and B.S. from MIT. While getting his PhD, he and his colleagues designed and built the Fates (Clotho, Atropos, and Lachesis) system for efficient execution of mixed database workloads with different I/O profiles. Jiri is also an adjunct professor at the Northeastern University where he teaches storage systems classes. He actively works with graduate students and supervises PhD theses.