FAWN : A Fast Array of Wimpy Nodes

Presented by Akhilesh Kotla
Three metrics to minimize

**Memory overhead** = Index size per entry
- Ideally 0 (no memory overhead)

**Read amplification** = Flash reads per query
- Limits *query throughput*
- Ideally 1 (no wasted flash reads)

**Write amplification** = Flash writes per entry
- Limits *insert throughput*
- Also reduces *flash life expectancy*
  - Must be small enough for flash to last a few years
Motivation

Why FAWN?

• We try to use energy efficient processors as nodes to make the cluster fast

What problem are we trying solve?

• Metadata is too large
• Improving power consumption
• Efficiently manage high query rates
Architecture

Figure 1: FAWN-KV Architecture.
Key-Design Choice

• Use of log-structured data stores

• This makes storing, reading, deleting easy
FAWN DataStore

<table>
<thead>
<tr>
<th>Constrained DRAM</th>
<th>Avoid Random Write in Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only maintain Hash Table in DRAM</td>
<td>Append-only log-structured filesystem</td>
</tr>
</tbody>
</table>

**DRAM**

- **160-bit Key**
  - KeyFrag
  - **Hash Index** $2^i$ buckets
    - **Chained hash entries in each bucket**

**Flash**

- **Log Entry**
  - Key
  - Len
  - Data

- **Data Log**
  - Offset
    - **Inserted values are appended**
1. “The workloads these systems support share several characteristics: they are I/O, not computation, intensive, requiring random access over large datasets, ..., and the size of objects stored is typically small.” Read the above statement, indicate why workloads of these characteristics represent a challenge to the system design?
Ans: These systems have to support large sets of metadata which are small in size but huge in number (billions).

When the object becomes small, the respective metadata size becomes more compared to the size of the data.

To keep all this metadata in memory (DRAM) is expensive and makes it challenging to design the system.
2. “The key design choice in FAWN-KV is the use of a log structured per-node datastore called FAWN-DS that provides high performance reads and writes using flash memory.” “These performance problems motivate log-structured techniques for flash filesystems and data structures” What key benefit does a log structured data organization bring to the KV store?
Ans: By using a log-structured datastore, random writes are replaced with sequential writes. This way the throughput of the SET and GET actions increases significantly.
3. “To provide this property, FAWN-DS maintains an in-DRAM hash table (Hash Index) that maps keys to an offset in the append-only Data Log on flash.” What are potential issues of the design? [Hint: consider metadata size and volatility of DRAM.]
Ans: To reduce the memory requirement.

For large metadata, the size of DRAM is not sufficient to hold the metadata. Therefore it maintains a hash table in the DRAM to map keys to offset in the Data log on flash.
4. “It stores only a fragment of the actual key in memory to find a location in the log;” Is there correctness concern in this design?
Ans: No. In terms of correctness, this design is not wrong.

Because first the FAWN-DS checks the first 5 bits to find a location in the log and verifies that the key it read was correct.

If the key matches then there would not be any problem, otherwise if the key does not match then the key fragment is searched again.
5. “Basic functions: Store, Lookup, Delete” Use Figure 2(a) to explain how these basic functions are executed?
Figure 2: (a) FAWN-DS appends writes to the end of the Data Log. (b) Split requires a sequential scan of the data region, transferring out-of-range entries to the new store. (c) After scan is complete, the datastore list is atomically updated to add the new store. Compaction of the original store will clean up out-of-range entries.
Ans: **Store**

Store adds an entry to the log, updates the corresponding hash table entry to point to this offset within the Data Log, and sets the valid bit to true.
Lookup

Lookup retrieves the hash entry containing the offset and index into the Data Log and then returns the value (data blob)
Delete

Delete invalidates the hash entry by clearing the valid flag and adds a Delete entry to the end of the log.
6. “As an optimization, FAWN-DS periodically checkpoints the index by writing the Hash Index and a pointer to the last log entry to flash.”. Why does this checkpointing help with the recovery efficiency? Why is a Delete entry needed in the log for a correct recovery?
Ans: Deletes do not immediately reclaim space. They periodically perform garbage collection and then reclaim the space through compaction.

The delete entry is necessary for fault tolerance.