FAWN

FAST ARRAY OF WIMPY NODES

VIRAJ SULE
• FAWN is a cluster architecture for **low-power** data-intensive computing.

• FAWN-KV is a consistent, highly available and high performance key-value storage system built over FAWN prototype.
(1) “The workloads these systems support share several characteristics: they are I/O, not computation, requiring random access over large datasets, they are massively parallel, with thousands of concurrent mostly independent operations 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?

• In I/O, CPU has to stall while waiting for data to be loaded or unloaded.

• Random access over large datasets would be inefficient in case we need to access the data sequentially.

• Size of objects is small then there will large amount of data; consequently, large metadata in terms of numbers.

• Systems requiring large clusters includes DRAM which are expensive and consume large amount of power.
(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?

• Log structured data organization provides with high write throughput because all the updates on data and metadata are written in sequential order in the log.
(3) “To provide this property (Writes are sequential and Read is random access), 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?

• Large number of key-value pairs will lead to large metadata.

• As DRAM is volatile, the hash table will be lost once we turn OFF the cluster.
(4) “It stores only a fragment of the actual key in memory to find a location in the log;” Is there a correction concern in this design?

• No

• With the 15-bit key fragment, only 1 in 32,768 retrievals from the flash will be incorrect.

• minor issue over drastically reduced memory requirements.
(5) “Basic functions: Store, Lookup, Delete” Use Figure 2(a) to explain how these basic functions are executed?

- **Store**
  - It appends an entry to the log, updates the corresponding hash table to point this offset within the Data Log, and sets the valid bit to true.

- **Lookup**
  - Retrieve the hash entry containing the offset, indexes into the Data Log, and returns the data blob.

- **Delete**
  - Invalidates the hash entry corresponding to the key by clearing the valid flag and writing a delete entry to the end of data file.
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
(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? How is a KV item deleted from the store?

• After a failure, FAWN-DS uses the checkpoint as a starting point to reconstruct the in-memory Hash Index quickly.
• This can be done because Data Log contains all the information necessary to reconstruct the Hash Index from scratch.
References:

• FAWN paper
• http://muratbuffalo.blogspot.com/2011/02/chain-replication-for-supporting-high.html
• Lectures Slides