Data Placement

Contents

Data Placement Modeling

See Destinations for destination definition and characteristics.

The Data Placement Problem: Given N storage tiers, a data placement policy P, a cost function F, and a BLOB, a data placement consists of a BLOB partitioning and an assignment of those parts to storage tiers that satisfies the constraints of the data placement policy and that minimizes the cost function.

- Epoch - interval within which we update destinations (status)

- Static (e.g., time interval or number of operations)
- Dynamic, i.e., computed by the delta of status

- Placement window - interval within which we make DPE decisions
  - Timer expired or I/O op. count reached, whichever comes first
- Google-OR tools (https://developers.google.com/optimization)

The Data Placement Loop

A *tiered schema* $TS(b)$ of a BLOB $b(>0)$ is a decompostion $b = s_1 + \cdots + s_k,\ s_i \in \mathbb{N} \setminus \{0\}$ together with a tier mapping $(s_1, \ldots, s_k) \mapsto (t_1(s_1), \ldots, t_k(s_k))$.

A sequence of buffer IDs $(ID_1, \ldots, ID_A)$ *conforms* to a tier assignment (s, t), iff $s = \sum_{i=1}^{A} Size(ID_i)$ and $\forall i\ Tier(ID_i) = t$.

An *allocation of a tiered schema* is a sequence of buffer IDs which is the concatenation of conforming tier assignments.

1. Given: a vector of BLOBs $(b_1, b_2, \ldots, b_B)$
2. The DPE creates tiered schemas $TS(b_i),\ 1 \leq i \leq B$
3. The tiered schemas are presented to the buffer manager, which, for each tiered schema, returns an allocation of that schema (or an error), and updates the underlying metadata structures.
4. I/O clients transfer data from the BLOBs to the buffers.

Problem to solve

Input:

- Vector of BLOBs $(b_1, b_2, \ldots, b_B)$
- Vector of destinations $(d_1, d_2, \ldots, d_D)$
- Vector of destination remaining capacities $Rem[d_k]$, $1 \leq k \leq D$
- Vector of destination speeds $Speed[d_k]$, $1 \leq k \leq D$

Output:

- A (sparse) placement matrix $X = (x_{ik})_{1 \le i \le B, 1 \le k \le D}$ where x_{ik} represents the fraction of b_i that is placed on destination d_k.

Objectives

Round-robin

1. Do we split? (coin flip)

- Blob split distribution (Uniform split distribution)
  - Do not split blob, if its size is <= 64KB
  - Blob size (64KB, 256KB] -> [2, 4]
  - Blob size (256KB, 1MB] -> [2, 4, 8, 16, 32]
  - Blob size (1MB, 4MB] -> [2, 4, 8, 16, 32, 64, 128]
  - Blob size (4MB, infinity) -> [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

2. Assign to "next" target

question: do we check total target capacity against blob size? (If not enough capacity, what is the possible error handling? going to node level or trigger buffer organizer? No decision required right now)

Random

1. Do we split? (coin flip) Split distribution refers to Round-robin

2. Sort the targets by remaining capacity and construct a multimap by <remaining_capacity, target> pair

3. Choose a destination that can accommodate the input at random => roll the dice

Minimize I/O Time

Constraints:

- The matrix entries must be non-negative, i.e., $x_{ik} \ge 0$
- Each blob must be placed in full, i.e., $\sum_{1 \le k \le D} x_{ik} = 1$ for all $1 \le i \le B$

- The placement must not exceed the remaining capacity, i.e., $\sum_{1\leq i\leq B} b_i x_{ik} \leq Rem[d_k]$ for all $1 \leq k \leq D$

Cost Function: $\sum_{i,k} \frac{b_i x_{ik}}{Speed[d_k]}$

Problem: Minimize the cost function!

Maximize Bandwidth

Input:

- Vector of BLOBs $(b_1, b_2, \ldots, b_B)$
- Vector of destinations $(d_1, d_2, \ldots, d_D)$
- Vector of destination remaining capacities $Rem[d_k]$, $1 \leq k \leq D$
- Vector of destination speeds $Speed[d_k]$, $1 \leq k \leq D$

Output:

- Vector of placements (vector of sparse vectors aka a sparse matrix)

Constraints:

- The matrix entries must be non-negative, i.e., $x_{ik} \geq 0$
- Each blob must be placed in full, i.e., $\sum_{1\leq k\leq D} x_{ik} = 1$ for all $1 \leq i \leq B$
- The placement must not exceed the remaining capacity, i.e., $\sum_{1\leq i\leq B} x_{ik} \leq Rem[d_k]$ for all $1 \leq k \leq D$

Cost Function: $\frac{\sum_{i,k} b_i x_{ik}}{\sum_{i,k} \frac{b_i x_{ik}}{Speed[d_k]}}$

$$\left(\sum_{i,k} b_i x_{ik} = \sum_i b_i\right)$$

Problem: Maximize the cost function!

Although the cost function is non-linear, it can be solved by linear programming. See Ratio Objectives.

Constraints

Minimum Remaining Capacity Constraint

There must be at least 10% of the total capacity remaining in each destination.

$$\sum_i b_i x_{ik} \leq Rem[d_k] - 0.1 \cdot Cap[d_k] \ \ \forall k = 1, \ldots, D$$

Remaining Capacity Change Threshold

The placement must not exceed 20% of the remaining capacity in each destination.

$$\sum_i b_i x_{ik} \leq 0.2 \cdot Rem[d_k] \ \ \forall k = 1, \ldots, D$$

Placement Ratio

Place at least 10 times as much data in destination d_{k_2} as in d_{k_1}

$$10 \sum_i b_i x_{ik_1} \leq \sum_i b_i x_{ik_2}$$

Other Potential Objectives

- Data balance: even distribution of data across media
- Load balancing: efficiently distribute I/O requests across media
- Throughput max.: optimize overall I/O throughput by taking advantage of the fastest tiers
- Fault tolerance: avoid data loss

- Maximize balance in destinations
  - Load: the number of (queued) requests
  - Capacity: the data payload (bytes)
- Maximize I/O "speed"
  - Maximize I/O bandwidth

- Minimize access latency
    - Device latency
    - Device load
- Maximize access concurrency
    - Device concurrency
    - Tier concurrency
    - Cluster concurrency (nodes)

- Maximize data locality
  - Spatial
    - Belongs to same logical unit (e.g., file, object, table, dataset, etc)
  - Temporal
    - Anticipation of future accesses
    - Historical accesses
- Maximize system utilization
  - Device
  - Tier
  - Node

Experiment Setup

- Produce graphs
  - x axis: #targets: 512, 1024, 2048, 4019, 8192
  - y axis Time elapsed in μs (solver time + I/O time estimations)

- Additionally collect:
  - How did each blob got split
  - Target remaining capacity at the end (y-axis MBs, x-axis targetIDs)
- 1st test case: scaling the number blobs, 10GB total size of IO
  - Small blobs: random(4KB,64KB)
  - Medium blobs: random(64KB-1MB)
  - Large blobs: random(1MB-4MB)
  - Xlarge blobs: random(4MB,64MB)
  - Huge blobs fixed at 1GB
- 2nd test case: scaling the blob size
  - 1000 total #blobs
  - Fixed size of: [4KB, 64KB, 1MB, 4MB, 64MB]
- Cluster configs
  - Capacity
    - DRAM targets: 128MB
    - NVMe targets: 1024MB
    - SSD targets: 4096MB
    - HDD target: 16384MB
  - BW
    - DRAM: 8GB/s
    - NVMe: 3GB/s
    - SSD: 550MB/s
    - HDD: 120MB/s
- Grouping based on target type
  - Homogeneous: total #targets divided by #groups (e.g., 1024/4,...)
  - Heterogeneous: 10% DRAM, 20% NVMe, 30% SSD, 40% HDD

Questions

- Is it a good idea to place a BLOB across tiers (as a matter of policy rather than by accident)?

This page was last edited on 2 September 2020, at 02:35.