LSM Tree Implementation

Nicolas Drizard

TF: Lukas Maas
May, 3^{rd} 2015

Outline

Table of Contents

LSM-Tree

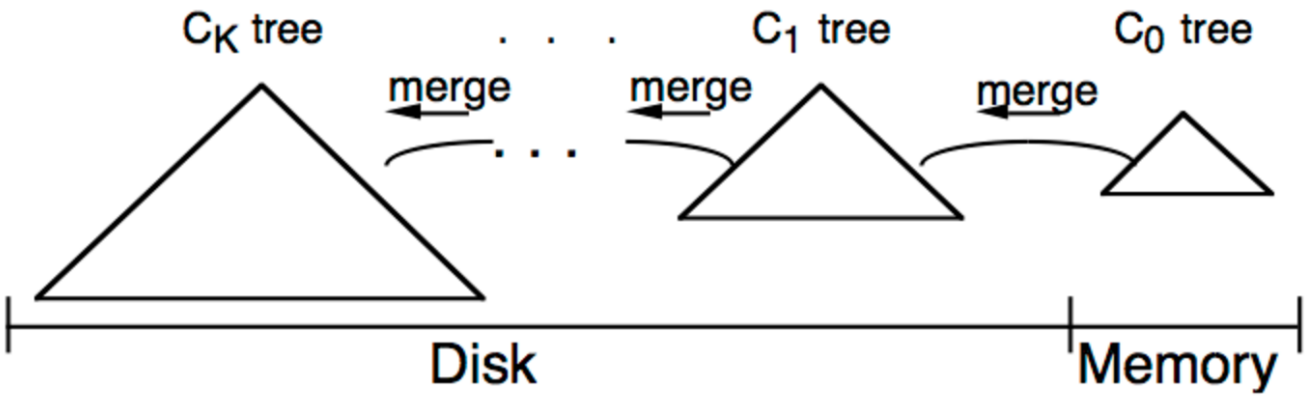

Figure: LSM Tree structure

Table of Contents

Layers Architecture

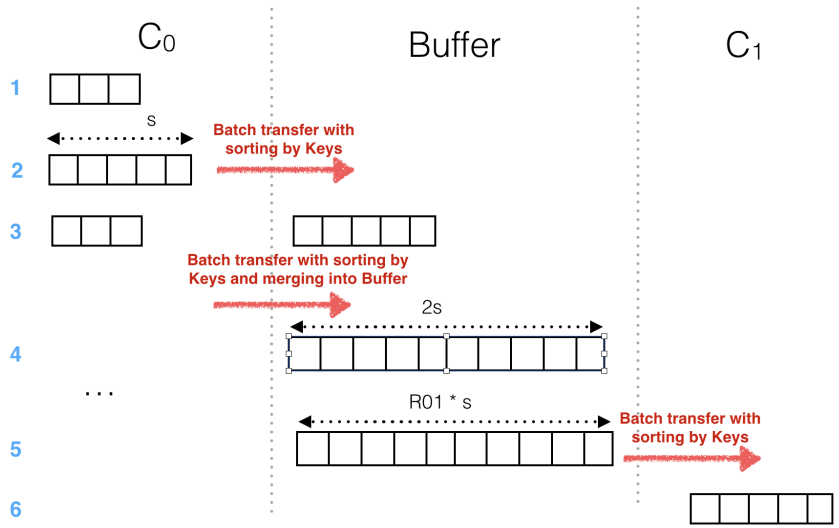

Figure: Merging Strategy

Operations

- READ
  1. Linear search in C_0
  2. binary search in other components
- APPEND (insert, update, delete)
  1. Linear search in C_0 (removing/updating keys)
  2. Merging if full

Each component contains two arrays: keys and values.
C_0 and buffer on memory

Optimizations

- Memory Mapped File
- Bounds Check
- Fault tolerance strategy
- Bloom Filter

Parallel Design

Two approaches are considered:

- Killing threads operating on deeper component once the key is found in its lowest component.
- Threads communication through a shared variable storing the lowest component on which the key has been found so far

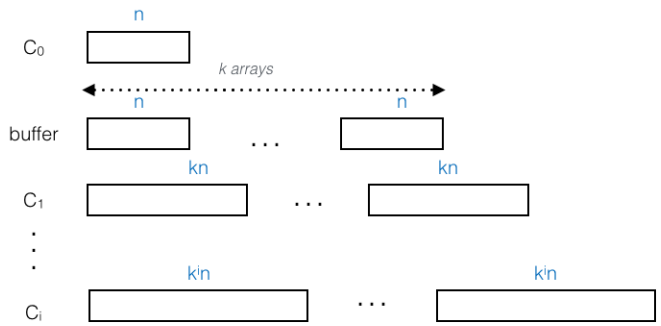

Figure: Alternative Merging Strategy

Table of Contents

Set Up

- 1 000 000 keys inserted
- keys are int, values string of size 32

Set Up

5 different LSM structures used:

1. C_0 size $= 1000$; size ratios $= 3$
2. C_0 size $= 2000$; size ratios $= 3$
3. C_0 size $= 1000$; size ratios $= 3$ and then 9
4. C_0 size $= 1000$; size ratios $= 5$
5. C_0 size $= 500$; size ratios $= 5$

Results

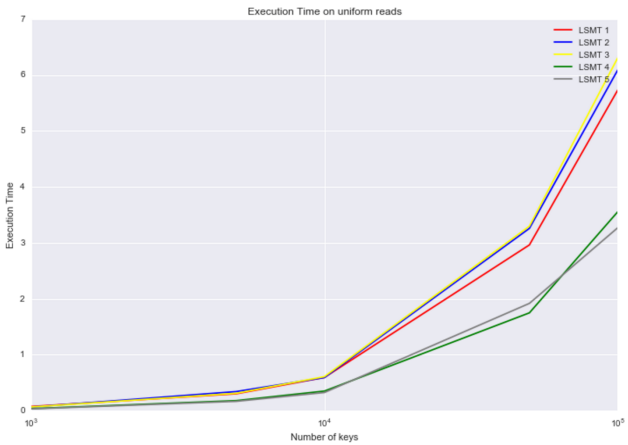

Figure: Uniform Reads

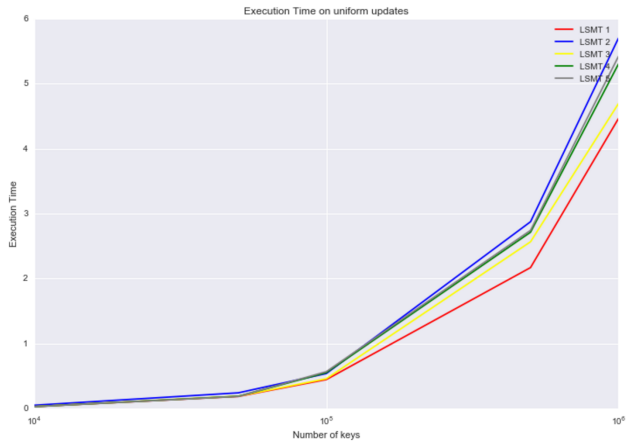

Figure: Uniform Updates

Results

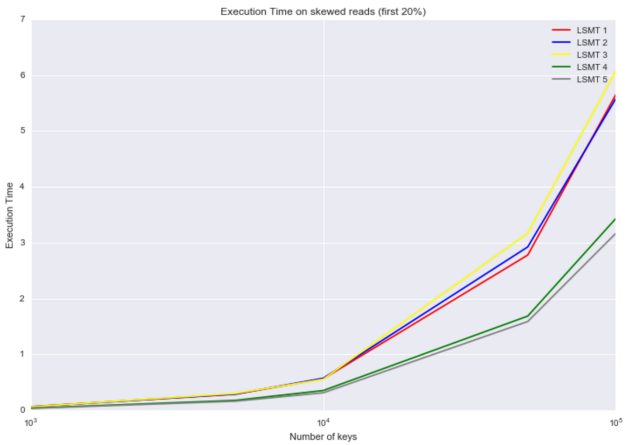

Figure: Skewed reads (20% start)

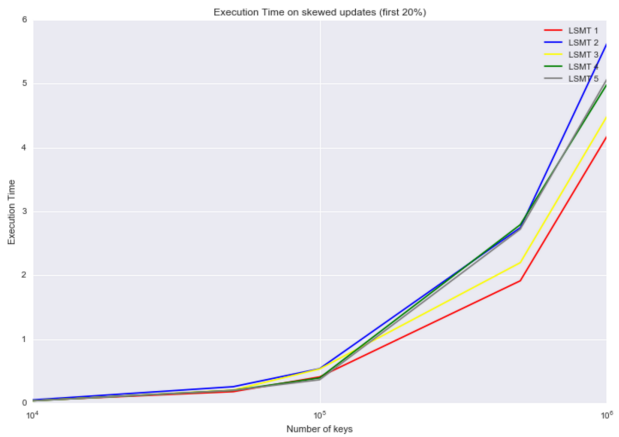

Figure: Skewed updates (20% start)

Results

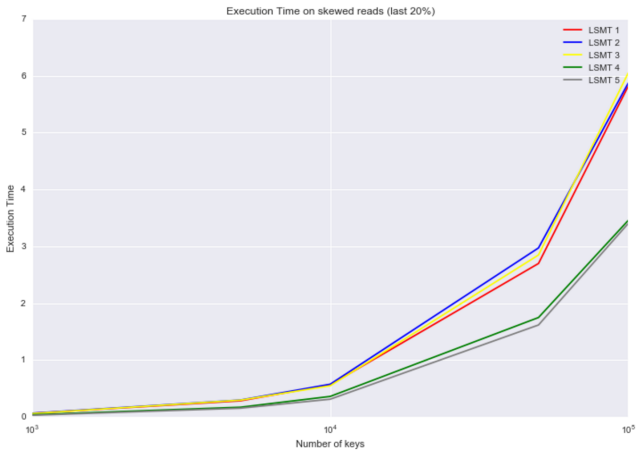

Figure: Skewed reads (20% end)

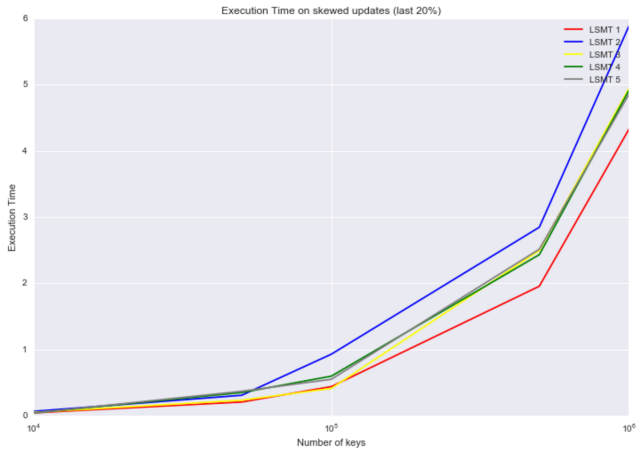

Figure: Skewed updates (20% end)

Results

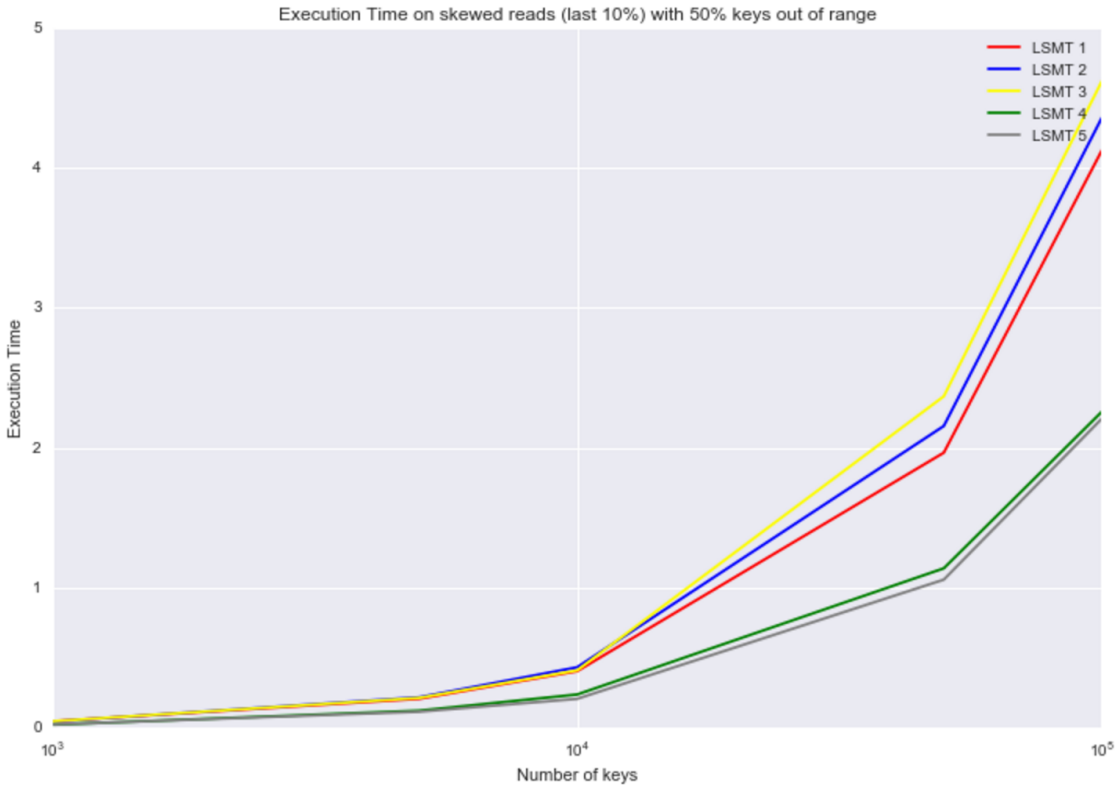

Figure: Skewed reads (10% end) and 50% out of range

Table of Contents

- Experiments on Bloom Filter and Parallel implementation
- Possibility to choose the merging strategy