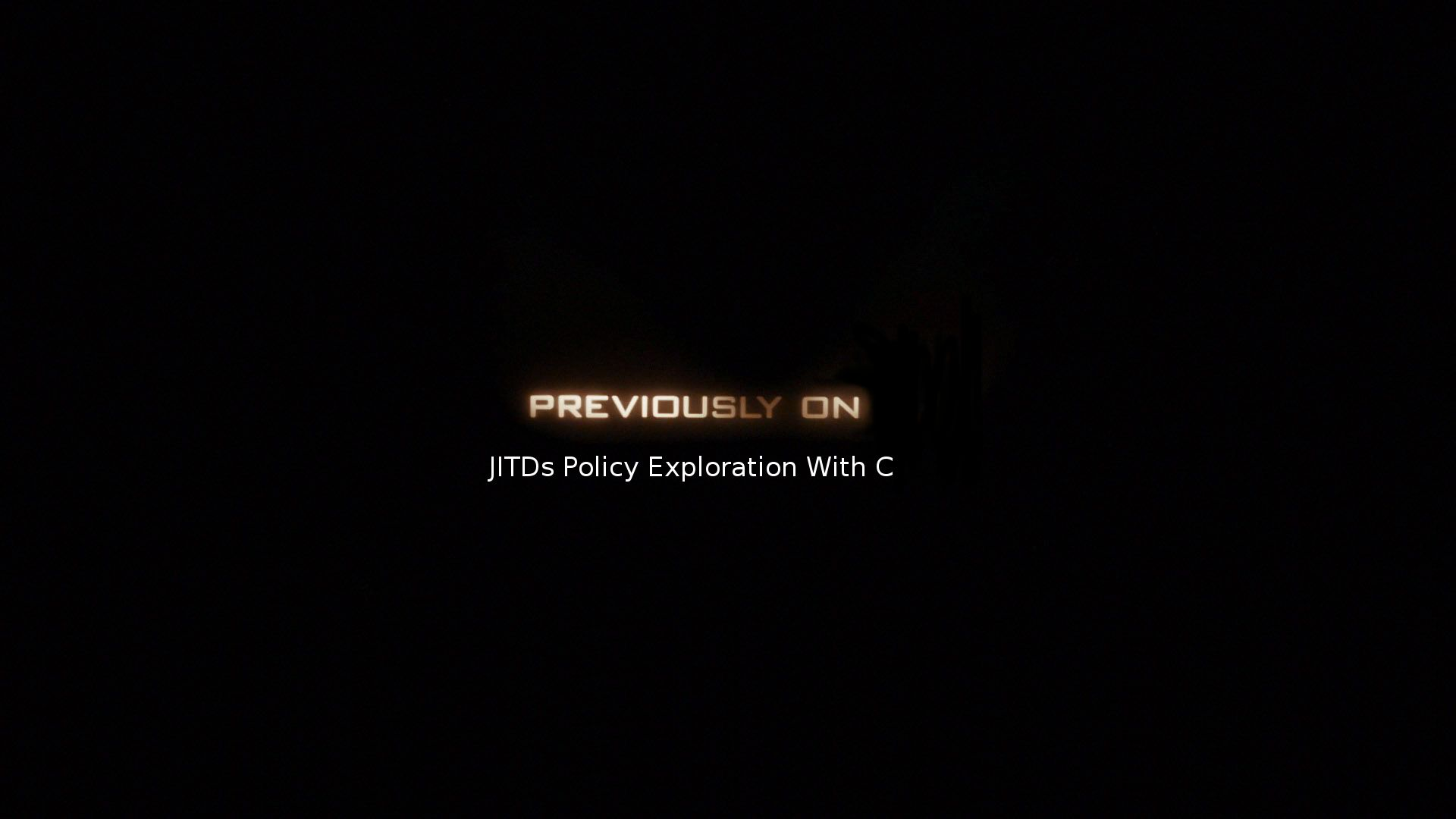
# JITDs Policy Exploration C

Team Twinkle

Nov 4, 2015


PREVIOUSLY ON
JITDs Policy Exploration With C

# Recap from Last Week

- How can we make things more interesting?
- Exploring a new paradigm of smart(er) JITDs + Policies
- Zipfian distributions & why they are interesting
- A proposed policy based on Zipfian and smart(er) JITDs
- Best case scenario and our actual aim
- A little bit of math & theory
- A bit more in depth look into the actual policy

# Implementation Details

**Naive implementation**

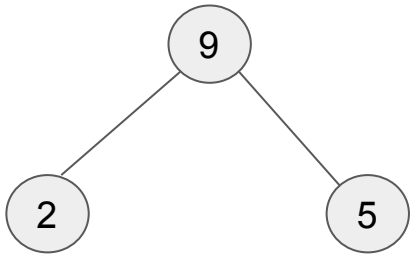
- Encode the read-count in the cog.(Make an extra element to store it).
- Increment the counter when ever the cog is accessed.
- Identify the cog with maximum read.
- Perform splay operation on that particular cog.
- Repeat it for the particular initially determined interval.

# Challenges

- Challenges?
  - How to rewrite the rewrite of read counts splay is done ?
  - How can we change the interval in a smart way so that the number of splays can be adjusted so that convergence is attained faster.
- Make smart decision on when to splay.

# Rewrite policy for readcount

- On each rotation maintain the invariance or the cumulative read count updated.
- Using cumulative read count make this problem a lot easier to work on.
- Figure showing cumulative read count.




- cumulative_root_count = lhs_count + rhs_count +root_count
- root_count = cumulative_root_count -( lhs_count + rhs_count )

# Updated Cog Structure.

struct { struct cog *lhs; struct cog *rhs; long sep; **long rds**; } btree;

struct { int start; int len; buffer records; **long rds;** } array;

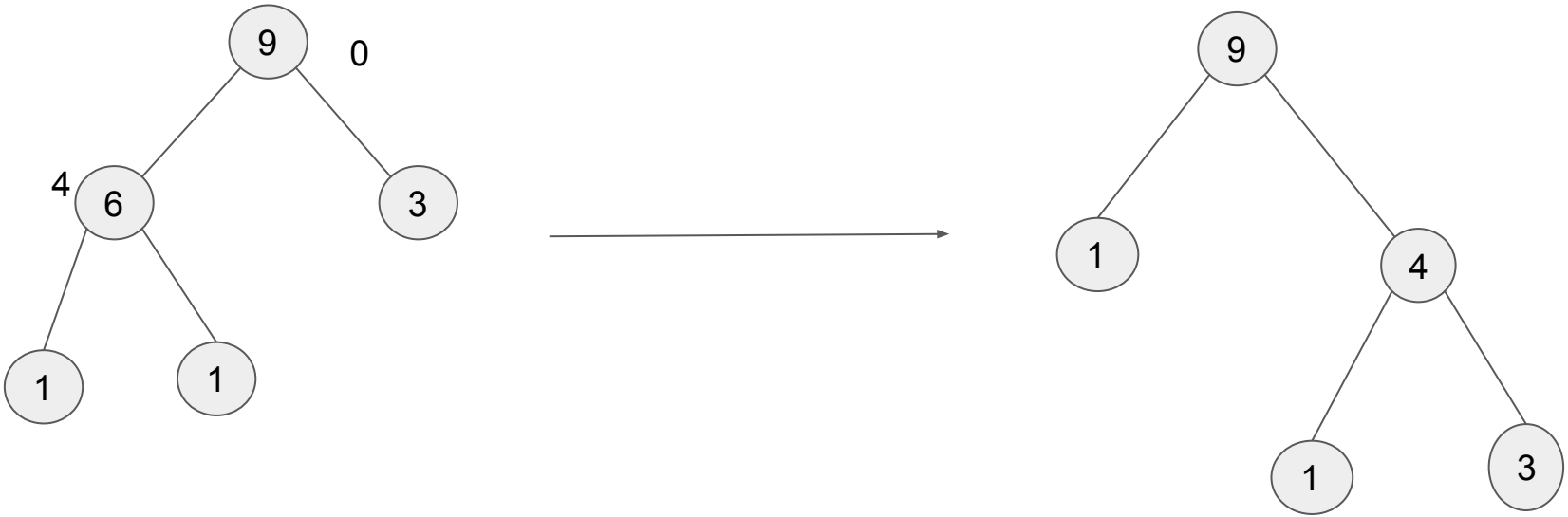
struct { int start; int len; buffer records; **long rds;** } sortedarray;

Memory overhead is only one extra long value.

# Sample code for Zig operation.

```
struct cog *zig(struct cog *root, struct cog *node) {

#ifdef __ADVANCED

  long root_count = root->data.btree.rds - (root->data.btree.rhs->data.btree.rds+root->data.btree.lhs->data.btree.
rds);

  long node_count = node->data.btree.rds - (node->data.btree.rhs->data.btree.rds + node->data.btree.lhs->data.
btree.rds);

  root->data.btree.rds = root->data.btree.rhs->data.btree.rds + node->data.btree.rhs->data.btree.rds+root_count;

  node->data.btree.rds = node->data.btree.lhs->data.btree.rds + root->data.btree.rds;

}
```

# Read Count Updation for Zig Operation



# Splaying heuristically

- Next big idea is varying splay interval
- Why vary the interval?
- More number of splays initially.
- Do the splay for particular number of levels passed as arguments.
- No need of splay after the convergence.

# How to do this?

- Keep track of the number of splays.
- If it is beyond the maxcount then it is an indication of further away from convergence. Decrease interval and splay more often.
- If the count is less it is an indication maximum count nodes are at the top of tree.Increase splay and splay less often.

# Identifying the data of interest:

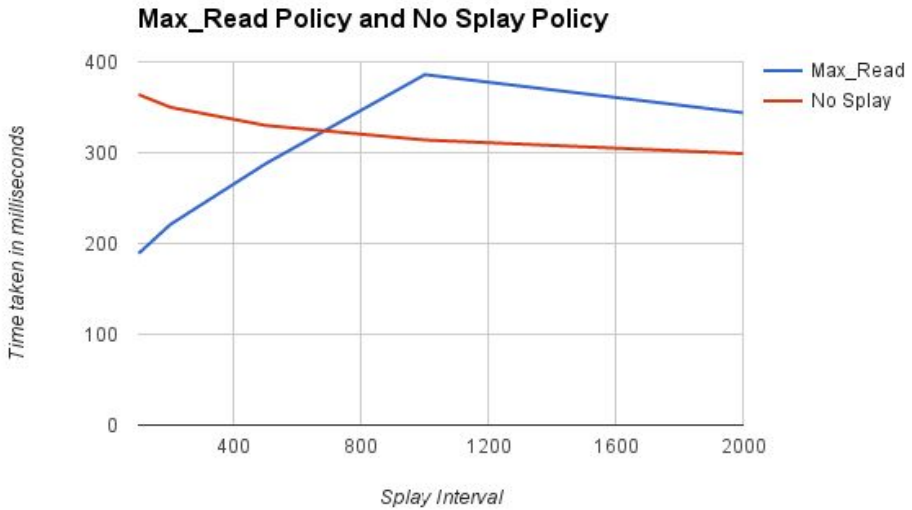
- For a 100000 elements and CDF .5:
  - n = 236.4 levels = 8
- CDF .6
  - n = 793.3 levels = 10
- CDF .7
  - n = 2658.8 levels = 12
- CDF .8
  - n = 8908.8 levels = 14

# Splaying over different levels.

- Use the formula to identify the levels then splay across the levels.
- Splay recursively across the different levels until the end of level of interest.
- Keep track of the number of splays after each interval and use it for updating interval.

# Interesting results

- Naive implementation performance improved to almost 50%.
- Graphs for splaying for max reads with different splay interval compared with no splaying.
- Splay for 10000 reads with selectivity of 100 reads.
- Splay interval varied from 100 to 2000.
- Found that splaying more often is good initially.


Max_Read Policy and No Splay Policy
Time taken in milliseconds
400
300
200
100
0
400
800
1200
1600
2000
Splay Interval
Max_Read
No Splay

# Future Work

- Working on making the splay adaptive rather than doing it for constant interval.
- Finding the best algorithm to update splay interval.
- If time permits figure out if we can have plug-in a machine learning model to update the splay interval learning on the data distribution.

# Questions ?