Tuning SNAP Segment Cache

June 2, 2018

There are a handful of **key** factors that have a huge impact on Query performance. On the physical execution side the **fast** processing of SNAP Segments is critical. The aspect that has a big impact on SNAP Segment processing is the location of the Segment w.r.t. the **Task** that is scanning it: this is because the movement of Segments from **deep store** to **local store** is significantly more expensive than scanning from **local disk**, and correspondingly scanning blocks from OS cache is much faster than disk reads(spinning or SSD).

Hence ensuring Tasks get scheduled such that they do *local* scanning of Segments is critical. Nothing new or surprising here, the Delay Scheduling Paper([1]) describes how the **Fair Scheduler** can be configured to ensure a desired **locality(** λ**)**. Here we briefly describe the **workload model** used in the paper, and how to tune SNAP based on it.

Fair Scheduler's Workload Model

A Workload is described by the following parameters:

Name	Description
M	number of processing nodes in the cluster
L	number of cores per processing node
S	$= M * L$ total number of processing slots
R	the *replication* factor of data Segments/Blocks
f	is a measure of concurrency of the workload, represented as the fraction of the total slots(S) allocated to each Query/Job.
N	is a measure of the work of each Query/Job, represented as the average number of segments/blocks processed in a Query.
T	represents the average time taken to process a task on the cluster; in a typical SNAP deployment the expectation for this will be a few $100 msecs$
D	is a measure of the delay scheduling introduces.

An explanation about parameter 'f'

- As a system administrator estimate the number of concurrent queries(C) running in the system on average.
- assuming all queries are equally important and have similar resource needs we assume each query can use $F = S/C$ slots

- we estimate $f = F/S$

When $D = 0$ the most locality we can achieve is $1 - (1 - f)^(RL)$. Figure 3 in the *Delay Scheduling* paper gives a sense about the locality for this case. Increasing R and L helps locality, but as f increases locality drops sharply. But we would typically not run with $D = 0$.

An explanation about parameter 'N'

N is impacted by the kind of queries in the workload, for example:

- the typical Query involves scanning a small portion(like last 3 months) of the segments of the entire multi-dimensional space.
- keep in mind N is impacted by SNAP Segment sizes; larger Segments will keep N lower but than may increase T to unacceptable values.
  - but look for cases that have a happy comprise: queries typically have a filter on a 'low' cardinality column(like country, part category,...) which ensures average scan times are low even for large Segments.
- As we will show increasing N(and keeping smaller segments) improves locality, but causes more *waves* to run for a Query which will adversely impact Query performance.

The Formula for locality

Formula 2 in the paper, reproduce below provides the mechanism by which we can tune an environment for a given locality expectation (λ)

$$D >= -\frac{M}{R} \ln\left(\frac{(1-\lambda)N}{1+(1-\lambda)N}\right) \tag{1}$$

So for a $\lambda = 0.95$ plug in M, R, N into the r.h.s. to get a value. D represents the number of times the scheduler should skip scheduling a non local task for a Query. To translate this to time multiply this by $\frac{T}{S}$ (this represents the rate at slots become free in the cluster). So set the *spark.locality.wait* to this value.

The process of tuning for SNAP

- in case of SNAP, R is initially set to 3 but can increase as the side-effect of performing a non local tasks is to convert a non local node into a local node for the Segment pulled.
- Each Segment is arbitrarily assigned 3 nodes; we ensure the global constraint that segment replicas are evenly distributed among the cluster nodes.
- The goal always is to have a high (λ), and a reasonable D. For example the goal could be $\lambda = 0.95$
  - then for a given cluster of size M we can tweak R and N to keep D low.

- When N is above 50 the multiplicative effect of the inner ratio drops considerably.
  - in this case $\frac{M}{R}$ is a good estimate of D. So for high values of M increasing R will help.
    - *TBD allow R to be user settable*
- When N is low
  - If M is large, try to increase N by reducing Segment sizes. For large M there is less of an issue of running many waves.
  - If M is also small then R is going to have to be close M or D should be allowed to increase.

Here is an **R** script that can be used to investigate the model:

Listing 1: calculating D

```
library(functional)

R <- 3
ld <- 0.99
M <- 100
D <- function(M,N){- M/R * log(((1-ld) * N)/(1 + (1-ld)*N))}
D_given_M <- Curry(D, M=M)
curve(D_given_M, ylim=c(0,M), from=5, to=M, xlab = "N",
      ylab = paste("D | M =", M, "lamda =", ld, "R = ", R))
```

Future Investigations for SNAP

- need a way to handle the growth of R over time. Today we ask customers to do a *clear cache* which resets the system to a clean slate. This obviously has issues, in the roadmap we have plans for a *rebalance* mechanism.
- The model assumes that a Query segment set is randomly chosen from the set of all Segments; this is definitely not true, as Query segment sets are based on **partition pruning**. So ensuring that segments within a partition are evenly distributed will increase the 'nicety' of the distribution: i.e. for the same value of R we will be able to λ locality at a lower level of delay D.

References

[1] Dhruba Borthakur et.al Matei Zaharia. "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling". In: *Eurosys* (2010). URL: `https://cs.stanford.edu/~matei/papers/2010/eurosys_delay_scheduling.pdf`.