Table of Contents

The Rules

This particular game (let's call it a *reading game*) requires N >= 2 players and M >= 6 buckets, and is played in rounds. At the start of the k-th round, a dealer secretly provides each player with m random buckets $i_1, \dots, i_m$, where i is the player and m is a fixed and previously agreed upon number in the range 2 <= m << M (default: 3).

Next, all players simultaneously take their turns to select 1 (one) bucket out of their respective m-sets:

$$s_k^i = S(i_1, \dots, i_m) \tag{1}$$

where s_k^i is the bucket selected by the i-th player.

The dealer then discreetly collects all private selections, sums them up by individual buckets, and

$$a_k^j = \sum_{i=1}^{N} (s_k^i == j)\ ?\ 1 : 0 \tag{2}$$

updates per-bucket scorecards, by feeding a_k^j tallies into the dual-function formula 3:

$$\begin{cases} u_k^j = U\left(a_k^j, u_{k-1}^j\right) \\ f_k^j = F(u_k^j) \end{cases} \tag{3}$$

for each j-th bucket in the range [1, M].

The $U()$ function in 3 reflects a measure of bucket *utilization* by the players, while the $F()$ function simply translates this utilization into player-visible scores. Both functions have a number of popular realizations – for instance, when the buckets are in fact leaky buckets, 3 converts into the familiar:

$$\begin{cases} u_k^j = a_k^j + \max(u_{k-1}^j - \text{svc\_rate}, 0) \\ f_k^j = \dfrac{u_k^j}{\text{svc\_rate}} \end{cases} \tag{3"}$$

where svc_rate is a constant *service rate* – the number of selections that each bucket can "leak" (i.e., process) within one round of the game. Finally, at the end of each round players are rewarded (or penalized, depending on the sign and nature of the function F):

$$\text{reward}_k^i = f_k^{s_k^i} \tag{4}$$

The goal of each i-th player is to maximize the accumulated rewards (or, minimize the accrued penalties):

$$\max\ E_k(\text{reward}_k^i) \tag{5}$$

where E is an expected value of the player's rewards over their respective distribution.

> Expected value is often approximated as a moving (cumulative, weighted, exponential, etc.) average.

The 3 and, especially, the 3" above illustrate one common trait shared by all *reading games*: a smaller per-bucket tally a_k^j produces a better outcome and, thus, a better aggregated result 5.

In other words, to win the player must devise a function $S(i_1, \dots, i_m)$ that would minimize *inter-player conflicts*, by picking the least utilized bucket out of the m-sets handed out by the dealer.

The problem is, the *other* m-sets as well as the choices made by other players are never known. There may also be a little (or limited) information in re the previous (historic) bucket utilizations u_k^j that may be affecting the penalties/rewards 3 and 4.

CH Cluster

```
All theory is gray
And green the golden tree of life
                    Goethe's "Faust"
```

Enough with the abstractions! Suppose, we have a very basic consistently-hashing (CH) cluster that runs a 100% read workload and serves same-size data chunks to concurrent readers. More exactly:

1) Each data chunk that is being read is stored in m distributed copies (replication factor m equals 3 by default)
2) The system time is quantized: an indivisible quantum of time is called an *epoch*
3) The storage initiators do the reading: at the start of each epoch initiator decides which of the m remote consistently-hashed copies to read
4) The storage targets do the storing: at the end of each epoch, each target computes its current latency to execute the newly arrived reads
5) The target's latency is based on the current and previous utilizations (of this target). In other words, the latency is based either on the entire history of target selections, or (a configured) part of thereof
6) Networking is instantaneous: target latency >> transmission time (which therefore can be safely ignored)
7) Last but not the least: the initiators do not disclose their target selections to other initiators (there's simply no time for it anyway).

The initiators are free to execute arbitrary policies to compute the optimal target selections:

$$s_k^i = S(i_1, \dots, i_m) \qquad (1)$$

The objective: minimum read latency.

Notice that even though the description 1 – 7 is totally devoid of all equations, this is the same *reading game* in disguise, except that:

- players are called *initiators*
- buckets are called *targets*
- number m is a *replication factor*
- round of the game – an *epoch*
- function $F()$ – *read latency*

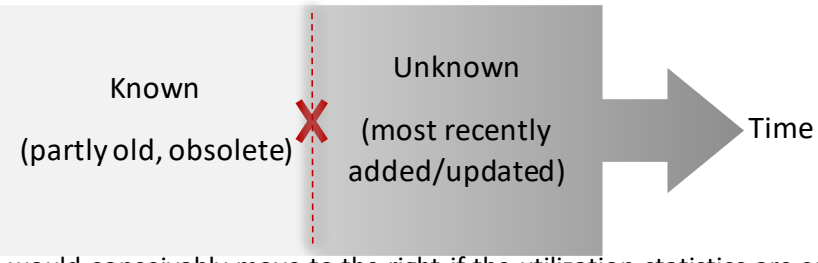
> For additional background on the *initiator* and *target* roles and functionality see, for instance, The Better Protocol.

When ignorance is bliss

Part of the challenge of playing the *reading game* is – not knowing. There is in fact, a known ⇔ unknown spectrum that can be placed between two opposite extremes:

- initiators that know almost nothing about target selections/utilizations (except maybe their own past choices), and
- initiators that know almost everything (except the most recent selections by other initiators)

Since there is a spectrum, there is a point X that properly divides it:



On the picture, the **X** divider would conceivably move to the right if the utilization statistics are somehow made available to whomever is interested (e.g., storage initiators) at the smallest possible delay, ideally – in no time at all. It would move to the left with increasing ratio between, say, service rate on one hand, and intra-cluster roundtrip time, on the other. Exacerbating, in turn, the risk (the curse!) of overfitting, whereby a carefully crafted model gets easily swayed by the quickly aging past...

> For in-depth introduction into the (quote) "*challenges of decision making under uncertainty from a computational perspective,*" please refer to Decision Making Under Uncertainty (a book and a namesake Stanford course).

For me, part of the beauty of modeling *reading games* (and, of course, *writing games*, but more about it later) – is the absolute ease with which the **X** point can be shifted left or right. This provides for a super-easy way to test all possible ML and non-ML based models for robustness with respect to all possible target-side "disturbances" (including both not yet available and old/obsolete information).

This does also answer the following, rather existential, questions: when does it stop making sense to utilize historic statistics? At which point, essentially, do we need to stop trying to be smart? And, finally, when does ignorance become bliss?

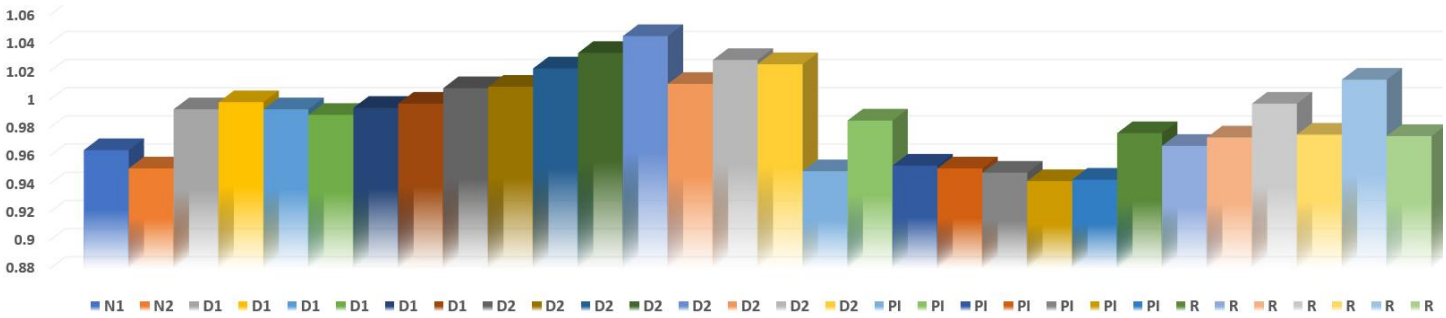
The Model

At the core of any storage cluster is the formula 3 that generates both target utilizations and initiator rewards. Since it'd be difficult to come up with any representative subset of possible (or even plausible) utilization functions, the idea would be to bypass this step altogether and use instead what's called *raw data*. Which in our case consists of straight target selection counts on a per-epoch basis. Let's call them "target history" or just history.

The tradeoff – the high(er) dimensionality of the model – is somewhat mitigated by limiting the history to a certain number of epochs, whereby everything outside the range is considered old, obsolete, just noise. The limit is configurable; in the pictures below, the history was limited to 100 epochs.

In addition, the model must be able to exercise a variety of target-selection strategies, run arbitrary numbers of competing initiators (players) and targets, and provide consistent results across configuration changes. So, let's see…

This, for instance, is a benchmark of a storage cluster of 30 initiators and 30 targets:



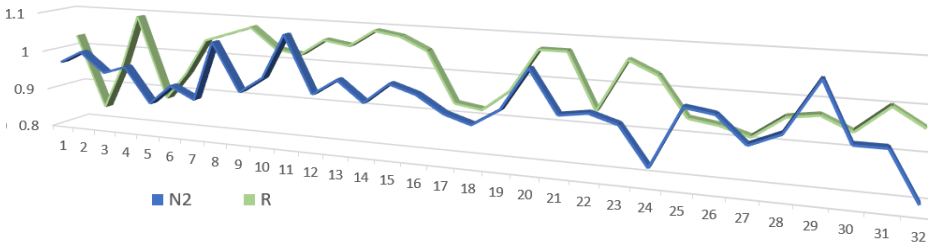
The Y-axis shows average read latencies measured in epochs (smaller is better), the X-axis: all 30 initiators grouped by the assigned (and labeled) target selection strategies that are summarized in the table below:

Label	Target Selection Strategy	Comment
R	Random (uniform) selection.	
D1	D1 and D2 use the actual target latency function (the one that is utilized by the targets), along with target selection history delayed by one and, respectively, two epochs.	
D2	See above.	
PI	Similar to D1/D2, a PI initiator makes its selections by using the target's own (and actual) latency function. In addition, the initiator has access to the **entire** target utilization history. Which is also why PI stands for Perfect Information.	Perfect Information (PI) strategy is employed by initiators that "see" everything – except the current selections by other initiators.
N1	Estimates latency using neural network #1.	Both N1 and N2 initiators have access to the target utilization information delayed by **two epochs** (which makes them similar to D2 and slightly disadvantaged as far as D1).
N2	Ditto, neural network #2.	As above.

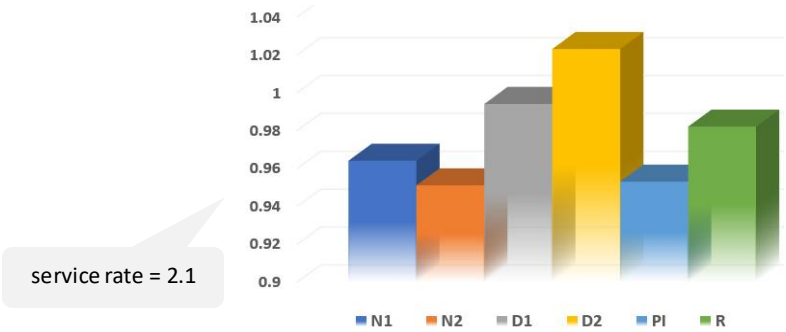
A note in re N1 and N2: these two are the neural networks that showed the best results in training, and that were trained *prior* to running the benchmark. The *prior* phase consisted of training N1 and N2, along with another 28 (in this case) competing networks of different shapes, sizes, and hyperparameters.

> For details on this separate (pretraining) phase, please see my previous text titled Parallel optimization via multiple neural networks.

Generally, vanilla CH clusters produce traffic patterns that must be expected to be a little spikey. Case in point: the next chart that compares the 50-point averaged runtime performances of the N2 with one of the "random" (R) initiators:



Finally, to be done with this particular 30-by-30 benchmark – a chart that averages initiator latencies across the entire respective group:



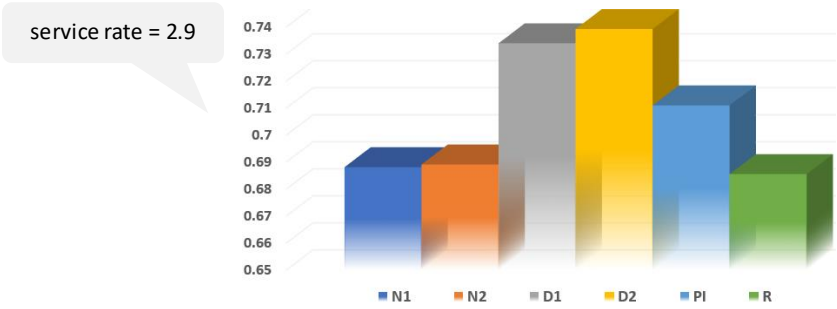
Ultimately, this shows that N2 and PI are the glorified winners, with N2 having a slight edge.

Losing to Randomness

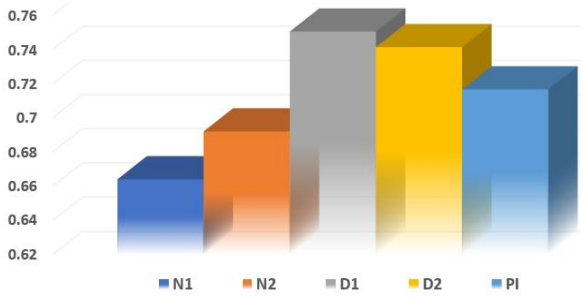
All things being equal, an increase in the service rate brings about a reduction in the target-side congestion. Is that… always a good thing? Well, a side effect that may be sounding like the proverbial "price to pay" is that it also de-values accumulated knowledge. And since at higher rates the aging of the target utilization history happens fast, and even *faster* when the target starts idling… well, visualize the overall value of a carefully assembled and preserved (statistical) past asymptotically approaching zero.

For example:

The following is a (60 initiators, 60 targets) benchmark that exemplifies the case when random ("dumb") strategy outperforms the "smart" one, albeit only by 0.4%:



However. When we remove the random (R) initiators altogether, and rerun the otherwise exactly the same 60-by-60 cluster, the result becomes different:



In summary:

The three pictures above (and a few more that I don't show, to keep it brief) – tell several stories. There is a story of a "confusion" that happens when a smart target selection strategy faces off with a random one – the latter cannot be learned! The story that it is maybe preferable to work with an old, possibly obsolete, largely irrelevant information – than to run against uniform randomness, intractable chaos, white noise…

To be continued.