Memory leak or calculus?

A graphic novel

stu@improbable.io

May 2016

1 Memory leak?

Making distributed systems is hard. At Improbable, we like it when our system works, and it makes us sad when it breaks. For this reason, we have a number of test simulations we deploy regularly on our cloud infrastructure and comprehensively measure, to give us some degree of confidence that everything is working properly and we haven't introduced performance or correctness regressions.

Recently, one of our engineers noticed something worrying: one of our test deployments had been running for several hours, and the graph tracking this particular deployment's memory usage was going up and to the right. The increase in memory usage was slow, but unmistakable: it looked pretty obviously like a memory leak[1]. In a system like SpatialOS that has to run for a long time without failure or loss of stability, any sort of memory leak, even a very small one, is certainly cause for concern. Worse, bugs like these can be quite tricky to investigate and sort out.

However, after studying our metrics more closely we noticed something odd. The offending graph was an aggregation over a metric that tracked memory usage in a spatial way. That is, we were looking at a graph of the *maximum* memory usage per unit of space over various sub-regions in the world. Drilling down, we saw that the so-called memory leak did not manifest uniformly throughout the world: in fact, there was a conspicuous pattern. The severity of the memory leak in a given sub-region was strongly correlated with the relative location in space of that sub-region. We observed that memory usage was more-or-less stable toward the *edges* of the world, and only began to trend upwards as we considered sub-regions closer to the *centre* of the world.

2 Test players

To understand what was going on, it's necessary explain what our test simulation actually *does*. It's designed to look a bit like a simplified version of any open-world online game:

[1]The SpatialOS kernel runs on the JVM, but this in no way precludes the possibility of memory leaks: for example, perhaps there is some internal data structure that we populate as necessary but (under certain circumstances) neglect to clean up—so it grows without bound over time.

we have a large world, filled with a thousand or so player entities controlled by dummy clients. These dummy clients connect from outside our cloud and, to SpatialOS, are indistinguishable from real clients controlled by actual humans—but they are automated. These "test players" are supposed to be reasonable simulacra of genuine players: they wander non-deterministically through the world, can observe each other, and SpatialOS has to carry out all the same data synchronisation work that would be necessary if they were real players.

What's important is exactly *how* these mock-players navigate the world. They start out distributed randomly throughout the world, and their subsequent behaviour is governed by the following steps:

1. Let the starting point x_i be the player's current position in the world.
2. Pick a destination point x_{i+1} uniformly at random in the world.
3. Proceed from x_i to x_{i+1} at a fixed speed.
4. Upon reaching x_{i+1}, go back to step 1.

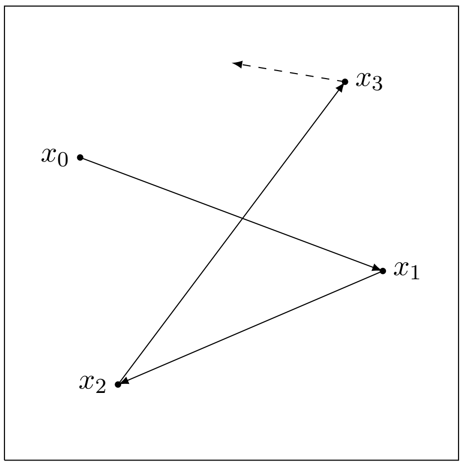

Figure 1: An example of a path a test player might take through the world.

Assuming this process repeats forever: on average, where can we expect the players to end up? That is, sampling a player's position at an arbitrary time gives rise to some probability distribution over the world itself—what does this distribution look like?

Our thought process was this: intuitively, it seems like the distribution should be weighted more heavily towards the centre of the world. After all, there are many possible routes a player might take through the middle of the world, but far fewer through points on the edge. If this is correct, then the upwards trend in memory usage could be explained simply by the fact that the thousand players start out distributed uniformly (so no matter where one looks in the world, there is a comparable amount of work to be done) but tend towards this limiting, centre-heavy distribution (so regions of space towards the centre of the world eventually become disproportionately complex, and require more resources to simulate).

Intuitive reasoning is all well and good, but to be sure we need something more rigorous.

3 Calculus

In the case of our test simulation, the world is a (large) three-dimensional cube, but for simplicity's sake we analyze the case where the world is the unit interval $[0, 1]$.

A player's trajectory through the world can be viewed as an (infinite) sequence of path segments, where each path segment has a pair of endpoints (x_i and x_{i+1}) and is traversed in time proportional to $|x_{i+1} - x_i|$. The probability distribution of a player's position in the world at some arbitrary time can be expressed as a probability density function[2] $p(z)$, where $z \in [0, 1]$ represents a position in the world. This can be computed by summing the contribution from each possible path segment, weighted by the likelihood that a player is currently traversing that path segment:

$$p(z) = \int_0^1 \int_0^1 \frac{f(x_0, x_1)\, \delta_{x_0,x_1}(z)}{|x_1 - x_0|}\, \mathrm{d}x_1\, \mathrm{d}x_0.$$

Here $\delta_{x_0,x_1}(z)$ is 1 if z lies on the path segment from x_0 to x_1 (i.e. $\min(x_0, x_1) \leqslant z \leqslant \max(x_0, x_1)$) and 0 otherwise, and $f(x_0, x_1)$ is the probability density function giving the distribution for which path segment is currently being traversed at an arbitrary time. This can be read as combination of probability density functions: $f(x_0, x_1)$ means "given we are on the way from x_0 to x_1", and $\delta_{x_0,x_1}(z)/|x_1 - x_0|$ is the probability density function for a single traversal between x_0 and x_1.

By symmetry we can assume $x_0 \leqslant x_1$ and rewrite this as

$$p(z) = 2 \int_0^z \int_z^1 \frac{f(x_0, x_1)}{x_1 - x_0}\, \mathrm{d}x_1\, \mathrm{d}x_0.$$

Now, path segments are chosen uniformly at random at each endpoint, but since the player takes a longer time to traverse longer path segments, the likelihood of a path segment being the current one at an arbitary *time* is proportional the length of the segment. Therefore, with normalisation,

$$f(x_0, x_1) = \frac{|x_1 - x_0|}{S}$$

with

$$\begin{aligned} S &= \int_0^1 \int_0^1 |x_1 - x_0|\, \mathrm{d}x_1\, \mathrm{d}x_0 \\ &= 2 \int_0^1 \int_{x_0}^1 (x_1 - x_0)\, \mathrm{d}x_1\, \mathrm{d}x_0 \\ &= \int_0^1 \left(1 - 2x_0 + x_0^2\right) \mathrm{d}x_0 = \frac{1}{3}. \end{aligned}$$

[2]https://en.wikipedia.org/wiki/Probability_density_function

Cancelling, we have

$$p(z) = 6\int_0^z \int_z^1 1 \, \mathrm{d}x_1 \, \mathrm{d}x_0 = 6\int_0^z (1-z) \, \mathrm{d}x_0 = 6z(1-z).$$

Note that this has all the properties we expect: it's symmetric around $z = \frac{1}{2}$, positive on $[0,1]$, and its integral over the whole world is 1. That is, $p(z)$ really is a probability density function, and the quadratic distribution[3] it describes is a plausible explanation for our "memory leak". Better still, we verified this result empirically with a simple one-dimensional simulation of our test players' behaviour (a simulation of a simulation, if you will).

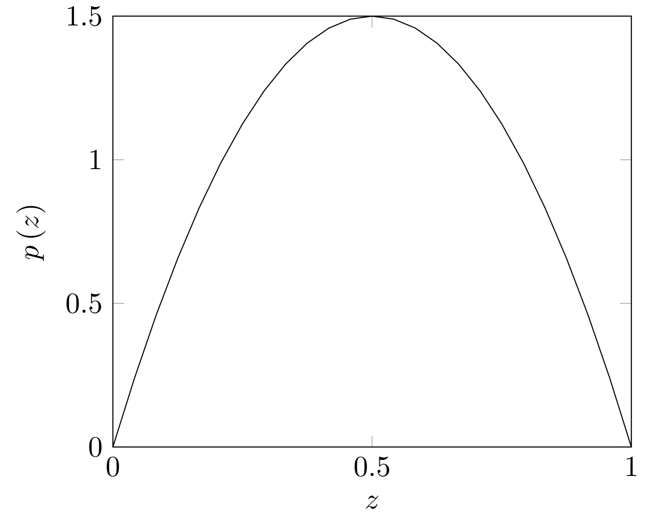

Figure 2: The graph of $p(z) = 6z(1-z)$.

4 Resolution

We now have an explanation, but the situation isn't ideal. What if this odd evolution in the test simulation's memory usage as the test players slowly congregate toward the centre of the world hides a *real* memory leak?

One obvious solution, now that we have this formula, is to *initially* distribute the players in the world according to $p(z)$ (or a higher-dimension generalisation)—then the simulation will be steady-state, at least.

On the other hand, it could cause some confusion if different regions in the test deployment have wildly different resource requirements. To avoid this, it might be even better if we could somehow modify the test players' behaviour such that they maintain a consistent uniform distribution despite their random movement. One might suppose

[3]`https://en.wikipedia.org/wiki/U-quadratic_distribution`

that we could weight each choice of destination point *away* from the middle in order to cancel out $p\left(z\right)$ and obtain a uniform distribution. Unfortunately, this does not seem to be possible: doing so would amount to finding a density function g with

$$\int_0^z \int_z^1 g\left(x_0\right) g\left(x_1\right)\, \mathrm{d}x_1\, \mathrm{d}x_0 = C,$$

where C is some constant. Letting G be the antiderivative of g with $G\left(0\right) = 0$ and $G\left(1\right) = 1$, the left-hand side is $G\left(z\right)\left(1 - G\left(z\right)\right)$. But $G\left(1 - G\right) = C$ is a quadratic equation and so has at most two solutions; so G is a step function and g must be zero almost everywhere.

By analogy, even in the discrete version (choosing from only n possible destination points spaced evenly along the unit interval) there is no way to obtain a uniform distribution except in the trivial cases where $n \leq 3$.

The only practical way to do fix the problem, then, is to use a qualitatively different process for controlling the players' behaviour: for instance, a random walk on a bounded n-dimensional lattice[4], which is well-known to converge to a uniform distribution.

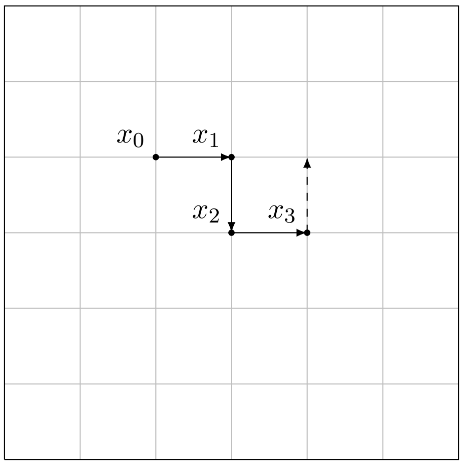

Figure 3: A random walk on a bounded 2-dimensional lattice. Points not actually on the grid will never be visited at all, but we can make the grid arbitrarily fine-grained, and distribution over the grid is uniform.

5 Appendix: a variant of the original problem

Mostly by accident, we also solved the case where the test players don't have constant *speed*, but instead each path segment is traversed in constant *time*. This is exactly the same thing as saying that each path segment is equally likely to be the current one

[4]`https://en.wikipedia.org/wiki/Random_walk#Lattice_random_walk`

at some arbitrary time, so $f(x_0, x_1) = 1$ and the density function becomes the (more difficult) integral

$$p(z) = 2 \lim_{t \to 0} \int_0^{z-t} \int_{z+t}^{1} \frac{1}{x_1 - x_0} \, \mathrm{d}x_1 \, \mathrm{d}x_0.$$

Naming the double integral in the above equation $I(z, t)$ we have

$$\begin{aligned} I(z,t) &= \int_0^{z-t} \int_{z+t}^{1} \frac{1}{x_1 - x_0} \, \mathrm{d}x_1 \, \mathrm{d}x_0 \\ &= \int_0^{z-t} \left[\ln(1 - x_0) - \ln(z + t - x_0)\right] \, \mathrm{d}x_0. \end{aligned}$$

Note that $-(a - x)\ln(a - x) - x$ is an antiderivative of $\ln(a - x)$, so

$$\begin{aligned} I(z,t) &= \left[(z + t - x_0)\ln(z + t - x_0) - (1 - x_0)\ln(1 - x_0)\right]_0^{z-t} \\ &= 2t\ln(2t) - (1 - z + t)\ln(1 - z + t) - (z + t)\ln(z + t). \end{aligned}$$

Taking the limit as $t \to 0$ we have

$$p(z) = 2\ln\left[\frac{(1 - z)^{z-1}}{z^z}\right].$$

Again, this turns out to be a well-behaved, positive, symmetric function over $[0, 1]$. Furthermore, its integral over this domain is 1, which can be seen by considering

$$\lim_{t \to 0} \int_t^{1-t} p(z) \, \mathrm{d}z = \lim_{t \to 0} \left(P(1 - t) - P(t)\right)$$

where $P(z) = \ln(1 - z) + z(1 + p(z)) + z^2 \ln(z/(1 - z))$ is an antidervative of $p(z)$.

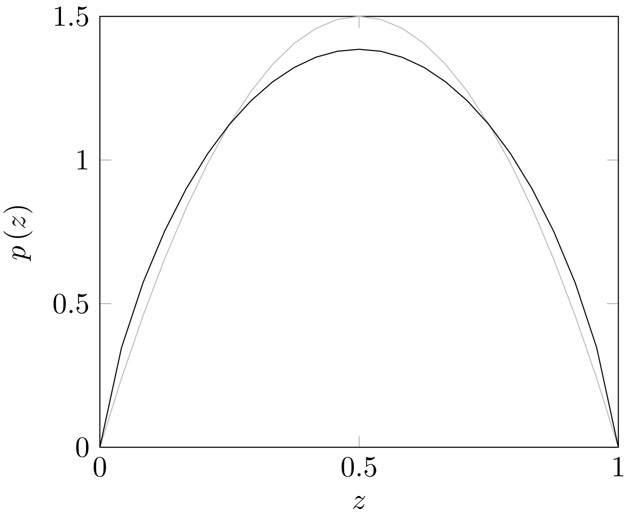

Figure 4: The graph of $p(z) = 2\ln\left[(1-z)^{z-1}\middle/z^{z}\right]$, shown here with $6z(1-z)$ for comparison in light grey.