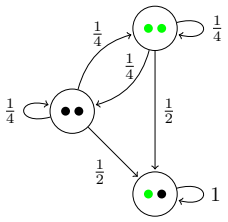
Singleton Election on Consensus Driven CLusters

Suppose we have two nodes in a cluster. We need them decide which of the two should run a particular service. It doesn't matter which node runs the service, but we need exactly one running.

We will assume that some form of consensus protocol is operating behind the scene to ensure the states of neighboring nodes are kept up-to-date.

One way we may attempt to program a solution, is to have each node randomly decide if they should run the service or not. In this instance, both nodes get have a $\frac{1}{2}$ chance of starting the service.

After enough time has passed and we can garuntee that the consensus algorithm has updated the global state, we compare the global state, to our desired state.



In this example, we would anticipate that after one round, we have a $\frac{1}{2}$ chance of succeeding, a $\frac{1}{4}$ chance of failing to start any services, and a $\frac{1}{4}$ chance of starting too many services.
(Note that once we achieve the desired state, the probability of a transition goes to one, and loops back to itself. Instead of continuing to execute state changes, we will just note that when we get to a full loop-back, we will stop execution.)

Figure 1: State Transitions for a Small Cluster

The diagram helps to visualize the system state over time, but it's not extremely helpful for answering immediate quesitons such as:

- Is the system garaunteed to reach a stable state?
- What is the average amount of time it will take for any given system to reach a stable state?
- Is there a better strategy?

More data needs to be collected in order to solve these problems. In the first example, the probabilites are fairly intuitive to generate, but that may not be the case for arbitrary examples.

For instance, what is the probability that in a system of 17 nodes where we need 5 of them to run services, that 9 of the nodes will switch state instead of 5?

Luckily, this exact type of question can be answered with a Binomial Distribution: $X \sim B(n,p)$ for $n \in \{1,2,...\}, p \in [0,1]$.

The probability mass function for $X \sim B(n,p)$ is $f(k,n,p) = \binom{n}{k}p^k(1-p)^{n-k}$ where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

We can translate our question of 17 nodes, 5 desired services, and 9 actual services into this expression and solve.

$$f(k=9,n=17,p=\frac{5}{17}) = \frac{1200925440000000000}{48661191875666868481} \approxeq 0.5\%$$

Let us plot some value for a less complex system, and examine the results.

s	n	r	$P(s,r,n)$
0	0	1	n/a - Note 1
1	1	1	1 - Note 2
0	1	2	1/4
1	1	2	1/2
2	1	2	1/4
0	1	3	8/27
1	1	3	4/9
2	1	3	2/9
3	1	3	1/27

s	n	r	$P(s,r,n)$
0	1	4	81/256
1	1	4	27/64
2	1	4	27/128
3	1	4	3/64
4	1	4	1/256
0	2	4	1/16
1	2	4	1/4
2	2	4	3/8
3	2	4	1/4
4	2	4	1/16

- *Note 1* - it makes no sense to calculate any probability when we request zero services.
- *Note 2* - if all nodes need to run a service, the probability is always 1.

The data is base on asking the question, given n nodes, and r desired services, what is the probability that s services will be turned on?

If we want to answer the question about the total time required to reach stability, we will need to chain these probabilities somehow.

There are three ways to answer the question of 'what is the next required state change?' They are

1. If all desired services are running, do nothing.
2. If there are more services running than what we desired, turn some of the running services off.
3. If too few services are running, then turn some more services on.

We can then make the assumption that we will be using a recursive piecewise function in order to calculated the average convergence time.

$$A(r,n) = \begin{cases} 1 + \text{recursive definition}, & s < r \\ 1, & s = r \\ 1 + \text{recursive definition}, & s > r \end{cases}$$

To actually make this function converge, the function will need to be weighted with the outcomes relevant probability. A more decriptive function starts to take shape...

$$f(k,n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$A(r,n) = \begin{cases} 1 + (\text{recursive definition}) * f(k?,n?,p?), & s < r \\ 1, & s = r \\ 1 + (\text{recursive definition}) * f(k?,n?,p?), & s > r \end{cases}$$

We need to actually decide on what it means for our future decisions when $s < r$, or when $s > r$.

Figure 2: Possible Outcomes for Three Services Across Five nodes

Figure 2 shows the relation between the desired service count r, and the actual service count s. Let us choose one of the states for when $s < r$ and determine what our next desried state should be.

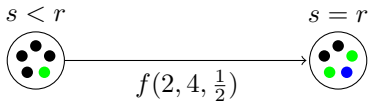


Figure 3: Next Desried State Transition

Figure 3 desribes the ideal next transition. The blue node represents the node that is already running the service we requested. We no longer need this node to make decisions as long as $s < r$.

Notice that the number of nodes needing to swicth service states is less though, from $n = 5$ it has dropped to $n = 4$. As such, the probability function was also changed.

We can model the next required state in terms of the first. Let the desried state definition by a tuple of $S_0 = (r, n)_0$ such that the next state is S_1. We can define S_1 with a piecewise function.

$$S_next = \begin{cases} (r-s, n-s), & s < r \\ (s-r, s), & s > r \end{cases}$$

Where s is the number of actual services acquired from S_i.

We can also update our probability function so that it's also piecewise. (Note that previously, we have used the Binomial Distributions notion of k, but we are changing it to s.)

$$f(s = k, n, p) = \binom{n}{s} p^s (1-p)^{n-s}$$

$$P(s, n, p) = \begin{cases} f(r-s, n-s, \frac{r-s}{n-s}), & s < r \\ f(r, n, \frac{r}{n}), & s = r \\ f(s-r, s, \frac{s-r}{s}), & s > r \end{cases}$$

NOT READY YET - INTRODUCE FUNCTION T

By combining our incomplete function A with our tuple pattern and our new probability function, we can finally start doing math with our Average Time to Converge Function.

$$A(r, n) = \sum_{s=0}^{n} (\text{Time to converge from } s) * (\text{Probability of } s)$$

$$\text{Time to converge from } S = \begin{cases} 1 + A(S_{\text{next}}) & s < r \\ 1 & s = r \\ 1 + A(S_{\text{next}}) & s > r \end{cases}$$

In order to know the Time to Coverge one state, we also need to know how to converge all following states. This is why T is defined in terms of A, and vice versa.

Where S is a tuple of (s, r, n)

By using our previous definition of S_next, we can define the function as follows.

$$\text{Time to converge from } (s,r,n) = T(s,r,n) = \begin{cases} 1 + A(r-s, n-s) & s < r \\ 1 & s = r \\ 1 + A(s-r, s) & s > r \end{cases}$$

By using our piecewise probability function, we can then define $A(r,n)$ as follows.

$$A(r,n) = \sum_{s=0}^{n} T(s,r,n) * P(s,r,n)$$

Testing the Function GARBAGE

Recall Figure 1. The diagram describes a state with $n = 2$ nodes and $r = 1$ desired service. The average time until convergence is $A(1,2)$.

$$A(r=1,n=2) = \sum_{s=0}^{n=2} T(s,r=1,n=2) * P(s,r=1,n=2) = \sum \begin{cases} T(0,1,2)P(0,1,2) \\ T(1,1,2)P(1,1,2) \\ T(2,1,2)P(2,1,2) \end{cases}$$

$T(s,r,n)$	Convergance Time	$P(s,r,n)$	Probability
$T(0,1,2)$	$1 + A(1-0, 2-0)$	$P(0,1,2)$	$f(1-0, 2-0, \frac{1-0}{2-0})$
$T(1,1,2)$	1	$P(1,1,2)$	$f(1, 2, \frac{1}{2})$
$T(2,1,2)$	$1 + A(2-1, 2)$	$P(2,1,2)$	$f(2-1, 2, \frac{2-1}{2})$

$$A(r=1,n=2) = \sum \begin{cases} [1 + A(1,2)] * 1/4 \\ 1 * 1/2 \\ [1 + A(1,2)] * 1/4 \end{cases}$$

$$= \frac{1}{2} + 2 * \left[[1 + A(1,2)] * \frac{1}{4} \right] = \frac{1}{2} + [1 + A(1,2)] * \frac{1}{2}$$

$$A(1,2) = \frac{1}{2} * [1 + 1 + A(1,2)]$$

$$2A(1,2) = 2 + A(1,2)$$

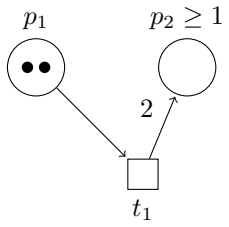
$$2A(1,2) - A(1,2) = 2$$

$$A(1,2) = 2$$

Garbage Below

dasd dsa dasasd

dasd



$$A(r,n) = \sum_{i=0}^{n} T(i,r,n) * P(\frac{i}{n}, r, n)$$

$$T(s,r,n) = \begin{cases} 1 + A(|r-s|, n-s), & s < r \\ 1, & s = r \\ 1 + A(|r-s|, s), & s > r \end{cases}$$

$$P(p,r,n) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$