Hello,

I have implemented a simulated version of the SGH demo to be able to tweak things.

Here is how the simulation works:

**INPUT:** A SGH Graph + A number X of servers to be killed (by monkeys)

For each simulation step:

a/ Randomly select X server and mark them as dead

b/ For each client, select \***ONE possible request**\* and check if it can be satisfied.

[i.e. Does the client has at least one linked server which can fulfil the request and is not marked as dead.]

c/ Count the ratio of requests which could not be satisfied = failure rate

**OUTPUT:** Average the failure rate for all different steps (which corresponds to different servers being down).

Using this simulation and different values for X we can get a percentage for the overall failure rate of the system, i.e. what is the percentage of requests which are not satisfied when some servers are down. From what I understand this should be very similar to what we have in the real experiment, isn't it?

**ASSUMPTION/THREATS TO VALIDITY:** The strong assumption of this simulation is that the request in the overall system are distributed uniformly from the clients. At each step all clients send one request so they have the same weight no matter how many types of request they can make. Aslo within each client, each type of possible request has the same weight, at each step one request is picked at random. These two uniform distributions are not realistic but the is no obvious realistic distribution we could use and which really makes sense.

In a real deployment, the values we are interested in are probably what is happening for low values of dead servers. In a real system we can imagine that we will often have a small fraction of the servers which are temporarily unavailable and we want to see how our strategy could help reduce the number of failures caused by the temporary outages. That is why in the simulation I focus on evaluating the failure rate for between 0% and 10% of server failures. We also simulate for 25% and 50% in order to see how the failure rate evolve in a more severe outage.

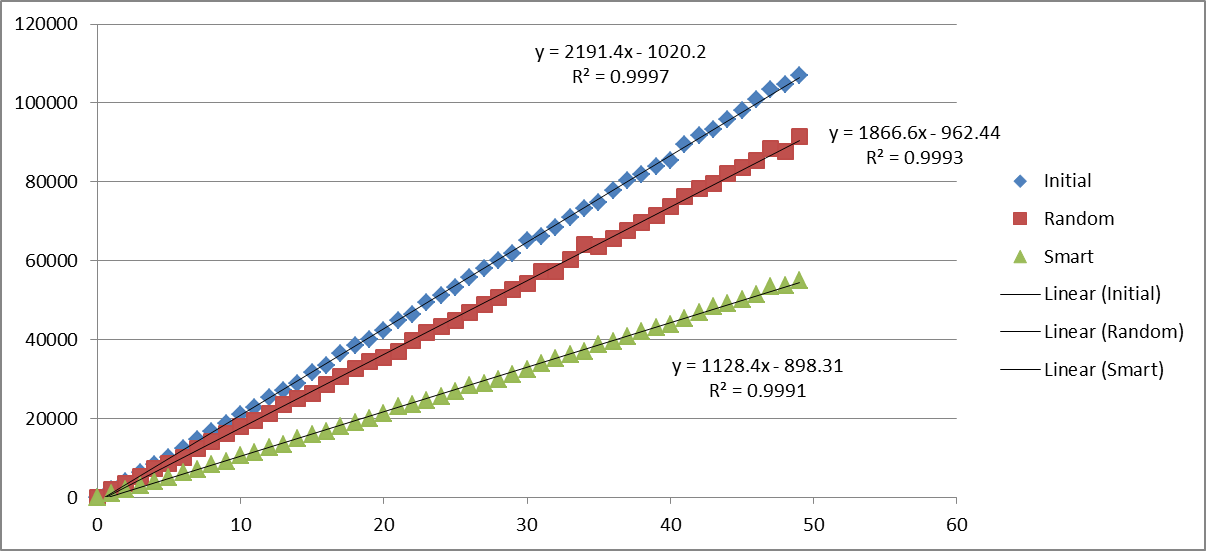
Below are some numbers for a simulation with 1891 Clients and 496 servers.

Here are the metrics for the initial random and final model:

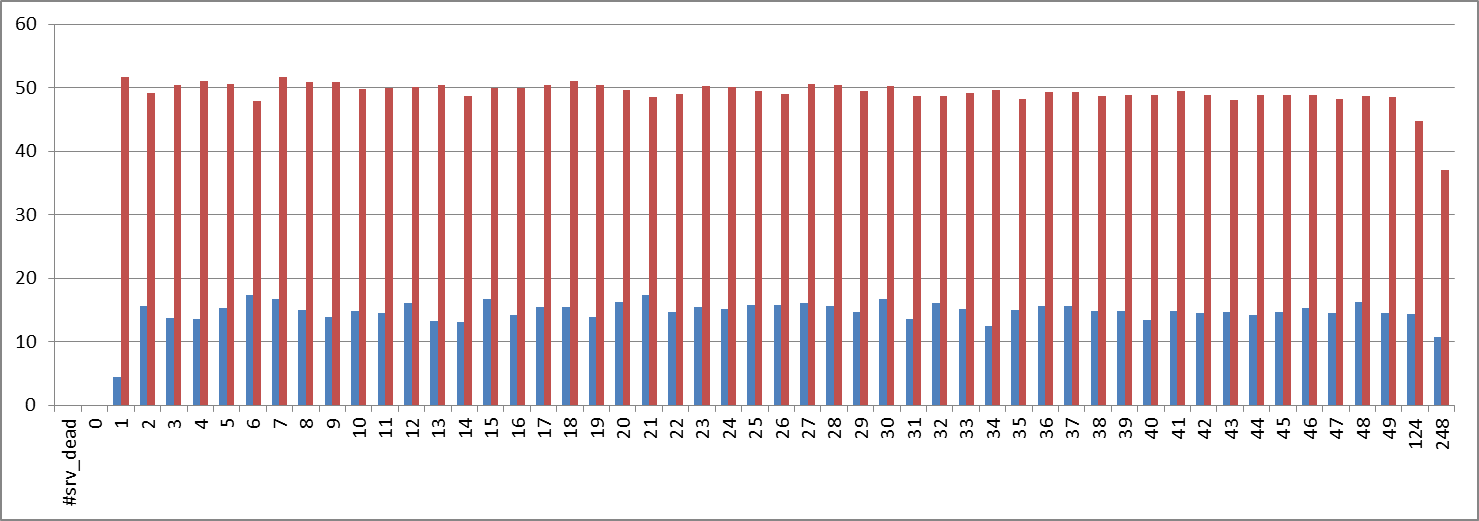


It is interesting to note from the numbers above that there seem to be no direct correlation between the diversity metrics and the robustness metric (computed using extinction sequences). The random strategy generally seem to be increasing diversity while the Smart strategy tends decrease the values of diversity metrics on the population of servers. Repeating the experiment with different models and at different scales produces comparable results.

The following graph shows the result for 50 simulations (with 1 to 50 server failures) with 1000 steps each for the 3 models. For each simulation the 1891 \* 1000 requests are evaluated and the number of failures is collected. The results are pretty linear: on the 0-10%, on average, the initial model drops about 2191 (out of 1891000) requests per dead server. This number is lowered to 1866 when using the Randomly evolved model and drops to 1128 on the model evolved using the Smart strategy.



If we now plot the improvement in percentages we obtain rougly a 15% reduction in the number of failures for the Random evolution and close to 50% for the Smart strategy.



The way to interpret those results is that the model evolved with the Smart strategy allows reducing the failure rate of the overall system to half its initial value while keeping the cost of the deployment constant.

What do you think?

André: do you see any differences with the real experiment? Can you get failure rate values for similar ratios of dead servers?

Cheers,

Franck