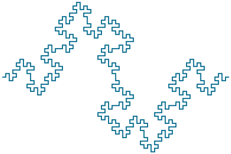
Injecting Diversity Into Running Software Systems

Vivek Nallur
Trinity College Dublin

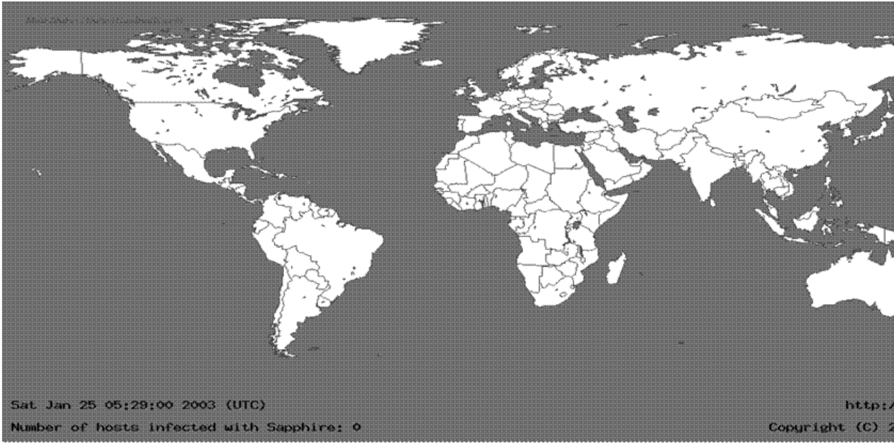


16-May-2014

Effects of Monoculture

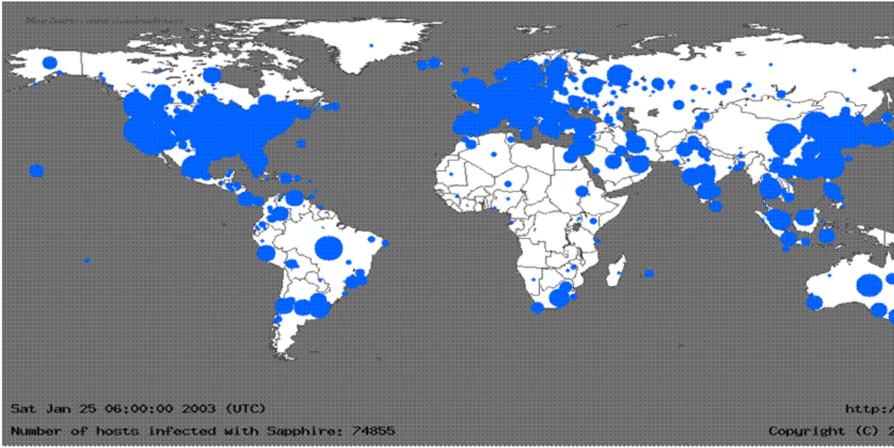
Figure: Phytophthora infestans

EVEN IN THE SOFTWARE WORLD



Slammer attacked *only* one combination: Win2k + MSSQL

EVEN IN THE SOFTWARE WORLD



- ~75k hosts in 30 mins!

FUNDAMENTAL PREMISE

1. Diversity is not just a *good-to-have*, but `essential`
2. Robustness is a quality attribute that we would like our systems to have
3. Robustness can be increased by injecting Diversity

DIVERSIFY - FET FP7 PROJECT

Partners Investigating Diversification at Various Levels

1. Inria (France)
2. Sintef (Norway)
3. Trinity College Dublin (Ireland)
4. Université de Rennes 1 (France)

Genetic Diversity

1. Not necessarily vastly different, but just different *enough*
2. An algorithm is the genetic heart of a software system
3. Algorithm diversification is a good candidate for genetic diversification

Algorithm Diversification

1. There exists natural diversity amongst algorithms
2. In any domain, there are multiple algorithms that do the same thing, better, faster, etc.
3. We use *load-balancing* as our domain, for now

LOAD BALANCING

1. Fundamental Idea: Distribute incoming traffic amongst pool of machines, such that two goals are satisfied:
   1.1 Response time is minimized
   1.2 Failure rate is minimized
2. Many algorithms exist: *round-robin*, *dynamic round-robin*, *leastconn*, *header-Hashing*, *parameter-Hashing*, *uri-Hashing*, *rdp-cookie*, etc.
3. Each makes assumptions about the nature of traffic being encountered

NATURE OF TRAFFIC

1. Traffic depends on type of content:
   1.1 Static web-pages, like wikipedia, blogs, articles, etc.
   1.2 Dynamic web-pages, like weather, traffic, news, youtube, etc.
   1.3 Sticky (personalized) like facebook, twitter, etc.
2. The algorithms mentioned previously, improve response times for these workloads
3. Specialist algorithms for specialist patterns

PATTERNS, NOISE, ETC.

1. In a DDoS attack, traffic pattern is random
2. Failure-rate rather than response time becomes more important
3. Generalist algorithm for all patterns of workload, doesn't exist

CHANGE ALGORITHMS

1. Currently, sysadmins have to consider their workloads and choose one algorithm
2. When pattern of traffic changes, or website gets hit by a DDoS attack, the prevailing algorithm's assumptions are invalid
3. What if we modify the algorithm when the traffic pattern changes?
4. Can we do better than random?

Adaptation via Algorithm Swapping

1. Modify load-balancer to work on a *pool of algorithms*, instead of *one*
2. Cycle through the pool, every n seconds
3. In the worst case:
    3.1 Algorithm completely unsuited for traffic pattern $\implies$ high failure
    3.2 But it lasts only for n seconds!

Creating a Pool of Algorithms

1. Choose `haproxy` as an industrial-strength load-balancer
2. Use all the algorithms implemented by `haproxy`
3. Number of combinations: ${}^{7}C_{2}$ —- ${}^{7}C_{7}$!!
4. Potential behavioural diversity is very high!

DOES THIS WORK?

1. We want to decrease failure-rate
2. So measure *dropped requests*
3. In the presence of a cloud of VMs hitting the load-balancer
4. Pools defined as:
   4.1 $^7C_1$ — class A — baseline
   4.2 $^7C_3$ — class B
   4.3 $^7C_4$ — class C
   4.4 $^7C_7$ — class D

EXPERIMENTAL CONDITIONS

1. Workload: 3 Virtual Machines
2. Load-Balancer: 1 haproxy
3. Load-Generators: 13 Virtual Machines

Note:
We want to overwhelm `haproxy`, not the workload machines

NORMAL PERFORMANCE OF HAPROXY

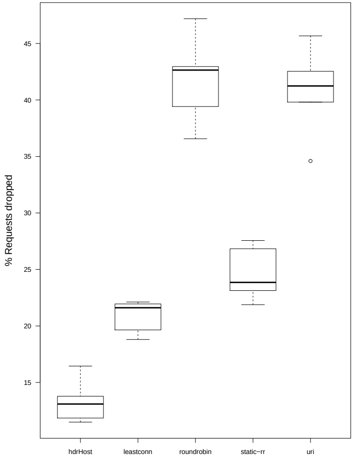


Figure: Each pool containing one algorithm – all of class A

DIVERSIFIED PERFORMANCE OF HAPROXY

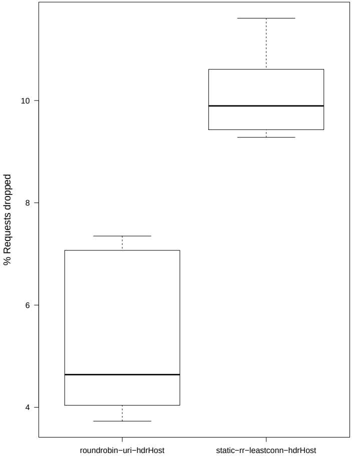


Figure: class B

DIVERSIFIED PERFORMANCE OF HAPROXY

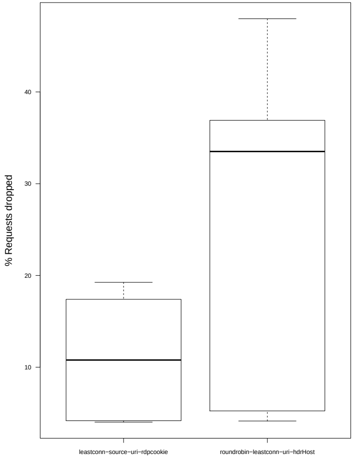


Figure: class C

Diversified Performance of Haproxy

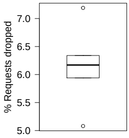


Figure: class D

ALL TOGETHER NOW

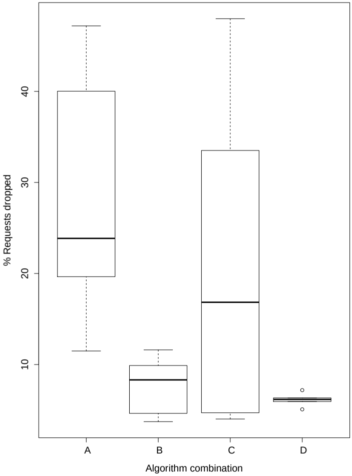


Figure: Robustness across pools

STATISTICAL EVIDENCE

	diff	lwr	upr	p adj
B- A	−20.622	−30.632	−10.612	0.00001
C- A	−9.329	−19.340	0.681	0.076
D- A	−22.160	−36.317	−8.004	0.001

Table: Significance of long-run differences in failure rate

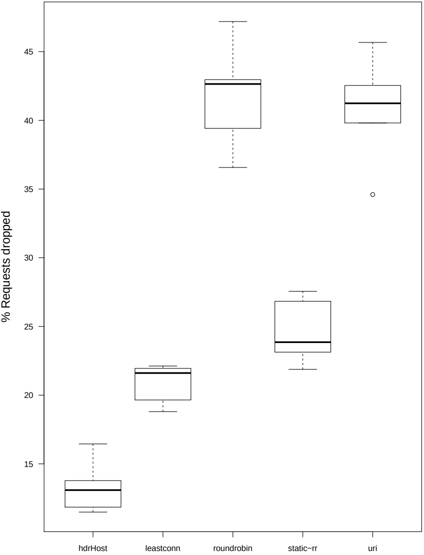
	diff	lwr	upr	p adj
B- A	-1, 073.833	-2, 638.443	490.777	0.276
C- A	50.333	-1, 514.277	1, 614.943	1.000
D- A	-1, 523	-3, 735.693	689.693	0.273

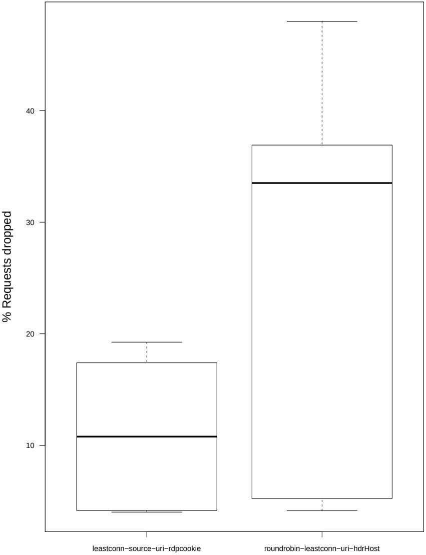
Table: No significance of long-run differences in median response time

EXPERIMENT VALIDITY

1. Sample size: 6 samples per pool
2. Anova & Tukey test pass for statistical significance
3. Failure-rate improved; Response time same!!
4. Only static workload
5. Dynamic & Sticky workloads missing

DIVERSITY ISN'T ALL GREAT :(





SO, IT'S STILL RANDOM CHOICE

1. Not exactly. We can measure inter-algorithm distance
2. Sort of.
3. We can use *Normalized Compression Distance*
4. Used in many free-text domains

$$NCD_Z(x, y) = \frac{maxK(x|y), K(y|x)}{maxK(x), K(y)}$$

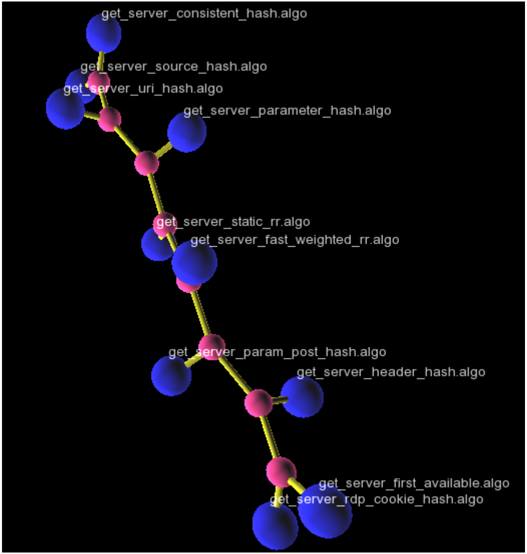


Figure: Clustering on code of algorithm implementation

Using NCD

1. Not all pools are created equal
2. Selecting from pool, might be better than random choice
3. Pre-compute pool diversity?

WHAT'S THE NET RESULT?

1. No definitive answers
2. But promising experiments
3. Obviously more required

THAT'S ALL, FOLKS!

Questions, Suggestions...