Zennet Pricing Algorithm

Ohad Asor

September 19, 2014

ohadasor@gmail.com

On this note we'll give a detailed description of Zennet's pricing algorithm. The purpose of the algorithm is to fairly price computational resources consumption, focusing only on massive distributed computationaljobs.

An intuitive approach is to measure accumulated *procfs* variables. But such variables are definitely illusive:

1. Variables are correlated: in real life, you cannot use only CPU or only RAM and so on.

2. Variables might often show same values, but they have totally different economical value on different machines: some CPU instruction set is much more efficient than of CPU from another model, and one can think of many other examples.

3. Both as a standalone statement and as consequences from the above, the true principal variables we're looking for, measuring the "real" computational resources consumption, are unknown.

Big troubles, but, *procfs* is all we have.

Assume that there exists n such physical unknown principal uncorrelated variables (abbrev. by UVs, "Unknown Variables"). Assume the existence of a function $\mathbf{f}:\mathcal{X}\to\mathbb{R}^n$ which maps a computer program $x\in\mathcal{X}$ to a vector which represents the accumulated consumption values of all n UVs.

Now, if the UVs were known, we would set a price for each one of them. Call this vector of prices $\mathbf{p}\in\mathbb{R}^n$. The total reward to be paid for a program x is therefore

$$s=\mathbf{p}^T\mathbf{f}\left(x\right) \qquad (1)$$

We now make the main assumption: we claim that the *procfs* variables are the result of a linear projection acting on the UVs. The rationale is that we do seek accumulated (hence additive) variables, but uncorrelated (=orthogonal). Denoting by $\mathbf{v}:\mathcal{X}\to\mathbb{R}^m$ a function that given a program returns a vector of

m accumulated *procfs* variables values as its components, we therefore assume the existence of a matrix M such that

$$f(\mathbf{x}) = M\mathbf{v}(x) \tag{2}$$

Plugging into (1) we get

$$s = \mathbf{p}^T M\mathbf{v}(x) \tag{3}$$

Take N programs $\{x_k\}_{k\in[N]}$ which altogether use all attached hardware in different forms, in which the user has set the price of each program as a whole, without dividing it among the UVs (or even set only one number for full PC usage, if this number is t then $\mathbf{s} = t\mathbf{1}$ on the upcoming notation). So we have N pairs of the form (x_k, s_k) where s_k is the desired price for program x_k. Let $\mathbf{s}$ be a vector with $s_{k\in[N]}$ as components, and V be a matrix with rows $\mathbf{v}(x_k)$. Our equation now reads:

$$\mathbf{s} = \mathbf{p}^T MV \tag{4}$$

our unknown is the vector $\mathbf{p}^T M$ which can be optimally estimated by the Moore-Penrose pseudoinverse:

$$\mathbf{p}^T M = \mathbf{s}V^+ = \begin{cases} \left(V^T V\right)^{-1} V^T & n < N \\ V^T \left(VV^T\right)^{-1} & n > N \\ V^{-1} & n = N \end{cases}$$

We do not know the value of n, so we just pick reasonably large N. In addition, if Singular Value Decomposition is used, the psuedoinverse of V can be calculated without dependence on $n > N$ etc.

Note that this pricing model is adequate for massively distributed applications only: say that a pending IO read was blocking for a long time. On singleton machine usage, this matters to both publisher and provider, and will break the linearity assumption. But when the big job is distributed into countless small jobs, we don't care if one host made 1000 small jobs per second, while another host made only one every second. This root assumption is therefore vital for the acceptability of the solution.

Those x_n are called, on the Zennet's framework, the Canonical Benchmarks.