Paper 2057-2018

Fluid and Dynamic: Using measurement-based workload prediction to dynamically provision your Cloud

Nikola Marković, Boemska UK

ABSTRACT

As the capabilities of SAS® Viya and SAS®9 are brought together, alongside the integration with open-source technologies such as R and Python, it appears that a set of technologies with wide-ranging resource requirements will inevitably end up sharing the same infrastructure. It's more than likely that some traditional SAS® batch jobs will continue to run throughout the night, probably joined by some newly scheduled Python programs; increasing amounts of data will be loaded into memory every day, but for many customers month-end processing cycles will always be a thing. As organizations naturally look to progress toward the Cloud, the resource requirements for such an environment will appear complex and might seem overwhelmingly expensive. Will high I/O instances be required or is it better to opt for compute-optimized? Will there be benefit from a Burstable Performance setup? While "all of the above" might be a common conclusion, once an environment is productionized and put through its paces, usage patterns should emerge that can be used to both optimize the use of the resources available and scale the infrastructure to what is actually required. This paper discusses the possible use of granular performance measurements to gauge periodic workload requirements, in order to both plan and execute appropriate dynamic alterations to provisioned cloud infrastructure, and to ensure that the infrastructure that is currently in place remains optimally utilized.

INTRODUCTION

This paper will discuss ideas for optimising execution of mixed analytical workloads on elastic cloud environments. It will describe a typical workload, discuss how that fits within the capacity and scaling options offered by current cloud computing providers, and propose data-driven techniques that optimise for maximum performance and minimum infrastructure cost.

The technicalities of *how* the relevant data is collected or *how* the scaling of either the applications or the cloud infrastructure is performed will not be discussed here. It is safe to assume that mechanisms and automation routines for scaling either exist or can be programmed fairly easily (we are in 2018 after all). Instead, this paper will contemplate *when* those scaling routines should be invoked, and *why then.*

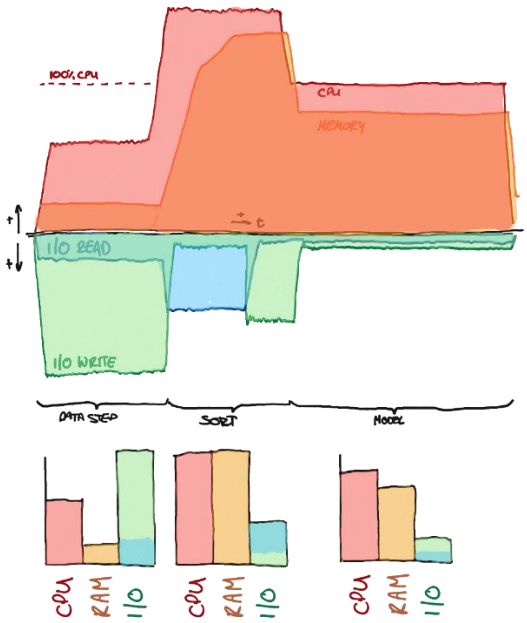
The ideas outlined here come from experience of analysing and optimising SAS workloads in large, relatively complex (5k+ user) enterprise environments. ESM, a performance tuning product we developed at Boemska, helps us profile analytical workload performance by collecting granular time-series metrics for every single user or job on a given platform. Over the last couple of years we have used it to improve the stability and capacity of many customer environments, and in the process learnt a considerable amount about the resource utilisation and behavioural patterns of real-world analytical workloads. It is those patterns and behaviours, alongside our own early research into things like ML-assisted batch schedule optimisation, that form the basis of this paper.

There will be very little practical take away here. Nevertheless, you may enjoy reading it if you have an interest in performance, if you are a Systems Architect or ponderous Design Authority type, or generally like to contemplate what the mechanisms for provisioning elastic clusters on Cloud infrastructure may look like in the not-too-distant future.

THE NATURE OF ANALYTICAL WORKLOADS

Analytical processing - SAS in particular – involves the use of a range of computational operations and procedures, each with varying computational complexity and resource requirements. Consider a typical

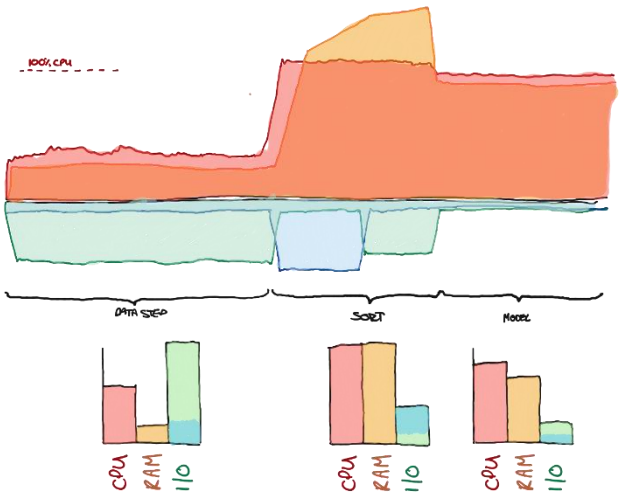
SAS program, which loads some data using DATA step, sorts that data or creates an index on it, and then run a typical analytical computation against it. From a resource usage perspective, that program would probably look something like this:



Notice how the resource requirements for each of these steps differ? How the load step uses very little memory but writes a lot to the disk, the sort is CPU and Memory intensive, and the final model just hits the CPU for a long time?

This represents a single job. Of course, in reality each node supports multiple users asynchronously triggering their code, or multiple batch jobs being triggered as their dependencies are met. Each of these user sessions or jobs behaves differently, depending on the code each user runs and when.

The resource utilisation pattern for the same job running on a busy node in the middle of a typical work day would look something like this:



During peak operation, workloads on bare metal hardware *by design* exhaust some of the resource available to them. It means that the physical hardware was sized properly. However, this also means that the workload is throttled and jobs take longer to execute than they would during off-peak times, when they don't have to compete for resource. Different resource bottlenecks can also often hit at various points during the day, and each time this happens the overall capacity is reduced in proportion with the bottleneck.

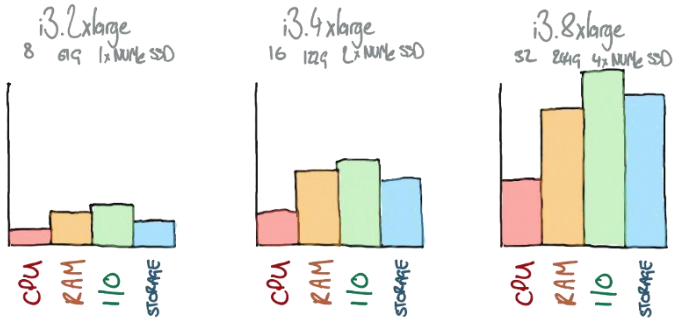
This happens because available resource is physically limited by the physical hardware that the workload is executing on. This is where the Cloud should make a difference.

THE CLOUD IS INFINITE

One of the main promises of Cloud computing is this idea of it offering an elastic, infinite resource pool where tenants *should* be able to only pay for the amount of resource they need at any given time.

The pricing models offered by today's cloud providers follow this premise. Amazon, for example, currently offer EC2 instances which range in price from a few dollars a month to a few thousand. Selected instances can be spun up in seconds, and scaling mechanisms enable users to adjust the size and cost of their instances as they see fit.

Here is an example cloud offering from Amazon showing different instance sizes:



Cloud providers also tend to offer different instance *types* to choose from in each price range. Amazon currently group their instance offerings into the categories of General Purpose, Compute Optimized, Memory Optimized, Storage Optimized, or 'Accelerated Computing' (and even within those categories there are instance subcategories that differ slightly). To illustrate this, here is a simplified example of three instance types which are in the same price bracket but feature different resource profiles:

These options exist because of a need to cover the range of requirements that various application stacks typically bring. Most common workloads, such as databases or web applications, tend to have a relatively

predictable resource footprint that scales with the number of site hits, users or transactions. With typical database applications, the count of IO operations (IOPS) tends to fluctuate predictably according to the volume of transactions, and individual users can do very little to affect this.

Understanding the resource requirements of any workload being hosted in the cloud is evidently important as it can substantially affect both performance and value. As mentioned earlier, these requirements can be very varied with analytical workloads, both in terms of capacity variability and the resource that's in demand at any point.

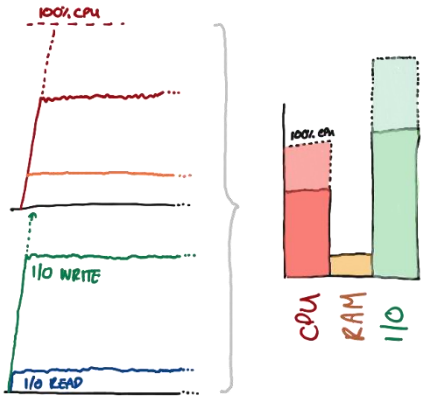
UNDERSTANDING THE REQUIREMENTS OF A WORKLOAD

Revisiting that 'typical SAS program', with the load, sort and model. Consider the first step, the data load. Notice how it doesn't quite utilise 100% of a CPU thread, even when it's run outside of peak time? The reason for that (in this example) is that the load step is computationally too simple - it's just reading some data from an off-host source, reformatting it and computing a few columns, and then writing it to a table on the disk.

Because of the way that computers work, single threaded programs will always *try* to execute the next instruction if there is CPU time available, up until they hit their limit and there are no more time slices in the same thread. A program generally fails hit that limit if it can't get enough data to operate on, when the database connection or disk device is unable to provide it with enough data to process, or it is unable to save the outputs to disk at the rate that they are being computed. For this example step it is the available disk throughput that is the *bottleneck*.

This paper is based around the idea that, with a Cloud environment, it *should* be possible to dynamically scale cloud infrastructure in a way that gives the workload all the resources it needs to operate optimally, while also minimising resource waste resulting from oversubscription. Temporarily doubling capacity so that a set of jobs can be executed in half the time *should* costs no more than leaving the same set of jobs to execute in twice the length of time on a node half the capacity.

If it is possible to scale resources up for no extra cost, it makes sense to calculate what the requirements of each job would be in an 'infinite resource' scenario.



By analysing these measurements, it is possible to extrapolate that increasing the available disk throughput capacity by 30% would enable this particular *step* to use the rest of the CPU thread and complete in a shorter timeframe. The minimum theoretical runtime for that job can also be calculated in this way.

It is important to remember that scaling the instance using currently available mechanisms requires the instance to be restarted so that the available virtual devices can be picked up by the operating system. it is not generally possible to simply scale the instance up and down in the middle of a job flow; instead, scaling tends to involve adding nodes and removing nodes from a cluster. Scaling down carries an

inherent cost penalty: each job running on an instance needs to complete before the instance is destroyed, and if one job overruns and keeps the instance alive, considerable resource can go unused.

A worthy goal for an optimisation would be to provide the *ideal* performance conditions for a workload while minimising total resource usage variability and therefore the frequency of these expensive scaling events. It therefore makes sense to consider other controls that can be applied from the application layer, in addition to instance scaling.

An obvious example of this is the orchestration of *which* jobs are triggered to run, exactly *when* a particular job is triggered to run, and *on which instance.*

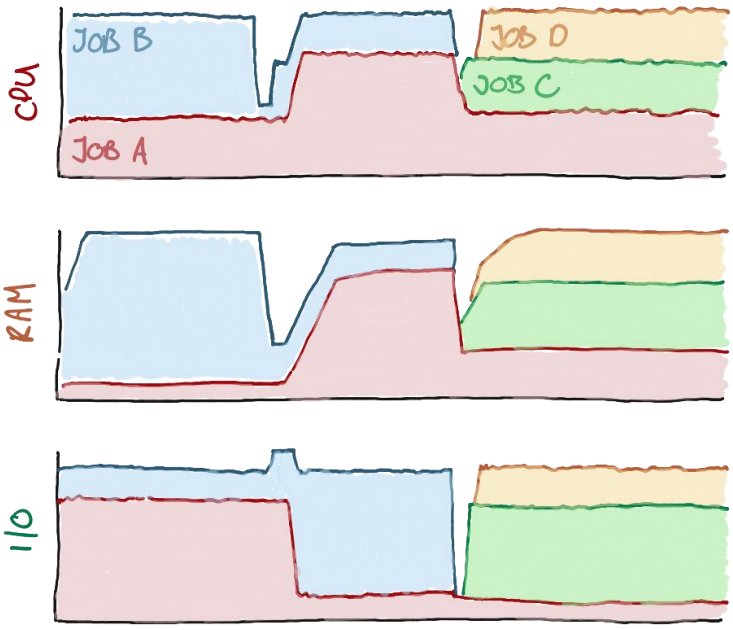
ORCHESTRATING THE PERFECT SCHEDULE

Batch workloads are an especially well suited candidate for optimisation, as both *what* runs and *when* it runs can be completely controlled at scheduler application level.

The more historical data we hold for a given job, the more accurately it should be possible to predict how it will behave and what it will require the next time it runs. The time and job dependency conditions for any given program are also a known and available control, as they will have been specified when the job was added to the schedule.

With jobs that behave predictably, where resource cost can be accurately estimated and have known trigger conditions and flow dependencies, it should be possible to plan a schedule of the workload in a way that makes best use of the resources available.

For example, consider the following ‘ideal resource usage’ scenario:



In this ‘optimised’ scenario, the program from the first example (Job A) executes with sufficient resource to complete in its best time. The remaining CPU, disk and memory resource perfectly satisfies the requirements of three other coinciding jobs, and minimal resource is left unused.

Calculating the ‘ideal batch schedule’ that would enable this kind of resource utilisation is a combinatorial optimisation problem. The same problem faced in other domains such as manufacturing and financial planning, and can be solved by applying existing algorithms. However, a few things must first be done to

enable the application of these algorithms.

There needs to be a way of profiling workloads for resource utilisation at the lowest logical level (ie. job or user session). Data for each job or user must be collected over a sufficient time period, and analysed for variances in resource usage, and where possible, how that variance is affected by differences in the volume of input data.

This will provide a confidence level for the historical model for that job, and also give a rough indication of the computational complexity of the job (and eventually each step within the job), which can in turn be used to project the long term growth of that job's requirements, estimated in line with the projected growth in data volume*. (Note, this is our current level of progress with ESM)*

Once these attributes are calculated, workloads can be *forward planned.* The necessary level of resource can be reserved in advance, and jobs can be distributed across multiple nodes of various types which satisfy the overall predicted workload requirements.

With responsive enough profiling mechanisms it should also be possible for a scheduler to monitor the performance of the workload in real time, comparing planned performance to actual performance and re-adjust the plan 'mid flight' if actual performance deviates, or an error condition is encountered.

All of this is just a question of implementation.

WHAT ABOUT INTERACTIVE WORKLOADS?

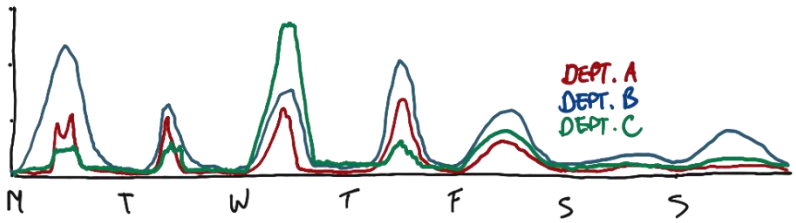
Up until this point this paper has mostly discussed batch workload analysis. An obvious pending question is 'how would I calculate the requirements for my user-generated daytime workload'. Before discussing that it is worth noting that there are other factors that make the management of interactive workloads considerably easier.

Most significant is the fact that there are no interdependencies to be concerned with. Any potential failure due to an out-of-resource condition is much more tolerable as a result, as it is very unlikely to cause subsequent failure of dependent jobs.

In addition, while interactive users traditionally shared bare metal hardware with their batch runs, there is rarely any actual *need* for them to really share the same nodes as the batch if their processing can be split up for negligible difference in infrastructure cost. In isolation, user behaviour can be modelled independently and managed using different mechanisms and algorithms, and batch resource protected from interactive user activity that may potentially destabilise it.

Last but not least, it is worth mentioning that modern analytical engines like the Cloud Analytic Server introduced as part of SAS Viya are by design much more predictable in terms of resource requirement, and far, far better at responsively scaling on cloud infrastructure than traditionally available technologies. CAS clusters can be resized on the fly, with data replication and node failure automatically taken care of. This makes them considerably easier to scale using existing on-the-fly auto-scaling mechanisms (more about these later).

With that in mind, interactive use also obviously follows a pattern, and therefore it makes sense that it should be modelled to enable more accurate resource reservation. Here is what a typical week of interactive usage looks like:



The same approach should be taken with interactive workloads: profiling the user behaviour over time, and analysing the data. While exceptions are a bit more likely, user behaviour on an interactive platforms is fairly repetitive and requirements are still predictable.

With simpler reporting applications behaviour will be more consistent, as users arrive at work at predictable times, generally look at refreshed versions of the same datasets and reports, and eat lunch at the same time. For more advanced ad-hoc data science exploration, advanced end users can be educated on resource requirements, and could subsequently be provided with controls that would allow them to estimate their own needs based on what they're looking to do, reserving cloud capacity interactively. Mechanisms can even eventually be put in place where users are rewarded for the predictability of their behaviour, much like cloud providers reward customers for up-front commitment to resource.

AI MACHINES LEARNING ABOUT BIG DATA CLOUD PLATFORMS WITH SOME INTERNET OF THINGS

The idea of modelling the interactive workload of users on Big Data platforms is far from new, and the problem is also a more interesting one. Big Data clusters can be *huge*, and a considerable amount of research and work has gone into the use of neural networks to model and predict user behaviour to enable predictive scaling of compute clusters.

However, most of this research has depended on feeding ML algorithms node-level timeseries data in the hope that they will be able to predict the immediate scaling requirements of that node. The aforedescribed complexity of the analytical workload and the resource variability it can introduce can render this node-level-metric approach less effective, and considerable gains in accuracy could be made by increasing the granularity of the data and adding it with logical attributes to aid feature selection, such as a process' user owner, and that user's department or other logical business grouping. These are the growth patterns that we have noticed on shared platforms, and by modelling at this level it should be possible to increase the accuracy of demand prediction considerably.

CONCLUSION:

One solution to the problem outlined here is an application-aware, backward-looking, forward-planning scheduler that can reserve and provision the appropriate cloud infrastructure autonomously. Control mechanisms exist that enable all of this to happen, and with the apparent recent advances in machine learning capacity, the accuracy of models should only get better.

The ability to forward plan and predict the workload also has its own benefits. Cloud computing providers are no doubt running their own models for this combinatorial optimisation, and they are already rewarding customers with considerable discounts in list price (50+%) if they are willing to commit in advance to a known volume of virtual infrastructure.

As Cloud provider pricing models inevitably grow in granularity, the landscape is likely to converge towards a model similar to futures trading, where tradeoffs will need to be made between the accuracy of demand forecast and the reduction in cost resulting from advanced-enough commitment to resource. With the way things are going, the ability to accurately predict Cloud resource requirements ahead of time is likely to increase in value considerably.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Nikola Marković
Boemska
20 Nugent Road
Guildford, GU2 7AF
United Kingdom
+44 (0) 20 3642 4643
nik@boemskats.com
http://boemskats.com