HIGH-PERFORMANCE COMPUTING ON AWS: CHOOSING THE PERFECT DATA STORAGE SERVICE FOR YOUR HPC WORKLOAD

Amazon S3 | Data and Analytics | High-Performance Computing (HPC)

beSharp | 4 March 2021

High-Performance Computing (HPC) has ballooned in importance in the last few years and produced truly remarkable results: from Fluidodynamics to protein shape and functional analysis, from Genomics to Computational quantum chemistry.

Even if the HPC began to boom well before cloud computing with the development of huge supercomputers starting in the '90s, the rise of cloud computing greatly democratized HPC. For example, a researcher studying a new material does not need to send a proposal to a supercomputer facility to calculate the electronic properties of its highly correlated material, he can just fire up 200 spot instances on his AWS account, run the workload, download the results, and finally switch off the cluster. All this for just a few bucks.

Going away with the lengthy and frustrating proposal review process needed to access a supercomputer facility, the scientist can obtain the same results in a fraction of the time with minimal effort, and act more quickly on the results: should a new experiment find out a slightly different composition with better properties than the original sample, the researcher can simply rerun the analysis overnight without any additional overhead.

Having the compute power you need, exactly when you need it, and pay only for what you use has a huge value in these situations.

Setting up a cloud-based HPC workload, while significantly simpler than its on-premise counterpart, is all but a trivial task. However, once the setup of a cluster capable of executing the task at hand is complete, its configuration can be easily saved and replicated using Infrastructure as Code (e.g. Cloudformation).

Every AWS HPC workload can be divided into two separate components: **Storage** and **Compute**.

This article will mainly focus on the Storage side and see how s3 can solve most of your problems. In a soon to come article we will instead focus on the creation of a real-life HPC cluster.

Storage solutions in AWS

Data storage is a long-standing problem in HPC computing: in an on-premise HPC cluster, you'll probably go for a distributed file system such as Lustre or a SAN-based solution as OCFS2 (or both) but on AWS you have many more options and you can combine them to get the best results for your workload.

Here is a shortlist of the storage options which can be used in an HPC cluster on AWS:

S3 – Simple Storage Server

Very powerful general-purpose object storage packed with lots of cool features. S3 is the second oldest AWS service (after SQS) and is now nearly **15 years old and very mature**.

Some of the most useful features are:

Event-based notification: you can receive notifications when an object is written or modified/deleted. Integrated with SQS.

Bucket replication: it is possible to use this feature to duplicate the contents of the S3 bucket in other AWS regions and/or Accounts for backup and application segregation purposes.

Storage classes: when you upload a file you can choose the storage class.

1. *Standard storage* is well suited for write seldom – read many applications.
2. *Infrequent access* is more suited for applications with not too many read request: storage for infrequent access cost half as much as standard storage but you pay twice as much for a GET API request.

Intelligent Tiering: let AWS determine the correct tier for a stored file from its access pattern and move it to the most convenient storage class accordingly.

Lifecycle policies: Data usage varies with time. You can set-up transitions to move old files to less expensive storage classes. Very old files can also be moved into AWS Glacier automatically for cold storage. You can even decide to delete ("expire") a file when it gets too old!

EBS – Elastic Block Storage

General-purpose single instance Network-based block store. It is a single-instance storage and comes with a wide range of flavors (Volume Types) each with different purposes and settings available.

For example, general-purpose (GP) disks can be used to host the root directory of an EC2, while more expensive provisioned IOPS (io1, io2) disks are more suited for DB engines data storage.

A multiple instances mount option is available for io1-2 disks but should be used with care, and only if the filesystem is a cluster-aware file system such as OCFS2 or if the disk is mounted as **read-only.**

Instance ephemeral Storage

Some of the AWS EC2 instance types come with scratch physical SSDs attached. The data in these disks is ephemeral and is lost when an instance shuts down. These types of instances are very useful.

EFS – Elastic File Storage

NFSv4 compatible storage. It is a very flexible solution since its storage capacity scales automatically, and it's even possible to move seldom accessed files to an infrequent access storage option that has a storage pricing identical to S3.

When using EFS one should always plan the needed performance, in fact, two different types of disks can be created: **Bursting** and **Provisioned IOPS**.

Bursting files systems can be used for applications that have a low flat disk usage with occasional usage spikes, while the much more expensive Provisioned IOPS option should be used for applications that need consistent performances and have a significant disk usage.

FSX for Lustre

A cloud-native version of the popular Linux Lustre Cluster file system; this distributed file system comes in very handy if you need to port to AWS a workload already using Lustre on an on-prem cluster.

However, as good as it can be, FSX Lustre comes with a few shortcomings: the file system is quite pricey (in particular for persistent storage), and the minimum size quite large (> 1 TB).

Furthermore, Lustre can be deployed in just one AZ so if the cluster spans multiple AZs you incur significant slowdowns and data transfer costs.

Despite these compromises, which were unavoidable to get the Lustre filesystem on AWS, Lustre also comes with some significant advantages: better pricing at scale and lower pricing than EFS, and extremely high performances up to hundreds of thousands of IOPS.

In general, an HPC workload may use a subset of all these storage solutions at the same time and decide from time to time which one to use.

Workload analysis

In most workflows, several steps are pretty mundane and similar to each other, even for very different applications.

Hereafter we'll refer to an infrastructure, we recently set-up, for a real-world genomics HPC data analysis workflow; many of the steps in this workflow fit very well on one storage type, so you don't need to overanalyze it, here is what works best in most scenarios:

App User Input/Output

Many analytics workflows need to receive bulky input data, and produce equally bulky analysis output to be shared with the end-users. For this step the natural storage choice is **S3.**

Objects can be uploaded directly to S3 from customer-facing applications, so you don't need to upload files to your backend before transitioning it to S3.

In our case, the genomics files are uploaded directly to S3 by the browser using the Javascript AWS SDK, and the AWS credentials needed to do so, are provided by a credential vending machine implemented in the web application backend.

The analysis output can also be downloaded from S3 using a pre-signed URL.

Nodes Root directories

Selecting a storage type for root directories is trivial: the only supported choice in AWS is EBS!

Code and Libraries

The code of your applications is the core of the HPC infrastructure, and the reason you decided to set it up in the first place!

But where should you store it to make it available to all the cluster nodes? The most obvious solution is to just store it on an EFS volume mounted on all the nodes. Unfortunately, this is often a terrible idea: EFS is an NFS volume, and thus has several performance pitfalls when working with a great number of small files, as very common in software libraries.

A better solution is to just use a dedicated EBS volume: you can install your app using a solution like Ansible, and use it to install and update the software on all the nodes.

Furthermore, you can simply create a new AMI for your nodes every time a new version of the application is released, if your cluster is scalable, so that each new node will start with the new version of your application.

If you want a simpler deployment solution, and the code bundle is small (no more than a few Gbs as a rule of thumb), you can just store it in S3, and download the code bundle when a new instance is started.

However, if:

1. the applications are relatively compact,

2. run mostly in RAM (e.g. python scripts),
3. do not need huge numbers of lib files,

distributing the code using EFS may work just fine, and save you a lot of configuration effort for the creation of a customized AMI builder pipeline.

This storage case is less clear-cut than the previous ones, thus real-world scenarios should be always analyzed with care, and performance tests carried out frequently.

Static assets

Static Assets are a wide category of artifacts used by HPC applications to carry out calculations. Examples of static files are genomics gene descriptions and precomputed APW representations in Quantum Chemistry.

These files can vary widely in size, ranging from a few MBs to several TBs. These types of files have several things in common:

1. They seldom change in time
2. They are read-only (write once read many)
3. They are needed by all the cluster nodes

The most “natural” location for these files is usually S3, but things often get much more complicated than it seems.

Storing these artifacts in S3, while in principle always preferable, only makes sense if you completely control the access pattern to the artifacts (you wrote all the code), or if the library you are using explicitly considers files in S3 (almost always not the case).

If you need to use a third-party library to access the artifacts, you may prefer having them in a block storage, because external libraries are usually created to manage files stored locally in a block file system.

What comes immediately to mind, is that we could use EFS, but this is usually impractical if the instances need to read a great number of files in a very short time, as several nodes doing this at the same time, will likely saturate EFS throughput.

While Lustre can also be chosen, it is often more cost-effective especially for multi AZs clusters, to just store the files in a dedicated EBS disk for each node.

The new GP3 disks are dirt cheap and provide good enough performances. This additional disk should be added to the node cluster AMI, and consequently updated with the same flow proposed for the application code.

Another smart solution for read-only file systems, is to use S3 as a block storage. To do this, you can leverage existing solutions like s3backer or write your own custom solution. The performances, for S3 buckets in the same region as the cluster, are surprisingly good and often rivals EBS.

Single Node Scratch

For single-node scratch, you can only choose between instance storage and EBS.

Instance storage is by far the faster solution, however, if you decide to scale your cluster using spot instances, the cluster scaling will be significantly constrained by the number of d-type instances available (e.g. m5d.large). Furthermore, you'll need to format the instance storage using user data when the instance starts.

Always keep in mind that each AWS d-type instance has a different SSD storage type, and you cannot change it (full table here).

For some workloads, where the size of the scratch can vary wildly, an instance dedicated EFS storage can also become a cost-effective solution.

Cross Node Scratch

Cross node scratch can be a very expensive mess, and should be reduced to the bare minimum. The best solution is always to store the data in S3, but as explained before, this is often not possible.

The only workable alternatives are Lustre and EFS, however, while EFS is only cost-effective for small-sized scratch sizes, Lustre wins hands down for high throughput shared scratch size, both for cost and speed.

Conclusion

In this article, we described the different storage options available for cluster storage in AWS, and analyzed how to use each of them. We learned that, when in doubt, S3 is most probably the right choice.

If you have any comments or suggestions feel free to write to us in the comment section, and as always, have a good AWS architecting day!

beSharp

Since 2011 beSharp has been guiding Italian companies on the Cloud. From small businesses to large multinationals, from manufacturing to the advanced service sector, we help the most advanced companies to implement innovative projects in the IT.

Get in touch

beSharp.it
proud2becloud@besharp.it