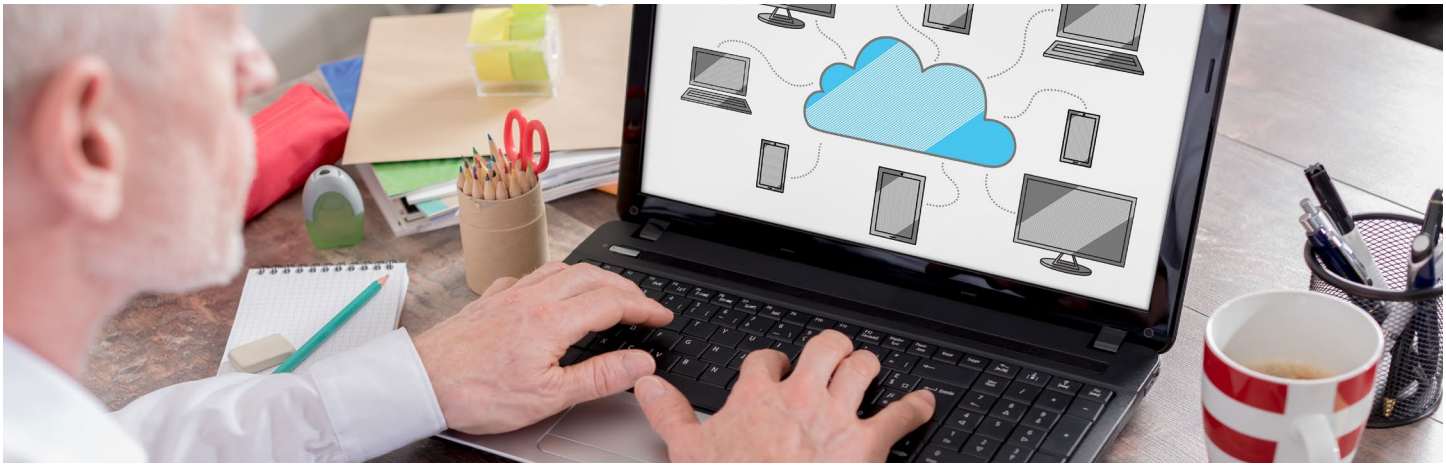
PLANNING CLOUD STRATEGY FOR HPC AND HIGH-THROUGHPUT APPLICATIONS

Dr. Rosemary Francis – Chief Scientist, Altair / April 14, 2021



When adopting flexible compute, half the challenge is in selecting a storage solution that suits the application. Whether you are tuning for performance, throughput, or cost, or a combination of all three, it's important to balance the compute nodes with the right access to data so that you are not paying for underutilized resources. Whether you get the storage architecture right or wrong can make or break the success of the cloud deployment; overprovisioning can escalate costs far beyond the expense of an in-house solution, but with reasonable planning there are big cost savings to be found.

Replicating in-house infrastructure can get you started, but it shouldn't be a long-term plan, as high-performance shared file systems are a very expensive way to use cloud storage. Sizing cloud storage is not just about knowing how much you need, but also about bandwidth, latency, peak performance, and fault tolerance. Putting all this together means designing a storage solution that suits the application, and the key starting point lies in understanding the storage requirements.

This white paper will present tried and tested means to help you take advantage of the elastic compute offered by the cloud. Establishing good in-house working practices from the beginning will help you to get much more from your in-house infrastructure and make it easier to move workflows to the cloud.

The Beginning: Don't Mirror Your In-house Environment

The temptation when moving an HPC application to the cloud is to start by replicating your on-premises environment with shared storage. Although there are shared storage solutions out there and you can install Lustre or Spectrum Scale (formerly GPFS) on the cloud, this is not the way to achieve the performance or cost tradeoffs that you get on-premises.

Instead, applications need to be untangled from legacy static compute environments and so allow them to start to use storage in a more flexible way that can take advantage of the solutions available in the cloud. Aside from being able to run high-performance applications in the cloud in a scalable, cost-effective way, the same strategy will make the in-house environment more efficient and you can take advantage of hybrid cloud setups.

Selecting the Right AWS Cloud Storage Option

There are many storage and compute options in the cloud and the list is growing every day. The following is a short introduction to what is available on AWS today. The ever-changing landscape of options makes it even more important to understand application requirements and to remain agile about how the applications are deployed. These options are specific to AWS, but other cloud vendors have comparable options.

Instance storage – Instance storage is physically attached to an EC2 instance. Therefore, it does not support data persistence once the instance has been terminated and is vulnerable to hardware failures. It's great to use for temporary scratch space because it can

support higher throughput and lower latency than networked EBS storage. The largest instance storage family, I3, can support 3.3 million random IOPS at 4KB block sizes with up to 16GB/s of sequential disk throughput.

Uses for local instance storage: Temporary files and scratch space. Storage for a short-lived high-performance shared file system that is implemented in EC2 instances.

Amazon EBS – This is for persistent volumes that are attached to EC2 instances over a high-speed network. It is reliable, high-performance storage that can be quickly resized as needs grow. EBS storage can achieve up to 80K IOPS or 1750 MB/s for a single EC2 image by striping multiple volumes together. The base and peak performance of each volume depends on the type of EBS volume selected. The maximum I/O available to an instance also depends on the type of that instance. It is therefore important to understand storage needs and select both the storage volumes and EC2 images that match the needs of the application.

NFS-based shared storage implemented using EBS is a good first step when migrating applications to the cloud as it mirrors your in-house environment and requires very little rearchitecting of your applications. It's easy to launch an EC2 image with EBS storage and configure it to be a storage server. For this to be a cost-effective way of running HPC applications, data that is no longer needed for current workloads needs to be tiered to S3 (see below).

Uses for networked block storage: Storage for data that needs to be persisted and will change throughout execution of the application. Storage for a medium-life high-performance shared file system that is implemented in EC2 instances.

Elastic File System – Amazon Elastic File System (EFS) provides a scalable NFS file system with high availability and redundancy. It is often used when moving HPC applications to the cloud because it can be set up as a drop-in replacement for in-house NFS storage. However, doing this will give disappointing results because it won't achieve the performance that much faster in-house shared storage can achieve. Only small amounts of data that needs to be uniformly available to large compute environments, but not accessed through a high-performance connection, should be stored on Amazon EFS.

Uses for cloud NFS: Small amounts of fairly static data that need to be available across all instances, such as home directories and application binaries.

Amazon S3 Object Storage – Object storage has a very simple web interface that makes it efficient to store or retrieve large data sets but inefficient for small changes. It can be accessed anywhere on the web so is ideal as a point of entry for large data sets or for use in backup. It's good practice to use S3 as the main data store, fetching datasets and storing results at the start and end of each major workflow. It's fast to pull data from S3 to EBS or Instance Storage so S3 can act as a long-term store for data that is not actively in use.

Uses for S3: Large datasets that will be infrequently updated and backup solutions.

Example Use Case: A Large Batch of Regression Tests

Storage persistence using S3, shared filesystem using EBS, temporary files on Instance storage:

Set up the shared file system using EC2 and EBS to start with. Run distributed regression tests, each one using local Instance storage for tmp files, but writing the results back to the EBS-based shared file system. Once the regression suite has finished running and the results analyzed, tear down the shared file system, storing results that need to be persisted back in S3.

Picking the Right Storage Options with Altair Breeze™ and Altair Mistral™

Altair's Breeze and Mistral solutions will help you untangle applications from legacy in-house infrastructure and architect the right solution for the cloud. Cost and performance are both key to making sure the cloud is a viable solution.

Step 1: Understand application dependencies with Breeze

The first step is to find out which programs, files, and network dependencies each application has and what the access patterns are. The Breeze solution will automatically identify temporary data, sequentially accessed data, and random I/O, as well as tell you everything you need to make sure that your application runs the first time in the new environment.

Often at this stage Breeze identifies some bad I/O patterns that could be fixed, but the choice to fix them or work around the bad I/O as a requirement is up to you. In addition, how much you redesign and how much you simply replicate at this stage is up to you, but Breeze will tell you everything you need.

Using Breeze is a two-stage process. First you need to trace your application. This is typically as simple as typing the following command and letting the application run as normal:

```
trace-program.sh -f <output_dir> <my_app> [my_options]
```

Once tracing is complete you can either use the interactive UI to explore the trace in detail or use the command line to export various reports, including a complete list of files and libraries the application touched.

Using the reports and the Breeze UI, you can determine not only which files are needed by your application, but also the access requirements. For example, random I/O can be a problem for applications, depending on the storage used. This can be identified by looking at the ratio of seek operations to other I/O. Sequential I/O that can be identified as this will have few, or no, seek calls in comparison to the number of reads and writes, whereas random I/O will have a much higher number of seek calls recorded.

This information is also listed in the Breeze Healthcheck report but can be looked at in detail in the Breeze UI either via the file usage information to identify those files that are randomly accessed, or via the I/O profiling statistics information that breaks down the behavior of processes over time.

Step 2: Specify application storage domains

Once Breeze has identified each of your dependencies you will be able to adjust the suggested storage strategy and architect your cloud solution. The following guidelines will get you started, but you may want to make domain-specific tradeoffs based on extra information that you have about the way you want to run your applications or the way you want to persist your storage:

- Config files and application binaries should be on EFS.
- Temporary files should be on Instance Storage.
- Large datasets should be stored on S3 and pooled to EBS.
- Sequential data should be on disk, and random I/O or small I/O needs to be on SSD.
- Distributed applications with shared data sets might want to set up an EBS-based shared file system for use during the run.
- Results should be stored on EBS and tiered to S3 once the data has been consumed by a human or otherwise.

Step 3: Profile compute and storage needs for each storage domain with Mistral

Now that data has been divided into domains, you need a lightweight profile that gathers accurate runtime information about the expected throughput and bandwidth requirements at scale. The Mistral tool delivers this for each of the storage dependencies of your application. It characterizes the peak and baseline storage needs of your application for each of the domains specified in the previous step.

Mistral monitors and summarizes the behavior of an application based on threshold rules defined by the user. As Mistral does not record all the fine-grained detail, it is much more lightweight than Breeze and can be used to run wider tests across a cluster. To identify jobs with bad I/O patterns, Mistral can be configured with rules that monitor all the areas of interest on a cluster, including mountpoints and typically hard-hit areas such as scratch directories.

For example, the following set of Mistral configuration rules monitors a user's shared directory for the number of file opens, seeks, and reads and writes, as well as small reads and writes that occur.

```
user_open,/shared/$USER/,open,all,count,1
user_seek,/shared/$USER/,seek,all,count,1
user_rw,/shared/$USER/,read+write,all,count,1
user_small_rw,/shared/$USER/,read+write,0B-4kB,count,1
```

This information can be used to identify how the files in each user's shared directory are being accessed and how often. Looking at the ratio of the seek calls to reads and writes will identify whether file access is random or sequential. Comparing the open counts to the reads and writes can tell you about the typical usage of files in this area, for example whether files are opened, read, or written to once

and then closed or kept open. By comparing the count of read and write calls to the count of those calls that were less than 4kB in size you will understand what proportion of I/O to this directory is made up of small requests.

These rules can be augmented with more information such as the complementary bandwidth rules to the read and write counts, which will add information related to file size and data throughput.

To run Mistral on a single application, simply edit the mistral_monitoring_setup.sh script that is shipped with the application to include your license details and config file. Mistral also comes with sample configurations for logging lightweight or detailed information. You can then wrap an application using the Mistral.sh script after sourcing the setup script to set up the environment. The simplest setup sees the profiling data written out to a log file.

To run Mistral on a larger set of applications, follow the instructions for setting up job launchers on your scheduler or add the mistral wrapper to your own job launcher scripts. For a lot of applications, it is better to set up a database such as InfluxDB or Elastic Search so you can see the results in Grafana. For a complete explanation of what is possible please refer to the Mistral User Guide, which is available on our website.

Step 4: Map storage needs to AWS instances
The last step is to select AWS solutions for each storage domain. This could be as simple as mapping the bandwidth and IOPS requirements to a specific instance, or you may need to aggregate the storage needs of thousands of tests or parallel applications to get the overall needs from a shared storage solution. Even if you go to a third party to implement the storage solution for your infrastructure, they will need the data you've gathered to design something that is right for you.

The data you get from Breeze and Mistral enables you to make the right choices and to ask the right questions of your storage solution providers.

Best Practices When Migrating Applications to the Cloud

Most of the time, it is possible to migrate applications to the cloud as they are, but it can be advantageous to look at optimizing the I/O patterns of the application as part of any other modifications to the runtime environment that are needed. Some I/O optimizations require code to be rewritten, but in other circumstances a simple change to the way an application is run can give big rewards. For example, redundant locations in a path variable can trigger thousands of failed metadata calls to open files that are not there. Simply changing the variable in the launch script can remove the load on the metadata service.

As well as being used to assess storage needs, the Breeze trace can also be used to produce a Breeze Healthcheck report that breaks down the behavior of the application and highlights areas where it behaves in ways that could indicate poor application design. For example, one section of the report lists the processes that had the worst file open to I/O operation ratio that may indicate a badly constructed loop, while another section lists details of directories used by the application that contain a large number of files.

Make Sure Your Code Is Fast, Agile, and Cloud-ready

Cleaning up bad I/O patterns is good practice for any application or workflow with long-term use. There is nothing about cloud platforms that particularly requires good I/O, but whenever the compute environment is changed, there is potential for bad I/O patterns to become a problem when previously they were masked by a bigger problem or a different architecture. By building I/O profiling into the quality assurance processes and continuous integration of your applications and deployments, you can ensure long-term flexibility within your compute environment.

For more information about Breeze and Mistral, please visit altair.com/contact-us.

