ADMIRE

malleable data solutions for HPC

Quality of Service at Scale

the ADMIRE approach

In large systems HPC, the common assumption that resource sharing can result from self-organization in the name of a common good tends to not match observation.

Fig. 2: Resource sharing as hoped, and as observed on a large HPC systems

The inherent complexity of HPC systems, the difficulties for end-users to get an in-depth understanding of the environment, and the numerous different resources which are effectively shared leads to suboptimal situations. Experience has shown that a very limited number of deviant jobs can produce large amounts of overhead, which end up significantly degrading the efficiency of the whole system. This is especially acute for the back-end's file system which is, by essence, a shared resource that ends up being used concurrently by all the jobs running in the system.

On the other side of the spectrum, however, the software stack of extreme-scale HPC systems is designed for high-throughput. Such software stacks are built for speed and lack the quality-of-service (QoS) guarantees required by a substantial class of data-intensive applications. An absence of coordinations leads to costly interferences: for instance large data transfers on behalf of simulations, big-data applications, or checkpointing can collide jobs exercising many small random file accesses, such as deep learning. The service of many small I/O requests typically degrades the average response time of the file system. As

a summary, available bandwidth and I/O bandwidth of the back-end file system are underutilized and applications suffer substantial performance penalties.

ADMIRE aims to provide QoS mechanisms to shield jobs from interferences. Either by isolating deviant jobs, as well as symmetrically enabling the shielding of vulnerable jobs from I/O interferences. To support this, the ADMIRE software stack collaborates with the Lustre Parallel File System (PFS) to establish both a controlled ADMIRE partition and a General Purpose partition. HPC applications choosing to rely on the ADMIRE software stack will benefit from its optimization loop, which uses real-time monitoring information of the system to intelligently optimize the usage of I/O resources. Thus, a fraction of the HPC system will operate using optimized I/O thanks to the ADMIRE stack, while the remainder of the cluster is considered as general purpose, and thus out of the perimeter of ADMIRE control.

Modular architecture and I/O interferences

ADMIRE is applying the concept of Modular Architecture to I/O, in order to segregate jobs depending on their interference profiles. A HPC system using the ADMIRE software stack eventually ends-up with a module optimized for I/O, where applications can be executed with a minimal level of interference and also deliver a predictable time to result. To achieve this, applications within the ADMIRE partition execute in the context of an ad-hoc storage system, which typically leverages the node-local storage from the application's compute nodes to provision its I/O necessities. This essentially moves normal application I/O from the shared backend file system into the ad-hoc storage system, effectively *containerizing it* and removing one of the major reasons for uncoordinated I/O and degraded performance.

Even so, complete isolation from the shared PFS is not possible, since an application needs to *consume data* as well as to *produce data* in order to be considered useful. This means that in order for ADMIRE to support such an I/O ecosystem, data needs to be moved back and forth between Lustre and the ad-hoc storage instances.

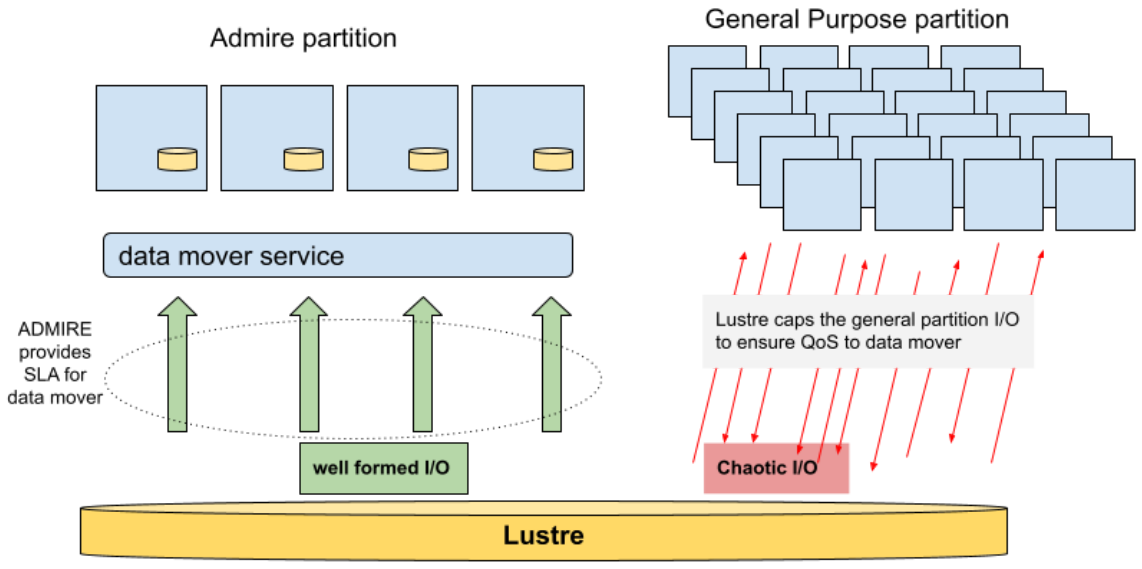

Fig. 2: Within an Exascale System, Admire is a sub-system of Compute node with Optimized IO both due to a better hardware infrastructure (local storage) and advanced software (Admire stack). Most of

the Exascale System is made of General Purpose compute nodes without I/O optimization. Lustre, the network fabrics and the ADMIRE monitoring solutions are elements which are shared between the Admire sub-system and the General Purpose solution.

In ADMIRE, this *Data Mover* is implemented by Cargo, a MPI-based parallel copy tool. Cargo receives the list of files to copy-in at the inception of the job and also of the list of files to copy-out to Lustre at the end of the job. This information is extracted by a dedicated Slurm SPANK plugin that allows users to provide this information, and is further curated with the help of application models provided by ADMIRE's Intelligent Controller. Once the application's data needs have been determined, the transfer requests are sent to ADMIRE's *Storage Coordinator* component (scord), which is responsible for scheduling the requests depending on the system and application needs and forwarding them to Cargo. Cargo will then execute the required data transfers on behalf of the application. Ultimately, at the end of the job, when the ad storage will have to be destroyed, Cargo will move the output files to Lustre. When compared to direct PFS access from HPC applications, this workflow offers knowledge over *which* data needs to be read/written from/to the PFS and, more importantly, control over *when* a PFS access should be executed.

SLA predictable Job inception

As we discussed above, knowing the list of files to copy in or out offers management advantages, one of which is that it is possible to estimate the volume of data that needs to be moved between the PFS and an ad-hoc storage instance. Furthermore, the I/O patterns exhibited by the Data Mover are well-controlled, since they will typically consist of sequential reads or writes in the shape of large transactions. Such I/O patterns offer high predictability, which further eases the estimation of bandwidth and RPC transactions to provision from the PFS.

Thus, in ADMIRE Slurm interacts with Lustre in order to compute and set-up the corresponding QoS limits to all jobs running on the General Purpose partition, so that the Data Mover is capable of enjoying a SLA. This SLA guarantees that all data transfers required by an ADMIRE job can be executed on time by Cargo, and thus, that a job scheduled on the ADMIRE partition runs in its due time. Furthermore, this allows a very precise control over the shared portion of the PFS used by an ADMIRE job, which can be dynamically adjusted depending on the needs of the system.

The exact means by which Slurm interacts with Lustre are still to be defined, but we envision that, at the very least, an API is required that allows setting the TBF (max amount of resources assigned) to all the jobs on the system, at the exception of the ones from the Data Mover itself. Our current line of work is to integrate this API into ADMIRE's Storage Coordinator component, firstly because it already supports a certain level of communication with Slurm using RPCs, and secondly because architecturally it makes sense.

With this in mind, the overall algorithm to compute the QoS limits would be the following:

RPC = estimation of the maximum amount of RPC available in the system
BW = estimation of the maximum amount of bandwidth available in the system

General Purpose RPC Allocation = RPC - (ADMIRE RPC need)
General Purpose BW Allocation = BW - (ADMIRE BW need)

This assumes that the General Purpose partition resources are distributed in an evenly manner, which is not an issue as resources are allocated only if needed. As an example, considering a 10 jobs General Purpose partition, if a job requires more than a tenth of the available bandwidth for the partition, it will be served as long as the other jobs do not consume their own tenth. Note, however, that we are not yet considering the cost of data eviction from the ad-hoc storage to the PFS. The best way to include such cost into the performance and QoS equations is still an open topic of discussion for the ADMIRE partners.

Estimation of Admire Gain of General Purpose execution

As we discussed above, ADMIRE addresses the I/O challenge at scale with a smart usage of ad-hoc storage. Ad Hoc storage is deployed on an ephemeral basis (job), and serves most if not all the I/O traffic from the job. This shields the Lustre file system from troublesome I/O patterns and delivers optimal performance for the job running in the ADMIRE partition. However, not all jobs exhibit complex I/O patterns, and scheduling a job with a simple I/O pattern is spending ADMIRE resources while the I/O traffic could be efficiently served directly from Lustre.

To solve this, ADMIRE relies on its monitoring and characterization tools, that allow the live monitoring of I/O patterns for all jobs. Thanks to Extra-P and other modelization tools, it is then possible to estimate the gain that ADMIRE would provide when compared to an execution in the General Purpose partition. The estimation of the gain will be reported by Slurm at the end of any job execution.

Data movement Optimization

Appropriate orchestration of data movements is the key to ADMIRE success. While the best way to do this is still an open research question at this stage, our current working idea is to monitor the way data is consumed by an application's ad-hoc storage in order to minimize the volume of data to stage in Job inception:

- Step 1: Stage-in data and files as requested from the Slurm command line
- Step 2: Check how files are consumed, partially or fully. Estimate the gain from not staging the not consumed data.
- Step 3: If the same job (or a similar job) is enqueued again, balance between the gain of not staging in all the data and the cost of misprediction. Depending of the relatively low cost of over-stage-in and the high cost of misprediction we may end-up with a similar behavior than data sieving techniques in MPI-IO

Moreover, the application models generated by ADMIRE also open up opportunities to determine when data is going to be needed (or dropped) by an ad-hoc storage instance. Provided that we are capable of estimating these needs with a high probability of success, it should be possible to attempt dynamic data transfers while the application is running.

Conclusion

Within ADMIRE we propose to implement a multi-layered Quality of Service. Firstly, by rewarding the end-users making the effort to express their I/O requirements. These end-users are promoted to the ADMIRE partition, where a subset of nodes are augmented with local storage and the deployment of an ad-hoc storage solution. Applications running within the ADMIRE partition are therefore protected from most of the general activities ongoing in the cluster.

The second layer of protection against interferences is the capping of all these general activities in order to guarantee a fraction of the file system resources (IO, and IOPS) to the essential data movement occurring at the beginning and the end of any of the ADMIRE jobs.

We believe that this approach, pragmatic and scalable, is suitable for Exascale class super-computers. With a minimal amount of centralization, hence congestion, ADMIRE will be able to deliver SLA for a large class of applications. An elusive problem up to now in HPC.