Contribute to other work packages

The control programs will be inserted into the ADMIRE components and applications at control points. Control points are places in the system that implement reconfiguration or scheduling actions [15]. The control points can be used for implementing application-local or system-wide policies.

WP3

3.1 Base mechanisms for malleability

- mechanisms for malleability of compute and io resources
- malleability protocol for expanding/shrinking ad hoc storage system
- runtime mechanisms to allow malleability overall
- API for programmers to indicate that the application can respond to external malleation requests

JGU will design malleability interfaces for the ad hoc storage system in three directions:

1. The ad hoc file system can be expanded or shrunk while it is running. Users can still access the ad hoc file system during this process, albeit with some performance penalties due to the relocation process. The entire ad hoc file system can also be re-located to an entirely different set of nodes and node numbers. This is beneficial when another job's application must access intermediate data but cannot be run on the same set of nodes (due to batch scheduling challenges). Further, this saves intermediate results being stored to the back-end parallel file system and then being reread in another job, causing unnecessary network traffic.
2. The ad hoc file system offers an API to throttle file system operations, that is, metadata operations and read/write operations. This can impact the overall CPU utilization of the file system and can be beneficial with regards to application performance when the application and the ad hoc storage system are co-located on the same nodes. Moreover, throttling I/O operations can reduce network interference of other applications.
3. The ad hoc file system offers a rich configuration interface to enable or disable certain file system features or file system protocols. This can include disabling certain metadata fields, e.g., permission bits, or relaxing file system protocols, e.g., the file creation process, with the goal to increase performance when a feature is not required by the application.

3.2 Scheduling algorithms and policies

- develop algorithms to maximize system throughput and minimize response times of individual jobs
- expand or shrink jobs and improve I/O and computation balance

JGU will contribute their knowledge concerning Quality of Service (QoS) to the decisions of the scheduling algorithms and policies. As the I/O bandwidth to the global parallel file system is limited, and a single application can severely degrade the overall system performance, it is necessary to fairly distribute the bandwidth on the users while taking priorities into account (some users have a higher priority than others and are privileged to more bandwidth). Taking the bandwidth information into account, e.g., via the QoS Lustre extensions developed by JGU and DDN, the scheduler can make informed scheduling decisions based on the current bandwidth usage of the system. Other Slurm plugins, defined and implemented in WP 4, allow users to ask for the required bandwidth when allocating a batch job, which the I/O scheduler can also use for their scheduling policies.

3.3

JGU will implement the designed interfaces into the ad hoc storage system and provide a corresponding API as a control point, allowing the malleability features outlined in T3.1

3.4

JGU will design a new Slurm plugin that extends today's Slurm API so that malleability features in T3.1 can be controlled through Slurm directly. For instance, the Slurm plugin could allow resizing the ad hoc file system or re-locate it to an entirely new set of compute nodes without further user actions. Moreover, JGU will consult with UC3M and TUDA with the goal to provide an API that can integrate the malleability protocols/policies with the malleable runtime and the scheduling policies into the Slurm plugin. With the help of PSNC system and application-centric metrics are evaluated in a real-world environment.

WP4

4.1 definitions of APIs, QoS metrics

- clients can give hints about their I/O requirements to the I/O scheduler when asking for compute resources
- They define which data is accessed, lifetime of data (should it be in gkfs or pfs), if other workflows reuse the data

- We define QoS metrics, such as how many I/O operations per second (of a specific size, say 1 MB), and how resources are prioritized to users
- These can be used to describe the QoS requirements for the job
- We define a syntax to convey the usage of the ad hoc file system
- User API to request data movement between storage tiers
- Add addiitonal user code if the data should be transformed in any way before storing it on the backend FS, e.g. compression.

JGU will lead this task due to their knowledge of the I/O requirements on their ad hoc file systems.

JGU will define the required APIs for the batch scheduler with the project partners so that users can convey their I/O requirements to the I/O scheduler. Example interfaces in the context of ad hoc file systems are

1. paths to the input data and paths where the output data should be placed within the PFS;
2. how long the data should be available on the ad hoc file system, that is, should the ad hoc file system be scheduled within or outside the boundaries of the batch job;
3. if other jobs need access to the data within the ad hoc file system. In this case, the ad hoc file system can run on a dedicated set of nodes, or the following jobs are scheduled to the same nodes of the node-local ad hoc file system. Nevertheless, I/O requirements can also include information about the data placement and distribution beneficial to the users' application.

Further, we define QoS metrics which allow insights on the used bandwidth of a user application. For instance, this can be based on the token-based system of Lustre's QoS extension that JGU and DDN developed in the past in which a user is allowed `x` amount of RPCs per second with each RPC being worth 1 MiB, regardless of wheather a user's I/O request uses the full megabyte of each RPC. This allows us to achieve insights into an application's behavior and allows users to ask for `x` amount of required bandwidth as a newly added the batch scheduler API. Based on the current usage and priorities (some users have a higher priority than others), the resources of the back-end storage system are fairly distributed. Such QoS metrics or, in general, the workload utilization should expand to the ad hoc file system so that users can make informed decisions on the ad hoc file system efficiency when running their applications.

Lastly, an API is defined for the batch scheduler, allowing users to ask for stage-in/out processes between storage tiers, e.g., PFS and ad hoc file system. In addition, the API should include ways to include custom intermediate code while moving the data between tiers. For instance, the job's output data could be processed and compressed before storing it on the PFS.

4.2 Scheduling algorithms and policies

- develop I/O scheduling algorithms and benchmarks to evaluate them.
- the current state of the system is used to drive scheduling decisions.
- Goal is to minimize storage backend congestion, enforce QoS constraits for each job, and reuse data on local storage to minimize data movement and leverage on locality.

- Algorithms to predict I/O behaviors and user/application behaviors with machine learning and other techniques
- Goal is to minimize waiting times

JGU will offer methods to reduce congestion by coordinating ad-hoc file systems and the storage back-end in three ways:

1. We'll define and implement optimized data movement strategies to minimize reading and writing data from and to the PFS, e.g., when staging-in/out data.
2. We enforce the QoS requirements in the batch scheduler defined in 4.1 by leveraging on the Lustre QoS extensions.
3. We implement the interfaces defined in 4.1 so that data can stay within the realms of the ad hoc file system across multiple jobs if they operate on the same input data or rely on the intermediate results of the previous job. In cases the same amount of nodes cannot be used, the data should be transferred between the compute nodes instead of storing them on the PFS.

4.3 On site, in transfer data transformations

- Create control points inside I/O scheduler
- Goal is that users can add custom code that is executed while data is transferred on the network or on-site when the data is on the local storage
- on site data should be resued by rescheduling connected simulation and analysis tasks to the same nodes
- control points will be offered to other admire components and ad hoc storage systems so that applications can use the functionalities through them (????)

JGU will implement interfaces and tools allowing users to execute their custom code for data processing at the compute node (in-situ). These users can then execute user-defined scripts or significantly extend certain file system I/O operations. For example, a modular file system interface could allow custom code to be executed before the ad hoc file system writes the back-end data to disk. One possible use case is the on-the-fly encryption of sensitive data so that raw data is never stored on node-local storage devices, which other users have access to in later scheduled batch jobs.

Revision #4
Created 29 March 2021 09:57:45 by Lefthy
Updated 28 June 2021 08:02:09 by Lefthy