SLAM: Scalable Locality-Aware Middleware for I/O in Scientific Analysis and Visualization

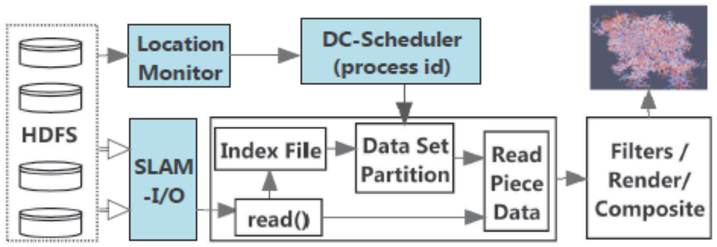
PI: Jun Wang, Jun.Wang@UCF.edu, NSF CCF-1337244
Jiangling Yin, Xuhong Zhang, Junyao Zhang, Dan Huang, Jun Wang, Wu-chun Feng*
Department of Electrical Engineering & Computer Science, University of Central Florida (co-*Virginia Tech)

Abstract

- Whereas traditional scientific applications are computationally intensive, recent applications require more data-intensive analysis and visualization to extract knowledge from the **explosive** growth of scientific information and simulation data.
- As the computational power and size of compute clusters continue to increase, the I/O read rates and associated network for these data-intensive applications have been unable to keep pace, suffering from long I/O latency in moving **"big data"** from the network/parallel file system and, in turn, resulting in a serious performance bottleneck.
- In this work, we propose "**Scalable Locality-Aware Middleware" (SLAM),** which allows scientific analysis applications to benefit from data-locality exploitation with the use of HDFS, while also maintaining the flexibility and efficiency of the MPI programming model.

Software Architecture

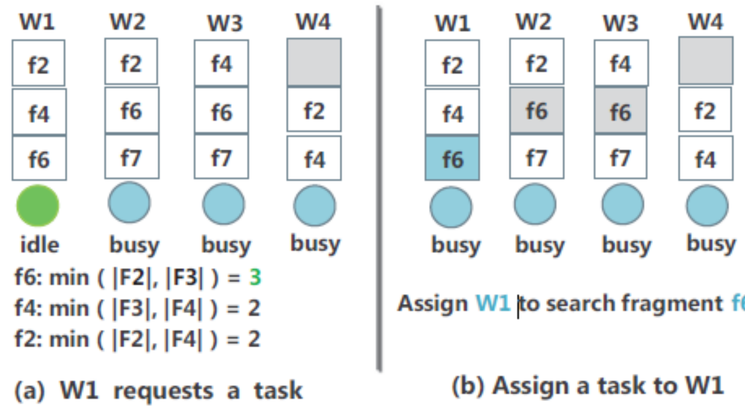
- Our SLAM consists of three components:
  (1) a process-to-data mapping scheduler (**DC-scheduler**), which transforms a compute-centric mapping into a data-centric one so that a computational process always accesses data from a local or nearby computation node,
  (2) a data **location-aware** monitor to support the DC-scheduler, and
  (3) a virtual **translation layer** enabling computational processes to execute on underlying distributed file systems.



Proposed SLAM for ParaView. The DC-scheduler assigns data processing tasks to MPI processes such that each MPI process could read the needed data locally.

A Data Centric Load-balanced Scheduler

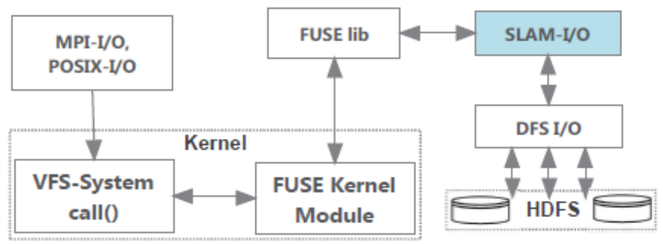
- DC-scheduler always tries to launch a **local** task of the requesting process, that is, a task with its corresponding data available on the node of the requesting process..
- The DC-scheduler algorithm takes the unprocessed data fragments and node distribution as input. The output of the algorithm is the assignment of a task for a process.



- A simple example where the DC-scheduler receives the task request of the process (W1). The scheduler finds the unassigned local tasks of W1 (f2, f4 and f6 in this example). The task f6 will be assigned to W1 since the minimum unassigned task value is 3 on W2 and W3, which also has f6 as a local task. After assigning f6 to W1, the number of unassigned local tasks of W1–4 is 2.

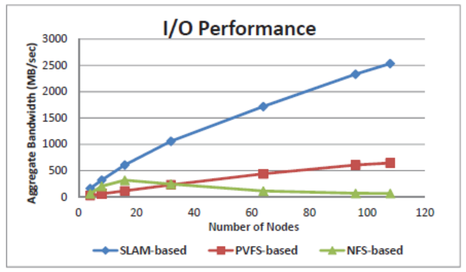
SLAM-I/O: A Translation Layer

- The I/O call in our prototype. A FUSE kernel module redirects file system calls from parallel I/O to SLAM-I/O. SLAM-I/O wraps HDFS clients and **translates** the I/O call to DFS I/O.

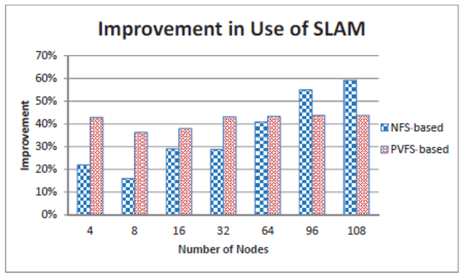


Performance Evaluation

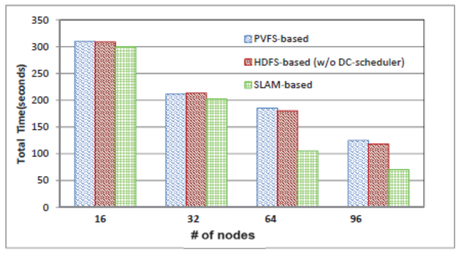
- Read bandwidth comparison of NFS, PVFS and SLAM based BLAST schemes varying the number of nodes on Marmot (NSF PF



- Performance gain of BLAST execution time when searching the *nt* database using SLAM, compared to NFS and PVFS-based.(NSF PRObE cluster)



- Execution time of PVFS, HDFS and SLAM based ParaView (NSF PRObE cluster)



Conclusion

- By testing our SLAM prototype system with real-world data, we saw significant performance increases of over traditional MPI architectures.
- We found performance and scalability increases in **mpiBLAST,** and **ParaView** by using our DC-scheduler algorithm.
- The easy integration of SLAM into MPI programming model allows MPI-based data-intensive applications to efficiently run on commodity clusters.

This work is conducted at a PRObE staging cluster—128-node Marmot cluster, which is supported in part by the National Science Foundation under awards CNS-1042537 and CNS-1042543 (PRObE)