ATDM Data Warehouse

WBS 1.3.4.05 (Scalable I/O Components)
WBS 1.3.4.06 (Data Warehouse)

Sandia National Laboratories
Craig Ulmer (PI), Ron Oldfield (PM)
Todd Kordenbrock, Scott Levy, Jay Lofstead, Shyamali Mukherjee, Greg Sjaardema, Gary Templet, Patrick Widener

Overview

Problem: ATDM's Asynchronous Many-Task (AMT) applications need an efficient way to exchange data with both *storage resources* and *external applications*.

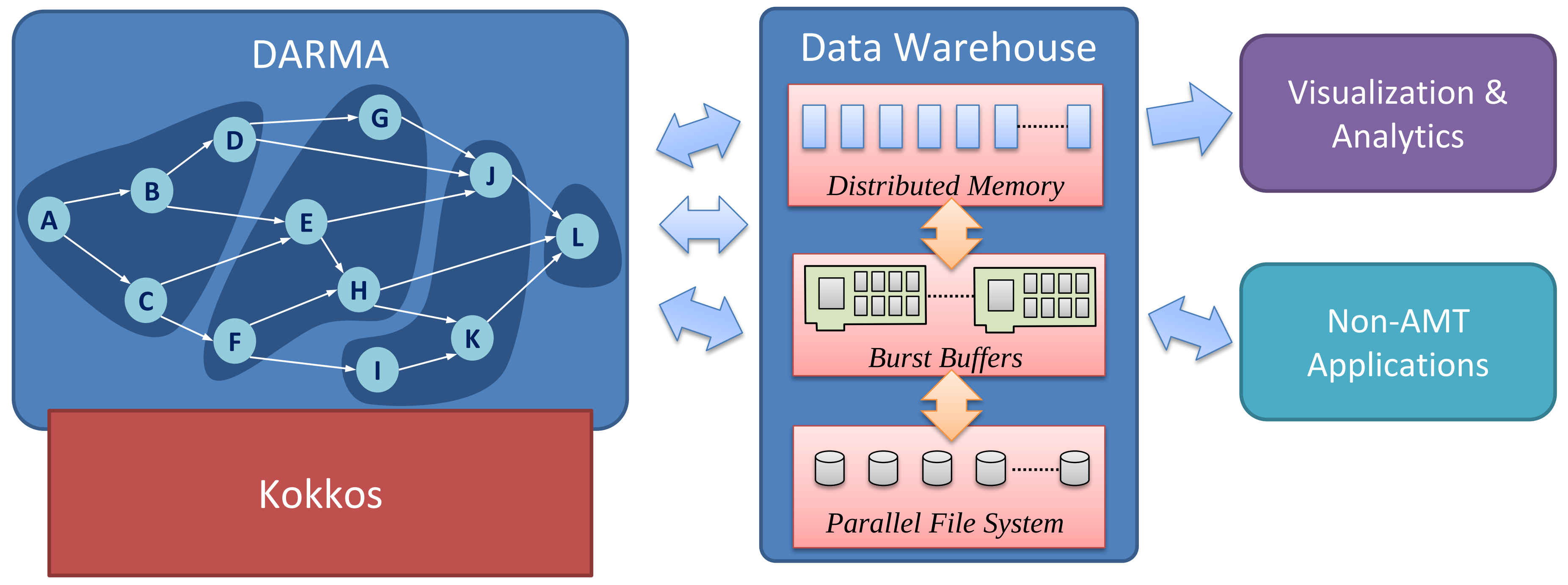
Approach: ATDM's Data Warehouse project is developing a collection of flexible *data management* services that can facilitate a variety of data flows on ATS platforms. These data management services use a collection of compute nodes to cache an application's data objects in a distributed, in-memory key/blob store. Application developers connect to the Data Warehouse through application-specific *Data Interface Modules* (DIMs). DIMs provide users with familiar APIs (e.g., IOSS for mesh data) and are responsible for converting application-specific data structures into key/blob items the Data Warehouse can manage. *I/O Modules* (IOMs) implement platform-specific mechanisms for exchanging key/blob items with storage resources such as burst buffers or the parallel file system. All communication in the Data Warehouse takes place through an asynchronous communication engine built on top of RDMA primitives. This approach ensures data management services will not interfere with an application's normal MPI operations, and provides a communication opportunity for job-to-job coupling.

Use Cases: Saving/Reloading datasets, in-memory handoff of application data to analysis tools, workflows, internal communication mechanism for an AMT runtime.

Non-ATDM Use: While intended for AMT use, the Data Warehouse services are also applicable for traditional HPC.

Two Projects:

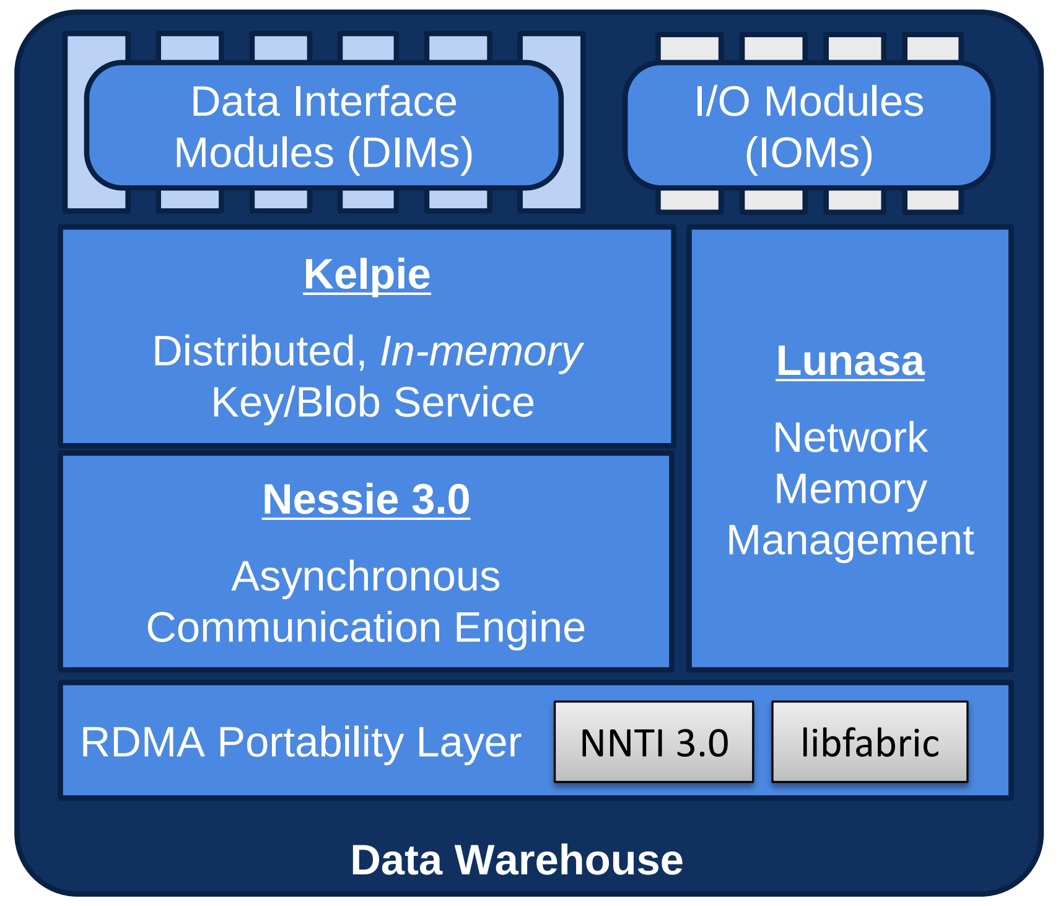
- **WBS 1.3.4.05**: Develop low-level components for use in Data Warehouse and traditional HPC.
- **WBS 1.3.4.06**: Integrate components into Data Warehouse and customize for AMT use.



FY17 Data Warehouse Workflow Demonstrations:

Workflow	Description	Schedule
App-to-Analysis	Demonstrate routing data from DARMA AMT application to an Analysis application using the Data Warehouse as intermediate memory.	Q3
App-to-Storage-to-Analysis	Demonstrate storing data results from DARMA AMT application to burst buffer for retrieval by an Analysis application.	Q4

Software Components for Data Management Services


Data Interface Modules (DIMs)
I/O Modules (IOMs)
Kelpie
Distributed, In-memory Key/Blob Service
Lunasa
Network Memory Management
Nessie 3.0
Asynchronous Communication Engine
RDMA Portability Layer
NNTI 3.0
libfabric
Data Warehouse

Data Interface Modules (DIMs)

The Data Warehouse uses DIMs to implement different user dataset APIs on top of the Data Warehouse. There are multiple DIMs currently in development:

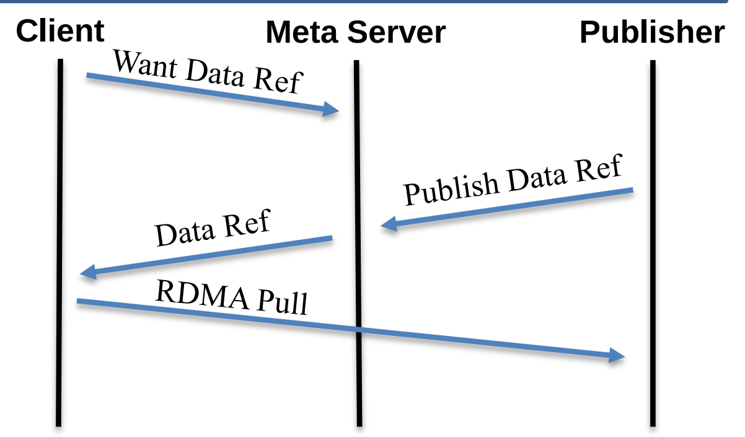
DIM	Description	Schedule
IOSS	IOSS is the standard interface to mesh datasets at Sandia. The IOSS DIM plugs into the IOSS library as an alternate backend for reading and writing data, and requires minimal changes to end applications.	Q1
PIC	The particle-in-cell codes being ported to DARMA track a large number of particles as they move through a meshed space. The particle DIM will export bundles of particles into the Data Warehouse for downstream inspection by Visualization tools.	Q2
SPARC	SPARC requires a way to save and reload state data. This implementation will be based on a prior prototype that snapshotted all data. This DIM will focus on interacting with SPARC to obtain the minimum dataset required.	Q2
Tempus	This DIM will provide a way to store and retrieve time integration data.	Q3

Kelpie: Distributed, In-Memory Key/Blob Service

Kelpie provides a way for data management services to coordinate how a dataset's individual objects are distributed across a collection of nodes. Users typically construct one or more **distributed hash tables** on top of the nodes to spread the dataset and use asynchronous publish/get operations to manipulate individual data components. Kelpie's bookkeeping allows communicators to stage operations to perform when a missing data object becomes available. This triggering can serve as a mechanism for implementing the dataflows found in AMT applications.

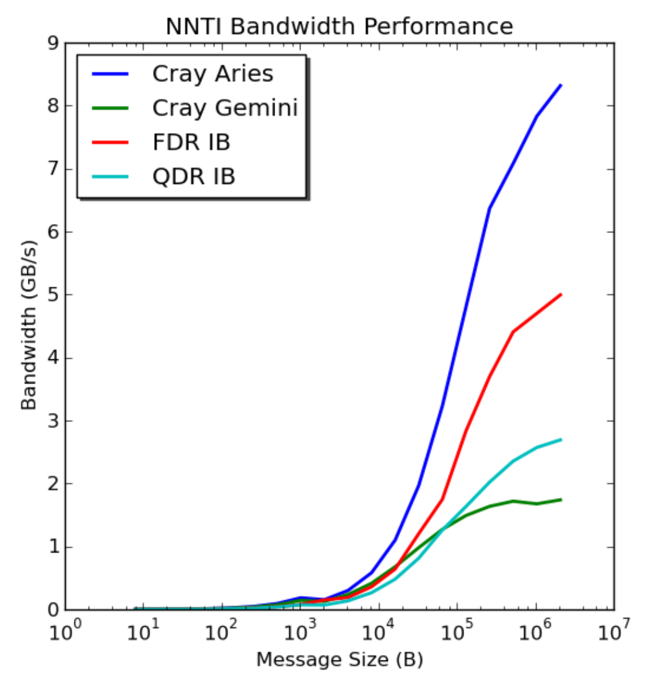
Nessie 3.0: Asynchronous Communication Engine

Data Management services such as Kelpie often need to execute a complex sequence of network operations in order to orchestrate high-level data transfers. Nessie 3.0 is a communication library that enables service developers to describe their operations as event-driven state machines. State machines allow progress to take place as events happen and without user intervention.



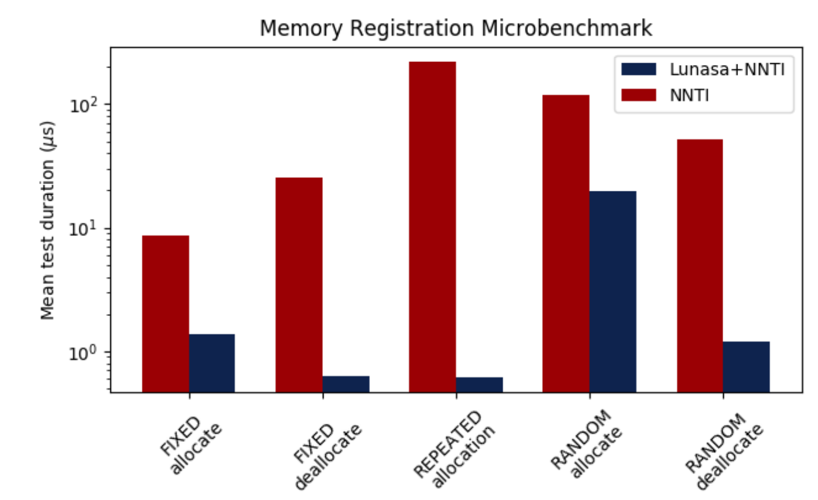
RDMA Portability

Data management services typically require a network layer that is independent of MPI in order to (1) avoid conflicts with the core application's communications and (2) allow an application to connect with external data management resources. The Data Warehouse currently supports two RDMA libraries: libfabric and the Nessie Network Transport Interface (NNTI). NNTI 3.0 has support for Aries, Gemini, and InfiniBand, and features event-based processing capabilities that can be exploited by Nessie. Initial experiments on Mutrino confirm that the overhead of this functionality does not impede general communication performance.



Lunasa: Network Memory Management

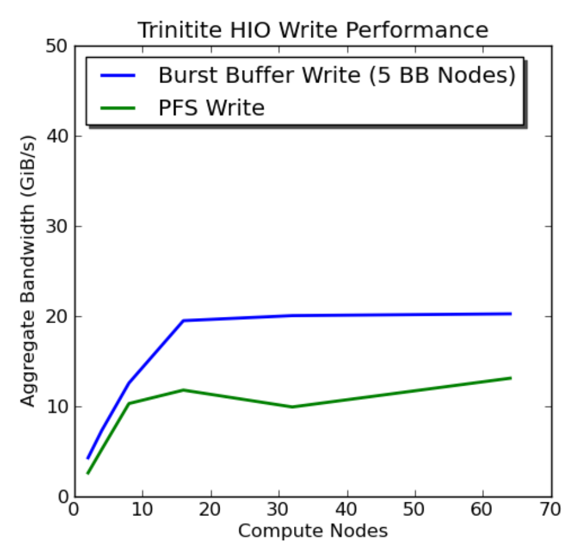
Data management services typically manage memory in an explicit manner for both performance reasons (e.g., NIC memory registration overheads) and practicality (e.g., allocation tracking). The Lunasa component provides a flexible registered memory management unit that is used throughout the Data Warehouse. Lunasa allocates large blocks of memory and then suballocates the memory to users via tcmalloc. Users may request their allocations be registered with the NIC in either an eager or a lazy manner. Experiments conducted on Mutrino confirm that Lunasa's memory management improves service performance in situations where applications frequently transmit data objects.

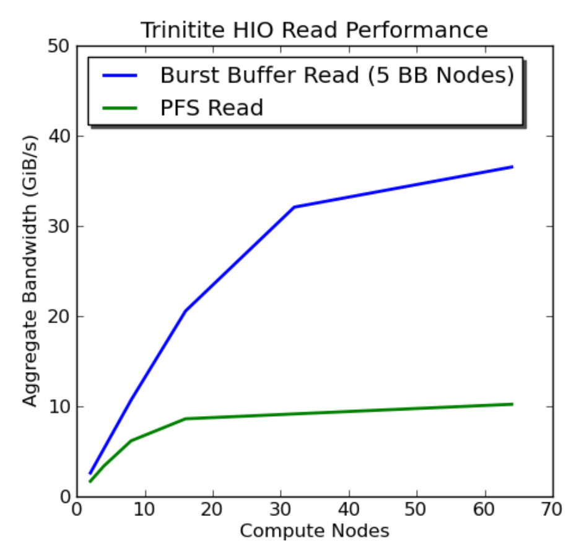


I/O Modules (IOMs)

The Data Warehouse is responsible for migrating data between in-memory resources and storage devices such as burst buffers and the parallel file system. I/O Modules (IOMs) implement platform-specific storage operations. We are currently developing an IOM for Trinity that uses LANL's Hierarchical I/O (HIO) library to exchange data with both the burst buffers and the PFS. This IOM maps Data Warehouse key/blob items into HIO objects that are persistent.

Initial small-scale experiments with HIO on Trinitite confirm that HIO is sufficient for the multi-node Data Warehouse environment, and that the number of Data Warehouse nodes should be scaled to match burst buffer resources.


Trinitite HIO Write Performance
Burst Buffer Write (5 BB Nodes)
PFS Write
Aggregate Bandwidth (GiB/s)
Compute Nodes


Trinitite HIO Read Performance
Burst Buffer Read (5 BB Nodes)
PFS Read
Aggregate Bandwidth (GiB/s)
Compute Nodes