MaDaTS: Managing Data on Tiered Storage for Scientific Workflows

Devarshi Ghoshal
Lawrence Berkeley National Lab
1 Cyclotron Road
Berkeley, California 94720
dghoshal@lbl.gov

Lavanya Ramakrishnan
Lawrence Berkeley National Lab
1 Cyclotron Road
Berkeley, California 94720
lramakrishnan@lbl.gov

ABSTRACT

Scientific workflows are increasingly used in High Performance Computing (HPC) environments to manage complex simulation and analyses, often consuming and generating large amounts of data. However, workflow tools have limited support for managing the input, output and intermediate data. The data elements of a workflow are often managed by the user through scripts or other ad-hoc mechanisms. Technology advances for future HPC systems is redefining the memory and storage subsystem by introducing additional tiers to improve the I/O performance of data-intensive applications. These architectural changes introduce additional complexities to managing data for scientific workflows. Thus, we need to manage the scientific workflow data across the tiered storage system on HPC machines. In this paper, we present the design and implementation of MaDaTS (Managing Data on Tiered Storage for Scientific Workflows), a software architecture that manages data for scientific workflows. We introduce Virtual Data Space (VDS), an abstraction of the data in a workflow that hides the complexities of the underlying storage system while allowing users to control data management strategies. We evaluate the data management strategies with real scientific and synthetic workflows, and demonstrate the capabilities of MaDaTS. Our experiments demonstrate the flexibility, performance and scalability gains of MaDaTS as compared to the traditional approach of managing data in scientific workflows.

KEYWORDS

Data management; scientific workflows; multi-tiered storage; burst buffer

ACM Reference format:
Devarshi Ghoshal and Lavanya Ramakrishnan. 2017. MaDaTS: Managing Data on Tiered Storage for Scientific Workflows. In *Proceedings of HPDC '17, June 26–30, 2017, Washington, DC, USA, ,* 12 pages.
DOI: http://dx.doi.org/10.1145/3078597.3078611

1 INTRODUCTION

Large amounts of data from simulations and data analyses is now processed on High Performance Computing (HPC) machines as complex workflows. Workflows are critical for the management of scientific applications on current and future systems.

Storage and workflow systems have evolved independent of each other, creating a disconnect between the two. Today, scientific workflows largely rely on applications to manage their file-based communication which presents performance challenges when moving to future HPC systems. Simultaneously, the memory storage hierarchy on HPC systems is getting deeper, driven by new technologies and the need to minimize I/O costs. The memory-storage hierarchy in future systems will have different performance and cost points. There is a need for workflow tools to explicitly and efficiently manage data on HPC systems.

The increasing amount of data in scientific workflows results in greater I/O demands from the memory/storage subsystem on HPC systems. Additional storage layers composed of flash devices (e.g., 'Burst Buffer') have recently shown to improve I/O performance of many HPC applications [18, 28]. Applications need to explicitly manage the staging of data to storage layers prior to execution [20, 33]. The global knowledge of data dependencies enables workflow tools to optimize data movement. Workflow tools also need to resolve and manage data dependencies, minimize overheads of data movement, and manage storage space efficiently [7].

The goal of this paper is *to explore the software ecosystem that is needed to manage data for data-intensive complex workflows on the multi-level storage hierarchy on HPC systems.* We describe the design and implementation of MaDaTS (Managing Data on Tiered Storage for Scientific Workflows) that provides an integrated data management and workflow execution framework on HPC resources. MaDaTS coordinates, tracks and manages the lifecycle of data in a science workflow on an HPC system with multi-tiered storage.

In the context of MaDaTS, we propose and develop an abstraction called virtual data space (VDS) that captures the intermediate representation of the data in the workflow. VDS is managed to capture the data elements of the entire workflow based on user provided "hints" and preferences. The complexities of the underlying storage and filesystem are hidden from the user and users can operate on the data space through an Application Program Interface (API). A workflow is often represented as a directed acyclic graph (DAG) and VDS provides a data-centric view of the workflow akin to a *data DAG.* The nodes in the graph represent data elements with their properties and the edges represent the processes that derive the relationship between the data elements.

We demonstrate the operation of MaDaTS with current HPC systems and an existing workflow tool. MaDaTS can be easily configured to work with future systems and tools as well. We evaluate MaDaTS and study the various properties of workflows and policies that influence the data management within workflows using a mix of science and synthetic workflows. Our evaluation is performed on Cori, a Cray XC40 supercomputer at NERSC (National Energy Research Scientific Computing Center). Cori is an exemplar machine that captures the storage hierarchy of future systems through its tier of Burst Buffers. Our experiments show that MaDaTS scales upto 1024 cores on Cori and is only limited by specific workflow characteristics.

Specifically, our key contributions are:

- We present the design of MaDaTS in the context of batch queue systems and an existing workflow tool.
- We describe the concept, design and API of the virtual data space (VDS) to manage data objects in a workflow.
- In the context of MaDaTS, we evaluate different data management strategies and show the capabilities of MaDaTS, comparing the results to the traditional approach of managing data in workflows.

The rest of the paper is organized as follows. We present the background and related work in Section 2. We present the design and implementation of MaDaTS in Section 3. We present our results in Section 4. Finally, we present the conclusions in Section 5.

2 BACKGROUND

In this section, we provide an overview of scientific workflows on HPC systems, the memory/storage hierarchy and related work.

2.1 Scientific Workflows

In HPC environments, workflows are widely used to execute simulations and data analyses for scientific discovery [27]. Today, workflows are written as ad-hoc scripts and/or using workflow tools and they chain together a series of tasks with inputs and outputs in a pipeline. Existing workflow tools [8, 29] do not provide mechanisms to implicitly manage data on HPC resources. Current user methods are less than optimal in performance and effectiveness. For example, users often 'stage' the input data to a storage layer prior to executing the workflow. However, some of this data can be asynchronously staged as the workflow executes. Also, selective staging of data might be sufficient for improving performance while balancing costs. Additionally, scientists face challenges in automating these due to the possible wastage of compute hours, file system user quota limits, etc. Thus, there is a need to automate or semi-automate the data management steps in workflows, which is the goal of our work.

The execution of a workflow in an HPC environment depends on the mapping of workflow tasks to job scripts that are submitted to a batch scheduler. Single-job execution mode (also called Pilot job [19]), consolidates workflow tasks into a single batch job. Multi-job execution mode creates distinct batch jobs for each task in a workflow. Single- and multi-job modes have trade-offs in wait time and utilization of resources and are preferred for different workflow types. For MaDaTS, we use the Tigres workflow library for managing workflows through job scripts on HPC resources and our evaluation tests both approaches.

2.2 Tigres Workflow Library

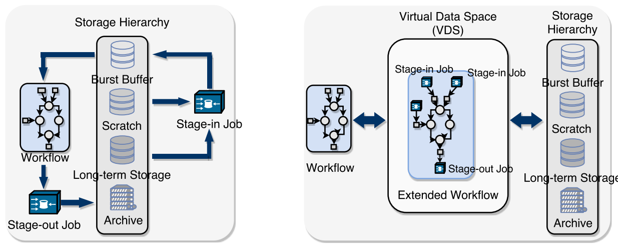
Tigres [12] is a programming library that allows users to compose large-scale scientific workflows and execute them on HPC environments. Tigres provides *"templates"* that enable scientists to easily compose, run and manage computational tasks as workflows. These templates define common computation patterns used in analyzing data and running scientific simulations. Currently, Tigres supports four templates – sequence, parallel, split and merge. Tigres can run on a variety of different platforms including desktops, clusters and supercomputers.

Tigres allows workflows to be composed from existing executable scripts and binaries. Tigres workflows are Python programs that are submitted as jobs to the batch scheduler directly. It manages workflow execution from within the job scripts, where resources are managed via different batch schedulers like Slurm, Torque, SGE etc. In this paper, we use Tigres as the workflow library to implement and evaluate MaDaTS.

2.3 Memory and Storage Hierarchy

Fast storage class memory devices are being added to the storage subsystem on current and future HPC systems. These devices (solid state drives or SSDs) currently use flash memory that is one form of non-volatile random-access memory (NVRAM). SSDs have added one more layer to the storage hierarchy in HPC systems in the form of Burst Buffer [18, 28]. It resides between compute nodes and the high-capacity parallel file system (PFS).

Current research on NVRAM technologies are also looking into alternate solutions that may add additional layers to the memory and storage hierarchy in future HPC systems. In this paper, we address the issue of data management across the tiered storage hierarchy in HPC systems. We use NERSC's most recent supercomputer Cori that provides a 'Burst Buffer' based on Cray DataWarp [13] for our evaluation. Cori's storage hierarchy is representative of the storage hierarchy that is expected in future systems. *MaDaTS has been designed to be generic and can be configured to be used with future HPC systems that have deeper storage tiers.*



(a) Traditional workflow execution. **(b) MaDaTS workflow execution.**

Figure 1: Traditional versus MaDaTS workflow execution. Today, in the traditional model of running workflows users manage the data stage-in and stage-out process at the beginning and end, independent of the execution. VDS provides a way to map the data of a workflow to the multi-tiered storage system. The data is managed with data movement tasks that are part of the workflow enabling greater efficiency throughout the workflow.

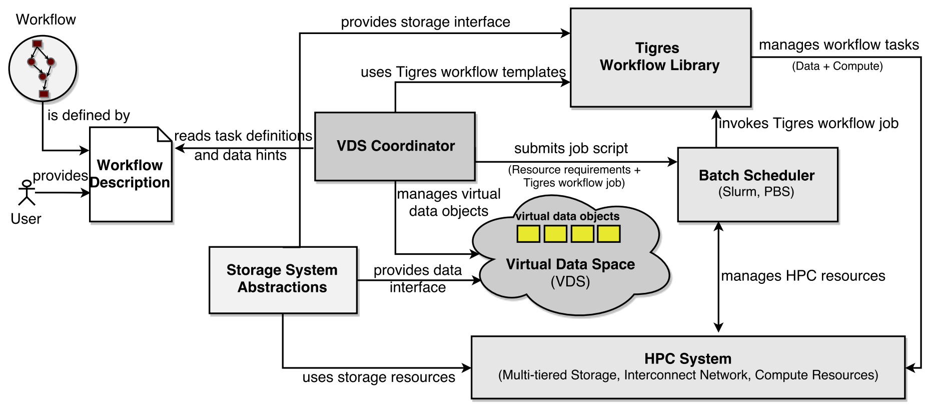

Workflow
is defined by
provides
User
Workflow Description
reads task definitions and data hints
VDS Coordinator
provides storage interface
uses Tigres workflow templates
Tigres Workflow Library
manages workflow tasks (Data + Compute)
invokes Tigres workflow job
submits job script (Resource requirements + Tigres workflow job)
Batch Scheduler (Slurm, PBS)
manages virtual data objects
virtual data objects
Virtual Data Space (VDS)
Storage System Abstractions
provides data interface
manages HPC resources
uses storage resources
HPC System (Multi-tiered Storage, Interconnect Network, Compute Resources)

Figure 2: MaDaTS architecture: VDS Coordinator creates a data management plan (i.e., an extended workflow DAG). The DAG includes data movement tasks abstracted by virtual data objects. Virtual data objects are mapped to file system data objects through storage system abstractions. The tasks in the extended workflow are managed by Tigres and the resources are managed by a batch scheduler like Slurm.

2.4 Related Work

Data management for large data-intensive science workflows on HPC resources is an active area of research, and is getting increased attention due to the evolving memory and storage hierarchy. We detail work related to the multi-tiered storage, and the various data abstractions and approaches for managing scientific data.

Multi-tiered Storage. Multi-tiered storage has shown to improve I/O performance across various domains [16, 22] However, the improvements are still limited by the data movement costs between the storage tiers, and the underlying limitations of the storage and the network subsystems. MaDaTS uses storage and data properties, along with the structure of the workflow to generate an efficient data management plan that minimizes the cost of moving the data between the storage tiers. Deadline-based [32] and I/O-aware [14] data migration techniques have been proposed to move data efficiently between SSDs and HDDs. In contrast, the focus of MaDaTS is on the efficient execution of scientific workflows on multi-tiered memory storage hierarchy in current and future HPC systems. MaDaTS's current focus is on three data management strategies - storage-aware, workflow-aware and user-controlled/passive. MaDaTS's extensible design makes it possible to explore other strategies relevant to HPC workflows in the future.

Data Abstractions. Franklin et al. [11] propose a concept of dataspaces that provides a data management abstraction for application developers hiding the underlying differences of the data sources through a common set of services.Docan et al. [9] define DataSpaces as a data exchange framework that is used to access whole or part of the data through a shared address space. Previous works have also proposed memory abstractions [21] including more recently the Resilient Distributed Datasets [30]. The concept of abstracting data objects into 'virtual data' has been earlier used in Grid environments and data catalogs [10, 17]. VDS abstractions in MaDaTS is similar to these abstractions but is more focused on capturing the data-centric view of the workflow, that is necessary for managing the data efficiently on the complex memory-storage hierarchy.

Data Management. Previous works have focused on the challenges [7] for data management of scientific workflows in HPC environments. Various aspects of distributed data management have been considered in the context of grid environments including tools for optimized wide area data transfer [2], replica management [5], and metadata catalogs [23]. Workflow systems managing data over a wide-area network [15, 26] make data management decisions based on network and storage parameters, and focus on strategies for efficiently transferring the data. These works address issues orthogonal to MaDaTS since they target distributed systems. MaDaTS's focus is on HPC systems and our focus is on determining appropriate storage layer for each data item in the workflow while allowing users control and hiding the complexities of the storage layers.

In-situ execution of workflows provide an alternative way to manage data and execute workflows by reducing the data movement costs [25, 31] or using resources on dedicated I/O transfer nodes [34]. While MaDaTS can address problems related to in-situ, our focus is more widely on workflows that have explicit I/O operations between stages of the workflow. Active Burst Buffer [4] proposes using a file system interface to minimize data movement between the Burst Buffer and parallel file system based on workflow patterns. MaDaTS operates at the middleware layer and will be able to work with technologies like the Active Burst Buffer.

3 MADATS DESIGN AND IMPLEMENTATION

Figure 1a shows the traditional way of executing workflows in an HPC environment. Users explicitly stage the input data to a storage layer (e.g., scratch), and then execute the tasks using the staged data. Finally, the output data is staged out to some persistent storage for long-term storage and analysis. The data movement between the storage layers is often constrained to the beginning and at the end of workflow execution.

We have designed MaDaTS to provide data management that works with existing workflow frameworks. MaDaTS generates a data management plan for moving the data between storage layers. It works in concert with workflow frameworks to execute the workflow and managing its data at run time in accordance to the data management plan. MaDaTS provides different strategies and policies that allow varying levels of control over the data management and movement during workflow execution without modifying the existing workflows and/or the batch schedulers. In the context of MaDaTS, we propose the concept of a Virtual Data Space (VDS) (Figure 1b). VDS is an abstract workspace that enables implicit data management for scientific workflows in a multi-tiered storage. It hides the complexities of the storage hierarchy and provides a simple and flexible interface to manage data for a workflow.

Figure 2 shows the high-level architecture of MaDaTS, with the Virtual Data Space (VDS) as its central component. In MaDaTS, users submit a workflow description containing the information about the workflow tasks and appropriate "hints" for the workflow data. For example, a user might specify that a particular input data set's size can be used for staging decisions. MaDaTS creates a VDS for every workflow. The VDS Coordinator manages VDS through virtual data objects that represents the data elements of a workflow and maps them to the underlying file systems. MaDaTS uses the hints and data management policies to create data tasks that orchestrates the movement of data across the memory-storage hierarchy. Next, an extended workflow consisting of compute and data tasks is created. The extended workflow is submitted through a job script to submit the workflow through a batch scheduler like Slurm. Once the job is submitted and resources are allocated, Tigres manages the execution of the workflow tasks on HPC resources. The data tasks in the workflow interact with the storage system to move data between the storage layers.

3.1 Workflow Description and Data Hints

MaDaTS uses a workflow description to derive data dependencies within a workflow, and generate policies to manage data during workflow execution. MaDaTS's workflow description extends current descriptions that specify the tasks and their dependencies by including their resource requirements (including number of CPUs, walltime etc.), and their input/output data sets. The workflow description may contain "hints" on the data that help MaDaTS to select the storage layer where the data can be moved during workflow execution. These hints can be provided by the users as additional annotations to a workflow description, describing various properties about the data. The hints provide information about the storage and quality of service requirements for workflows. For example, a data hint `persist = true` suggests that the data needs to be persisted for long-term storage and MaDaTS needs to stage the data out to a persistent storage. Similarly, a user might specify a data size that helps MaDaTS decide the appropriate storage layer for the workflow data. In addition to the data hints, MaDaTS uses different data management strategies - storage aware, workflow aware and passive (more details in Section 3.5) to decide if and when to move the data. In the absence of data hints, if the data management strategy requires the data to be moved, MaDaTS always moves the data to the storage layer with the highest throughput. Thus, users can skip specifying the data hints and MaDaTS still tries to aggressively optimize the I/O performance of the workflow.

Figure 3 provides a partial example of a workflow description, annotated with data hints. In this workflow description, a task `task1` is defined with certain inputs and outputs. Each input and output of the workflow task maps to a unique virtual data object which has an identifier. Each task definition contains information

```
task_definitions = ({
    task = 'task1';
    inputs = ['vdo_in1', 'vdo_in2'];
    outputs = ['vdo_out1'];
    cpus = 1024;
    walltime = '00:30:00'; ...}, ...);
data_properties = ({
    name = 'vdo_out1';
    size = '1G';
    persist = true; ...}, ...);
data_management_strategy = 'STRATEGY';
```

Figure 3: A workflow description contains task definitions, hints to the data elements and a data management strategy for MaDaTS.

about resource requirements for executing the task. In this example, the size and persistence hints are specified for `vdo_out1`. Our initial implementation accepts hints about the data-size, persistence and replication. These hints help MaDaTS to select the appropriate storage layer for accessing the data in the workflow. The existing set of hints are identified by our use cases, where workflows have both small (in KBs) and large (in GBs) data sets and where only part of the output data needs to be preserved for long-term storage. However, this is extensible to include other properties/hints since the data hints are specified as name-value pairs.

3.2 VDS Coordinator

The VDS Coordinator in MaDaTS manages virtual data objects and hides the complexities of managing data between the storage layers. The VDS Coordinator is responsible for managing the data on the memory-storage hierarchy and prepare a data execution plan. Its goal is to minimize data movement by keeping the data 'alive' in the selected storage layer throughout the lifetime of their associated tasks. Essentially, it copies and keeps the data in a selected storage layer until all the associated tasks of the virtual data object finish execution. Based on the data management strategy (described in Section 3.5), VDS Coordinator creates data movement tasks and associates them with the corresponding virtual data objects. Next, the VDS Coordinator generates an extended workflow consisting of compute and data tasks. It creates a directed acyclic graph (DAG) of the extended workflow and generates an execution order of the workflow tasks. Finally, it creates and submits a job script, consisting of a Tigres workflow, to the batch scheduler.

Tigres workflow library is used to execute the directed acyclic graph (DAG) generated by the VDS Coordinator. Since MaDaTS submits the workflow through a batch scheduler, it currently supports two job execution modes – i) single-job, which uses a single resource allocation for the entire workflow, and ii) multi-job, which uses separate allocation of resources for each task of the workflow. In the single-job execution mode, the VDS Coordinator creates a single job submission script for the entire workflow and uses Tigres to manage the execution of tasks on the allocated HPC resources. In the multi-job execution mode, VDS Coordinator creates a separate job submission script for each task of the workflow which is managed by the batch scheduler.

In order to maximize task throughput and resource usage, MaDaTS uses a *binning* technique to combine independent parallel

Algorithm 1 Task binning and deferring for just-in-time staging.

```
Input: Workflow DAG, $W$ and task bins $bin$
1: Initialize $bin_j = 0 \; \forall \; j \in jobs_W$
2: for $j$ in $jobs_W$ do
3:     for $d$ in $dependents_j$ do
4:         if $bin_d < bin_j + 1$ then
5:             $bin_d = bin_j + 1$
6:         end if
7:     end for
8: end for
9: for $j$ in $jobs_W$ do
10:    $bin_{min} = n_{bins}$
11:    for $d$ in $dependents_j$ do
12:        $bin_{min} = Min(bin_d, bin_{min})$
13:    end for
14:    $bin_j = Max(bin_j, bin_{min} - 1)$
15: end for
```

tasks of the workflow into a single bin. First, MaDaTS determines the execution order of the tasks within a workflow by topologically sorting the DAG to combine tasks into bins. This step ensures that the order of the workflow tasks is based on their dependencies. MaDaTS then puts the tasks into separate bins by doing a breadth-first search (BFS) on the DAG. Hence, each bin ends up with workflow tasks that are independent of each other. The bins are sorted and executed in sequence from lower to higher, corresponding to starting and ending tasks of a workflow respectively. MaDaTS further optimizes the execution order of the tasks by *deferring* some tasks to a higher bin such that each task executes just prior to its dependent task. This minimizes the wait time between the subsequent tasks in the workflow and enables *just-in-time* data staging.

Algorithm 1 is used for binning and deferring workflow tasks to higher bins. Lines 1-8 finds the earliest (minimum) bin of each dependent task. Lines 9-15 then assigns the task to the bin that is just before the lowest bin of its dependent tasks. This ensures that the data staging occurs just prior to executing the tasks using it. Line 14 defers the stage-out tasks to higher bins. This is to ensure that in case of synchronous stage-out, the compute tasks do not unnecessarily wait for their completion.

The workflow tasks and bins are executed by Tigres. Each bin, consists of independent parallel tasks, is executed using the *parallel* template. The bins themselves, on the other hand, are executed in sequence. In the multi-job mode, MaDaTS manages the execution order of bins by specifying job dependencies through the batch scheduler. Each bin of the workflow is executed as a single batch job and all the bins are submitted at the same time through Slurm using the dependency specifier `--dependency=afterok:<job-id>`. The dependency specifier explicitly notifies Slurm not to execute a job until all of its dependencies have finished execution.

3.3 Virtual Data Object

A virtual data object is an abstract entity in VDS that is uniquely identified by an object identifier. Each object identifier is a combination of a storage identifier and path to the data object relative to the storage mount point. Figure 4 shows the mapping of a data object in the storage system to a virtual data object in VDS. The example maps the data object `/global/scratch/sample/foo` to a virtual data object `scratch:sample/foo`. Since `/global/scratch/sample/foo` represents the absolute path of the data object `foo` on the `scratch` storage (mounted on `/global/scratch`), the corresponding object identifier for the virtual data object in VDS is `scratch:sample/foo`. The storage identifier uniquely identifies the storage layer in the multi-tiered storage hierarchy. Storage system abstractions (more details in Section 3.4) resolve the virtual data object identifier to the corresponding physical location of the data object on the file system.

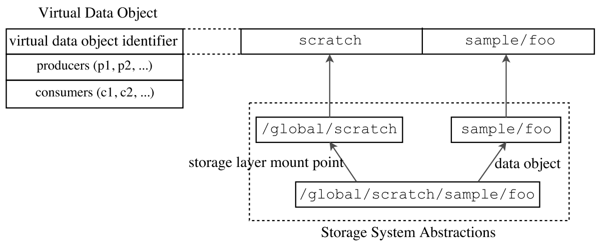


Figure 4: Example virtual data object in VDS that corresponds to a data object in the storage system.

In addition to the virtual data object identifier, each virtual data object consists of *producer* and *consumer* tasks. Hence, a collection of virtual data objects within a VDS provides a **data-centric** representation of workflows, where each data object is a node and the tasks are its edges. Producers are the tasks that generate the data object and consumer tasks use the data object. Using the data object from a different storage layer in a workflow now simply means replacing the storage layer for the virtual data object. Hence, this provides a simple abstraction for managing the data on multiple storage tiers without explicitly modifying the tasks.

VDS supports the data-centric model of workflows. In the era of big data and multi-tiered hierarchical storage, we need to move towards the data-centric model of workflows [1]. Data management is going to be the principal challenge with increasing volume and rate of data. Performance and efficiency of workflows now depend on efficiently utilizing the storage resources to manage data during execution. The data-centric model of workflows allows us to make data management decisions by examining each data object over the lifecycle, rather than just looking at individual tasks. In the data-centric model of a workflow, data objects are treated as first-class parameters to the workflow tasks. This results in a data-driven workflow execution, where a task can be executed as soon as the data it uses becomes available.

3.4 Abstractions

MaDaTS provides a two-level abstraction – at the VDS and system levels. The abstractions provide flexibility for managing workflow data in tiered storage. VDS provides different operations on virtual data objects to abstract the data management across multiple storage layers. These abstractions hide the complexities of data management and yet provide users control to manage data for scientific workflows in a tiered storage. Storage system abstractions hide the low-level complexities of the multi-tiered hierarchical storage system. These abstractions also provide the interface to connect different workflow engines and tools.

Abstraction	Description
$create()$	creates a virtual data space (VDS)
$destroy()$	deletes the VDS for a workflow
$add(V)$	adds virtual data object V to the VDS
$copy(V, s)$	copies virtual data object V to storage layer s
$delete(V)$	removes virtual data object V from the VDS

Table 1: Data abstractions in MaDaTS.

Data Abstractions. MaDaTS provide data abstractions to manage a virtual data space (VDS) and to manage virtual data objects within the VDS. Table 1 lists these abstractions to manage a VDS through two simple interfaces – i) `create` and ii) `destroy`. Once a VDS is created, it provides three basic operations on virtual data objects to abstract data management across the storage layers using – i) `add`, ii) `copy`, and iii) `delete`. The `add` and `delete` are used to create and remove the virtual objects from VDS. The `copy` operation on a virtual data object creates a new virtual data object on a specific storage layer and copies all the producers and consumers of the original virtual data object to the new one. These operations provide the foundation for the VDS Coordinator to implement different data management strategies through VDS. Although these operations are sufficient for managing the workflow data across the storage layers, new operations can be added to VDS when MaDaTS interfaces with other storage systems and workflow engines. Once the workflow execution ends, the associated VDS and its virtual data objects are deleted.

Algorithm 2 Creating data tasks to manage virtual data objects in VDS.

```
Input: A virtual data object $v$, destination storage layer $s$, and
    associated virtual data space $VDS$
 1: $v' = VDS.copy(v, s)$
 2: if $v$ has no $producers$ and has $consumers$ then
 3:     $task = T(v \rightarrow v')$
 4:     $producers_{v'} = [task]$
 5:     $consumers_{v} = [task]$
 6: else if $v$ has no $consumers$ and has $producers$ then
 7:     $task = T(v' \rightarrow v)$
 8:     $producers_{v} = [task]$
 9:     $consumers_{v'}.append(task)$
10:     $consumers_{v} = []$
11: else
12:     $VDS.delete(v)$
13: end if
```

VDS Coordinator manages the virtual data objects in VDS through the data operations. The VDS Coordinator creates and adds a virtual data object to a VDS, using the `add` operation. Based on the data management strategy, it creates a data task (Algorithm 2) that copies the data between storage systems. The data task abstracts the data movement between storage systems from the user. It creates a data task corresponding to each data movement that is introduced by MaDaTS. It copies the source virtual data object to the destination storage system using the `copy` operation (Line 1). If data needs to be moved, it creates a data task for both stage-in (Lines 2-5) and stage-out (Lines 6-10) operations. If no data needs to be moved, it simply deletes the old virtual data object using the `delete` operation and replaces it with the new data object (Line 11-12). This corresponds to intermediate workflow data that need not persist beyond the workflow execution.

Storage System Abstractions. Storage system abstractions provide interfaces to access data objects across multiple layers in the storage hierarchy. They provide translation of virtual data objects into file system data objects through various file system interfaces over the multi-tiered storage. They provide different mount points corresponding to the different storage layers in the hierarchy. A data object in the storage layer is accessed via the file system path relative to the mount point of the storage layer.

Our initial prototype implementation provides storage system abstractions through a storage configuration, which maintains the hierarchy of storage systems. Storage tiers can be added with hints on its properties or removed from the configuration file and MaDaTS moves the workflow data between the storage tiers specified through the configuration file. MaDaTS implements a hashtable mapping of the storage identifiers to the corresponding mount points in the storage system. For example, if `/global/scratch` is the mount point of the `scratch` file system, then the hashtable contains the mapping `scratch` → `/global/scratch`. This mapping is used to create virtual data objects in VDS corresponding to data objects in the file system. There can be multiple file system interfaces to a storage system. In the future, we envision that a tighter protocol between storage systems, file systems and VDS can be developed which would further improve the performance and hide the complexity.

Workflow Plug-ins. MaDaTS is implemented in C and Python, and prepares the data management plan through a description of a workflow provided by the user. The description is a set of key-value pairs describing the tasks and the data for the workflow. This provides a simple interface to represent different workflows into a format that MaDaTS understands. Standard workflow engines have workflow description formats and XML schemas that can be easily translated into the workflow description that MaDaTS uses by parsing the description files. The description also contains resource requirements pertaining to a workflow and the HPC system. This allows for an easy adaptation of the workflow and data management strategies on different HPC systems, without modifying the underlying architecture of MaDaTS.

The workflow DAG that is generated by the VDS Coordinator provides a workflow execution and data management abstraction to Tigres. Tigres templates are used to execute the workflow DAG that includes the data management tasks through a Tigres workflow script. MaDaTS currently provides different interfaces to optimize the execution order of the workflow DAG by binning and deferring task executions. Other workflow engines can be plugged into MaDaTS by including translators. First, we would need a translator that takes the user provided workflow DAG and translates it to MaDaTS's workflow description format. Second, a workflow engine could translate MaDaTS's data plan and convert it to its workflow description format. In previous work, Tigres workflows have been converted to other formats [3].

Type	Criteria	Strategy	Effectiveness
Storage-aware (VDS-STA)	storage and data properties	data for which the I/O performance improves	data caching
Workflow-aware (VDS-WFA)	structure of the workflow	only if data movement can be overlapped	data staging
User-driven (VDS-Passive)	user-defined rules	explicit data movement	application-specific
Traditional	-	explicitly move data before and after execution	-

Table 2: Data management strategies: the first three highlight the features of MaDaTS strategies. The traditional approach highlights managing data manually through separate scripts or tools that move data before and after running workflows.

3.5 Data Management Strategies

The VDS Coordinator in MaDaTS manages the data in a workflow by creating data tasks and each data task is created based on selected data management strategy. MaDaTS currently provides three data management strategies that provide varying levels of control – i) storage-aware (VDS-STA), which uses data hints and storage layer properties to define data management policies, ii) workflow-aware (VDS-WFA), which considers the structure of the workflow to optimize the data management policies, and iii) user-driven (VDS-Passive), which lets users describe the policies for managing the data. Table 2 summarizes the three data management strategies that MaDaTS supports. MaDaTS has been designed such that other strategies can be implemented as needed.

Storage-aware (VDS-STA). MaDaTS uses a storage function to select the appropriate storage layer based on certain properties, for each data object in the workflow. The storage function selects the 'lowest' storage layer for a data object that satisfies the properties. These properties consist of storage capacity read-write throughput, data-size, persistency etc. VDS-STA generates data movement tasks only if the storage layer satisfying the data requirements is different from the original data object storage. This strategy is useful when the workflow will benefit from 'caching' the data on a faster storage.

Workflow-aware (VDS-WFA). MaDaTS considers the structure of a workflow to define the 'when' and 'where' of the data movement. With VDS-WFA, VDS coordinator overlaps data movement with workflow tasks. If there are preceding tasks to the consumers of a virtual data object, then a data task is created that copies the virtual data object to a faster storage layer. A data task can also be created if there are succeeding tasks to the producers of a virtual data object, which copies the data out from a fast storage to a specific persistent storage. If no overlap is possible, then no data is moved between the storage layers. VDS-WFA is useful when data staging is part of the workflow execution. This data management strategy optimizes the I/O performance by selectively moving the data, thereby reducing the overheads of data movement.

User-driven (VDS-Passive).: MaDaTS directly maps the virtual data object to the workflow data. Data is moved, only when specified by the user. Users can use virtual data objects to create data tasks and explicitly move data between storage layers. All the data objects are generated and used from the storage layer specified by the user, for a workflow task. When users need to explicitly place the data based on application choices, VDS-Passive is the best choice.

4 EVALUATION

In this section, we evaluate the performance and resource usage of scientific workflows using MaDaTS. We compare our results against the traditional approach of executing workflows and managing data, where data is explicitly moved before and after the workflow execution using ad hoc scripts or data movement tools.

4.1 Evaluation Setup

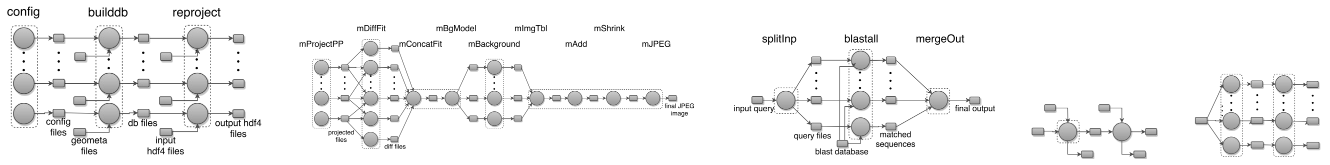
We evaluated our system on Phase I of NERSC's Cori supercomputer. It is a Cray XC40 supercomputer with 1630 compute nodes. Each node has 32 cores and has 128 GB DDR4 2133 MHz memory and four 16 GB DIMMs per socket. Each core has its own L1 and L2 caches, with 64 KB and 256 KB, respectively.

For compute jobs, Cori provides multiple storage options with three different file systems that provides different throughput, storage space, and period for data retention. The GPFS based 'project' file system has a peak performance of 40 GBps and typically used for longer term storage of frequently accessed files. The filesystem used for I/O during job execution is 'scratch' which is a Lustre file system with peak performance of approximately 700 GBps. Cori's 'Burst Buffer' architecture is built upon discrete *Burst Buffer nodes* (BB nodes), each containing two Intel P3608 SSDs that deliver 6.4 TB of usable capacity and 5.7 GBps of bandwidth. Currently, Cori has a total of 144 BB nodes with over 800 GBps of peak performance. The BB nodes are managed by the Cray Datawarp API. Through a job script users can either specify datawarp directives or use Linux commands to move the data between the Burst Buffer and other parallel file systems. The job scripts are submitted and scheduled to run through the Slurm batch scheduler. MaDaTS uses the DataWarp API to manage data on BB nodes on Cori.

4.2 Workloads

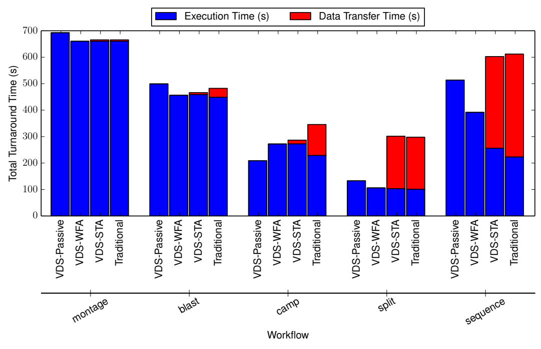
We evaluate VDS Coordinator using synthetic and real I/O intensive science workflows. Figure 5 shows the different data and execution patterns of the workflows. The workflows use data sets of varying sizes, accessed at different stages in the workflows. Each workflow has a different data placement and access requirement, where data can be either shared or distributed across the workflow tasks. The workflows also cover fairly different execution patterns that can be composed using the basic templates in Tigres.

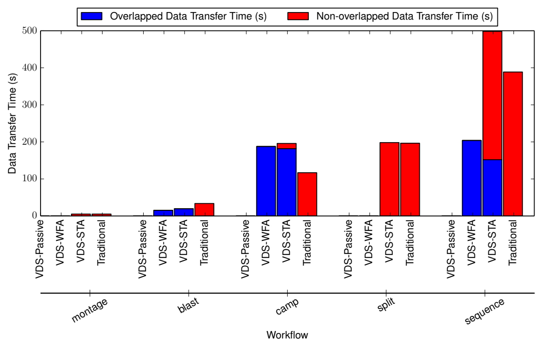
Science Workflows. We use three science workflows with varying I/O and execution characteristics and are a combination of one or more of the basic workflow patterns (i.e., sequence, parallel, split, merge). *CAMP* is an I/O and metadata intensive workflow. We use an optimized version of the CAMP workflow, where each computation task builds its own sqlite database (builddb) and a subsequent task reprojects the input data (set of 1-2 MB HDF4 files) from a coordinate system to a tiling system (reproject). The workflow processes one month's input data of size approx. 54 GB. CAMP has three stages of parallel tasks (Figure 5a). Each task in



(a) CAMP (b) Montage (c) BLAST (d) Sequence (e) Split

Figure 5: Scientific and synthetic workflows: Each workflow exhibits different data access and execution pattern. CAMP transforms NASA MODIS satellite data from a coordinate system to a tiling system. Montage assembles an image for survey M17 on band j from 2mass Atlas images. BLAST performs protein sequence matching using a shared large protein database. Sequence and split workflows are synthetic I/O intensive workflows where each task reads and writes files using the 'dd' utility.





(a) Turnaround Time (b) Data Transfer Time

Figure 6: Comparison of workflow performance under different data management strategies: VDS-WFA performs better than the other data management strategies for all the workflows except CAMP. For CAMP, VDS-Passive performs better than VDS-WFA because CAMP is dominated by data transfers, and not the computation tasks.

the subsequent stages use the data from the previous stage and some additional input files, which are used only during that stage.

Montage is a data-intensive workflow that assembles a jpeg image from sky survey data (fits files). Montage is combination of sequential and parallel tasks (Figure 5b) and requires all the input data in a single directory prior to executing the workflow. Each fits file is 1 MB in size, and a total of 23 GB of data is processed for degree 5.0 of the Montage workflow.

BLAST is a compute-intensive workflow that matches protein sequences against a large database (> 6 GB). BLAST splits an input file (7500 protein sequences for our tests) into multiple small files (a few KBs) and then uses parallel tasks to compare the data in those files to that of a large shared database (Figure 5c). It finally merges all the outputs from the parallel tasks into a single file.

Synthetic Workflows. The synthetic workflows are based on the basic workflow patterns as described in [24]. We use *sequence* and *split* synthetic workflows. In the sequence workflow the output of one task is used as an input to another task. In addition to previous task's inputs, each task also uses an independent external input. A split workflow has a single task that sets up parallel tasks. Each task of the synthetic workflow uses dd to read and write files. The workflows also correspond to different file I/O patterns in scientific workflows. The sequence workflow corresponds to sequential read and write pattern. The split workflow uses a shared and parallel read write pattern.

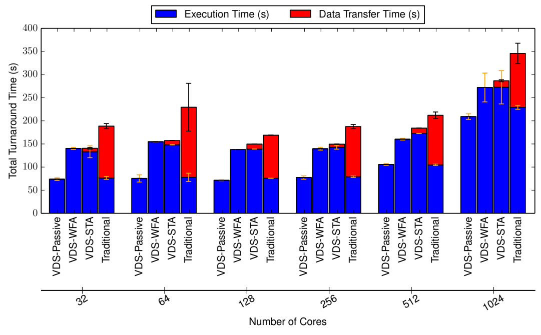
Metric (unit)	Description
Execution time (s)	Time to complete all the tasks in a workflow
Data transfer time (s)	Time to move data between the storage layers
Turnaround time (s)	Workflow execution time + Data transfer time
Total space used (GB)	Amount of space used on a storage layer
Total CPU time (hrs)	Task execution time * Number of cores allocated to task

Table 3: Metrics for evaluation.

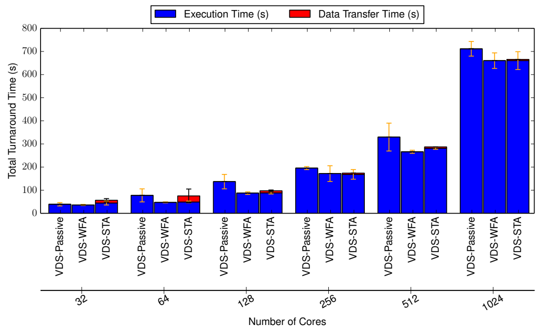
Table 3 lists the metrics used for our evaluation. The metrics include execution time, data transfer time, turnaround time, space used, resources used, measure the performance and efficiency of the workflows with respect to the run time and storage space requirements.

4.3 VDS Coordinator Data Management Strategies

In this section, we evaluate the three MaDaTS data management strategies (Table 2). We compare the results against the traditional approach of managing data and executing workflows that enables data movement through job scripts outside of the actual workflow



(a) CAMP



(b) Montage

Figure 7: Weak scaling results for CAMP and Montage: All the MaDaTS strategies scale uniformly with increasing data size and number of compute resources. One or more of the MaDaTS strategies scale consistently better than the traditional approach. For Montage, traditional approach shows similar results to VDS-STA. The overall performance starts decreasing at 1024 for MaDaTS strategies due to the limitations of the application, but they always perform better or at par to the traditional approach.

execution (i.e., data is moved only before and after the workflow execution). We show the average over five runs in the plots except for BLAST. BLAST encountered system issues since the BB usage modes are still evolving, We were only able to run BLAST experiments once each. We used one job-script per workflow when using single-job execution mode unless otherwise mentioned.

Workflow Turnaround Time. Figure 6 shows the turnaround time and overlapping data transfer time under different data management strategies (Table 2) for different synthetic (sequence and split) and science workflows (Montage, BLAST, CAMP). Workflow turnaround time is calculated from start of the first workflow task (including the data movement) to end of the workflow execution. For each workflow, we compare the results of storage-aware (VDS-STA), workflow-aware (VDS-WFA) and user-driven (VDS-Passive) data management strategies to the traditional approach. While using VDS-Passive, we use the Lustre parallel file system to store workflow data and avoid moving data to the Burst Buffer. Figure 6a shows that VDS-WFA always performs better than the traditional approach because it overlaps the movement of data and the execution of tasks, whenever possible (Figure 6b). In addition to the overlapping data movement, VDS-WFA is opportunistic and also moves the data selectively based on the workflow structure. Hence, it always performs better than the traditional approach. VDS-STA moves all the data larger than a threshold (> 1 MB in this case) to the Burst Buffer. Since all the workflows process large amounts of data, VDS-STA moves almost all the data to the Burst Buffer, but overlapping any computation, whenever possible. Its worst case matches the traditional approach when all the data needs to be moved but when data movement does not overlap with the compute stages.

Figure 6 also shows that VDS-Passive results in lower turnaround times than the traditional approach for all workflows except Montage and BLAST. These two workflows have very small data transfer overheads and have I/O patterns that can benefit from the Burst Buffer. VDS-Passive performs better overall by not moving the data, when compared to the traditional approach. VDS-Passive also performs better than VDS-WFA for CAMP because CAMP is dominated by data transfers, and not the the execution time of the

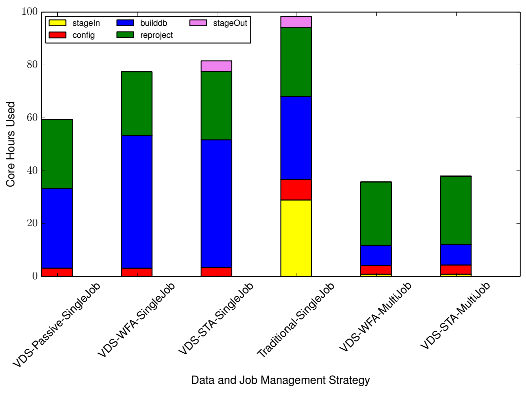
Workflow	Space Saved (GB)	% Space Saved
sequence	400.0	57.06
split	200.0	66.44
camp	1.0	18.18
blast	0.49	4.55

Table 4: Amount of space saved with VDS-WFA as compared to using the traditional approach of executing workflows.

tasks. Overall, we see that the different data management strategies in MaDaTS provide significant benefits for all the workflows and allow users to select the appropriate strategy based on the workflow requirements. Typically, workflows that benefit from caching the data to fast storage like the Burst Buffer, benefit from using VDS-STA. Workflows that need to stage the data during workflow execution benefit from using VDS-WFA (e.g., sequence workflow). Finally, workflows that are dominated by data transfers (e.g., CAMP) benefit by using custom data rules for moving data.

Storage Requirements. The Burst Buffer provides additional performance. However, it also comes at an additional usage cost that is calculated differently by different sites. Thus, it is important that it is used optimally. Table 4 shows the amount of space saved for each workflow when using VDS-WFA relative to the traditional approach. The storage requirements for each strategy is calculated based on the total space used on the Burst Buffer. The amount of space saved is the difference between the storage used by the MaDaTS strategies and the traditional approach. VDS-WFA uses 4% - 66% less space as compared to the traditional approach.

VDS-WFA strategy allows MaDaTS to selectively move the data between the different storage layers, thereby reducing the total space used on Burst Buffer while executing the workflows. The storage space savings for BLAST are smaller because large part of the total data (i.e., the BLAST database > 6 GB) gets copied to the Burst Buffer in both approaches. Only the inputs and the outputs (which are only a few MBs) are used directly from the final storage (scratch Lustre file system). The output data in CAMP, which is large part of the total data, gets generated directly at the target storage layer, saving 18% of the Burst Buffer space. For split and


(a) CAMP

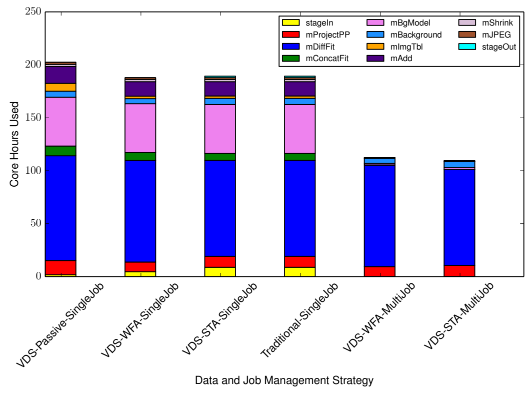
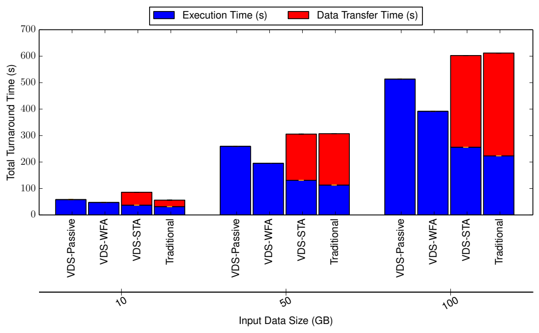

(b) Montage

Figure 8: Total CPU time (in core hours) for CAMP and Montage under different MaDaTS strategies: Multi-job execution mode with VDS-WFA uses the least amount of CPU time, implying maximum usage and minimum wastage of compute resources during workflow execution.


(a) Sequence

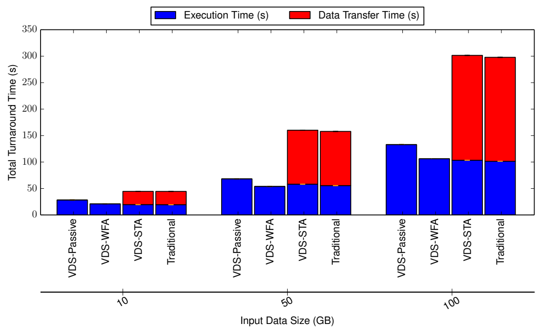

(b) Split

Figure 9: Effect of increasing data-size on sequence and split workflows: Both sequence and split workflows benefit from overlapping and selective data movement of VDS-WFA. It minimizes the overheads of data movement, whereas the overheads continue to increase with increasing data sizes for the traditional approach.

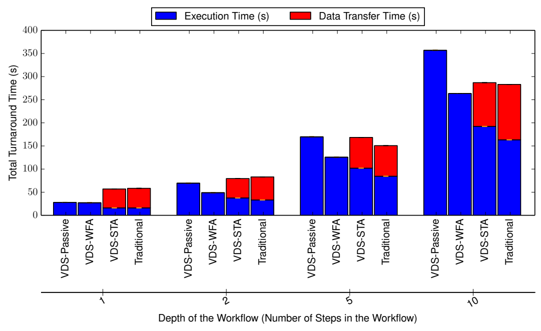
sequence workflows, only the intermediate data sets are generated and used from the Burst Buffer, whereas the large input and output data sets are stored on the target storage system. Thus, this leads to a saved storage space of to up to 66%. This shows that the storage savings from MaDaTS's strategies will be higher for workflows with larger input and output data.

Scalability. Figure 7 compares the performance of different data management strategies with weak scaling (from 32 to 1024 cores) for CAMP and Montage. In weak scaling, we increase the number of tasks with increasing cores and increasing data sizes. All three MaDaTS data management strategies scale better than the traditional approach for the two workflows. However, at 1024 cores the performance deteriorates due to application limitations (i.e., load on the metadata server [6]).

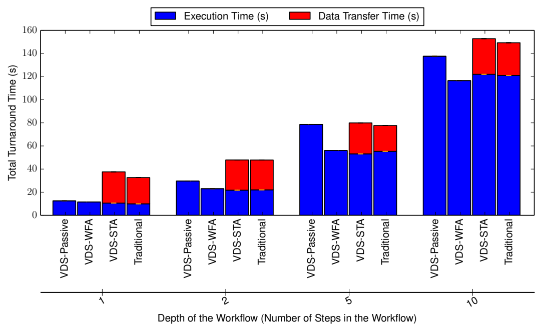
Montage requires its data to be present in a single working directory, and much of the performance gain is due to the intermediate files being written on to the fast storage. Figure 7b shows that with increasing data size and number of tasks, VDS-WFA and VDS-STA tend to perform better as compared to VDS-Passive since cost of data movement is low and the I/O performance improves significantly when the data is on Burst Buffer. For Montage, the traditional approach shows similar results to VDS-STA while scaling up, because the data movement stages are similar for both.

Total CPU Time. In this section, we evaluate the two job execution modes (single-job and multi-job) in MaDaTS along with the data management strategies for CAMP and Montage. We compare the core hours used under both execution modes for different data management strategies (Figure 8). X-axis of the graph shows the data management strategies with the job execution mode. Y-axis shows core hours for each strategy. The maximum number of nodes requested for the parallel tasks in both workflows was set to 32 (= 1024 cores). Hence, the single-job execution mode requested 1024 cores for the entire execution of the workflow job-script, whereas the multi-job execution requested as many cores as are required by each stage of the workflow, up to a maximum of 1024 cores.

Figure 8a shows that the traditional approach for CAMP consumes large core hours for data transfers. It uses a single allocation of resources for the entire workflow execution. MaDaTS' multi-job strategy uses less core hours as each task of the workflow consumes only as many resources as required by the task. Figure 8b shows



(a) Sequence



(b) Split

Figure 10: Effect of the depth of workflow on sequence and split workflows: Both the workflows benefit from VDS-WFA due to overlapping and selective data movement. However, with the increasing depth of the workflow, the initial cost of moving the data for VDS-STA and the traditional approach is overshadowed by the performance gains of caching the intermediate data.

significant differences in core hours used between the single-job and multi-job strategies. The large number of sequence stages in Montage have resources allocated but not used in a single-job mode.

4.4 Factors Affecting Data Management Strategies

In this section, we study the impact of the data management strategies in MaDaTS. Specifically, we evaluate three workflows characteristics – a) size of data, b) depth of workflow, and c) width of workflow. We use synthetic sequence and split workflows.

Effect of Workflow Data Size. Figure 9 shows the effect of increasing data size on workflow turnaround time. X-axis shows the aggregate input size for each workflow. Each workflow has two dependent stages and there are ten parallel tasks in each stage of the split workflow. As can be seen from the figure, for both sequence and split workflows VDS-WFA performs better with increasing data sizes by minimizing the data movements between the storage layers. The difference in turnaround times between VDS-WFA and the other data management strategies increase as we increase the input data sizes because larger data sets are moved between the storage layers when using the other strategies.

Effect of Workflow Depth. Figure 10 shows the effect of different data management strategies on turnaround time with increasing depth of the workflows. The depth of the workflow refers to the number of stages in the workflow. X-axis shows the number of stages in the workflow. The input data size is 10 GB per task and the split workflow has ten tasks per stage. As shown in the figure, VDS-WFA performs better for both the workflows with increasing data sizes. VDS-STA and the traditional approach have significantly lower execution times because of the improved I/O performance through Burst Buffer. However, due to large data transfer times both the strategies suffer a slowdown in turnaround time. VDS-Passive I/O performance degrades when not using Burst Buffer due to increasing data sizes that affects the overall performance of the workflow.

Effect of Workflow Width. Figure 11 shows the effect of increasing width of the split workflow on turnaround time. The X-axis of the graph shows the number of parallel tasks (from 2 to 10) for each stage in the workflow. The total input data size is 100 GB

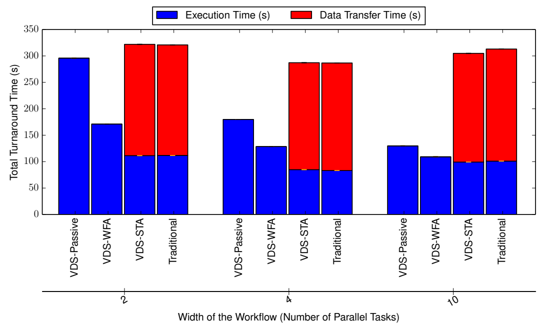


Figure 11: Effect of the workflow width on the split workflow: VDS-WFA and VDS-Passive continue to perform better by avoiding unnecessary data movements. With increasing number of parallel tasks, the I/o performance of VDS-WFA and VDS-Passive continue to improve, whereas for the traditional approach the performance remains the same due to staging overheads.

that is split between the tasks equally and the workflow has two stages (i.e., depth=2). The results show that with increasing number of parallel tasks, MaDaTS strategies perform significantly better or at par to the traditional approach. Again, due to selective and overlapping data movement, VDS-WFA performs much better with increasing number of parallel I/O tasks in the workflow.

4.5 Summary

MaDaTS improves the performance of scientific workflows by providing different data management strategies for different requirements. Workflows with significant I/O and that can benefit from caching data (e.g., Montage) benefit by using VDS-STA. Sequence and split workflows that need to manage data as part of the workflow execution benefit from VDS-WFA. Workflow with dominant data transfer stages (e.g., CAMP) benefits from VDS-Passive. VDS-WFA used up to 66% less storage space as compared to the traditional approach, when using multiple storage layers. Our weak scaling experiments up to 1024 cores showed that MaDaTS as well as the underlying software systems scale to run multi-job workflows on HPC resources and is only limited by application characteristics.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we present the design, implementation and evaluation of MaDaTS, that enables management of workflow data on multi-tiered storage hierarchy on HPC systems. MaDaTS uses virtual data space to map user-level requirements to system-level abstractions. We demonstrate through the use of science and synthetic workflows that our infrastructure provides the users flexibility while transparently managing data. Our evaluation shows that MaDaTS enables workflow tools to achieve better turnaround time and resource usage through various data management strategies.

Our current implementation of MaDaTS is built to work with current software stack in HPC systems. However, our design is extensible and enables other workflow engines and batch schedulers to be used with MaDaTS. Additionally, the deeper memory-storage hierarchy is spearheading advancements in file systems and programming models that MaDaTS will be able to leverage in the future.

ACKNOWLEDGMENTS

This work and the resources at NERSC are supported by the U.S. Department of Energy, Office of Science and Office of Advanced Scientific Computing Research (ASCR) under Contract No. DE-AC02-05CH11231, program manager Lucy Nowell. The authors would also like to thank Christopher Daley from NERSC, Harinarayan Krishnan from LBL, and Sudharshan S. Vazhkudai from ORNL for their valuable suggestions and feedback.

REFERENCES

[1] Asif Akram, J Kewley, and Rob Allan. 2006. A Data centric approach for Workflows. In *2006 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06)*.

[2] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. 2005. The Globus Striped GridFTP Framework and Server. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC '05)*. IEEE Computer Society, Washington, DC, USA, 54–.

[3] Javier Rojas Balderrama, Matthieu Simonin, and Cédric Tedeschi. 2015. *GinFlow: A Decentralised Adaptive Workflow Execution Manager.* Ph.D. Dissertation. Inria.

[4] Chao Chen, Michael Lang, Latchesar Ionkov, and Yong Chen. 2016. Active Burst-Buffer: In-Transit Processing Integrated into Hierarchical Storage. In *Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on.*

[5] Ann L. Chervenak, Robert Schuler, Matei Ripeanu, Muhammad Ali Amer, Shishir Bharathi, Ian Foster, Adriana Iamnitchi, and Carl Kesselman. 2009. The Globus Replica Location Service: Design and Experience. *IEEE Trans. Parallel Distrib. Syst.* 20, 9 (Sept. 2009).

[6] Christopher Daley, Devarshi Ghoshal, Glenn Lockwood, Sudip Dosanjh, Lavanya Ramakrishnan, and Nicholas Wright. 2016. Performance Characterization of Scientific Workflows for the Optimal Use of Burst Buffers. In *11th Workshop on Workflows in Support of Large-Scale Science (WORKS'16)*.

[7] E. Deelman and A. Chervenak. 2008. Data Management Challenges of Data-Intensive Scientific Workflows. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on.*

[8] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, and others. 2005. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* 13, 3 (2005), 219–237.

[9] Ciprian Docan, Manish Parashar, and Scott Klasky. 2012. DataSpaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing* 15, 2 (2012).

[10] Ian T. Foster, Jens-S. Vöckler, Michael Wilde, and Yong Zhao. 2002. Chimera: AVirtual Data System for Representing, Querying, and Automating Data Derivation. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM '02)*. IEEE Computer Society.

[11] Michael Franklin, Alon Halevy, and David Maier. 2005. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record* 34, 4 (2005).

[12] Valerie Hendrix, James Fox, Devarshi Ghoshal, and Lavanya Ramakrishnan. 2016. Tigres workflow library: Supporting scientific pipelines on hpc systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on.*

[13] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, and N.J. Wright. 2016. Architecture and Design of Cray DataWarp. In *Cray User Group CUG.*

[14] Stephen Herbein et al. 2016. Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC '16).*

[15] Chen Jin, Scott Klasky, Stephen Hodson, Weikuan Yu, Jay Lofstead, Hasan Abbasi, Karsten Schwan, Matthew Wolf, W Liao, Alok Choudhary, and others. 2008. Adaptive io system (adios). *Cray User's Group* (2008).

[16] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. 2011. HybridStore: A Cost-Efficient, High-Performance Storage System Combining SSDs and HDDs. In *Proceedings of the 2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '11).* Washington, DC, USA.

[17] David T. Liu and Michael J. Franklin. 2004. GridDB: A Data-centric Overlay for Scientific Grids. In *the 30th International Conference on Very Large Data Bases.*

[18] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn. 2012. On the role of burst buffers in leadership-class storage systems. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST).*

[19] A. Luckow, L. Lacinski, and S. Jha. 2010. SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing.*

[20] Henry M. Monti, Ali R. Butt, and Sudharshan S. Vazhkudai. 2013. On Timely Staging of HPC Job Input Data. *IEEE Transactions on Parallel and Distributed Systems* 24, 9 (2013).

[21] Bill Nitzberg and Virginia Lo. 1991. Distributed Shared Memory: A Survey of Issues and Algorithms. *Computer* 24, 8 (Aug. 1991).

[22] Ramya Prabhakar, Sudharshan S Vazhkudai, Youngjae Kim, Ali R Butt, Min Li, and Mahmut Kandemir. 2011. Provisioning a multi-tiered data staging area for extreme-scale machines. In *2011 31st International Conference on Distributed Computing Systems (ICDCS).*

[23] Arcot Rajasekar, Reagan Moore, Chien-yi Hou, Christopher A Lee, Richard Marciano, Antoine de Torcy, Michael Wan, Wayne Schroeder, Sheau-Yen Chen, Lucas Gilbert, and others. 2010. iRODS Primer: integrated rule-oriented data system. *Synthesis Lectures on Information Concepts, Retrieval, and Services* 2, 1 (2010), 1–143.

[24] Lavanya Ramakrishnan and Beth Plale. 2010. A Multi-dimensional Classification Model for Scientific Workflow Characteristics. In *the 1st International Workshop on Workflow Approaches to New Data-centric Science (Wands '10).* ACM.

[25] Melissa Romanus, Fan Zhang, Tong Jin, Qian Sun, Hoang Bui, Manish Parashar, Jong Choi, Saloman Janhunen, Robert Hager, Scott Klasky, Choong-Seock Chang, and Ivan Rodero. 2016. Persistent Data Staging Services for Data Intensive In-situ Scientific Workflows. In *Proceedings of the ACM International Workshop on Data-Intensive Distributed Computing (DIDC '16).* ACM, New York, NY, USA, 8.

[26] Masahiro Tanaka and Osamu Tatebe. 2010. Pwrake: A Parallel and Distributed Flexible Workflow Management Tool for Wide-area Data Intensive Computing. In *the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10).* ACM, New York, NY, USA.

[27] Ian J Taylor, Ewa Deelman, Dennis B Gannon, and Matthew Shields. 2014. *Workflows for e-Science: scientific workflows for grids.* Springer Publishing Company.

[28] Teng Wang, Sarp Oral, Michael Pritchard, Kevin Vasko, and Weikuan Yu. 2015. Development of a Burst Buffer System for Data-Intensive Applications. *CoRR* (2015).

[29] Michael Wilde, Mihael Hategan, Justin M Wozniak, Ben Clifford, Daniel S Katz, and Ian Foster. 2011. Swift: A language for distributed parallel scripting. *Parallel Comput.* 37, 9 (2011).

[30] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12).* USENIX Association, Berkeley, CA, USA, 15–28.

[31] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi. 2012. Enabling In-situ Execution of Coupled Scientific Workflow on Multi-core Platform. In *26th International Parallel Distributed Processing Symposium (IPDPS).*

[32] G. Zhang, L. Chiu, C. Dickey, L. Liu, P. Muench, and S. Seshadri. 2010. Automated lookahead data migration in SSD-enabled multi-tiered storage systems. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST).*

[33] Zhe Zhang, Chao Wang, Sudharshan S. Vazhkudai, Xiaosong Ma, Gregory G. Pike, John W. Cobb, and Frank Mueller. 2007. Optimizing Center Performance Through Coordinated Data Staging, Scheduling and Recovery. In *the 2007 ACM/IEEE Conference on Supercomputing (SC '07).* ACM, New York, NY, USA.

[34] Fang Zheng, Hasan Abbasi, Ciprian Docan, Jay Lofstead, Qing Liu, Scott Klasky, Manish Parashar, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. 2010. PreDatA–preparatory data analytics on peta-scale machines. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on.* IEEE.