2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00045

A Comprehensive Study of In-Memory Computing on Large HPC Systems

Dan Huang*
Sun Yat-sen University
Guangzhou, China

Zhenlu Qin, Qing Liu
New Jersey Institute of Technology
Newark NJ, USA

Norbert Podhorszki, Scott Klasky
Oak Ridge National Laboratory
Oak Ridge TN, USA

***Abstract*—With the increasing fidelity and resolution enabled by high-performance computing systems, simulation-based scientific discovery is able to model and understand microscopic physical phenomena at a level that was not possible in the past. A grand challenge that the HPC community is faced with is how to handle the large amounts of analysis data generated from simulations. In-memory computing, among others, is recognized to be a viable path forward and has experienced tremendous success in the past decade. Nevertheless, there has been a lack of a complete study and understanding of in-memory computing as a whole on HPC systems. This paper presents a comprehensive study, which goes well beyond the typical performance metrics. In particular, we assess the in-memory computing with regard to its usability, portability, robustness and internal design trade-offs, which are the key factors that of interest to domain scientists. We use two realistic scientific workflows, LAMMPS and Laplace, to conduct comprehensive studies on state-of-the-art in-memory computing libraries, including DataSpaces, DIMES, Flexpath and Decaf. We conduct cross-platform experiments at scale on two leading supercomputers, Titan at ORNL and Cori at NERSC, and summarize our key findings in this critical area.**

***Index Terms*—High-performance computing, data analytics, workflow, in-memory computing**

I. Introduction

It is well recognized that on high-performance computing (HPC) systems, the compute has become increasingly cheaper as compared to the storage and I/O [1]. This architectural trend has continued to motivate and drive the HPC community to look for alternative solutions that allow data analysis to be done efficiently, rather than post-processing scientific data at persistent storage. In-memory computing, among others, aims to address this challenge by analyzing data while they are still in memory so that the cumbersome post-processing over persistent storage can be avoided. Unlike Big Data domain, in-memory computing in HPC is apt to reduce the abstraction layers of data object and I/O subsystems, for achieving high performance on data management and movement. For example, ORNL ADIOS [2] configures an external XML file for depicting raw data (dimensions, offsets, sizes and types), rather than encapsulates the raw data with metadata and semantics via high-level data abstraction, e.g. RDD in Apache Spark. Broadly, in-memory computing on HPC systems can be in the form of in situ [3], [4], [5], [6], [7], [8], [9], [10], in transit [11], [12], [13], [14], [15], [16], or the combination of the two, depending on how the simulation and data analytics are deployed. Particularly for in situ, the simulation and data analytics can execute on the same compute node. As such, the data analytics can directly retrieve raw data from the simulation memory and there is short data movement associated with it. A key disadvantage of this approach is that simulation and analytics must be tightly coupled via shared memory mechanism instead of saving data into a dedicate data staging area. In contrast, the in transit approach stages data from the simulation memory to a dedicated off-node staging area where the data can be further analyzed. While this approach incurs explicit off-node data movement, a major benefit is that the simulation can run asynchronously with the data analytics, thus posing much lower impact on the simulation.

Despite the multitude of efforts and the demonstrated success in both in situ and in transit, there has been a lack of complete evaluations and understanding of in-memory computing as a whole on emerging HPC architectures. This is particularly needed for general-purpose in-memory computing libraries that provide generic APIs for coupling scientific workflows in various application scenarios and HPC systems. Without expertise on system design and performance tuning, these libraries can significantly impact the workflow performance and scalability. Our evaluation demonstrates that the in-memory libraries may yield lower performance and scalability than persistent file I/O under default configuration. Therefore, our study is conducted to help domain scientists understand the internal mechanism of these libraries, tune workflow performance, and also benefit library developers on how to achieve better performance, usability, robustness and etc. To this end, we evaluates state-of-the-art general-purpose in-memory libraries, including DataSpaces [13], DIMES [13], Flexpath [14], and Decaf [15], with some of these tested through the ADIOS [2] framework, using two realistic scientific workflows, i.e., LAMMPS [17] and Laplace [18]. Specifically, our paper makes the following contributions:

- We believe this work presents the most comprehensive study of in-memory computing on HPC systems, and assesses a broad spectrum of metrics that are to the interest of domain scientists, including the end-to-end performance (Section III-B1) of scientific workflows, software usability (Section IV-A), portability (Section IV-B), and robustness (Section IV-C), thus being much broader than the scope of prior work [16], [3]. These dimensions are important for broad

* The work was performed at NJIT.

2575-8411/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDCS47774.2020.00045

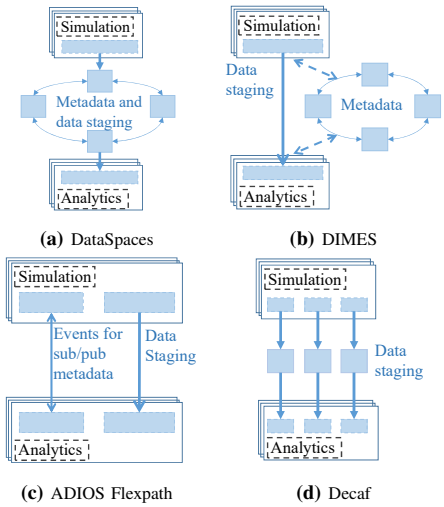


Fig. 1: In-memory computing libraries. Data staging stores the simulation output in an intermediate storage for further processing. For DataSpaces and DIMES, the metadata maintains the descriptive information of staged data, such as data dimension, size, location, type and etc.

adoption of in-memory computing by domain scientists, but have rarely been discussed in the literature.

- We conduct in-depth qualitative and quantitative analyses of the behavior of in-memory computing, and summarize a number of key findings that we hope can shed a light on the weaknesses and possible areas for future research. We verify our findings through experiments, deep code review, debugging as well as discussions with some of the library developers.
- We conduct cross-platform studies at scale on two leading supercomputers, Titan [19] at Oak Ridge National Laboratory (ORNL) and Cori [20] at National Energy Research Scientific Computing Center (NERSC). With adjusting the building configurations and running modes, we show the insights of how to tune the performance of in-memory libraries on the two distinct systems.

The rest of this paper is organized as follows. We present a survey on in-memory libraries and methodologies along with the related work in Section II. A comprehensive performance evaluation and analysis are presented in Section III. In Section IV, we present a quality assessment on the usability, portability and robustness of those libraries. Finally, we conclude this work in Section V.

II. Background and Related Work

A. In-memory Computing

For the traditional post-processing on HPC systems, a simulation dumps the analysis output onto persistent storage. After the run is over, data analytics will retrieve and analyze the data. In contrast, in-memory computing allows the simulation and analytics to be coupled through the memory layer, which delivers a much higher throughput and lower latency than persistent storage. Figure 1 illustrates the high-level design of general-purpose in-memory computing libraries, which can couple applications and analytics in various scenarios, such as feature extraction, anomaly detection, visualization and etc. DataSpaces and DIMES construct a shared virtual space to stage data and metadata respectively. In contrast, Flexpath stages data at the simulation side and uses the subscription/publication mechanism to notify analytics with regard to where and when to retrieve the staged data. Decaf is a dataflow system that depicts a dataflow graph, where an edge denotes the direction of dataflow and a node represents where data resides, e.g. at simulation, analytics and data staging area.

DataSpaces [13] provides a set of high-level declarative APIs, such as put() and get(), to allow analysis data to be placed into a shared virtual space, indexed and subsequently queried by various workflow components. DataSpaces deploys dedicated staging and metadata servers to manage the distributed datasets, and utilizes DART [21] as the underlying communication layer to achieve highly-optimized data movement over interconnect. As of now, DataSpaces has customized data transport over a variety of remote direct memory access (RDMA) implementations, such as Infiniband, Cray Gemini [22] and Aries [23].

DIMES [13] is another in transit method offered from the DataSpaces library and similarly provides put() and get() to access data staging servers. As compared to the baseline, it places the shared virtual space directly into the simulation memory in a distributed fashion, and provides direct memory-to-memory data exchange, as opposed to moving data to the dedicated staging servers first. However, metadata are still maintained by the stand-alone DIMES servers.

Flexpath [14] adopts a publisher/subscriber based model to exchange data between the simulation and data analytics. To support a range of communication protocols, Flexpath uses a network abstraction layer, EVPath [24], which currently supports TCP sockets, Sandia NNTI [22], Infiniband, Cray Gemini, and the BlueGene interconnect. Flexpath adopts Fast Flexible Serialization (FFS) [25] for data serialization, which creates self-describing events to support flexible data types.

Decaf [15] is a dataflow system that enables the parallel communication between the coupled components within an HPC workflow. In particular with Decaf, workflows can perform data transformations on-the-fly, including serialization and complex data redistribution. The communication layer of Decaf is entirely based upon message passing over MPI, thus being portable across different platforms. In contrast to the libraries mentioned above, Decaf adopts a simple Python API for mapping a workflow to a graph, e.g. *add_node()*, *add_edge()* and *processGraph()*. Users can also assign roles to the nodes of dataflow graph, such as producer and consumer.

ADIOS [2] is a framework level library that provides a range of I/O methods, including both file I/O and in-memory computing. A key contribution of ADIOS is that it provides an open framework that allows new I/O methods to be easily plugged and played. On the other hand, ADIOS designs a binary-packed mechanism that allows for the self-describing data format. As of now, ADIOS has demonstrated its high performance and scalability over 1 million cores

on leadership class systems. ADIOS has integrated nearly all general-purpose in-memory libraries, including Flexpath, DataSpaces and DIMES, and hides the complexity of usage. It also provides a set of descriptive APIs, e.g. *adios_write()* and *adios_read()*, and users can determine the underlying in-memory library to be used typically through an XML configuration file.

B. Other Related Work

Researchers have proposed various in situ and in transit methodologies to allow science to be done more efficiently on HPC systems, for example, for feature extraction [26], [27], parallel visualization [12], [8]. Prior research [5], [9], [3] have improved the efficiency of running in situ workflows at the framework level. Malakar et al. [28] propose a mathematical model to choose the optimal frequency of data transfer between workflow components under a given resource constraint. Zipper [16] identifies and resolves the performance bottleneck of in-memory frameworks through deep performance analysis. SENSEI [3] is primarily an in situ library that provides VTK data model [29] for performing analysis and visualizations. Meanwhile, a set of in transit techniques at the system level is designed and implemented to enable data analytics to run more efficiently, such as PreDatA [30]. Larsen et al. [31] present a statistical modeling based technique to accurately predict the runtime cost of in situ volume rendering. Our work differs from all prior work in the sense that we comprehensively compare the state-of-the-art in-memory computing on two different platforms with a set of metrics, including end-to-end performance, usability, portability and robustness. Through deep analysis on these metrics, we identify the design and scalability challenges and opportunities.

III. Performance Evaluation

A. System Setup

This paper evaluates two realistic scientific workflows, LAMMPS and Laplace, along with a synthetic workflow on two different HPC systems: Titan [19] at ORNL and Cori [20] at NERSC. The Titan system contains 18,688 physical compute nodes, with each containing a 16-core 2.2 GHz AMD Opteron (Interlagos) processor, 32 GB of RAM, and an NVIDIA Kepler accelerator with 6 GB of DDR5 memory. The interconnect on Titan is Cray Gemini in 3D Torus and is capable of delivering 5.5 GB/s peak injection bandwidth [32] on each node. The parallel file system on Titan is Lustre, which is configured with 32 PB disk space and 1 TB/s peak performance. The Cori system has 2,388 Haswell nodes and 9,688 Knights Landing (KNL) nodes. Our experiments were conducted on KNL nodes, each of which is a single-socket Intel Xeon Phi processor with 68 1.4 GHz cores. Each core supports up to 4 hardware threads, thus totaling 272 threads per node. Each node has 96 GB 2.4 GHz DDR4 memory in six DIMMs. Each Haswell node has two 2.3 GHz 16-core Haswell processors with 128 GB memory. With the die-stacked memory, the total memory capacity of Cori is 1.09 PB. The interconnect on Cori is Cray Aries with the Dragonfly topology, and each node is capable of delivering 15.6 GB/s peak injection bandwidth [33]. Similar to Titan, Cori uses Lustre file system that has 248 storage targets with 744 GB/s peak performance and 30 PB disk space.

The build and runtime configurations of each in-memory library are detailed in Table I, and the descriptions of the scientific workflows used throughout this paper are presented in Table II. The LAMMPS workflow consists of the LAMMPS simulation [17], a molecular dynamics code, and computing the mean squared displacement (MSD) [16], [28]. In particular, the LAMMPS simulation models the clusters of Lennard-Jones atoms and studies the melting process of materials from a low-energy solid structure to a set of higher energy liquid structures. The coupled data analytics computes MSD, which characterizes the deviation between the position of a particle and a reference position. The Laplace workflow runs a Laplace based computational fluid dynamics simulation [34], [18], and its analysis output is further processed by a parallel n-th moment turbulence data analysis (MTA) [16]. Particularly, the output data of the two workflows are multi-dimensional floating-point arrays that are representative of HPC data [35].

In the evaluation, we configure the in-memory staging area based upon the workflow problem size and processor count [13], [14], [15], [16]. In particular, the number of Decaf servers is set to the number of analytics processors used. Unless otherwise specified, the numbers of DIMES and DataSpaces servers are set to 4 (since only metadata servers are involved in DIMES) and $(\#\ of\ analytics\ processors)/8$, respectively. Each DataSpaces server deals with 16 simulation and 8 analytics processors. For the runs throughout this paper, we vary either the processor count, i.e., the number of MPI processors used by the simulation and analytics in Figure 2, respectively, or the problem size in Figure 3, which is the simulation output size per MPI processor, to study the scaling behavior.

B. Performance

In this section, we study a set of in-memory libraries including Flexpath, DataSpaces with ADIOS, native DataSpaces, DIMES with ADIOS, native DIMES, and Decaf using the two real scientific workflows (Table II). For comparison, we also discuss the MPI-IO method, which dumps data from the simulation directly to persistent storage.

1) Overall Performance: Figure 2a and Figure 2b show the end-to-end times of running LAMMPS and Laplace workflows, respectively, using each of the in-memory libraries. The runs of "simulation only" and "analytics only" measure the time spent on the simulation and analytics, respectively, without I/O. Since the CPU frequency of Cori is only 63.6% of Titan, the finish times of compute-intensive Laplace workflow and MTA on Cori are much longer than those times on Titan. These provide a baseline for us to understand the scalability of the workflow itself. Overall, in-memory libraries, except DataSpaces on Titan, exhibit better scalability than MPI-IO due to the fact that the workflows are executed in the faster memory tier with the size of staging area scaling up with the processor count. For example, when the LAMMPS workflow scales from (32, 16) to (8192, 4096), the end-to-end time

TABLE I: Build and runtime configurations.

Method	Version	Build options	Runtime configurations
DataSpaces/ADIOS and DIMES/ADIOS	DataSpaces 1.7.2, ADIOS 1.13	–with-dataspaces, –with-dimes, –with-mxml, –with-flexpath, –enable-dimes, –with-dimes-rdma-buffer-size=1024, –enable-drc	lock_type=2, hash_version=2, max_versions=1
DataSpaces/native and DIMES/native	DataSpaces 1.7.2, ADIOS 1.13	–enable-dimes, –enable-drc, –with-dimes-rdma-buffer-size=2048	lock_type=2, hash_version=2, max_versions=1
MPI-IO/ADIOS	ADIOS 1.13	–with-mxml	lfs setstripe –stripe-size 1m –stripe-count -1, ADIOS XML: stats=off
Flexpath/ADIOS	ADIOS 1.13, EVPath for ADIOS 1.13	–with-flexpath	CMTransport=nnti, ADIOS XML: queue_size=1
Decaf	version as of 06/20/2018	transport_mpi=on, build_bredala=on, build_manala=on	prod_dflow_redist='count', dflow_con_redist='count'

TABLE II: Workflow description. Note that $nprocs$ is the number of MPI processors used in the simulation.

Workflow	Simulation	Analytics	Output data
LAMMPS	LAMMPS (version as of 08/22/2018), a molecular dynamics simulator	mean squared displacement (MSD)	$5 \times nprocs \times 512000$ double-precision data
Laplace	Solving Laplace's equation in a rectangle region	moment turbulence data analysis (MTA)	$4096 \times nprocs \times 4096$ double-precision data
Synthetic	An MPI based writer that outputs data to the staging servers in parallel	An MPI based reader that retrieves data from the staging servers in parallel	Configurable, e.g. a 3D array and each MPI processor access a portion.

increases only by 60% for Flexpath. In comparison, the end-to-end time of MPI-IO increases linearly with the number of processors allocated to the workflow. We note that such a trend of MPI-IO is a result of raw storage performance, for which there are only a fixed amount of Lustre storage targets available in the systems, as well as the metadata service, for which a very limited amount of Lustre metadata servers are deployed, with four on Titan and one on Cori.

The exception is DataSpaces running on Titan, in which the end-to-end time of LAMMPS increases rapidly with the processor count. Our diagnosis indicates that there exists a data decomposition mismatch between the writers in LAMMPS, the data layout in the staging servers, and the readers in MSD (further illustrated in Figure 8), which leads to the situation that all processors in the simulation and analytics have to access one staging server to put/get the data. This results in expensive and unscalable N-to-1 data movement at memory layer (detailed in Subsection III-B4). However, due to the higher throughput of the Aries interconnect on Cori (15.6 GB/s versus 5.5 GB/s on Titan), this overhead does not appear on Cori at the problem size (4096×2048).

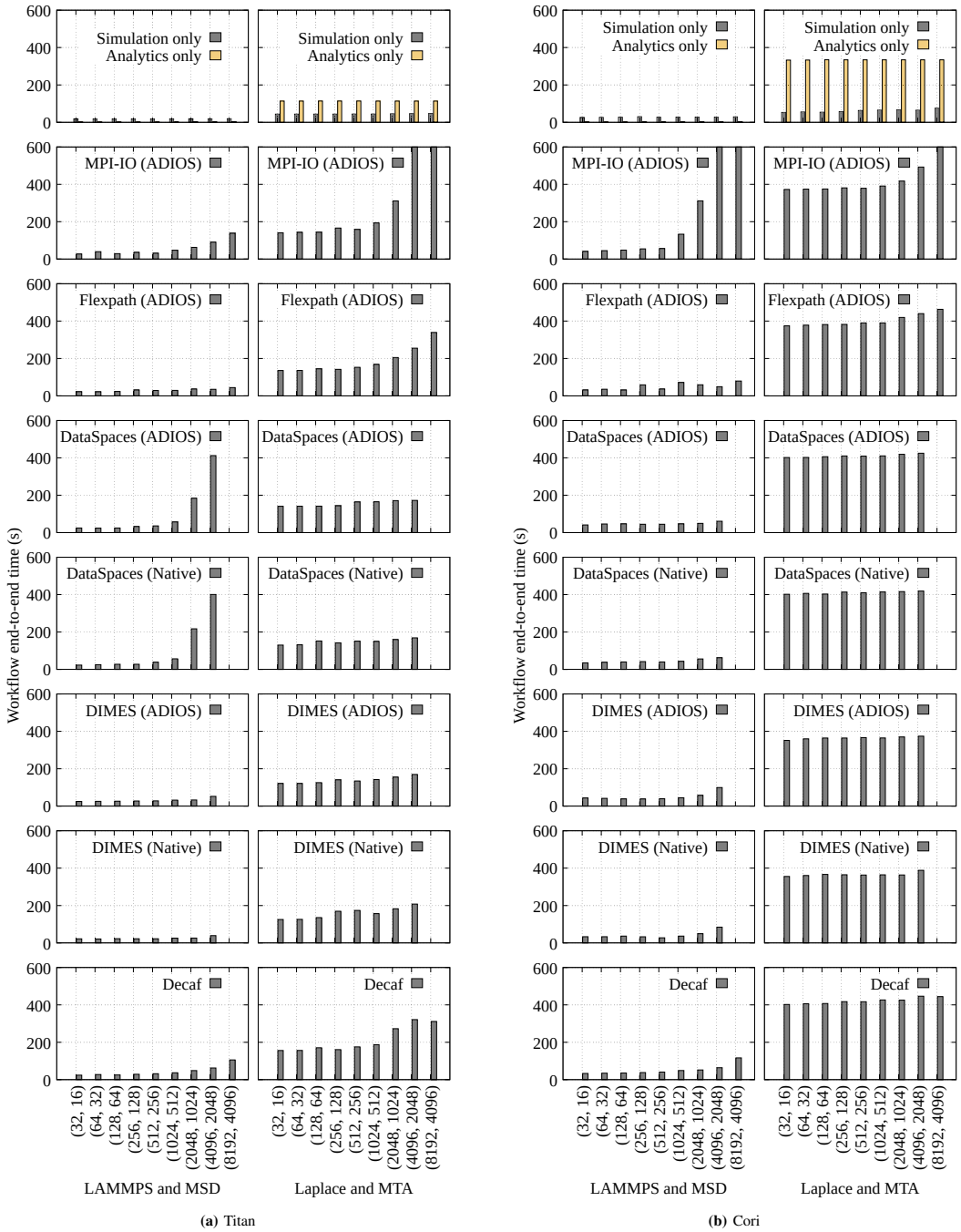
Finding 1: In-memory libraries do not always yield higher performance than persistent file I/O due to the expensive N-to-1 data movement at memory layer involved.

Note that both LAMMPS and Laplace workflows fail to run at (8192, 4096) on Cori, which uses the dynamic RDMA credentials (DRC) [36] to allow for the shared access between applications. If configured with RDMA, the workflow runtime needs to access the DRC service to acquire RDMA credentials prior to the communication. For a large-scale run that issues large amounts of parallel requests, the DRC server can be overwhelmed and result in failures. We will elaborate more on the impact of dimension mismatch in Section III-B4.

In Figure 3, we further examine the scalability by varying the problem size of the Laplace workflow from 512 KB (256×256) to 128 MB (4096×4096) per processor. It is observed that the end-to-end time increases proportionally with the problem size. We note that at the problem size of 128 MB, DataSpaces and DIMES are out of RDMA memory on Titan. Therefore, in this figure we double the amount of the staging servers in order to make the runs successful. As a matter of fact, a key obstacle to scaling up the problem size on Titan is the RDMA memory constraint. DataSpaces and DIMES adopt a low-level Cray RDMA interface uGNI to synchronously acquire and release RDMA memory. If requesting more RDMA resources than what is available in the system, then the acquire operation will fail and crash the application. What happened in the Laplace workflow is that two staging servers are launched on each node and each server handles 8 analytics and 16 simulation processors. A simulation or analytics processor calls the DataSpaces/DIMES API (datapspaces/DIMES_put() and datapspaces/DIMES_get()) to send/receive 128/256 MB data to/from the servers. When all simulation or analytics processors perform data movement concurrently, 2 GB RDMA memory is needed on each server and this exceeds the maximum RDMA capacity per node on Titan. Additionally, the RDMA resources on Titan are constraint not only by the total capacity, but also the number of RDMA memory handlers. The latter leads to the DataSpaces and DIMES failure at (8192, 4096) in Figure 2a, even with a reduced problem size (e.g., 20 MB).

To further quantify the limitation of RDMA resources, we run a synthetic test on Titan to acquire and release RDMA memory by varying the request size and concurrency, as shown in Figure 4. If the request size is less than 512 KB, the run can concurrently register at most 3,675 RDMA memory handlers. If the request is larger than 512 KB, the maximum concurrency of RDMA memory requests is constraint by the RDMA memory capacity (1,843 MB per node on Titan).

2) Memory Usage: As compared to persistent storage, memory is a precious resource on HPC systems. This is particularly true in the context of in-memory computing where the simulation and data analytics may share the same physical memory space, and sophisticated highly optimized data movement can further consume memory space. In light of the out of memory issues we experienced as well as the trend that the memory has become increasingly bottlenecked as compared to compute, we aim to further understand the memory usage



(a) Titan **(b)** Cori

Fig. 2: End-to-end time of scientific workflows (LAMMPS and Laplace) on different architectures (Titan and Cori KNL). The x-axis represents the the number of processors assigned to the simulation and analytics, respectively. For example, the notation of (32,16) represents the case that 32 and 16 processors are assigned to the simulation and analytics, respectively. LAMMPS and Laplace produce 20 MB and 128 MB per processor, respectively.

patterns of in-memory computing.

In Figure 5, we profile the memory usage of each library using the profiling tool, Valgrind, with millisecond resolution. In particular, we measure the memory usage at the simulation, data analytics, and staging servers (if any), respectively. For Flexpath, there are no stand-alone staging servers, and for DIMES, a staging server only manages metadata and therefore incurs a small memory consumption on the server side. Carefully note that the total memory consumption includes those consumed by the main calculation as well as those by calling the in-memory libraries. For DataSpaces, DIMES and Flexpath, LAMMPS uses approximately 400 MB of memory

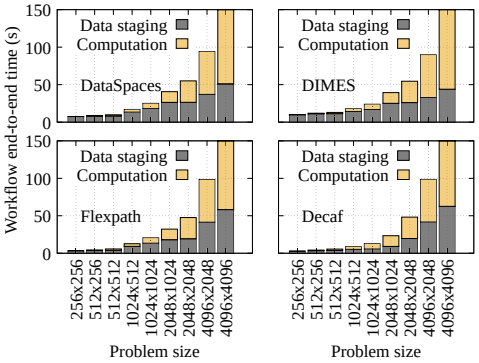


Fig. 3: Problem size scaling (Laplace). The simulation and analytics processors are 1,024 and 512, respectively. Note that the simulation and analytics overlap their computations.

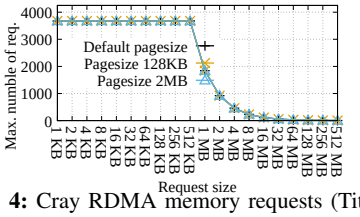


Fig. 4: Cray RDMA memory requests (Titan).

per MPI processor (Figure 5a, 5b and 5c), of which 173 MB is consumed by the numerical calculation and 227 MB is consumed by the in-memory libraries to perform data movement and staging. In contrast, Decaf needs 40% more memory (Figure 5d) due to the extra overhead incurred by flattening and buffering high dimensional data. Both DataSpaces and Decaf allocate up to 560 MB to stage the LAMMPS output. We note that in Figure 5a and 5e, the spike at 40 second marks the creation of DataSpaces staging servers.

To further diagnose the memory consumption, we break down the memory usage of the Laplace workflow in Figure 7. With each DataSpaces server handling 16 processors in the Laplace simulation, each of which generating 128 MB data, we expect 2 GB data in total to be staged in a server. However, in our measurement, we observe the total consumption is more than 2 GB due to the additional buffering used by DataSpaces. For Decaf, each Decaf server stages the output from two Laplace processors totaling 256 MB. As a result of extra data transformation between raw data and the internal object with rich semantic information, the total memory consumption of Decaf is 7 times that of the raw data size.

> Finding 2: The raw data transformation to high-level data abstraction with rich metadata and semantics can be overly expensive with regard to the memory consumption, and therefore needs to be carefully managed.

3) Cost of Indexing: To identify the location of target data, DataSpaces and DIMES build a spatial index to serve the queries from data analytics. Figure 6 shows the memory usage using the Hilbert space-filling curve (SFC) [37] to index data. When the problem size scales to 64 MB (4096×2048) per processor, each DataSpaces server allocates around 6 GB memory to index and stage data from 16 Laplace processors.

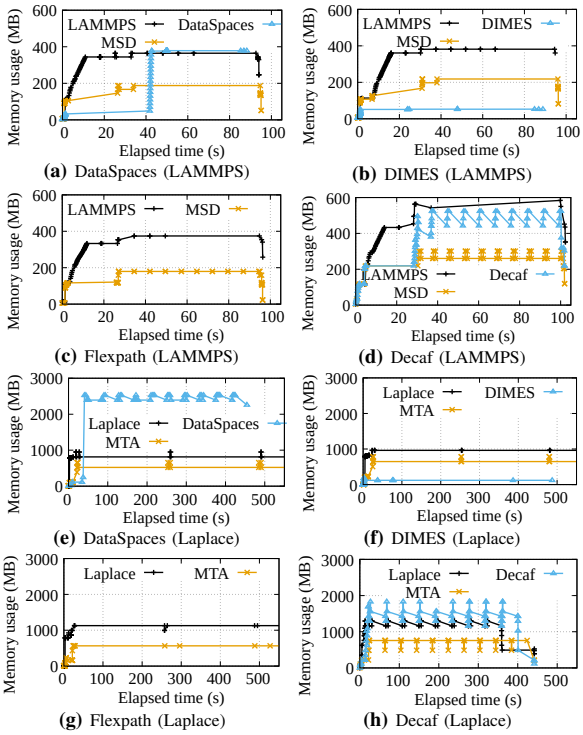


Fig. 5: Memory usage per processor in LAMMPS and Laplace workflow (Cori). Note that the output data sizes are 20 MB/processor for LAMMPS and 128 MB/processor for Laplace, respectively.

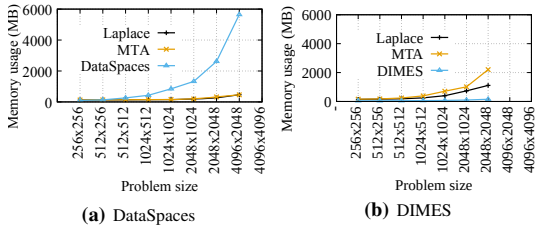


Fig. 6: Memory usage of the Laplace workflow.

Such a quadratic trend of memory usage is partially due to the construction of SFC index, which requires an n-dimensional index space with the size of each dimension being 2^k, where n is the number of dimensions of the raw data and k is the smallest integer that satisfies 2^k greater than the size of longest dimension. Each dimension is then divided into smaller buckets, which are mapped to DataSpaces servers evenly. For example, for the problem size of 4096×2048 per processor (i.e., with the global problem size of $4096 \times (64 \times 2048)$ for 64 Laplace processors), SFC will construct a mapping between the 2D data and index space with the size of 262144×262144, which is then evenly mapped to DataSpaces servers. This leads to a high indexing cost. In contrast, the DIMES servers use much smaller memory (about 154 MB) than DataSpaces due to the fact that DIMES stores the index at the simulation processors, rather than the metadata servers.

4) Data Decomposition and In-memory N-to-1 Access: In our experiments, data decomposition is observed to be another factor that substantially affects the performance. As aforemen-

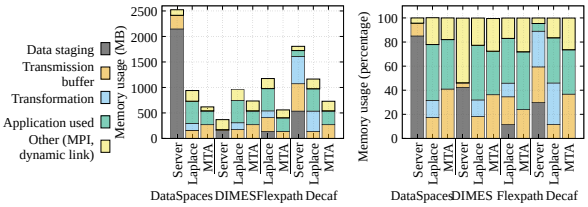
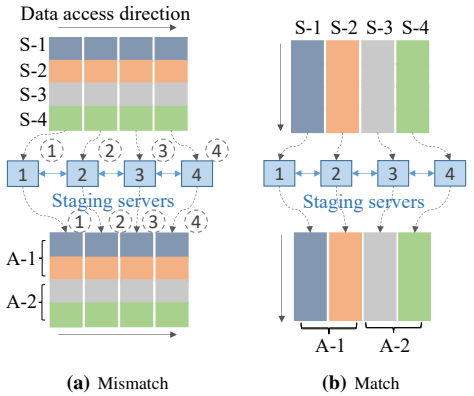


Fig. 7: Memory usage breakdown (Laplace).



(a) Mismatch **(b)** Match

Fig. 8: Data layout in the staging area. The suffixes "S-" and "A-" denote a processor in the simulation and data analytics, respectively.

tioned in Figure 2a, the end-to-end time of the LAMMPS workflow increases by up to 2X using DataSpaces/DIMES, when the processor count scales up to (4096, 2048). This performance degradation is attributed to the data decomposition at simulation side, which enforces N processors accessing one staging node. In particular, the domain decomposition of LAMMPS occurs in the second dimension, and the global dimensions of the LAMMPS output are $5 \times nprocs \times 512000$. However, DataSpaces/DIMES decomposes the problem domain into $2^{\lceil log(n) \rceil}$ regions in the longest dimension, where n is the number of staging servers. This decomposition strategy can result in a mismatch between the decomposition and processor scaling, with an example illustrated in Figure 8a. Internally each processor of simulation and analytics accesses its data region from begin to end in each iteration, without enabling multi-threads to split and concurrently access that region. As such, the regions are decomposed into sub-regions, which are mapped to the staging servers sequentially. Then all processors must access the corresponding staging servers in the same sequence as that of the sub-regions, leaving other staging servers in idleness. Thus, this results in the expensive N-to-1 communications. For example in Figure 8a, since the first sub-regions of all 4 'S-' processors are placed on staging *server 1*, the 4 processors have to concurrently access *server 1* with other three servers idle. By adjusting the domain decomposition, all processors can access all staging servers (N-to-N access), as shown in Figure 8b. Figure 9 shows that by matching the decomposition dimension to the processor scaling dimension, e.g. $5 \times 512 \times (1000 \times nprocs)$, the performance of the synthetic workflow can be improved by up to 5.3X.

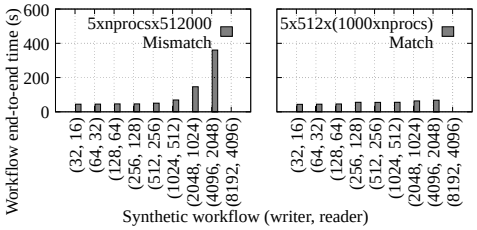


Fig. 9: The impact of data layout.

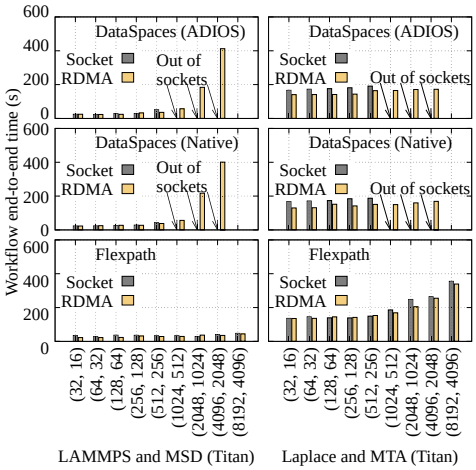


Fig. 10: Workflow end-to-end time using socket (Titan).

Finding 3: The mismatch between staging data layout and the decomposition strategy can result in the unexpected N-to-1 access to data staging area. This can introduce a significant performance penalty at scale (5.3X performance degradation in our experiments).

5) Transport Layer: Data movement is a critical step within an in transit workflow. In Figure 10, we compare the performance of using different transport layers for moving data between the simulation and analytics. As expected, RDMA results in a shorter end-to-end time than TCP sockets. The average performances of the LAMMPS and Laplace workflows are improved by 15.8% and 3.82% using Flexpath with NNTI, and by 8.4% and 17.3% using DataSpaces with Cray uGNI. The performance loss incurred by socket is mainly due to the cost of memory copy across the network stack [38], which is well acknowledged in the literature [38], [39], [40], [41].

Note that when using socket beyond (1024, 512), the workflows failed to establish socket connections between the staging servers and simulation/analytics. This is attributed to out of socket descriptors on the server side. After digging into the system logs and source code, we found that the socket connections are requested by the following operations: 1) simulation staging data to DataSpaces/DIMES servers; 2) analytics retrieving data from the servers; and 3) servers updating metadata among peers. As a result, even if we try to increase the number of data staging servers to reduce the connection load per server from simulation and analytics, a server still needs to establish more socket connections with its peers to manage distributed metadata.

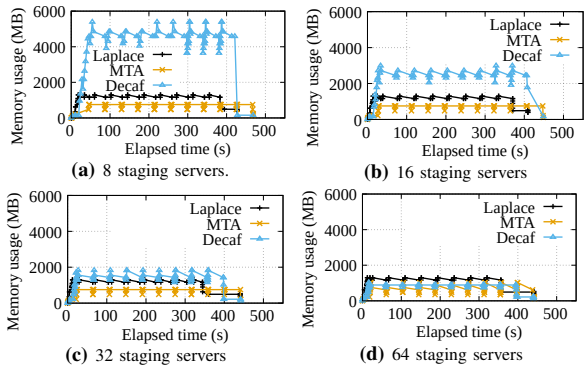


Fig. 11: Memory usage vs. the number of Decaf servers, Laplace workflow with processor scale (64, 32), (Titan).

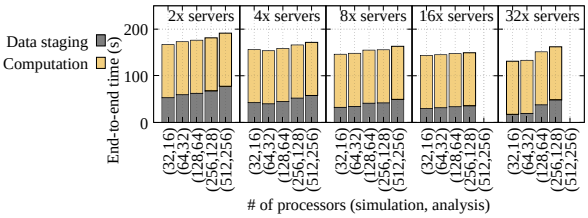


Fig. 12: End-to-end time vs. # of DataSpaces servers using socket (Titan). The baseline is that we use one DataSpaces server for (32, 16) and we maintain this ratio of the number of servers to the processor count when scaling up.

Finding 4: While using high-level protocols, e.g., RPC and sockets over RDMA, is more convenient and portable, proprietary low-level RDMA implementations, such as Cray uGNI, yield substantial performance gains, e.g., up to 17.3% in DataSpaces. The accompanied challenge is that the non-trivial engineering effort on adapting the low-level implementations to various application scenarios.

6) Number of Staging Servers: Intuitively, by increasing the number of staging servers in Decaf and DataSpaces, one would expect higher performance and lower memory usage per server. To quantify the impact of the number of staging server, we run a set of experiments by varying the number of staging servers. For Decaf, as shown in Figure 11, the memory usage per server is proportionally decreased by 83.5%, when the number of servers increases from 8 to 64. However, the end-to-end time of the Laplace workflow is only reduced by 5.5%, quite insensitive to the number of servers. Figure 12 shows the end-to-end time of the Laplace workflow under different number of DataSpaces servers. The overall performance improvement by doubling the number staging servers is about 5.4% and the performance gain on data staging (versus computation) can be up to 20.1%.

7) Shared Memory: Figure 13 shows the results of running the LAMMPS and Laplace workflows using shared memory on Cori. Compared to the results of Figure 2b, the performance of using Flexpath improves by 12.7% and 17.0%, respectively, for LAMMPS and Laplace. Similarly, the performance of using DataSpaces improves by 11.0% and 8.9%, respectively. The gain is attributed to the shortened I/O path from off-

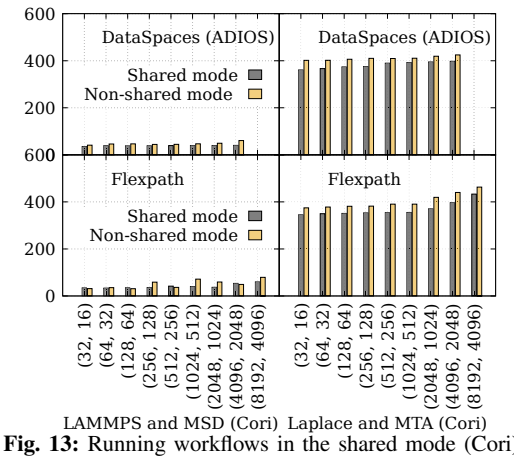


Fig. 13: Running workflows in the shared mode (Cori).

node data movement to local memory copy. Nevertheless, two executables sharing memory on a node is not universally supported on HPC systems. For example, Titan does not allow multiple jobs running on the same compute node. Even though Cori supports multiple jobs sharing one node, it does not support *heterogeneous running*, which wraps multiple jobs in one MPI communicator and allocates resources to each job. Without this capability, Decaf cannot allocate compute resources to the MPI wrapped workflow. In addition, the DRC service may prohibit some workflows from running in the shared mode. By default, DRC does not allow multiple jobs on the same node to use the same credential to access a shared network domain, unless its *node-insecure* option is enabled. As a result, in Figure 13, we had to run DataSpaces using socket rather than uGNI in order to avoid acquiring and releasing DRC on Cori.

Finding 5: Despite the substantial performance improvement (about 10%), shared memory is a restricted running mode on some leadership HPC systems due to security.

IV. Library Quality Assessment

In this section, we further evaluate in-memory computing more broadly regarding usability, portability, and robustness.

A. Usability

In-memory computing ultimately needs to be in the hands of domain scientists to allow analytics to be done more efficiently for science productions. Therefore, the aspect of software usability, for long being ignored and less favored as compared to the software performance and robustness, would matter for the broad adoption of in-memory computing. For example, ADIOS is a framework that incorporates a number of third party libraries to accommodate a broad spectrum of science scenarios, making it nontrivial for domain scientists to configure and build ADIOS properly on their local systems. As of now, most of these in-memory libraries still need extensive support from HPC administrators or library developers in order to be seamlessly integrated in science production. Herein we thoroughly examine the complexity of their build systems and software interfaces, two key factors to the overall usability of in-memory libraries.

TABLE III: Lines of code for configuration and API invocation.

Category	LOC	Functionality
DataSpace and DIMES (ADIOS)		
Build options	13	Enable RDMA, socket and etc.
Runtime config.	8	Define staging area: dimensions, size, offset and etc.
ADIOS XML config.	18	Data description in ADIOS: dimensions, size, offset and etc.
ADIOS data staging API	30	Server and client init, put/get data, and finalize
DataSpace and DIMES (native)		
Build options	13	Enable RDMA, socket and etc.
Runtime config.	8	Define staging area: dimensions, size, offset and etc.
Data staging API	81	Server and client init, lock/unlock, put/get data, and finalize
Flexpath		
Build options	5	RDMA API options, compiler and flags.
ADIOS XML config.	18	Data description in ADIOS: dimensions, size, offset and etc.
Data staging API	30	Init, put/get data and finalize
Decaf		
Build options	8	Enable transport layers, e.g. MPI
Bootstrap script	21	Define and link producer, consumer and staging processes
Data staging API	32	Init, dynamical load libs, data transformation, staging and finalize

By and large, there are dozens of building options and switches to properly configure these libraries, and some of these options require users to be knowledgeable about the HPC hardware, such as the details of HPC interconnect, or the library internals, such as the hashing scheme used. Additionally, they may require domain scientists to compose the library configuration files (e.g. XML file for Flexpath), or write nontrivial code (for Decaf) for application integration. In particular, DataSpaces, DIMES and Flexpath connect simulation, staging servers and analytics via the communication APIs, while Decaf adopts python or C++ to wrap them into one MPI communicator. The engineering efforts involved can be substantial, e.g., defining and accessing staging area and defining staging object (dimensions, size and data type) and etc. Therefore, unless the I/O cost of an application is prohibitively high, in-memory computing may see resistance from domain scientists for production usage. In Table III, we summarize the number of lines of code for build and runtime configurations as well as calling the APIs to perform in-memory computing on Cori and Titan.

Finding 6: In terms of usability, in-memory libraries are still far from being plug-and-play for domain scientists, and most of them require substantial support from HPC administrators or library developers, e.g. choosing the optimal build options and runtime I/O configurations.

B. Portability

The HPC systems have become increasingly diverse in terms of the architecture, hardware and software. The ability to be able to function seamlessly across platforms and fully exploit the hardware and software ecosystem is crucial for production science. This is particularly true for in-memory computing as it needs to deal with the interconnect and memory subsystem which are often different across machines.

At the hardware level, we investigate the portability of in-memory libraries between CPUs and GPUs. We found that GPU is mostly not supported by the current in-memory libraries, and data staging is assumed to be done at main memory between compute nodes, rather than at GPU memory. As a result, GPU-enabled workflows are required to take care of the movement between GPU and CPU memory. We comment that, given the recent development in new interconnects, e.g., NVLink, between GPUs, we believe this will an attractive area for future research and development for in-memory computing.

At the transport layer, DataSpaces, DIMES and Flexpath support both TCP sockets as well as high performance protocols including RDMA, Cray Gemini via Sandia NNTI [22] and uGNI [23]. In contrast, Decaf wraps the workflow components into an MPI communicator and thus the communication is done through MPI message passing. Therefore, Decaf relies on MPI to be portable. For all the in-memory computing libraries, we have not seen official releases that explicitly support new memory devices, such as non-volatile memory and die-stacked memory, for memory allocation and migration. In the libraries, the usage of new memory devices are mainly handled by the operating system and are used no differently from the conventional disks and DRAM.

At the application layer, both DataSpaces and DIMES can be configured as services to applications through generic API calls. Additionally, DataSpaces, DIMES and Flexpath are integrated into the ADIOS framework as transport method, which allows them to interact with a range of analytics packages, such as FastBit, Paraview, VisIt, using the ADIOS API as the interface of choice [14]. Rather than providing generic APIs, Decaf is designed to build a dataflow system for coupled scientific workflows and rely on MPI for communications. Therefore, Decaf is applicable to any MPI-based (versus socket-based) scientific workflows.

Finding 7: To achieve high performance and portability for expert users, most of in-memory libraries can be configured to reduce the layers of I/O stack and ported to low-level APIs. For non-expert users without the knowledge of performance tuning, these libraries can be ported to high-level abstraction API, e.g. TCP socket over RDMA, so as to hide the low-level API complexity.

C. Robustness

We view robustness as an area that needs continued investment and improvement in order for in-memory computing to be fully production ready. This is attributed to the following: 1) compared to the enterprise sector, the HPC user communities are much smaller, particular for those who are supercomputer users. As a result, the opportunities of leveraging the user communities to provide feedback and potentially have them contribute to the code development are insignificant. We looked into some of the library repositories and noticed that the code contributions are dominated by the library developers, with occasional bug fixes initiated from domain scientists. 2) HPC systems are rapidly evolving with new architectures and hardware, and in some cases the hardware interfaces are proprietary, such as the Cray interconnect. 3) Resilience mechanisms for machine failures have not been constructed in existing in-memory computing libraries. This is vital to large-scale runs of workflows, since we know the machine failures are quite common in the extreme-scale cluster. In conjunction with the complexity in scale and application, this results in substantial engineering challenges associated with hardening the software infrastructure on HPC systems.

TABLE IV: Lessons of running in-memory workflows.

Issue	Description	Suggested resolve
Out of RDMA memory	Data movement between simulation and data analytics can deplete the shared RDMA resources on a compute node.	1. Better error handing, e.g., adding wait and re-try. 2. Add a layer of indirection that manages the requests from applications so that the RDMA resource constraint can be checked and observed in advance.
Data dimension overflow	The dimension size can be overflown, if it is set to 32-bit unsigned integer	Switch to 64-bit unsigned long int.
Out of main memory	In-memory libraries might incur huge memory footprint (e.g., 7 times the size of analysis data (1.8 GB versus 256 MB)), resulting in unexpected out of memory aborts.	1. Better understand the memory consumption through profiling so that we can allocate sufficient memory resources. 2. Optimize memory usage by freeing the memory region that may not be used immediately.
Out of sockets	A reader in data analytics may read data from all processors in the staging servers. In this case, the socket descriptors can be depleted on a compute node.	1. If possible, adjust the communication pattern so that a reader only reads data from a small number of processors. 2. Alternatively, we can design a socket pool that is responsible for communication so that only a small number of sockets are used. However, this may compromise the data movement efficiency.
Out of DRC	If uGNI is used as the RDMA layer, an application need to access the DRC service to acquire RDMA credentials prior to communication. A large scientific workflow may overwhelm the DRC.	1. Add a layer of indirection that manages requests to the DRC service. 2. Re-sign the DRC service to be distributed so that it can handle a large amount of requests at a time.

Table IV lists the robustness-related issues we encountered in deploying and running these workflows on Titan and Cori. In addition to the out of RDMA memory issue as aforementioned, there are a handful of hardware and software issues. First, while the data dimension overflow issue is not erroneous from the library level, we note that this often results in ugly crashes that may not be easily debugged. Second, the out of memory error appears much more often than one would expect. This is mainly due to the optimizations that these libraries employ to trade space for improving usability and robustness via high-level abstraction. In Decaf, we observed in our runs that memory consumption can be as high as 7 times the raw data size. For extreme-scale dataset, this can result in unexpected out of memory errors. In addition to the capacity related issues, resource descriptors for both RDMA and sockets can also run out quickly for large runs. Last but not the least, a single entity, such as the DRC server, in a massively parallel environment, can be easily overwhelmed.

Finding 8: Using sophisticate high-level abstractions, e.g., wrapping all components into MPI or using TCP over RDMA, dose not always improve usability and robustness. In an extreme run, available resources, e.g. memory and socket capacity, might be overwhelmed by high abstraction overhead and lead to crash, particularly while running those data intensive workflows.

TABLE V: Qualitative summary. '+' denotes that the finding is relevant to a particular library, while '-' denotes otherwise and '+/-' denotes being conditionally relevant.

Findings	DataSpaces	DIMES	Flexpath	Decaf
Finding 1	+	-	-	-
Finding 2	+/-	-	-	+
Finding 3	+	-	-	-
Finding 4	+	+	+	-
Finding 5	+/-	+/-	+/-	-
Finding 6	+	+	+	-
Finding 7	+	+	+	-
Finding 8	-	-	-	+

D. Qualitative Analysis of Findings

Herein, we conduct a qualitative analysis on the above findings in Table V. This qualitative analysis can further reveal the challenges and opportunities of using in-memory computing to potential users including both domain scientists and library developers. Note that some findings are conditionally relevant to in-memory libraries. For example, finding 2 is relevant to DataSpaces on the condition of using SFC to index staging data. Finding 5 needs supports from HPC job scheduler and system administrators. To avoid the undesirable situations in finding 1 and 3, users need to understand the inner mechanisms of how application data is decomposed and moved into the staging servers, so as to lower the cost of data movement. Finding 2 can be a lesson learned for library developers - simply using high-level abstractions can be overly expensive. Finding 4 can be beneficial to both library developers and domain scientists. Although using sockets and RPC is more convenient and productive for code development, using proprietary low-level RDMA implementations can result in substantial performance gains. This is an area that needs continued investment so that proprietary RDMA can be fully exploit and hardened. Similarly in finding 5, shared memory is not well supported on HPC systems and needs to hardened. Finding 6, 7 and 8 evaluate in-memory libraries beyond the common performance metrics, and reveal the design trade-offs among usability, portability and robustness.

V. Conclusion

This paper conducts a comprehensive study of in-memory computing on two large supercomputers, Titan and Cori, using two realistic scientific workflows. Our results suggest that in general, in-memory computing offers much higher scalability and performance than the traditional post-processing. However, the scalability of in-memory is often constrained by the availability of resources on HPC systems, such as RDMA memory and sockets. These libraries generally have excellent portability across platforms, leveraging the sophisticated software stack, such as MPI, and open frameworks, such as ADIOS. Usability and robustness are the two areas that need continued investment and improvement due to the complexity of HPC systems and the design trade-off among usability, portability and performance. Furthermore, we assess the memory usage, the impact of data layout, transport layer, sharing mode and etc, for in-memory computing.

VI. Acknowledgments

The authors wish to acknowledge the support from the US NSF under Grant No. CCF-1718297, CCF-1812861, and NJIT research startup fund. This research used resources of the Oak Ridge Leadership Computing Facility and National Energy Research Scientific Computing Center, which are supported by the Office of Science of the U.S. Department of Energy.

References

[1] I. Foster and et al., "Computing just what you need: Online data analysis and reduction at extreme scales," in *Euro-Par 2017: Parallel Processing*. Springer International Publishing, 2017.

[2] Q. Liu and et al., "Hello adios: the challenges and lessons of developing leadership class i/o frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014. [Online]. Available: http://dx.doi.org/10.1002/cpe.3125

[3] U. Ayachit and et al., "Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis SC'16*. Piscataway, NJ, USA: IEEE Press, 2016, pp. 79:1–79:12.

[4] A. C. Bauer, B. Geveci, and W. Schroeder, "The paraview catalyst user's guide," *Clifton Park, NY: Kitware*, 2013.

[5] Y. Wang, G. Agrawal, T. Bicer, and W. Jiang, "Smart: A mapreduce-like framework for in-situ scientific analytics," in *SC'15*. IEEE, 2015, p. 51.

[6] D. Huang, Q. Liu, S. Klasky, J. Wang, J. Y. Choi, J. Logan, and N. Podhorszki, "Harnessing data movement in virtual clusters for in-situ execution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 615–629, 2019.

[7] T. Fogal, F. Proch, A. Schiewe, O. Hasemann, A. Kempf, and J. Krüger, "Freeprocessing: Transparent in situ visualization via data interception," in *Eurographics Symposium on Parallel Graphics and Visualization: EG PGV:[proceedings]/sponsored by Eurographics Association in co-operation with ACM SIGGRAPH. Eurographics Symposium on Parallel Graphics and Visualization*, vol. 2014. NIH Public Access, 2014, p. 49.

[8] M. Dorier and et al., "Damaris/viz: a nonintrusive, adaptable and user-friendly in situ visualization framework," in *2013 IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*. IEEE, 2013, pp. 67–75.

[9] F. Zheng and et al., "Goldrush: resource efficient in situ scientific data analytics using fine-grained interference aware execution," in *SC'13*. ACM/IEEE, 2013, p. 78.

[10] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi, "Enabling in-situ execution of coupled scientific workflow on multi-core platform," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 1352–1363.

[11] P. Malakar and et al., "Optimal scheduling of in-situ analysis for large-scale scientific simulations," in *SC'15*. IEEE, 2015, pp. 1–11.

[12] J. C. Bennett and et al., "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *SC'12*. IEEE, 2012, pp. 1–9.

[13] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: an interaction and coordination framework for coupled simulation workflows," *Cluster Computing*, vol. 15, no. 2, pp. 163–181, 2012.

[14] J. Dayal and et al., "Flexpath: Type-based publish/subscribe system for large-scale science analytics," in *2014 14th IEEE/ACM CCGrid*. IEEE, 2014, pp. 246–255.

[15] M. Dreher and T. Peterka, "Decaf: Decoupled dataflows for in situ high-performance workflows," Argonne National Lab.(ANL), Argonne, IL (United States), Tech. Rep., 2017.

[16] Y. Fu and et al., "Performance analysis and optimization of in-situ integration of simulation with data analysis: zipping applications up," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2018, pp. 192–205.

[17] S. Plimpton, P. Crozier, and A. Thompson, "Lammps-large-scale atomic/molecular massively parallel simulator," *Sandia National Laboratories*, vol. 18, p. 43, 2007.

[18] (2011) Laplace's equation in a rectangle, solved with mpi. [Online]. Available: https://people.sc.fsu.edu/\~jburkardt/c_src/laplace_mpi/laplace_mpi.html

[19] (2019) Titan user guide. [Online]. Available: https://www.olcf.ornl.gov/for-users/system-user-guides/titan

[20] (2019) Cori user guide. [Online]. Available: https://www.nersc.gov/users/computational-systems/cori

[21] C. Docan, M. Parashar, and S. Klasky, "Dart: a substrate for high speed asynchronous data io," in *Proceedings of the 17th international symposium on High performance distributed computing*. ACM, 2008, pp. 219–220.

[22] J. Lofstead, R. Oldfield, and T. Kordenbrock, "Unconventional data staging using nssi," in *In Proceedings of IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, Delft, The Netherlands*, 2013.

[23] Y. Sun, G. Zheng, L. V. Kale, T. R. Jones, and R. Olson, "A ugni-based asynchronous message-driven runtime system for cray supercomputers with gemini interconnect," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 751–762.

[24] G. Eisenhauer, M. Wolf, H. Abbasi, and K. Schwan, "Event-based systems: opportunities and challenges at exascale," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*. ACM, 2009, p. 2.

[25] G. Eisenhauer, M. Wolf, H. Abbasi, S. Klasky, and K. Schwan, "A type system for high performance communication and computation," in *2011 IEEE Seventh International Conference on e-Science Workshops*. IEEE, 2011, pp. 183–190.

[26] V. Vishwanath, M. Hereld, and M. E. Papka, "Toward simulation-time data analysis and i/o acceleration on leadership-class systems," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*. IEEE, 2011, pp. 9–14.

[27] A. G. Landge and et al., "In-situ feature extraction of large scale combustion simulations using segmented merge trees," in *SC'14*. IEEE, 2014, pp. 1020–1031.

[28] P. Malakar, V. Vishwanath, C. Knight, T. Munson, and M. E. Papka, "Optimal execution of co-analysis for large-scale molecular dynamics simulations," in *SC'16*. IEEE Press, 2016, p. 60.

[29] M. D. Hanwell, K. M. Martin, A. Chaudhary, and L. S. Avila, "The visualization toolkit (vtk): Rewriting the rendering code for modern graphics cards," *SoftwareX*, vol. 1, pp. 9–12, 2015.

[30] F. Zheng and et al., "Predata–preparatory data analytics on peta-scale machines," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[31] M. Larsen, C. Harrison, J. Kress, D. Pugmire, J. S. Meredith, and H. Childs, "Performance modeling of in situ rendering," in *SC'16*. IEEE, 2016, pp. 276–287.

[32] S. S. Vazhkudai and et al., "The design, deployment, and evaluation of the coral pre-exascale systems," in *SC'18*. IEEE Press, 2018, p. 52.

[33] D. Doerfler and et al., "Evaluating the networking characteristics of the cray xc-40 intel knights landing-based cori supercomputer at nersc," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 1, p. e4297, 2018.

[34] G. C. Fox, "Solving problems on concurrent processors," *Old Tappan, NJ; Prentice Hall Inc.*, 1988.

[35] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *2016 ieee international parallel and distributed processing symposium (ipdps)*. IEEE, 2016, pp. 730–739.

[36] J. Shimek, J. Swaro, and M. Saint Paul, "Dynamic rdma credentials," 2016. [Online]. Available: https://cug.org/proceedings/cug2016_proceedings/includes/files/pap108s2-file1.pdf

[37] T. Bially, "Space-filling curves: Their generation and their application to bandwidth reduction," *IEEE Transactions on Information Theory*, vol. 15, no. 6, pp. 658–664, 1969.

[38] N. S. Islam and et al., "High performance rdma-based design of hdfs over infiniband," in *SC'12*. IEEE Computer Society Press, 2012, p. 35.

[39] N. S. Islam, M. Wasi-ur Rahman, X. Lu, and D. K. Panda, "High performance design for hdfs with byte-addressability of nvm and rdma," in *Proceedings of the 2016 International Conference on Supercomputing*. ACM, 2016, p. 8.

[40] J. Jose and et al., "Memcached design on high performance rdma capable interconnects," in *2011 International Conference on Parallel Processing*. IEEE, 2011, pp. 743–752.

[41] X. Lu, D. Shankar, S. Gugnani, and D. K. D. Panda, "High-performance design of apache spark with rdma and its benefits on various workloads," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 253–262.