A Decoupled Execution Paradigm for Data-Intensive High-End Computing

Yong Chen [1], Chao Chen [1], Xian-He Sun [2], William D. Gropp [3], Rajeev Thakur [4]
[1] Department of Computer Science, Texas Tech University, Lubbock, TX
[2] Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois, USA
[3] Department of Computer Science, University of Illinois Urbana-Champaign, Urbana, Illinois, USA
[4] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, USA
Email: {yong.chen@ttu.edu, chao.chen@ttu.edu, sun@iit.edu, wgropp@illinois.edu, thakur@mcs.anl.gov}

***Abstract*— High-end computing (HEC) applications in critical areas of science and technology tend to be more and more data intensive. I/O has become a vital performance bottleneck of modern HEC practice. Conventional HEC execution paradigms, however, are computing-centric for computation intensive applications. They are designed to utilize memory and CPU performance and have inherent limitations in addressing the critical I/O bottleneck issues of HEC. In this study, we propose a decoupled execution paradigm (DEP) to address the challenging I/O bottleneck issues. DEP is the first paradigm enabling users to identify and handle data-intensive operations separately. It can significantly reduce costly data movement and is better than the existing execution paradigms for data-intensive applications. The initial experimental tests have confirmed its promising potential. Its data-centric architecture could have an impact in future HEC systems, programming models, and algorithms design and development.**

Keywords- decoupled execution paradigm, high-end computing, data-intensive computing, storage

I. INTRODUCTION

Many high-end computing (HEC) applications in critical areas of science and technology are becoming more and more data intensive [CLGN09, DBMA11]. For instance, twelve out of twenty-six INCITE applications run at Argonne National Laboratory generate and store terabytes of data on-line [RLUW09]. These applications contain a large number of I/O accesses, where large amounts of data are written to and retrieved from storage. In addition to newly emerged data-intensive applications, such as information retrieval and transaction collection, conventional high-end computing applications have also become increasingly data intensive due to the ever increasing of computing power, and new needs such as animation and data mining. Input/output has become the key performance factor in modern computing.

The rapid advance of semiconductor process technology and the evolution of microarchitectures, such as multicore/manycore architectures, have drastically increased the computing power of microprocessors. However, compared to the computational performance improvement, data-access performance (latency and bandwidth) improvement has been at a snail's pace. The disk drive speed has only increased by roughly 7% each year over the past two decades, which is significantly lagging behind the nearly 50% per year improvement of processor speed [CLGN09, DBMA11]. This disparity of performance improvement is expected to continue in the near future. Figure 1 compares the disk drive bandwidth improvement (left vertical axis) and the computational capability improvement of well-known supercomputers (right vertical axis) for the past several decades [RLUW09]. The computational performance improvement rate is magnitudes higher than the bandwidth improvement rate of disk drives, which causes a so-called "I/O-wall" problem.

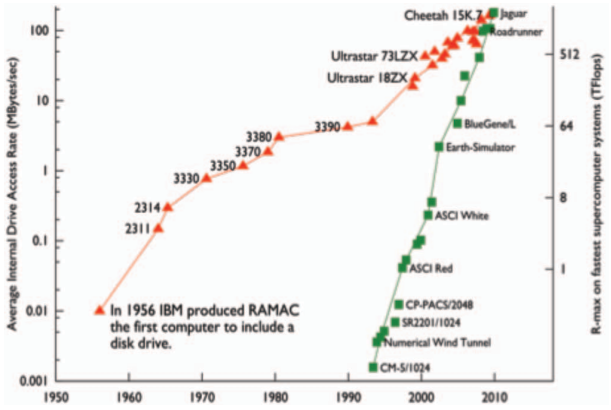

Figure 1. FLOPS of HEC Systems v.s. Disk Drive Bandwidth [RLUW09]

Data access has become the bottleneck of computing. However, existing HEC execution models, and their associated runtime systems, are computing-centric [GrLT99, GA, UPC]. They are not ready to support efficient input/output. For instance, MPI, the dominant parallel programming model of HEC, focuses on exchanging in-memory data for parallel computations. HEC architecture and system research often consider I/O devices as peripheral and leave them to someone else. HEC performance is commonly measured in terms of peak performance of small computation kernels that can fit into memory and cache well. The data-driven IT industry has developed a new paradigm, MapReduce, for their needs

978-0-7695-4807-4/12 $26.00 © 2012 IEEE
DOI 10.1109/CLUSTER.2012.80

[DeGh04]. There is a great need for the HEC community to rethink the execution models for the coming data-intensive HEC era.

In this study, we propose an innovative *decoupled execution paradigm* (DEP) and the notion of the separation of computing nodes and data (processing) nodes. The novelty of the new DEP execution paradigm is that the data nodes, collectively, take care of the data-intensive operations of the application. The computing nodes, collectively, take care of the computation-intensive operations. The application is executed in a decoupled but fundamentally more efficient manner for data-intensive HEC with the collective support from data processing nodes and computing nodes. The data processing nodes proposed in the DEP design are extension of the prior work of server-push architecture [SuBC07a, SuBC07b] that employs dedicated data-access servers to proactively push data to compute nodes instead of a traditional pull-based architecture. The DEP execution paradigm is an evolutionary, if not revolutionary, execution model where I/O intensive operation is as important as computation. The current results have shown the DEP approach is promising and has a potential.

The rest of the paper is organized as follows. Section II reviews important existing studies in related areas. Section III presents the design and notion of the proposed decoupled execution paradigm. Section IV discusses implementation issues and introduces the initial prototyping implementation. Section V presents the experimental evaluation results. Section VI concludes this study and discusses the future work.

II. Related Work

Extensive studies have focused on improving the performance of data-intensive HEC systems at various levels. This section discusses existing studies along three lines: architecture improvements, programming model improvements, and runtime system improvements. To the best of our knowledge, there is no study that rethinks the execution paradigm to address fundamental I/O issues.

A. Architecture Improvements for Data-Intensive High-End Computing

At the hardware level, the emerging nonvolatile storage-class memory devices such as flash-memory based solid-state drives and phase-change memory can provide more promising performance than hard disk drives, especially for random accesses [ChKZ11, DoXi11]. However, they cannot reduce the data movement across the network, and they help to mitigate the performance gap between CPU and I/O but will not be able to solve the I/O bottleneck problem alone.

Active storage [RiGF98, SLCR10, XMFL11], active disks [RiGi97, ChMa02], and smart disks [ChLC03] have gained increasing attention recently. Active storage leverages the computing capability of storage nodes and performs certain computation to reduce the bandwidth requirement between storage and compute nodes. Active disks and smart disks integrate a processing unit within disk storage devices and offload computations to embedded processing unit. However, these architecture improvements are designed to explore either the idle computing power of storage nodes or an embedded processor, and have limited computation-offloading capability. It is easy to see that DEP provides a much more powerful platform for the same purpose. I/O forwarding (both hardware and software solutions) [ACIK09, IRYB08] and data shipping [ScHa02] provide approaches to offloading I/O requests to dedicated nodes, aggregating the requests, and carrying out them on behalf of compute nodes. The data nodes proposed in the DEP design can carry all these functions and do more.

B. Programming Model Improvements for Data-Intensive High-End Computing

Current parallel programming models are designed for computation-intensive applications. These programming models include Message Passing Interface (MPI) [GrLT99], Global Arrays [NiHL94], Unified Parallel C [ElSm06], Chapel, X10, Co-array Fortran, and data parallel programming models such as High Performance Fortran (HPF). These programming models primarily focus on the memory abstractions and communication mechanism among processes. I/O is treated as a peripheral activity and often a separate phase in these programming models and execution paradigms, which is often achieved through a subset of interfaces such as MPI-IO [TRLG04].

Advanced I/O libraries, such as Hierarchical Data Format (HDF), Parallel netCDF (PnetCDF), and Adaptable IO System (ADIOS), provide high-level abstractions, map the abstractions onto I/O in one way or another, and complement parallel programming models in managing I/O activities. The recent MapReduce programming model [DeGh04, SMWB10] is an instant hit and has been proven effective for many data-intensive applications. The MapReduce model, however, is typically layered on top of distributed file systems and is not designed for high performance computing semantics. It requires specific Map and Reduce abstractions as well [DeGh04, SMWB10]. DEP is designed for general parallel applications, with an increased programming capability.

C. Runtime System Improvements for Data-Intensive High-End Computing

There has been significant amount of research effort in optimizing I/O performance using runtime libraries, such as collective I/O [ThGL99, LiCh08, CSTR11], two-phase I/O, extended two-phase I/O, data sieving, server-direct I/O, disk-directed I/O, lightweight I/O [OWRM06], partitioned collective I/O [YuVe08], layout-aware collective I/O [CSTR11], ADIOS library [LKSP08], and resonant I/O [ZhJD09]. These strategies collect and aggregate small requests into larger ones at the I/O client/middleware/server level.

Many caching, buffering, staging, and prefetching optimization strategies exist at runtime as well, such as collective caching [LCCC07], collective buffering [NiLo97], active buffering [MWLY02], discretionary caching [VSKT06], SpecHint prefetching [ChGi99], transparent informed prefetching (TIP) [PGGS95], adaptive prefetching based on time series modeling [TrRe04], multiple-level caching and prefetching for Blue Gene systems [BICL09], and our prior work in pre-execution based prefetching [CBST08, CBST08a] and a signature based prefetching with post-execution analysis

[BCST08]. Abbasi et. al. recently proposed a DataStager framework with data staging services that move output data to dedicated staging or I/O nodes prior to storage, which has been proven effective in reducing the I/O overheads and interferences on compute nodes [AWEK10]. Zheng et. al. proposed a preparatory data analytics (PreDatA) approach to preparing and characterizing scientific data when generated (e.g. data reorganization and metadata annotation) to speedup subsequent data access [ZADL10]. These approaches have shown considerable performance improvement with dedicated output staging services and preparatory analysis. Our proposed DEP approach, built upon server-push architecture [SuBC07a, SuBC07b], leverages dedicated nodes as well, but is different. The dedicated data processing nodes work for both reads and writes, and can provide buffering or staging, but more importantly on reduction. The notion of data processing nodes in DEP is a rethinking of HEC systems architecture to provide balanced computational and I/O capability. The DEP considers to address the I/O bottleneck issues fundamentally from the execution paradigm including systems architecture and programming model, not only from runtime optimizations.

Parallel file systems (PFS), such as Lustre, GPFS [ScHa02], PanFS, PVFS, and PPFS2, enable concurrent I/O accesses from multiple clients to files. Numerous optimizations exist to improve the file system performance, such as data staging services [AWEK10], latent asynchrony I/O [WPBW09], and a log-structured interposition layer [BGGM09]. A comprehensive comparison between PVFS and distributed file system HDFS was presented in [TSPL11].

III. Decoupled Execution Paradigm

A. A Motivating Example

In scientific applications, data is commonly represented by a multi-dimensional array-based data model. For instance, the widely used Community Earth System Model (CESM) software package consists of four separate modules simultaneously simulating the earth's atmosphere, ocean, land surface and sea-ice, and each module uses the multi-dimensional arrays data model [CESM]. Figure 2 shows a 3-dimensional temperature data with longitude, latitude, and time dimensions. It is often needed to compute the moving average, median, lowest and highest temperature with specified conditions such as areas and periods of time. Such computed results will be further correlated with the computed results from other parameters, such as the humidity and wind speed, to predict weather conditions.

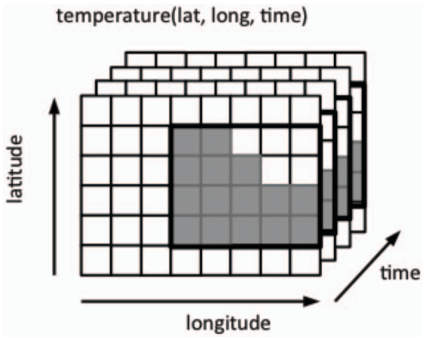

Figure 2. Processing 3-dimensional Temperature Data

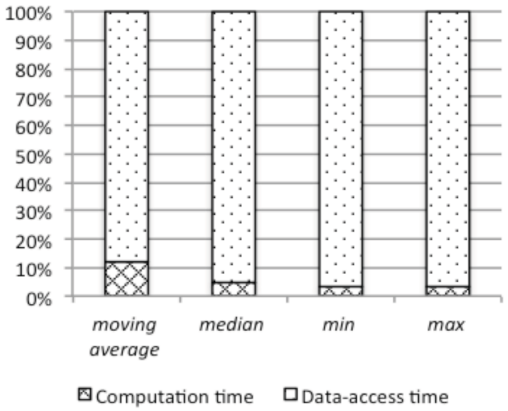

Figure 3. Comparison of Computation Time and Data-access Time

The current way of conducting such processing is to read the required data (e.g., a sub-array with the bold border, as shown in Figure 2) from storage servers to compute nodes, perform computations on desired data with specified conditions, such as those data shown in shaded area, and then write the output back to storage. For CESM, an experimental test shows that the data access and movement time for the calculation of the moving average, median, lowest and highest degrees can occupy 88.2%, 95.4%, 96.6%, and 96.6% of the total execution time on a cluster, where 128GB of data are retrieved to 272 nodes for processing (Figure 3).

CESM clearly has data retrieval and processing phases and computing and simulation phases, as many applications do. The basic idea of DEP is to handle these two phases differently on different nodes. DEP decouples the execution operations into computation-intensive operations and data-intensive operations. Computation-intensive operations are executed on massive compute nodes. Data-intensive operations are executed on dedicated data processing nodes.

B. Decoupled Execution Paradigm Design

The *decoupled execution paradigm (DEP)* consists of three components: *system architecture*, *programming model* and *runtime system*. The architecture view of DEP is shown in Figure 4.

1) System architecture. DEP decouples the nodes into *data-processing (data) nodes* and *compute nodes*. Data-processing nodes are further decoupled into *compute-side data nodes* and *storage-side data notes*. Compute-side data nodes are compute nodes that are dedicated for data processing. Storage-side data nodes are specially designed nodes that are connected to file servers with fast network. Compute-side data nodes reduce the size of computing generated data before sending it to storage nodes. Storage-side data nodes reduce the size of data retrieved from storage before sending it to compute-side data nodes.

Writes will go through compute-side data nodes, whereas reads will go through the storage-side data nodes. Data nodes can provide simple data forwarding without any data size reduction, but the idea behind data nodes is to let the data nodes conduct the decoupled data-intensive operations and optimizations to reduce the data size and movement.

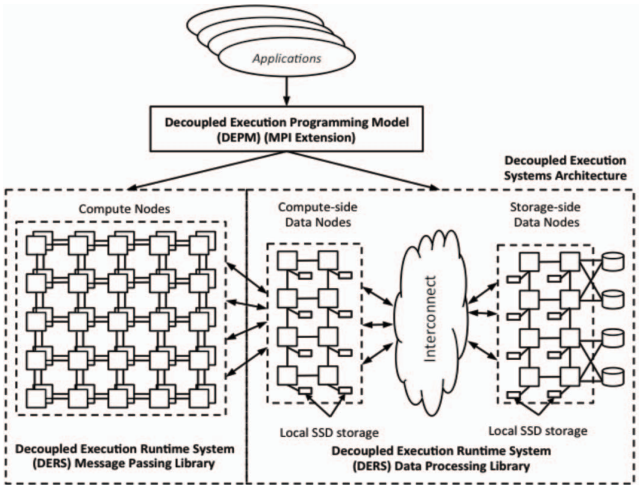

Figure 4. Decoupled Execution Paradigm (DEP) for Data-Intensive HEC

2) Programming model. What operations should be passed to the data nodes are determined by users and supported by the decoupled execution programming model (DEPM). The DEPM component is an MPI extension, allowing users to specify operations conducted on data nodes, instead of on compute nodes as the normal MPI library does. The purpose of the MPI extension and the DEPM component are similar to the netCDF Operators [ZeWa07] in some sense, allowing data-intensive operations to be decoupled and processed on data nodes, and the results being sent back to compute nodes for further processing. For instance, an *ncwa* operator in netCDF computes the weighted average on specified data and returns the result for further computations, reducing the unnecessary data movement. Different from netCDF Operators, however, the DEPM is extended and much more powerful. It allows *operations* to be decoupled not only *operators*, which essentially allows general piece of code to be executed on data nodes, beyond operators. In addition, the DEPM allows optimizations across operations, which is impossible in the netCDF Opertaors.

3) Runtime system. At runtime, the DEP relies on two libraries, *message passing library* and *data processing library*, to support computation-intensive operations and data-intensive operations respectively. The message passing library focuses on the memory abstraction of massively parallel processes and provides the runtime support for computation-intensive operations to be run on massive compute nodes. We leverage the existing MPI library for this purpose. The data processing library focuses on the I/O abstraction and provides runtime support for data-intensive operations to be run on data nodes. These two libraries are tightly coupled, and the message passing library manages the interaction between these two libraries as well. The runtime system can optimize user-defined data-intensive operations and other I/O optimization operations on data nodes as well.

The proposed decoupled execution paradigm changes the current execution paradigm by *balancing the computation and data-access capabilities*. This new paradigm separates computation-intensive operations and data-intensive operations and handles them concurrently and in a coordinated manner, but on different hardware and software environments for best performance.

C. Comparison of Execution Paradigms

The proposed DEP, in other words, reshapes the current execution paradigm of "retrieve - compute - store" cycles into "retrieve - reduce - compute -reduce - store" cycles as shown in Figure 5, where the "reduce" phases are designed to conduct data-intensive operations and reduce data size before moving data across the network. These retrieval, reduce, compute, and store phases can be pipelined to overlap the I/O, communication, and computation times. From one point of view, DEP is an enhanced version of MapReduce, where the "reduce" is not conducted by one node with its local storage, but a set of (data) nodes and the global storage, so that parallel computing features can be maintained. From another point of view, the data nodes are the data-access accelerators, to speed up the storage data-access delay and reduce data size before sending data across the network.

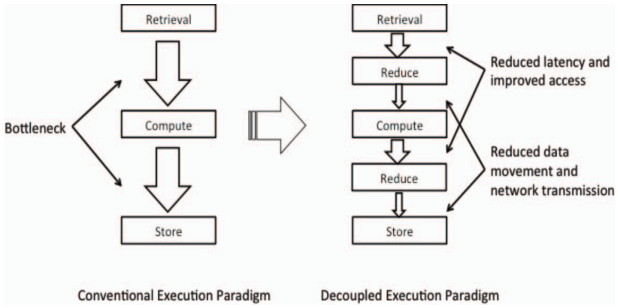

Figure 5. Comparison of Execution Paradigms

IV. Impementation of Decoupled Execution Paradigm

A. Systems Architectures

Data nodes perform the data-intensive operations and data-access optimizations with runtime library support. They sit close to the data source physically. The two kinds of data nodes, compute-side nodes and storage-side nodes, are different. The storage-side data nodes are predefined and static. The compute nodes and the compute-side data nodes can be either predefined or dynamically assigned. The predefined strategy configures the compute nodes and compute-side data nodes statically and in advance. For instance, a subset of nodes in a rack can be predefined as compute-side data nodes, and the

rest act as normal compute nodes. The selection of the data node should consider the physical location and network topology. For instance, if we assume to have 16 nodes on one card, 4 cards in one plane, and two planes in one rack, we can have one data node in each card, and 4 data nodes in one plane and 8 in one rack. The compute-side data nodes can dynamically join the compute nodes group and act as compute nodes as well to make the best use of resources. The dynamic configuration of nodes is instructed by users.

B. Programming Model

An ideal implementation for the decoupled execution paradigm would be automatically identifying those data-intensive operations and shipping them to data-processing nodes, while keeping computation-intensive operations on the compute nodes. This solution is challenging as it requires a precise understanding of the code and an automatic separation process. Instead, a more practical solution is to rely on programmers' hints and knowledge. We currently take an MPI extension approach and rely on programmer's knowledge to instruct the operations that are decoupled and to be executed on data nodes. The processed results are transferred via MPI communications as well. This approach is a manual approach, but we plan to build a semi-automatic decoupling tool to assist the application decomposition. This is a two-step process. The first step enables programmers to leverage the existing SIGIO tool [SIGIO] to identify these statements that cause intensive data accesses with profiling runs and trace-based analysis. The second step enables programmers to identify data-intensive operations and the essential computation related with these data-intensive operations that can be shipped to data nodes for processing. The implementation of this semi-automatic decoupling tool is expected to ease the needs from programmers' knowledge.

C. Runtime System

The current implementation of the runtime system is at an early stage. The implementation carries out the decoupled code specified by programmers on the data nodes and transmit the results between compute nodes and data nodes. The prototyping system is simple but focuses on verifying the idea and potential. An ideal implementation can leverage existing data-intensive processing library, such as a MapReduce library, but customize it for the DEP. The rationale is that these data-intensive processing library naturally fits the requirement of the runtime system on the data nodes (processing data intensively) in the DEP. In addition, we plan to incorporate and develop the filtering, caching, and prefetching components for the runtime system on data nodes. These components can build a staging area on data nodes, further reduces data transfer over network, and also removes redundant data movement.

V. Experimental Results

A. Experimental Platform

We have performed initial experimental tests on a 640-node Linux cluster, Hrothgar cluster [HPCC]. We dedicated a portion of nodes as data processing nodes to evaluate the DEP potential. The experimental tests varied the configuration and ratio of compute nodes and data nodes for evaluating different scenarios. Each node of the Hrothgar cluster is equipped with Intel(R) Xeon(R) 2.8GHz CPUs (12 cores per node) and 24GB memory. The cluster has a 600TB global parallel file system.

We performed the tests with two application cases. One is the kernel calculation of the CESM that computes the moving average of selected area of specified data as discussed in Section III. Another application is a geographic information system that predicts the rainfall accumulation for an area by processing the geospatial data from the GIS and the data collected from rain sensors [TWAC01, PostGIS]. This application calculates the water flow directions and analyzes the impact by processing the terrain and rain sensors data, as shown in Figure 6. It has two primary operations, *flow routing* and *flow accumulation* operations. The *flow routing* is first

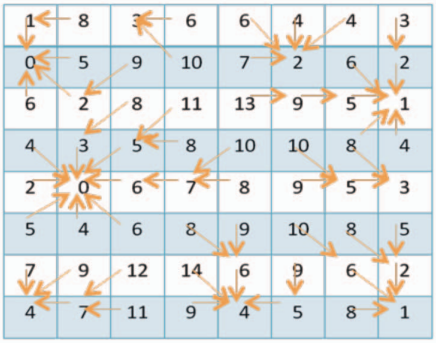

Figure 6. Flow Directions in a Grid of Terrains. Numbers represent the gradient of each terrain. Arrows represent the direction of water flow.

used to calculate the *flow direction* at every point of the terrain in order to model global flow of water. The terrain data is obtained in raster (grid) form: the coordinates of the data correspond to a uniform lattice, and elevations are given for each cell in the grid. It defines the neighbors of a grid cell *s* are the eight cells around *s*; and a neighbor who owns strictly lower elevation than *s* is called *downslope neighbor*; as well as the *gradient* of *s* towards one of its neighbors can be estimated as the ratio of the height difference of the cells and the horizontal distance between them. The *steepest downslope neighbor* of s is the *downslope neighbor* with the largest *gradient*. The *flow direction* of a cell is the directions in which water would flow if poured into that cell. With these directions information, flow accumulation algorithm computes to quantify the flow through each point by summing all the flow that passed through. The rainfall accumulation can thus be analyzed and predicted.

B. Results of the CESM Kernel Code

The first set of experiments that we have conducted is to evaluate the execution time results of the CESM kernel code with 12GB, 24GB, 48GB, and 96GB of data sets respectively on total 48 nodes. Figure 7 reports the results with 12 and 24 storage-side data nodes respectively, comparing the conventional execution model and the DEP model. The DEP model clearly outperforms the former. It can also be observed, with the growing data sizes, the performance gain of the DEP further increases. Figure 8 reports the results with the same data sets on 96 total nodes, with 24 and 48 storage-side data nodes respectively, and compares the DEP against the conventional execution model. The performance trend we can

observe is similar to that in previous tests. The DEP improves the apparition run time by more than three folds. On average, it achieved speedup of 229%.

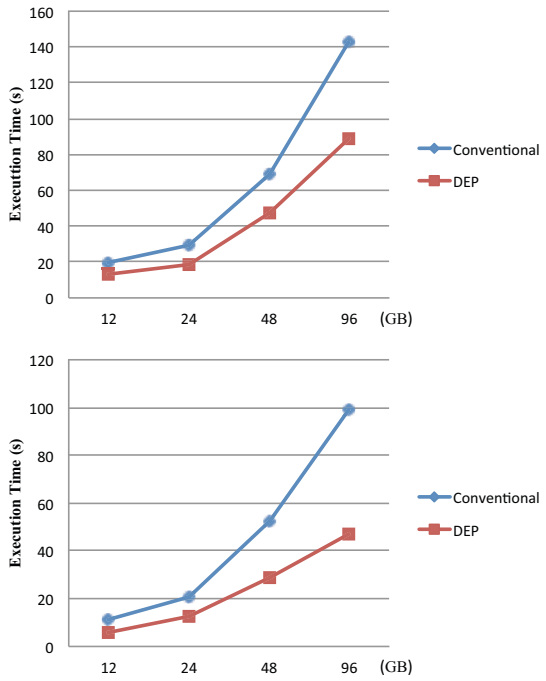

Figure 7. Execution Time of CESM Kernel Code with Different Data Sets on 48 Nodes, with 12 and 24 storage-side data nodes respectively.

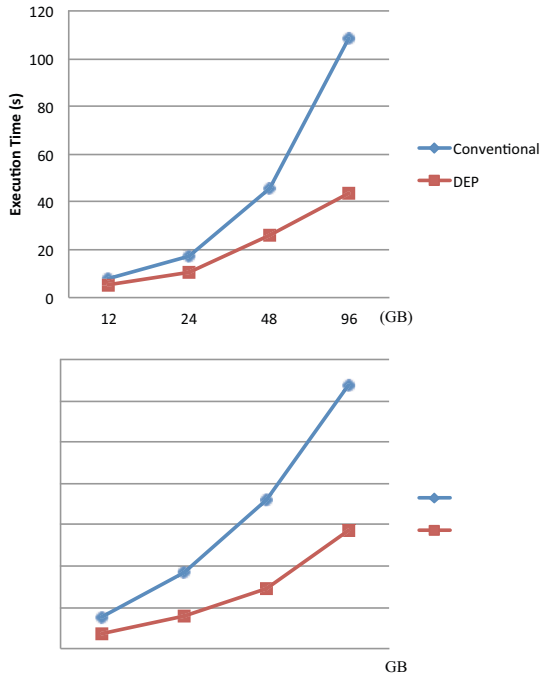

Figure 8. Execution Time of CESM Kernel Code with Different Data Sets on 96 Nodes, with 24 and 48 storage-side data nodes respectively.

We can also analyze the results from the effective bandwidth obtained for different scenarios tested. The effective bandwidth is calculated by the actual amount of data processed (the amount of data that is supposed to be moved) divided by the time taken. Figure 9 illustrates the results with different data sets tested on 96 nodes. It can be observed that the DEP significantly increases the effective bandwidth for data processing. The performance advantages primarily come from the decoupled operations and significantly reduced data movement. The DEP tends to be more scalable as well in terms of the amount of data processed, while the conventional execution paradigm suffers a decreasing performance trend. The further investigation reveals that the performance degradation of the conventional case is mainly caused by the contention from competing nodes, while the DEP achieved stable performance improvement.

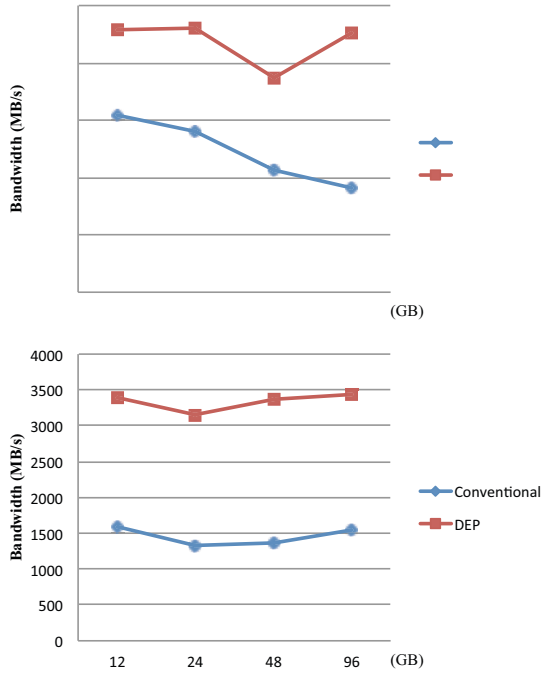

Figure 9. Effective Bandwidth of CESM Kernel Code with Different Data Sets on 96 Nodes.

C. *Results of the GIS Kernel Code*

We have also performed extensive tests with the GIS kernel code that computes the flow routing and accumulation.

Figures 10, 11, and 12 report the effective bandwidth results of the flow routing and accumulation code with different data sets on 24, 48, and 96 nodes respectively. Each figure reports the results with two cases, one case with one fourth of nodes as storage-side data nodes, and the other case with half of nodes as the storage-side data nodes. The results are mostly consistent across various runs under different scenarios.

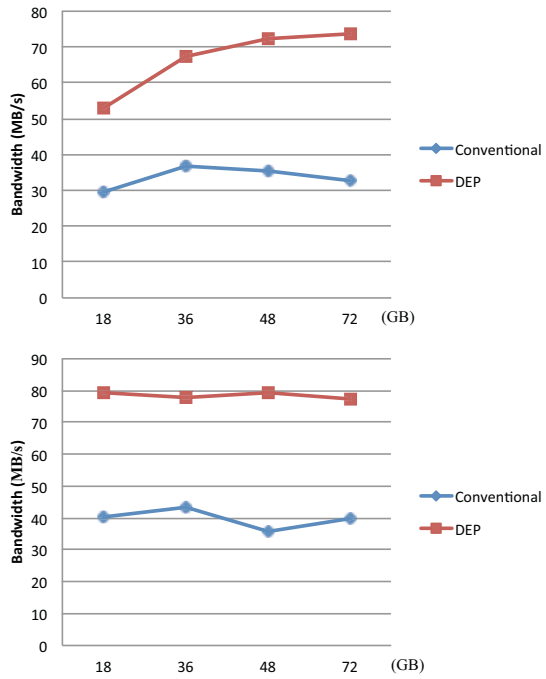

Figure 10. Effective Bandwidth of Flow Routing and Accumulation Code with Different Data Sets on 24 Nodes. Left: with 6 storage-side data nodes. Right: with 12 storage-side data nodes.

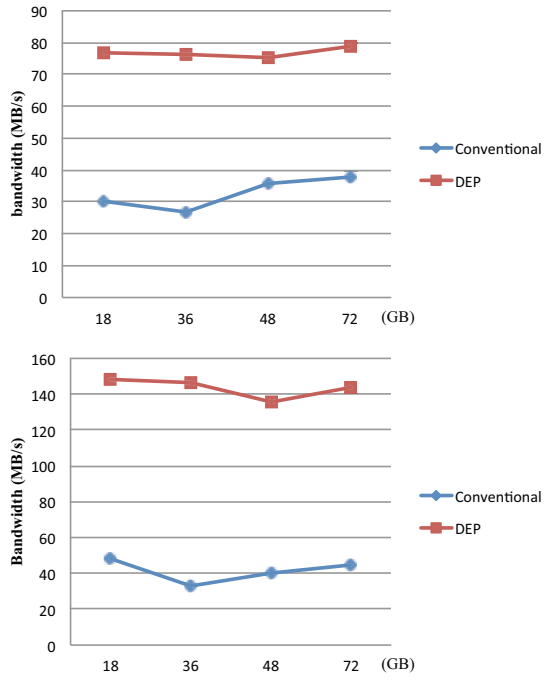

Figure 11. Effective Bandwidth of Flow Routing and Accumulation Code with Different Data Sets on 48 Nodes. Left: with 12 storage-side data nodes. Right: with 24 storage-side data nodes.

It can be observed that the DEP achieved clear better performance in all cases due to decoupling and reducing data movement. The DEP achieved up to six folds of speedup when compared with the conventional execution model, which is a promising result. Overall, the DEP achieved stable performance improvement and clearly outperformed the existing execution model. The results are encouraging.

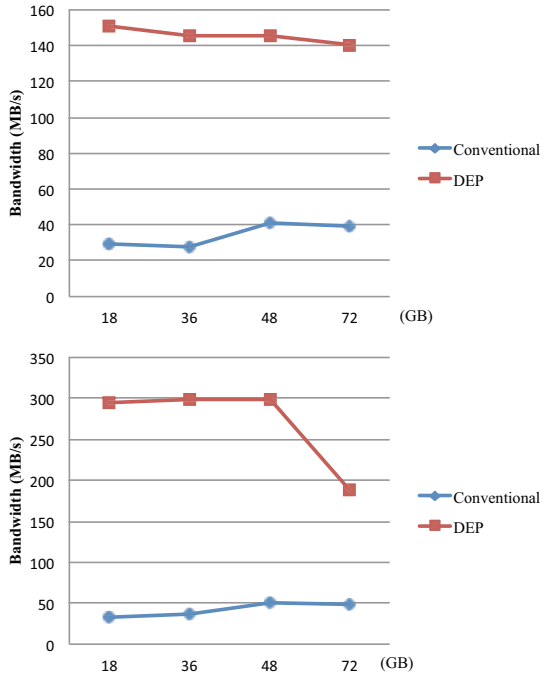

Figure 12. Effective Bandwidth of Flow Routing and Accumulation Code with Different Data Sets on 96 Nodes. Left: with 24 storage-side data nodes. Right: with 48 storage-side data nodes.

Figure 13 plots the results of various tests together for easy comparison across various tests. These tests have confirmed the clear benefits of the DEP.

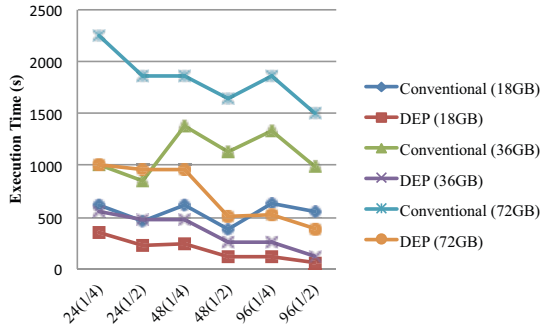

Figure 13. Comparison among Various Tests for the Flow Routing and Accumulation Code.

VI. CONCLUSION

With the tremendous advance in processor architectures and the computational capability, I/O has been widely recognized as the bottleneck in high-end computing for data-intensive applications. These data-intensive applications are critical for scientific discovery and innovations. However, the I/O bottleneck issue and massive amount of data movement for these applications can largely limit the productivity of data-intensive sciences.

In this study, we propose a decoupled execution paradigm (DEP) for data-intensive high-end computing. The DEP builds separate data-processing nodes and compute nodes, decomposes application operations into computation-intensive and data-intensive operations, and maps these decoupled operations onto compute nodes and data-processing nodes respectively. The data-processing nodes and compute nodes collectively provide a balanced system design and deliver the best performance for data-intensive applications. We have verified the idea with an initial prototype. The results are promising: for both climate model kernel code and the flow routing and accumulation code in the GIS, the prototype has shown significantly better results than the conventional execution paradigm due to decoupled operations and reduced data movement. While this study is an initial step of building a new execution paradigm for data-intensive HEC, the current results are encouraging. The current study confirms that the DEP has a great potential for data-intensive HEC. Given the growing importance of supporting data-intensive sciences, the DEP can have an impact.

VII. ACKNOWLEDGMENT

This research is sponsored in part by the National Science Foundation under grant CNS-1162540, CNS-1162488, and CNS-1161507. The authors acknowledge the High Performance Computing Center (HPCC) at Texas Tech University at Lubbock for providing HPC resources that have contributed to the research results reported within this paper. URL: http://www.hpcc.ttu.edu.

REFERENCES

[AWEK10] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan and F. Zheng. DataStager: Scalable Data Staging Services for Petascale Applications. Cluster Computing 13(3): 277-290, 2010.

[ACIK09] N. Ali, P. H. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. B. Ross, L. Ward and P. Sadayappan. Scalable I/O Forwarding Framework for High-performance Computing Systems. In Proc. of the 2009 IEEE Intl. Conf. on Cluster Computing, 2009.

[BGGM09] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte and M. Wingate. PLFS: A Checkpoint Filesystem for Parallel Applications. In Proc. of ACM/IEEE Supercomputing Conference, 2009.

[BICL09] J. G. Blas, F. Isaila, J. Carretero, R. Latham and R. Ross. Multiple-Level MPI File Write-Back and Prefetching for Blue Gene Systems. In Proc. of PVM/MPI, 2009.

[BCST08] S. Byna, Y. Chen, X.-H. Sun, R. Thakur and W. Gropp. Parallel I/O Prefetching Using MPI File Caching and I/O Signatures. In Proc. of the ACM/IEEE Supercomputing Conference (SC'08), 2008.

[CESM] http://www.cesm.ucar.edu/

[ChGi99] F. Chang and G. A. Gibson. Automatic I/O Hint Generation Through Speculative Execution. In Proc. of the 3rd Symposium on Operating Systems Design and Implementation (OSDI), 1999.

[ChKZ11] F. Chen, D. A. Koufaty and X. Zhang. Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems. ICS 2011, 2011.

[CSTR11] Y. Chen, X.-H. Sun, R. Thakur, P. C. Roth and W. Gropp. LACIO: A New Collective I/O Strategy for Parallel I/O Systems. In Proc. of IEEE International Parallel and Distributed Processing Symposium (IPDPS' 11), May, 2011.

[CBST08] Y. Chen, S. Byna, X.-H. Sun, R. Thakur, and W. Gropp. Hiding I/O Latency with Pre-execution Prefetching for Parallel Applications. Best paper award finalist, in Proc. of the ACM/IEEE SuperComputing Conference (SC'08), Nov. 2008.

[ChMa02] G. Chockler and D. Malkhi. Active Disk Paxos with infinitely many processes. In Proc. of the 21th annual symposium on Principles of distributed computing, pp. 78-87, 2002.

[CLGN09] A. Choudhary, W.-K. Liao, K. Gao, A.Nisar, R. Ross, R. Thakur, and R. Latham. Scalable I/O and Analytics. Journal of Physics: Conference Series, 180 (012048), 2009.

[CSTR11] Y. Chen, X.-H. Sun, R. Thakur, P. C. Roth, W. D. Gropp. LACIO: A New Collective I/O Strategy for Parallel I/O Systems. IPDPS 2011: 794-804, 2011.

[CBST08a] Y. Chen, S. Byna, X.-H. Sun, R. Thakur, W. Gropp. "Exploring Parallel I/O Concurrency with Speculative Prefetching," in Proc. 37th International Conference on Parallel Processing (ICPP'08), 2008.

[ChLC03] S. Chiu, W.-K. Liao and A. Choudhary. Design and Evaluation of Distributed Smart Disk Architecture for I/O-Intensive Workloads. In ICCS, 2003.

[DeGh04] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In Proc. of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04), pp. 137 - 150, December, 2004.

[DoXi11] X. Y. Dong and Y. Xie. AdaMS: Adaptive MLC/SLC Phase-change Memory Design for File Storage. ASP-DAC, 31-36, 2011.

[DBMA11] J. Dongarra, P. H. Beckman, et. al. The International Exascale Software Project Roadmap. IJHPCA 25(1): 3-60 (2011)

[ElSm06] T. E. Ghazawi and L. Smith. UPC: Unified Parallel C. ACM/IEEE conference on Supercomputing (SC'06), 2006.

[GrLT99] W. D. Gropp, E. Lusk and R. Thakur. Using MPI-2. MIT Press, 1999.

[GTC] Gyrokinetic Particle Simulations Gyrokinetic Toroidal Code (GTC) http://w3.pppl.gov/theory/proj_gksim.html

[HPCC] High Performance Computing Center, http://www.hpcc.ttu.edu

[IRYB08] K. Iskra, J. W. Romein, K. Yoshii and P. Beckman. ZOID: I/O Forwarding Infrastructure for Petascale

Architectures. In Proc. of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 153 -162, 2008.

[LCCC07] W.-K. Liao, A. Ching, K. Coloma, A. Choudhary and L. Ward. An Implementation and Evaluation of Client-Side File Caching for MPI-IO. In Proc. of IEEE International Parallel and Distributed Processing Symposium, 2007.

[LKSP08] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki and C. Jin. Flexible I/O and Integration for Scientific Codes Through the Adaptable I/O System (ADIOS). In Proc. of the 6th International Workshop on Challenges of Large Applications in Distributed Environments, 2008.

[MWLY02] X. S. Ma, M. Winslett, J. Lee and S.-k. Yu. Faster Collective Output through Active Buffering. IPDPS, 2002.

[NiHL94] J. Nieplocha, R. J. Harrison and R. J. Littlefield. Global arrays: a Portable "Shared-Memory" Programming Model for Distributed Memory Computers. ACM/IEEE Supercomputing Conference (SC'94), 1994.

[NiLo97] B. Nitzberg, V. Mary and Lo. Collective. Buffering: Improving Parallel I/O Performance. HPDC, 1997.

[RiGF98] E. Riedel, G. Gibson and C. Faloutsos. Active Storage For Large-Scale Data Mining and Multimedia. In Proc. of the 24rd Intl. Conference on Very Large Data Bases, 1998.

[RiGi97] E. Riedel and G. Gibson. Active Disks – Remote Execution for Network-Attached Storage Abstract. Carnegie Mellon Univ. Pittsburgh, 1997.

[RCDG06] W. W. Ro, S. P. Crago, A. M. Despain, J.-L. Gaudiot: Design and evaluation of a hierarchical decoupled architecture. The Journal of Supercomputing, 38(3): 237-259, 2006.

[OWRM06] R. Oldfield, L. Ward, R. Riesen, A. B. Maccabe, P. Widener and T. Kordenbrock. Lightweight I/O for Scientific Applications. In Proc. of IEEE International Conf. on Cluster Computing, 2006.

[PGGS95] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky and J. Zelenka. Informed Prefetching and Caching. In Proc. of the 15th ACM Symposium on Operating Systems Principles (SOSP'95), 1995.

[PostGIS] PostGIS: postgis.refractions.net

[RLUW09] R. Ross, R. Latham, M. Unangst and B. Welch. Paralell I/O in Practice. Tutorial in the ACM/IEEE Supercomputing Conference, 2009.

[ScHa02] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In 1st USENIX Conference on File and Storage Technologies, 2002.

[SMWB10] S. Sehrish, G. Mackey, J. Wang and J. Bent. MRAP: A Novel MapReduce-based Framework to Support HPC Analytics Applications with Access Patterns. In Proc. of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC), 2010.

[ShDM10] J. Shalf, S. S. Dosanjh and J. Morrison. Exascale Computing Technology Challenges. VECPAR, 1-25, 2010.

[SIGIO] http://www.cs.iit.edu/~scs/iosig/

[SLCR10] S. W. Son, S. Lang, P. Carns, R. Ross, R. Thakur, B. Ozisikyilmaz, P. Kumar, W.-K. Liao and A. Choudhary. Enabling Active Storage on Parallel I/O Software Stacks. In Proc. of the 26th IEEE Symp. on Massive Storage Systems & Technologies, 2010.

[SuBC07a] X.-H. Sun, S. Byna and Y. Chen. Server-based Data Push Architecture for Multi-processor Environments. Journal of Computer Science and Technology, Vol. 22, No. 5, 641 – 652, 2007.

[SuBC07b] X.-H. Sun, S. Byna and Y. Chen. Improving Data Access Performance with Server Push Architecture. In Proc. of the NSF Next Generation Software Program Workshop (with IPDPS'07), 2007.

[TSPL11] W. Tantisiriroj, S. W. Son, S. Patil, S. Lang, G. Gibson, R. B. Ross: On the duality of data-intensive file system design: reconciling HDFS and PVFS. In Proc. of ACM/IEEE Supercomputing Conference (SC'11), 2011.

[TRLG04] R. Thakur, R. Ross, E. Lusk and W. Gropp. Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation. Technical Memorandum ANL/MCS-TM-234, Mathematics and Computer Science Division, ANL, 2004.

[TWAC01] Laura Toma, Rajiv Wickremesinghe, Lars Arge, Jeffrey S. Chase, Jeffrey Scott Vitter,Patrick N. Halpin, Dean Urban: Flow Computation on Massive Grids. ACM-GIS 2001: 82-87

[TrRe04] N. Tran and D. A. Reed. Automatic ARIMA Time Series Modeling for Adaptive I/O Prefetching. IEEE Trans. Parallel Distrib. Syst. 15(4): 362-377 (2004).

[VSKT06] M. Vilayannur, A. Sivasubramaniam, M. T. Kandemir, R.Thakur and R. Ross. Discretionary Caching for I/O on Clusters. Cluster Computing 9(1): 29-44, 2006.

[WPBW09] P. M. Widener, M. Payne, P. G. Bridges, M. Wolf, H. Abbasi, S.McManus and K. Schwan. Exploiting Latent I/O Asynchrony in Petascale Science Applications. ICPP Workshops, 105-112, 2009.

[XMFL11] Y. Xie, K.-K. M. Reddy, D. Feng, D.D.E. Long, Y. Kang, Z. Niu and Z. Tan. Design and Evaluation of Oasis : An Active Storage Framework based on TIO OSD Standard. In 27th IEEE Symp. on Mass Storage Systems and Technologies (MSST), 2011.

[YuVe08] W. Yu and J. S. Vetter. ParColl: Partitioned Collective I/O on the Cray XT. ICPP, 562-569, 2008.

[ZhJD09] X. Zhang, S. Jiang, and K. Davis. Making Resonance a Common Case: A High-performance Implementation of Collective I/O on Parallel File Systems. IPDPS, 2009.

[ZADL10] F. Zheng, H. Abbasi, C. Docan, J. F. Lofstead, Q. Liu, S. Klasky, M. Parashar, N.Podhorszki, K. Schwan and M. Wolf. PreDatA - Preparatory Data Analytics on Peta-scale Machines. IPDPS, 2010

[ZeWa07] C. S. Zender and D. L. Wang. High performance distributed data reduction and analysis with the netCDF Operators (NCO). Presented to the 23rd AMS Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology, January 14–18, San Antonio, TX, January 14–18, 2007.