Semantics-based Distributed I/O with the ParaMEDIC Framework*

P. Balaji
Math. and Comp. Science
Argonne National Laboratory
balaji@mcs.anl.gov

W. Feng
Dept. of Computer Science
Virginia Tech
feng@cs.vt.edu

H. Lin
Dept. of Computer Science
North Carolina State University
hlin2@ncsu.edu

Abstract

Many large-scale applications simultaneously rely on multiple resources for efficient execution. For example, such applications may require both large compute and storage resources; however, very few supercomputing centers can provide large quantities of both. Thus, data generated at the compute site oftentimes has to be moved to a remote storage site for either storage or visualization and analysis. Clearly, this is not an efficient model, especially when the two sites are distributed over a wide-area network.

Thus, we present a framework called "ParaMEDIC: Parallel Metadata Environment for Distributed I/O and Computing" which uses application-specific semantic information to convert the generated data to orders-of-magnitude smaller metadata at the compute site, transfer the metadata to the storage site, and re-process the metadata at the storage site to regenerate the output. Specifically, ParaMEDIC trades a small amount of additional computation (in the form of data post-processing) for a potentially significant reduction in data that needs to be transferred in distributed environments.

1 *Introduction*

With the rapid growth in the scale and complexity of scientific applications over the past few decades, the requirements for compute, memory and storage resources are now greater than ever before. While system sizes have also grown, most researchers do *not* have local access to systems of the scale that they need for their applications. Hence, they remotely access such large systems to perform the required computations. Further, many of these applications require multiple resources simultaneously for efficient execution. For example, applications that perform large computations to generate massive amounts of output data are becoming increasingly common. While several large-scale supercomputers provide either the required compute power or storage resources, few provide both. Thus, data generated at a remote compute site oftentimes has to be moved to a local storage site for either storage or visualization and analysis. Clearly, this is not an efficient model, especially when the two sites are distributed over a wide-area network.

In this paper we present a novel framework called "ParaMEDIC: Parallel Metadata Environment for Distributed I/O and Computing" to effectively utilize both large-scale remote computational resources as well as local compute resources to perform efficient data movement in distributed environments. ParaMEDIC uses application-specific semantic information and converts the generated data from a given application to orders-of-magnitude smaller metadata at the compute site. It then transfers the metadata to the storage site and re-processes the metadata at the storage site to regenerate the output. Specifically, ParaMEDIC trades a small amount of additional computation (in the form of data post-processing) for a potentially significant reduction in data that needs to be transferred in distributed environments.

At a high level, ParaMEDIC is similar to standard compression algorithms. However, the term "compression" typically has a connotation that the data is dealt as a generic byte-stream. Because ParaMEDIC uses a more abstract application-specific representation of the data to achieve a **much** larger reduction in data size, we instead use the term "metadata transformation" in this case.

Together with a detailed description of the ParaMEDIC framework, this paper also evaluates ParaMEDIC on three distributed systems. The first system utilizes TeraGrid nodes at the University of Chicago and San Diego Supercomputing Center (30-Gbps network connection). The second system consists of a secure filesystem hosted between the Argonne National Laboratory and Virginia Tech over an Internet2 connection (1-Gbps connection). The third system is a local cluster connected with a dedicated 10-Gbps network that has the ability to vary both bandwidth and latency between any two nodes. We use this last system to emulate various forms of high-latency and high-bandwidth distributed computing infrastructures. Our experimental results demonstrate *order-of-magnitude improvements* in performance with ParaMEDIC compared to the traditional approach of blindly moving data in distributed environments.

2 *Sample Distributed Environments*

A wide variety of distributed environments exist today, ranging from high-latency, high-bandwidth *LambdaGrids*, to low-bandwidth environments connected over the Internet, to unsecure environments requiring data encryption before transmission. Here we present two sample distributed environments.

2.1 *NSF TeraGrid*

NSF TeraGrid is a distributed computing facility that combines leadership-class resources at 11 partner sites within the U.S. to form the world's largest distributed cyberinfrastructure for open scientific research. It includes 750+ teraflops of computing capability and 30+ petabytes of data storage, along with rapid access and retrieval over high-performance networks to form the world's largest distributed

*This work was funded in part by the Mathematical, Information, and Computational Sciences Division of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357 and the Dept. of Computer Science at Virginia Tech.

cyber-infrastructure for open scientific research.

The TeraGrid comprises several sites including the University of Chicago/Argonne National Laboratory (Illinois), San Diego Supercomputing Center (California), Purdue University (Indiana), Texas Advanced Computing Center (Texas), and others. The San Diego Supercomputing Center (SDSC) also doubles as a host for a global parallel filesystem (GPFS) that is visible and usable by all TeraGrid compute servers. All sites are connected using high-bandwidth optical links. However, the large physical distance between the sites forces the latency to be high (up to tens of milliseconds) as well.

Scientists use the computational power available at different locations to run computations and then write the final output to the globally shared filesystem to be viewed or post-processed at a later time. While such a system provides good computational capability for I/O-rich applications, the distributed filesystem can form a significant bottleneck.

2.2 *Argonne-VT Distributed System*

The Argonne-VT distributed system is a small-scale research infrastructure built to share application output data for post-processing and visualization. The system consists of 200 processors of shared compute resources, and about 200 gigabytes (GB) of main memory spread across the two sites. Apart from this, the Argonne site also provides 10 terabytes of storage resources to be utilized by the compute systems.

The storage resources available at Argonne are shared across the two sites using a distributed file-system that utilizes the shared Internet2 connectivity (1 Gbps). This network connectivity is *much* slower than the NSF TeraGrid infrastructure. Furthermore, though this network is mostly a dedicated infrastructure, it occasionally experiences large traffic bursts, causing further performance degradation in the network. Finally, because the connection between the two sites is over the regular wide-area network, it is considered to be unsecure, and thus, the data transmission might require encryption, which can add substantial overhead as well.

3 *ParaMEDIC Design Overview*

This section presents a detailed description of the ParaMEDIC framework (short for *Para*llel *M*etadata *E*nvironment for *D*istributed *I*/O and *C*omputation).

3.1 *The ParaMEDIC Framework*

ParaMEDIC provides a framework for *decoupling* computation and I/O in applications that rely on large quantities of both. Its architecture, shown in Figure 1), contains three major components that are abstracted to the applications through the ParaMEDIC API: (a) ParaMEDIC data tools, (b) communication services, and (c) application plugins. The ParaMEDIC data tools include efficient support for data-touching operations such as data encryption and data integrity that might be required in unsecure distributed environments. The communication services include data-movement mechanisms such as direct communication via TCP/IP. The semantics-based metadata creation for ParaMEDIC is done by the *application plugins*. Most application plugins are specific to each application, and thus rely on knowledge of application semantics. These plugins provide two primary functionalities:

1. Functionality to perform post-processing on the data generated by the application in order to create metadata. This metadata can either be *intermediate data* generated during the application execution or data that is separately extracted by processing the generated output.
2. Functionality to convert the metadata back to the final output. This conversion can be 100% accurate or lossy, depending on the application requirements. However, in this paper, we only deal with lossless data conversion.

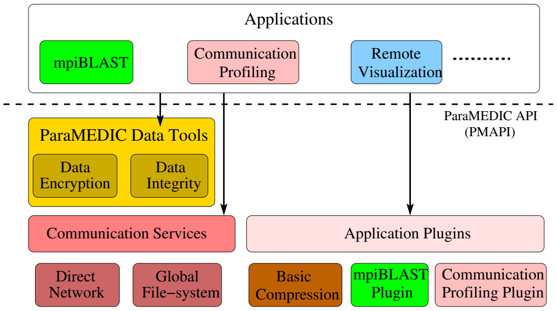

Figure 1: ParaMEDIC Architecture

Though the development of application-specific plugins is mostly straightforward, they require application knowledge to be developed. Consequently, the onus of writing these plugins is on the application writers, though the ParaMEDIC framework provides tools to ease this effort. As an intermediate solution, ParaMEDIC also provides a standard compression plugin that views the output data as a byte stream and performs non-application-specific compression on it. This standard plugin is similar to regular archiving tools that move and store data without interpreting it. While this compression plugin is straightforward to use and does not require any application knowledge, the performance benefits it can achieve are small compared to application-specific plugins. Thus, we do not discuss this approach further in this paper and focus only on application-specific plugins.

While tasks such as data encryption and data integrity typically add substantial overhead in communication, the encryption and integrity components in the ParaMEDIC framework are applied on compressed metadata which is much smaller than the actual data. Thus, within the context of ParaMEDIC, the overhead added by these components is minimal.

3.2 *Trading Computation with I/O Cost*

The amount of computation required in the ParaMEDIC framework is higher than what is required by the original application. For example, after the output is generated by the application processes, it has to be further processed to generate the metadata, sent to the destination nodes, and processed yet again to re-generate the final output. However, the I/O cost achieved can potentially be significantly reduced by using this framework. In other words, the ParaMEDIC framework aims at trading (a small amount of) additional computation for reduced I/O cost.

With respect to the additional computational cost incurred, the ParaMEDIC framework is quite generic with respect to the metadata processing that is required by the different processes. For many applications, it is possible to tune the amount of post-processing that is performed on the output data, with the general trend being—the more the post-processing computation, the better the reduction in the metadata size. That is, an application plugin can perform more processing of the output data to reduce the I/O cost at the cost of the additional processing overhead. Similarly, it can perform very little processing of the output data to save on the processing overhead, but at the cost of a larger amount of data that might need to be transferred. In the rest of this section, we formalize this capability of ParaMEDIC to trade additional computational cost for larger savings in I/O.

Consider a distributed environment connecting a large-scale supercomputer that is performing the computation and generating the data and a small-scale system where the data has to be stored or visualized. In this context, consider the following definitions:

B: Network Bandwidth in the above distributed environment.

T: Overall application execution time.

D: Total data generated by the application.

f(x): Time taken to convert output data to *x* units of metadata.

g(y): Time taken to convert *y* units of metadata to final output.

Based on the above definitions, the actual time (A) taken for the application to be executed and the data to be moved to the target system is given by the sum of four components: (a) the computation time of the application, (b) the time to convert the generated output to metadata, (c) time to transfer the metadata over the network and (d) time to post-process the metadata to generate back the required output. This is formalized by the following equation:

$$A = T + f(x) + \frac{x}{B} + g(x) \tag{1}$$

Conversely, the actual time taken by the application without using ParaMEDIC is the summation of the application computation time and the time to transfer the overall data:

$$A' = T + \frac{D}{B} \tag{2}$$

Clearly, ParaMEDIC is only beneficial when A (equation 1) is small than A' (equation 2). That is,

$$T + f(x) + \frac{x}{B} + g(x) < T + \frac{D}{B}$$

Simplifying the above equation, we get:

$$D - x > B \times (f(x) + g(x)) \tag{3}$$

Thus, as illustrated by Equation 3, ParaMEDIC would be beneficial only when the difference between the actual data size and the metadata size is larger than the network bandwidth times the amount of time taken for the post-processing (output-to-metadata and metadata-to-output conversion). Accordingly, ParaMEDIC is expected to significantly outperform the native implementations of the applications when either the network bandwidth of the distributed environment is low, or the post-processing cost is low. For example, even in distributed environments such as the NSF TeraGrid which has 30-Gbps high-speed network connectivity between the sites, ParaMEDIC can improve performance if the application data post-processing time is low enough.

Figure 2 illustrates the implications of Equation 3. Applications in *Quadrant 1* have a large difference in size between the actual data and the processed metadata, while requiring only a small amount of metadata post-processing. Such applications can take ideal benefit of the ParaMEDIC framework. Applications in *Quadrant 2* have a large difference in size between the actual data and the processed metadata, but they require a large amount of metadata post-processing as well. So, it is not clear whether such applications can fully benefit from ParaMEDIC. Applications in *Quadrant 3* almost never benefit from ParaMEDIC as they use a lot of post-processing, but do not reduce the size of the data output too much. Finally, for applications in *Quadrant 4*, again, it is not clear whether they would benefit from ParaMEDIC, since though they require only a small amount of post-processing, they do not reduce the amount of data significantly.

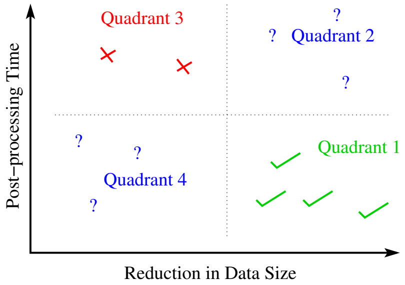

Figure 2: Application Output Data Patterns

3.3 *Managing Computational Resources*

In ParaMEDIC, the conversion of the actual output to metadata and the conversion of metadata back to the actual output does not come free. Computational resources have to be expended for such post-processing. Thus, if the total number of compute resources are fixed, managing how the resources are partitioned between the actual application computation and the metadata post-processing determines the tradeoff in the amount of time spent in computation versus the amount of time saved in I/O.

If too many compute resources are used for application processing and too little for post-processing, the time taken for post-processing either increases significantly or only superficial post-processing will be possible, resulting in larger amounts of metadata that needs to be communicated. On the other hand, if too few compute resources are used for the application processing and too many for the post-processing, the application execution time increases rapidly.

In general, since the post-processing resources are mainly restricted to the client system (i.e., the system where the final data has to be moved to), these resources are limited. For example, a remote large-scale cluster where the actual application is executed might have about 10000 processors, while the local small-scale cluster used by the scientist might only have about 1000 processors. Thus, in this case, it is ideal to maintain a 10:1 ratio between the number of compute resources allocated to the actual application and the number of compute resources allocated to the post-processing. Deviating from this ratio will result in degradation in performance. This forces ParaMEDIC to use algorithms that can generate metadata from the result and re-generate the result from the metadata with minimal processing requirements. That is, for most applications, the framework ensures that the post-processing cost required to process the metadata and generate the final output is significantly less than the actual computation needed by the application.

An evaluation describing the tradeoffs associated with managing the number of computation resources provided to application computation vs. post-processing is presented in §5.2.2.

4 *Case Studies in Scientific Computing*

As described in §3, different applications can *plug-in* to the ParaMEDIC framework by providing application-specific plugins using the ParaMEDIC API. In this section, we discuss two sample applications that can take advantage of this framework, namely mpiBLAST in §4.1 and the MPI Parallel Environment (MPE) profiling library in §4.2.

4.1 *mpiBLAST Sequence Search Application*

Here we present an overview of mpiBLAST and discuss how we integrate it with the ParaMEDIC framework.

4.1.1 *Application Overview*

mpiBLAST [4] parallelizes the NCBI BLAST toolkit [1], the de facto "gold standard" for sequential pairwise sequence-search that is ubiquitously used in biomedical research. The BLAST tool searches one or multiple input query sequences against a database of known nucleotide (DNA) or amino acid sequences. A similarity score is calculated for each close match based on a statistical model. The similarity of the comparison is measured by the match with the highest score. As a result, the sequences in the database that are most similar to the query sequence are reported, along with their matches scored beyond a certain threshold. Therefore, the BLAST process is essentially a top-k search, where k can be specified by the user, with a default value of 500.

The core of the mpiBLAST algorithm is based on *database segmentation*. Before the search, the raw sequence database is formatted, partitioned into fragments, and stored in a shared storage space. mpiBLAST then organizes parallel processes into one master and many workers. The master breaks down the search job in a Cartesian-product manner and maintains a list of unsearched tasks, each represented as a pair of a query sequence and a database fragment. Whenever a worker becomes idle, it asks the master for a unsearched task and copies the needed fragment to its local disk (if the fragment has not been cached locally) and performs a BLAST search on its assignment. Upon finishing a task, a worker reports its local results to the master for centralized result merging. Once the results of searching a query sequence against all database fragments have been collected, the master calls the standard NCBI BLAST output function to format and print out results of this query to the output file. By default, those results contain the top 500 database sequences with highest similarity to the query sequence along with their matches. The above process repeats until all tasks have been searched. With database segmentation, mpiBLAST can deliver super-linear speed-up when searching sequence databases larger than the memory of a single node. In recent developments, mpiBLAST has evolved to use a more scalable parallel approach that allows different queries to be concurrently searched as well [8].

4.1.2 *Integration with ParaMEDIC*

In a *cluster* environment, most of the mpiBLAST execution time is spent on the search itself, i.e., comparing input query sequences to the database fragments, because the search requires a full scan of the database fragment and the BLAST string-matching algorithm is computationally intensive (quadratic complexity in the worst case). In contrast, the cost of formatting and writing the results that are generated from the search is much less significant, especially when many advanced clusters are configured with high-performance parallel filesystems.

However, in *distributed* environments such as those presented in §2, the execution profile of mpiBLAST differs significantly from cluster environments because mpiBLAST output needs to be written over a wide-area network to a remote filesystem. Hence, the cost of writing the results can easily dominate the execution profile of mpiBLAST, and thus, become a severe performance bottleneck.

This is where "ParaMEDIC comes to the rescue" for mpiBLAST. By replacing the traditional global parallel filesystem over a wide-area network with the ParaMEDIC framework (as shown at the top of Figure 3), we can still support the basic functionality of a global parallel filesystem, e.g., transferring large volumes of data to a distant filesystem (if needed), but more importantly, we can also provide advanced functionality that trades a small amount of additional computation for a potentially significant reduction in data that needs to be transferred in distributed environments. For example, as we will see in §5, a mpiBLAST-specific instance of ParaMEDIC reduces the volume of data that needs to written across a wide-area network by *more than two orders of magnitude*.

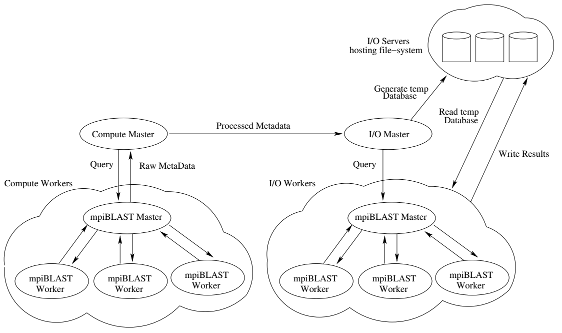

Figure 3: ParaMEDIC and mpiBLAST Integration

Specifically, Figure 3 depicts how mpiBLAST can be integrated with the ParaMEDIC framework. First, on the compute site (the left cloud in Figure 3), instead of having mpiBLAST collect and write all the result sequences and their matches of a query sequence, the mpiBLAST application plugin in ParaMEDIC, as shown in Figure 1, generates semantics-based metadata based on the mpiBLAST output at the compute site. ParaMEDIC then transfers this metadata to the I/O site (the right cloud in Figure 3), metadata that is orders of magnitude smaller than the actual data output that would have been transferred in a traditional global parallel filesystem. Next, at the I/O site, a small amount of additional computation must then be performed on the metadata in order to re-generate the actual output data. That is, a temporary (and much smaller) database that contains only the result sequences is created by extracting the corresponding sequence data from a local database replica. ParaMEDIC then re-runs mpiBLAST at the I/O site by taking as input the same query sequence and the temporary database to generate and write output to the local filesystem. (Note: The overhead in re-running mpiBLAST at the I/O site is quite small as the temporary database that is searched is substantially smaller with only 500 sequences in it by default, as opposed to the several millions of sequences in large DNA databases.

4.2 *MPE Communication Profiler*

The MPI Profiling Environment (MPE) is a suite of performance analysis tools consisting of profiling libraries, utility programs, graphical tools and checking libraries. MPE is a part of the MPICH2 distribution of the Message Passing Interface (MPI) communication standard, but can be used by any MPI implementation that provides the MPI profiling interface. The MPE profiler generates logs viewable by the integrated Jumpshot visualization tool. The datatype and collective verification library finds argument inconsistencies in MPI calls. The tracing library records all MPI calls and the animation and X-graphics libraries provide a real-time program animation of the trace. In this paper, we are primarily interested in the profiling capabilities of MPE. MPE performs postmortem performance analysis based on trace file generated during parallel program execution. It utilizes CLOG2 as a low-overhead logging format, a simple collection of single time-stamp events.

4.2.1 *Output Data Management in MPE*

Depending on the communication pattern, number of processors used and the length of execution, the output data generated by MPE can be enormous. For example, a large-scale application such as FLASH when executed on a 16384 processor cluster can easily generate 40GB of data every second. That is, for an hour-long run, the amount of data generated is close to 150 terabytes. Thus, the output data has to be managed carefully in order to avoid performance overheads.

MPE allocates a default 8MB memory buffer in each process during initialization time. During the run, MPE profiles its communication pattern and stores this information in this memory buffer. As the memory buffer fills up, the content of the buffer is written to the local storage. At the end of the run, all processes form a binary tree. Each process reads in its local trace data from the disk and performs a three-way merge of its own buffer with the buffers sent from its children. When a merged buffer is filled up, it will then send to its parent. At the root process, the overall merged buffers are written to local storage.

Once the profiled data is obtained, it can viewed through visualization tools such as Jumpshot. However, for this, the data has to be available locally for the scientists to visualize. Thus, if the application was executed and profiled at a remote location, the profile data has to be transferred (oftentimes over a distributed environment) to be processed and viewed.

4.2.2 *Integration with ParaMEDIC*

As described in §4.2.1, the amount of data generated by the MPE profiling tool is enormous. However, most scientific applications follow a very periodic pattern where they repeatedly perform the same task to refine a data set, or to process different portions of the data set. Accordingly, the communication pattern for such applications is periodic as well.

While most application communication patterns are periodic, the periodicity for each application is different. Thus, the main idea for integrating MPE with ParaMEDIC is two-fold:

1. Find the periodicity of the profiled data using a Fast Fourier Transform (FFT) of the data.
2. Use this periodicity information to convert the actual output into metadata that breaks the output into periods, and only store differences between the periods, rather than the entire communication profiling information.

Finding Output Periodicity with FFT: Though FFT is a popular approach for finding the periodicity of data sets, it can be quite compute intensive to evaluate, especially for large data sets. Specifically, FFT is an `O(N.log(N))` time operation, where `N` is the number of points in the data set. Further, several applications also have recursive periodicity. That is, within each recurring communication pattern, these applications might have a separate recurring pattern embedded. While identifying such patterns recursively can allow us to improve the I/O savings even further, this can significantly increase the amount of computation required as well. Thus, in our approach, we only consider the first level of periodicity.

Given the importance of FFT in scientific computing, several parallel implementations of FFT have been designed and developed. FFTW [7], Parallel 3D FFT (P3DFFT) [3] and BG/L 3D FFT are some popular implementations. Such implementations can reduce the time taken to compute the periodicity of the data set by utilizing multiple compute nodes in parallel. Also, since the periodicity detection has to only take place on the computational site, this process can be heavily parallelized using the large number of compute resources available on the site, thus further reducing the total time taken. Note that the post-processing required to convert the metadata back to the actual output does not require another FFT calculation, and can be done much more efficiently even with the lesser compute resources on the storage site. In our approach, we used the Parallel 3D FFT (P3DFFT) implementation of FFT.

Metadata Storing Only Differences between Periods: Once the periodicity of the data set is calculated, the repeating information in each period does not have to be stored and can be regenerated. In the metadata format for this application, we store the complete information for the first period, and store only the differences for the remaining periods.

For most applications, the differences between each period is quite minimal even when storing the timing information to up to two decimal places. For systems such as IBM BlueGene, where the system noise level is extremely low, the accuracy is even higher. Thus, the size of the differences is also very small, allowing for larger savings in the amount of data that needs to be communicated.

Our experimental results demonstrate about 3-5X improvement in performance for the MPE profiling library by using ParaMEDIC. However, due to space constraints, we do not present detailed performance results for this library in this paper and restrict ourselves to detailed performance results with the mpiBLAST application.

5 *Experimental Results*

This section presents a performance evaluation of mpiBLAST in its native form, as compared to ParaMEDIC-enhanced mpiBLAST (hereafter referred to as simply ParaMEDIC). All experiments were performed with the Nucleotide (NT) database downloaded from the NCBI website. NT is a nucleotide sequence database that contains the GenBank, EMB L, D, and PDB sequences. At the time when our experiments were performed, it contained over 5 million sequences with a total raw size of about 20GB. All queries were synthesized by randomly sampling sequences from NT itself.

5.1 *Experimental Testbeds*

Testbed 1 (Local Cluster): This testbed consists of 24 dual-2.8GHz-Opteron-processor dual-core nodes. Each processor has 2MB of L2-cache, and each node has 4GB of 667MHz DDR2 SDRAM and four SATA disks using a software RAID0. The nodes were connected with NetEffect NE010 10-Gigabit Ethernet adapters. We used NetEm to emulate the various distributed computing infrastructures. This allowed us to emulate high-latency, high-bandwidth distributed computing environments by separating the nodes in the system into two sub-clusters, where communication within the sub-cluster is fast but between sub-clusters is slow.

Testbed 2 (Argonne-VT Distributed System): This testbed consists of two clusters (one at Argonne and one at VT) connected over Internet2. The Argonne cluster is the one described in *Testbed 1*, while the VT cluster consists of a 24-node Orion Multisystems DT-12 system that contains 12 individual x86 compute nodes in a 24” x 18” x 4” (or one cubic foot) pizza-box enclosure. Each compute node contains a Transmeta Efficeon processor, its own memory, and Gigabit Ethernet interface. The nodes share a power supply, cooling system, and external 10-Gigabit Ethernet network connection.

Testbed 3 (NSF TeraGrid): This testbed consists of a subset of the TeraGrid, using nodes at U. Chicago and SDSC. The U. Chicago site consists of two sets of nodes. The first set has 96 2.4GHz Intel Xeon 32-bit dual-processor systems, each with 3GB memory and 512KB L2 cache, while the second set has 64 1.5GHz Intel Itanium II 64-bit dual-processor systems, each with 4GB memory. The SDSC site has 64 1.5GHz Intel Itanium II 64-bit dual-processor systems, each with 4GB memory. The two sites are connected with a 30-Gbps high-bandwidth network, and the end-to-end delay between the two sites is approximately 10ms.

5.2 *Local Cluster Evaluation*

Here we compare ParaMEDIC to native mpiBLAST on a local cluster, which emulates various distributed infrastructures.

5.2.1 *Impact of High-Latency, High-Bandwidth Networks*

We analyze the impact of distributed environments connected with high-latency, high-bandwidth networks on the performance of ParaMEDIC and basic mpiBLAST. For this experiment, we divide the local cluster into two logical sub-clusters. While all the nodes are connected with a 10Gbps network, we artificially delay the communication between nodes belonging to different sub-clusters. The performance of the application for different network delays is measured (the amount of delay is fixed within a run and is illustrated on the x-axis). The socket buffer sizes are set to be equal to the bandwidth-delay product of the network so as to maximize the performance that the network subsystem can provide. Four nodes (each node with 4 SATA disks connected with a software RAID0) in the second sub-cluster host a PVFS2 filesystem which is visible to all nodes in both the sub-clusters.

For the evaluation, 80 processors are used for performing the computation. For mpiBLAST, all processors were used for performing the actual application computation. However, for ParaMEDIC, to keep the overall computational resources constant, 76 processors hosted on the first sub-cluster were used for performing the actual computation, while 4 processors on the second sub-cluster were used for the post-processing.

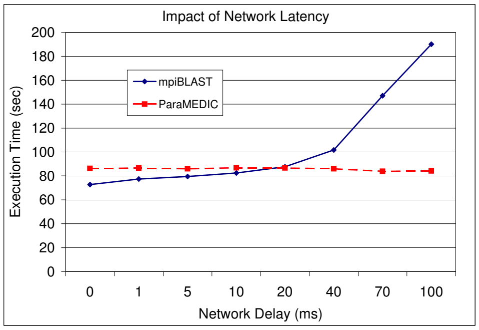

Figure 4: Impact of High Latency Networks

As shown in Figure 4, when the network delay between the two sub-clusters is low, mpiBLAST outperforms ParaMEDIC. This is expected since ParaMEDIC requires additional computation for converting the search results to metadata and converting the metadata back to the final output. However, as the network delay increases, ParaMEDIC outperforms mpiBLAST. In fact, for a network delay of 100ms, ParaMEDIC outperforms mpiBLAST by a factor of 2.26. This improvement is attributed to two factors. First, high network latency causes degradation in the filesystem communication and synchronization operations required when data needs to be written or read from the server. Second, the total amount of data written in mpiBLAST over the I/O subsystem is much higher as compared to ParaMEDIC, since ParaMEDIC only writes metadata which is significantly smaller than the final results to the filesystem.

To further understand these results, we show the performance breakdown of the time taken by mpiBLAST and ParaMEDIC in Figure 5. As shown in Figure 5(a), for mpiBLAST, as the network delay increases, the I/O time increases very quickly. Thus, though the computation time does not change much, the overall execution time suffers. On the other hand, for ParaMEDIC (Figure 5(b)), the computation time, the I/O time, and the post-processing time required to handle the metadata are nearly constant for all values of network delays. This is expected since the only component in ParaMEDIC that would be affected by the network latency is the post-processing, since it requires moving the metadata from the compute workers to the I/O workers. And because the metadata amount is very small (few KB), this time typically does not make any difference to either the post-processing time or the overall execution time of the application.

5.2.2 *Trading Computation to I/O*

We analyze the performance of mpiBLAST and ParaMEDIC by varying the ratio of computational resources allocated to the actual application processing vs. the resources allocated for post-processing. For all experiments in this section, for ParaMEDIC, we allocate four processes for performing the post-processing. The number of processes allocated for the actual application computation is varied from 16 to 80. That is, the ratio of resources allocated for actual application computation to post-processing is varied from 4:1 to 20:1. For mpiBLAST, on the other hand, all of the processes are allocated for the actual application computation. That is, the native mpiBLAST implementation gets four extra processes for application computation as compared to ParaMEDIC.

The ratio of resources allocated to application and post-processing essentially determines the trade-off in computation time to I/O time. Specifically, a large ratio means that more resources are given for application processing, thus the resources given for metadata generation and management is minimal. This implies that the application execution time will be lesser, while the amount of data that needs to be moved over the distributed environment will be large. Similarly, a small ratio means that lesser resources are given for application processing, thus the resources given for metadata generation and management is high.

Figure 6 shows the performance of the two schemes as the ratio of resources alloted to application and post-processing is varied. As shown in the figure, when this ratio is low, mpiBLAST outperforms ParaMEDIC. However, as the ratio increases the performance of mpiBLAST degrades faster than ParaMEDIC, and it is eventually outperformed by ParaMEDIC.

This behavior is related to the available compute resources for execution. Specifically, when the total number of compute resources is *N*, ParaMEDIC uses (N-4) of them for application computation and 4 processes for metadata post-processing. Thus, when N is very large, the increase in computation time caused by using resources (approximately N / (N - 4)) is not very high. However, when N is small, the increase in computation time can be substantial. For example, when N is 8 processes, ParaMEDIC uses only 4 processes for the application processing while mpiBLAST uses 8. Thus, the computation time taken by ParaMEDIC would be nearly twice that of mpiBLAST. This overshadows any benefit in the I/O time ParaMEDIC can bring about, causing it to deliver worse performance than mpiBLAST. In summary, ParaMEDIC is most effective only when the number of resources used for application processing are sufficiently large as compared to the number of resources used for post-processing.

5.2.3 *Impact of Output Data Size*

In this section, we vary two parameters that affect the output data size: (i) number of input query sequences provided by the user and (ii) number of output query sequences requested by the user, and study their impact on the performance of mpi-

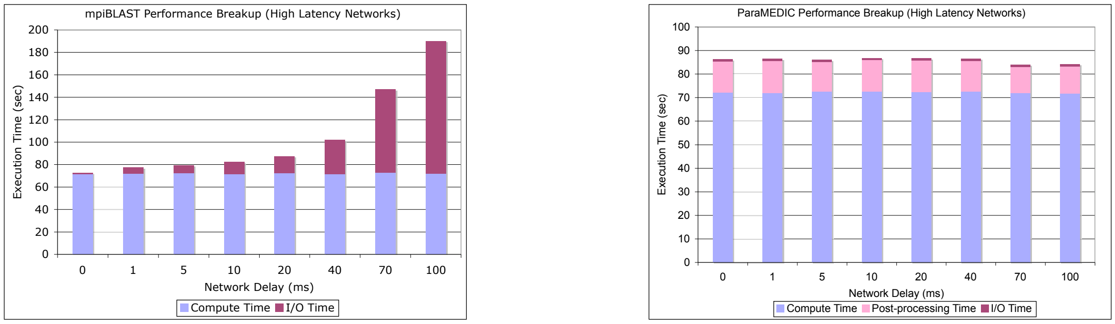

Figure 5: Breakup of Performance with Network Delay: (i) mpiBLAST and (ii) ParaMEDIC

BLAST and ParaMEDIC. Varying the number of sequences in the input query increases the search time for both mpiBLAST and ParaMEDIC. However, it is also expected to impact the post-processing time for ParaMEDIC. Thus, increasing the query size is expected to affect the computation time of ParaMEDIC more than that of mpiBLAST. At the same time, an increase in the query size also typically results in more output. This, on the other hand, can potentially impact mpiBLAST more than ParaMEDIC. Figure 7(a) shows the performance of the two schemes with increasing number of input query sequences (depicted by input query size). We see that while the increase in the input query size increases the execution time of ParaMEDIC, it has a more drastic effect on mpiBLAST. Thus, as the query size increases, we notice that the performance difference between the two schemes increases, with ParaMEDIC outperforming mpiBLAST by about 66% for a 100KB query file size.

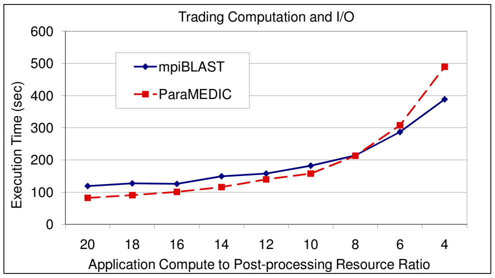

Figure 6: Varying the Number of Worker Processes

Figure 7(b) shows the impact of increasing the number of requested output result sequences. Increasing the number of output result sequences does not increase the computation too much, while it can increase the amount of I/O. Thus, because the I/O cost for ParaMEDIC is very low, increasing the number of output result sequences does not vary its performance too much. On the other hand, since the I/O cost for mpiBLAST is very high, increasing the number of output result sequences significantly affects its performance.

5.2.4 *Impact of Encrypted Filesystems*

For distributed filesystems that span unsecure network connections (such as the Internet), using data encryption to protect transmitted data is a common occurrence in several environments such as government national laboratories and other secure facilities such as those demonstrated in §5.3.

Figure 8 shows the impact of such data encryption on the performance of the two schemes. As shown in the figure, the performance of the two schemes is similar to the case where there is no file encryption (except that the performance of mpiBLAST degrades faster). This is attributed to the data encryption overhead. That is, since all the data that is being transmitted has to be encrypted and the amount of data transmitted by mpiBLAST over the unsecure network is significantly larger than ParaMEDIC, encryption affects mpiBLAST more significantly as compared to ParaMEDIC.

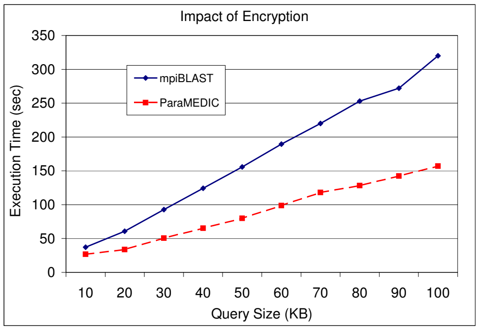

Figure 8: Impacted of Encrypted Filesystems

5.3 *Distributed Setup from Argonne and VT*

Here we evaluate mpiBLAST and ParaMEDIC on a distributed system between Argonne National Laboratory and Virginia Tech connected over Internet2. Since the network connecting the two clusters is *not* secure, data encryption is used to protect the data transmitted over this network.

As shown in Figure 9, ParaMEDIC significantly outperforms mpiBLAST in this environment. Further, as the query size increases, the performance difference between the two schemes increases. For a query size of 100KB, we observe more than a *25-fold improvement* in performance for ParaMEDIC as compared to mpiBLAST. This difference is attributed to multiple aspects. First, given that the network connection between the two sites is shared by other users, the effective network per-

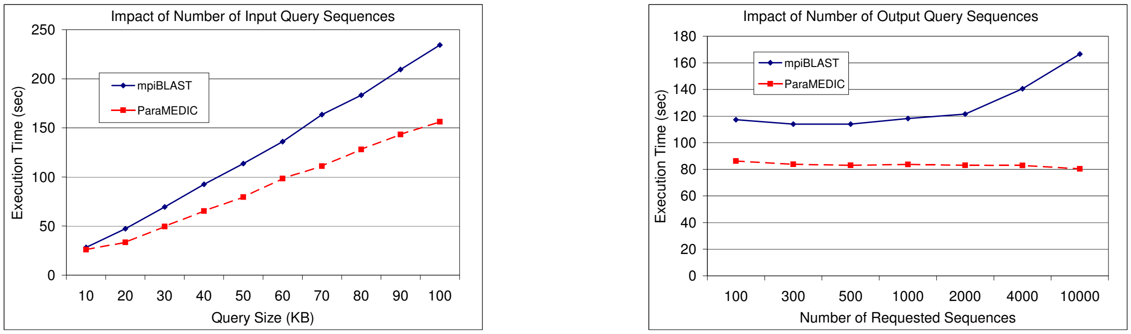

Figure 7: Varying the Number of Requested Sequences: (i) Input Query Sequences and (ii) Output Result Sequences

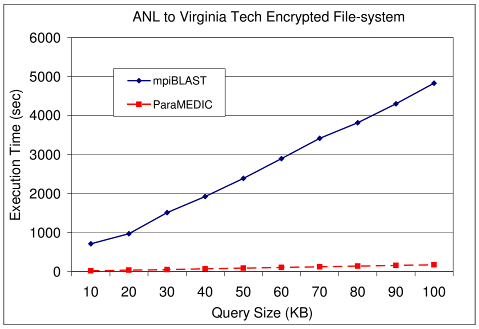

Figure 9: Argonne to Virginia Tech Encrypted Filesystem

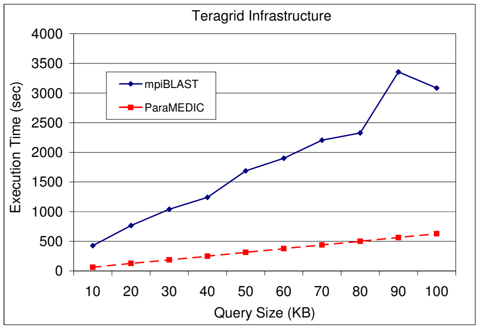

Figure 10: NSF TeraGrid using U. Chicago and SDSC

formance achievable is usually much lower than within the cluster. Thus, with mpiBLAST transferring the entire output result over this network, its performance would be heavily impacted by the network performance. Second, since data communicated is encrypted, mpiBLAST also has to pay the penalty for such encryption. Though ParaMEDIC also pays such data encryption penalty, the amount of data it transfers is significantly lesser, and hence the penalty is lesser as well. Third, the distance between the two sites causes the communication latency to be high. Thus, file-system communication and synchronization messages tend to take a long time to be exchanged resulting in further loss of performance.

5.4 *TeraGrid Infrastructure*

The TeraGrid infrastructure represents a widely used real environment for several compute- and I/O-intensive applications. As described in §2, a GPFS-based distributed filesystem is hosted at San Diego Supercomputing Center (SDSC), which can be accessed from all facilities, and forms a part of the TeraGrid facility. For the experiments in this section, we utilized the nodes at the University of Chicago and SDSC.

In this experiment, both mpiBLAST and ParaMEDIC perform their application computation on the University of Chicago nodes. However, mpiBLAST directly writes the output data to the global GPFS file-system. ParaMEDIC, on the other hand, converts the output data to metadata, transfers the metadata to SDSC, and re-converts the metadata to the final output at SDSC.

Figure 10 shows the performance of mpiBLAST and ParaMEDIC on TeraGrid. While the final output is written to the same global filesystem in both cases, mpiBLAST suffers because the application processing nodes at University of Chicago are performing the I/O for the output results. Since these nodes reside on a remote cluster as compared to the physical filesystem, their I/O performance is limited resulting in an overall degradation in execution time. For ParaMEDIC, on the other hand, since the post-processing nodes are performing the I/O for the output results, the amount of time taken is significantly smaller. For a query file size of 100KB, ParaMEDIC outperforms mpiBLAST by five-fold.

Figure 11 shows the performance breakdown of the two schemes. As the query size increases, the computation time for both mpiBLAST as well as ParaMEDIC increases. However, for mpiBLAST, the I/O time also increases very quickly. On the other hand, for ParaMEDIC, there is practically no difference in the I/O time with increasing query sizes. That is, ParaMEDIC is only minimally impacted by the limited I/O of the subsystem and it efficiently distributes its computational resources across the system to achieve high performance.

6 *Related Work*

Providing efficient remote I/O for scientific applications has been an ongoing subject of research. RIO [6] introduced a proof-of-concept library that allowed application to access remote files through ROMIO [12]. However, it relied on a legacy communication protocol and required setting up extra

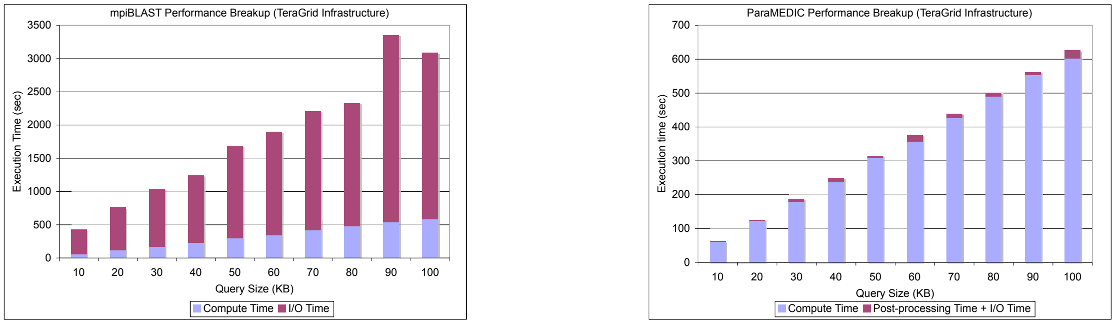

Figure 11: TeraGrid Infrastructure Performance Breakup: (i) mpiBLAST and (ii) ParaMEDIC

nodes dedicated for message processing. Those limitations were addressed in a more recent study, RFS [9], that adopted the active buffering technique to reduce the visible remote write cost, thus optimizing overlap between application I/O and computation. Other approaches of translating remote I/O requests into operations of general data transferring protocols such as Grid FTP [2] and Logistic Network [10] have also been investigated. ParaMEDIC, on the other hand, focuses on aggressively reducing the amount of I/O data that needs to be shipped across the wide area network with efficient utilization of application semantic knowledge.

Example frameworks of decoupling computation and I/O include MapReduce and TCP Linda. MapReduce is a programming model and an associated implementation for processing and generating large datasets [5]. TCP Linda is a virtual shared memory system whereby parallel processes execute simultaneously and exchange data be generating, reading, and consuming data objects [11]. Both frameworks decouple the different phases of an algorithm and transfer intermediate objects to the appropriate processes. In ParaMEDIC, these intermediate objects are the metadata. That MapReduce has been shown to achieve high performance when working with large datasets in [5] is indicative that ParaMEDIC, a similar model, can substantially improve performance as well.

7 *Concluding Remarks*

In this paper, we presented a novel framework called “ParaMEDIC: Parallel Metadata Environment for Distributed I/O and Computing” which uses application-specific semantic information to convert the generated data to orders-of-magnitude smaller metadata at the compute site, transfer the metadata to the storage site, and re-process the metadata at the storage site to regenerate the output. In other words, ParaMEDIC trades a small amount of additional computation (in the form of data post-processing) for a potentially significant reduction in data that needs to be transferred in distributed environments. We presented the detailed design of the framework and presented experimental evaluations on different experimental as well as real distributed systems. Our results show an *order-of-magnitude* improvement in performance with ParaMEDIC in some cases.

Acknowledgment

We wish to recognize and thank Jeremy S. Archuleta for his technical support on this project.

References

[1] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSIBLAST: A New Generation of Protein Database Search Programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[2] T. Baer and P. Wyckoff. A parallel i/o mechanism for distributed systems. In *Cluster*, 2004.

[3] San Diego Supercomputing Center. Parallel 3D FFT Library. http://www.sdsc.edu/us/% linebreak[0]resources/p3dfft.php.

[4] A. Darling, L. Carey, and W. Feng. The Design, Implementation, and Evaluation of mpiBLAST. In *International Conference on Linux Clusters: The HPC Revolution 2003*, 2003.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[6] I. Foster, D. Kohr, R. Krishnaiyer, and J. Mogill. Remote I/O: Fast access to distant storage. In *Proceedings of the Fifth Workshop on I/O in Parallel and Distributed Systems*, 1997.

[7] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[8] M. Gardner, W. Feng, J. Archuleta, H. Lin, and X. Ma. Parallel Genomic Sequence-Searching on an Ad-Hoc Grid: Experiences, Lessons Learned, and Implications. In *SC*, 2006.

[9] J. Lee, X. Ma, R. Ross, R. Thakur, and M. Winslett. RFS: Efficient and flexible remote file access for MPI-IO. In *Cluster*, 2004.

[10] J. Lee, R. Ross, S. Atchley, M. Beck, and R. Thakur. MPI-IO/L: efficient remote i/o for mpi-io via logistical networking. In *IPDPS*, 2006.

[11] TCP Linda. `http://www.lindaspaces.com/products/linda_overview.html`.

[12] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective I/O in ROMIO. In *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, February 1999.