Distributed I/O with ParaMEDIC: Experiences with a Worldwide Supercomputer*

P. Balaji[†] W. Feng[‡] H. Lin[§] J. Archuleta[‡] S. Matsuoka[¶] A. Warren[£ ‡]
J. Setubal[£‡] E. Lusk[†] R. Thakur[†] I. Foster[†] D. S. Katz[ò] S. Jha[ò] K. Shinpaugh[‡]
S. Coghlan[†] D. Reed[õ]

[†]Mathematics and Computer Science, Argonne National Laboratory
{balaji, lusk, thakur, foster, smc}@mcs.anl.gov

[‡]Dept. of Computer Science, Virginia Tech
{feng@cs., jsarch@cs., kashin@}vt.edu

[§]Dept. of Computer Science, North Carolina State University
hlin2@ncsu.edu

[£]Virginia Bioinformatics Institute, Virginia Tech
{anwarren@, setubal@vbi.}vt.edu

[¶]Dept. of Mathematical and Computing Science, Tokyo Institute of Technology
matsu@is.titech.ac.jp

[ò]Center for Computation & Technology, Louisiana State University
{dsk, sjha}@cct.lsu.edu

[õ]Scalable Computing and Multicore Division, Microsoft Research
daniel.reed@microsoft.com

Abstract

Achieving high performance for distributed I/O on a wide-area network continues to be an elusive holy grail. Despite enhancements in network hardware as well as software stacks, achieving high-performance remains a challenge. In this paper, our worldwide team took a completely new and non-traditional approach to distributed I/O, called *ParaMEDIC: Parallel Metadata Environment for Distributed I/O and Computing*, by utilizing application-specific *transformation* of data to orders-of-magnitude smaller meta-data before performing the actual I/O. Specifically, this paper details our experiences in deploying a large-scale system to facilitate the discovery of missing genes and constructing a genome similarity tree by encapsulating the mpiBLAST sequence-search algorithm into ParaMEDIC. The overall project involved nine different computational sites spread across the U.S. generating more than a petabyte of data, that was "teleported" to a large-scale facility in Tokyo for storage.

Keywords: distributed I/O, bioinformatics, BLAST, grid computing, cluster computing.

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. With the onset of petascale and exascale computing, issues related to managing such grand-scale resources, particularly related to data I/O, need to be carefully studied. For example, applications including genomic sequence search and the emergent field of metagenomics, large-scale data mining, data visual analytics and communication profiling on ultra-scale parallel computing platforms, generate massive amounts of data that needs to be managed for later processing or archival.

To add to the complexity of this problem is the issue of resource locality. While system sizes have certainly grown over the past few years, most researchers do not have local access to systems of the scale required by their applications. Therefore, researchers access such large systems remotely to perform the required computations and move the generated data to their local systems after the computation is complete. Similarly, many applications tend to require multiple resources simultaneously for efficient execution. For example, applications that perform large computations and generate massive amounts of output data are becoming increasingly common. While several large-scale supercomputers provide either the required compute power or the storage resources, very few provide both. Thus, data generated at one site often has to be moved to a different site for storage and/or analysis.

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

To alleviate issues related to moving such massive data across sites, there has been a lot of monetary and intellectual investment in high-speed distributed network connectivity [2, 5, 33]. However, the utility of these investments is limited in the light of three primary observations: (i) in spite of the effort put into high-speed distributed networks, such infrastructure is scarce and does not provide end-to-end connectivity to a very high percentage of the scientific community, (ii) the amount of data generated by many applications is so large that even at 100% network efficiency, the I/O time for these applications can significantly dominate their overall execution time and (iii) based on recent trends and published results, existing distributed I/O mechanisms have not been able to achieve a very high utilization for "real data" on high-speed distributed networks, particularly for single stream data transfers [3, 36].

To resolve such issues on a *global* scale, we proposed a new non-traditional approach for distributed I/O known as *ParaMEDIC* (Parallel Metadata Environment for Distributed I/O and Computing) [11, 12]. ParaMEDIC uses application-specific semantic information to process the data generated by treating it as a collection of high-level abstract objects, rather than as a generic byte-stream. It uses such information to transform the data into orders-of-magnitude smaller metadata before transporting it over the distributed environment and regenerating it at the target site. All data transformation, movement, and regeneration is done while the application is executing, giving the illusion of an ultra-fast teleportation device for large-scale data over 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. Since ParaMEDIC uses a more abstract application-specific representation of the data to achieve a **much** larger reduction in the data size, we use the terminology of "metadata transformation" in this case.

Because ParaMEDIC utilizes application semantics to generate metadata, it loses some portability compared to traditional byte-stream based distributed I/O. For example, an instance of ParaMEDIC's metadata transformation in the context of the mpiBLAST sequence search application is described in Section 3.2. However, by giving up some portability, ParaMEDIC can potentially attain tremendous savings in the amount of actual distributed I/O performed, consequently resulting in substantial performance gains. Further, through the use of a generic framework with an application plugin model, different applications can utilize the overall framework in an easy and flexible manner.

In this paper, we demonstrate how we used ParaMEDIC to tackle two large-scale computational biology problems—discovering missing genes and adding structure to genetic sequence databases—on a worldwide supercomputer. The overall worldwide supercomputer comprised nine different supercomputers distributed at seven sites across the U.S., and one large-scale storage facility located in Japan. The overall experiment consisted of sequence searching the entire microbial genome database against itself generating more than a petabyte of data that was transported to Tokyo for storage. We present several insights gained from this large scale run which will be of immense value to other researchers performing such large global-scale distributed computation and I/O.

2 *Large-Scale Computational Biology: A Peek at Compute and Storage Requirements*

In this section we discuss details on different aspects of computational biology with a focus on the compute and storage requirements of large-scale applications in this domain.

2.1 *Sequence Searching*

With the advent of rapid DNA sequencing, the amount of genetic sequence data available to researchers has increased exponentially [9]. The GenBank database, a comprehensive database that contains genetic sequence data for more than 260,000 named organisms, has exhibited exponential growth since its inception over 25 years ago [13]. This information is available for researchers to search new sequences against and infer homologous relationships between sequences or organisms. This helps progress a wide range of projects, from assembling the Tree of Life [18] to pathogen detection [20] and metagenomics [21].

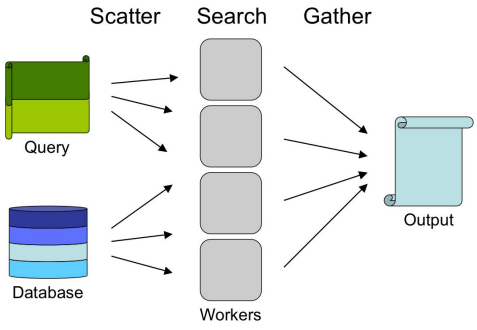

Figure 1: High-level Algorithm of mpiBLAST

Unfortunately, the exponential growth of sequence databases necessitates faster search algorithms to sustain reasonable search times. The Basic Local Alignment Search Tool (BLAST), which is the de facto standard for sequence searching, uses heuristics to prune the search space and decrease search time with an accepted loss in accuracy [7, 8]. mpiBLAST parallelizes BLAST using several techniques including database fragmentation, query segmentation [16], parallel input-output [27], and advanced scheduling [38]. As shown in Figure 1, mpiBLAST uses a *master-worker* model and performs a *scatter-search-gather-output*

execution flow. During the scatter, the master splits the database and query into multiple pieces and distributes them among worker nodes. Each worker then searches the query segment against the database fragment that it was assigned. The results are gathered by the Master, formatted, and output to the user. Depending on the size of the query and that of the database, such output generated can be large (Table 1). Thus, for environments with limited I/O capabilities, such as distributed systems, the output step can cause significant overheads.

Query Size (KB)	Number of Queries	Estimated Output (GB)
0-5	3,305,170	1,139
5-50	87,506	593
50-150	25,960	23,555
150-200	26,524	3,995
>200	9,840	[1]
Total	3,455,000	>29,282

Table 1: Estimated Output of an All-to-All NT Search

2.2 *Discovering Missing Genes*

Genome annotation is the process of associating information with a genomic sequence. Part of this process includes determining (by computational analysis) the location and structure of protein-encoding and RNA-encoding genes, also known as making gene calls. It is important to be as accurate and as sensitive as possible in making gene calls: avoiding false positives and missing real genes. Gene prediction in prokaryotes (bacteria and archaea) typically involves evaluating the coding potential of genomic segments which are delimited by conserved nucleotide motifs. The most widely used gene finding programs [17, 28] build a sequence model based on statistical properties of genes known to be (very likely) real genes in a given genome. This model is then used to evaluate the likelihood that an individual segment codes for a gene. In using this method it is almost always the case that some genes with anomalous composition are missed. Another popular method for locating genes is to compare genomic segments with a database of gene sequences found in similar organisms. If the sequence is conserved, the segment being evaluated is likely to be a coding gene (this is the "similarity method"). Genes that do not fit a given genomic pattern and do not have similar sequences in current annotation databases may be systemically missed.

One way to detect missed genes is to use the similarity method and compare raw genomes against each other, as opposed to comparing a raw genome to a database of known genes. If gene a in genome A and gene b in genome B have been missed and a is similar to b, then this method will find both. However, this involves performing an all-to-all comparison of the entire microbial genome database against itself. This task is heavily compute and data intensive, requiring thousands of compute processors and generating more than a petabyte of output data that needs to be stored for processing.

2.3 *Adding Structure to Genetic Sequence Databases*

One of the major issues with sequence searching is the structure of the sequence database itself. Currently, these databases are unstructured and represented as a flat file and each new sequence that is discovered is simply appended to the end of the file. Without more intelligent structuring, a query sequence has to be compared to every sequence in the database forcing the best-case to take just as long as the worst-case. By organizing and providing structure to the database, searches can be performed more efficiently by discarding irrelevant portions entirely.

One way to structure the sequence database is to create a sequence similarity tree where "similar" sequences are closer together in the tree than dissimilar sequences. The connections in the tree are created by determining how "similar" the sequences are to each other determined through sequence searches. To create *every* connection, however, the entire database has to be searched against itself resulting in an output size of N^2 values (where N is the number of sequences in the database).

3 *Overview of ParaMEDIC-enhanced mpiBLAST*

In our previous work [11, 12], we provided a detailed description of ParaMEDIC. Here we present a brief summary of this work.

3.1 *The ParaMEDIC Framework*

ParaMEDIC provides a framework for *decoupling* computation and I/O in applications relying on large quantities of both. Specifically, it does not hinder application computation. However, as the output data is generated, the framework differs from traditional distributed I/O; it uses application-semantic information to process the data generated by treating it as a collection of high-level application-specific objects rather than as a generic byte-stream. It uses such information to transform the data into orders-of-magnitude smaller metadata before transporting it over the distributed environment and regenerating it at the target site.

As shown in Figure 2, ParaMEDIC provides several capabilities including support for data encryption and integrity as well as data transfer in distributed environments (either directly via TCP/IP communication or through global file-systems). However, the primary semantics-based metadata creation is done by the *application plug-ins*. Most application plugins are specific to each application, and thus rely on knowledge of application semantics. These plugins provide two functionalities: (i) processing output data generated by the application to create metadata and (ii) converting metadata back to the final output. Together with application-specific plug-ins, ParaMEDIC also provides application-independent components such as data compression, data integrity, and data encryption. These can be used in conjunction with the application-specific plugins or independently.

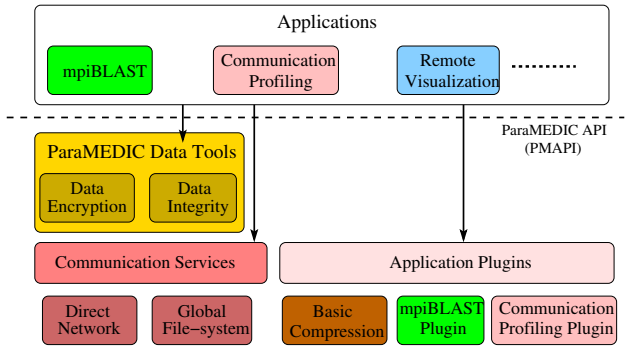

Figure 2: ParaMEDIC Architecture

Trading Computation with I/O: The amount of computation required in ParaMEDIC is higher than what is required by the original application; after the output is generated by the application processes, it has to be further processed to generate the metadata, sent to the storage site, and processed yet again to regenerate the final output. However, the I/O cost achieved can potentially be significantly reduced by using this framework. In other words, ParaMEDIC trades (a small amount of) additional computation for (potentially large) reduction in I/O cost. With respect to the additional computational cost incurred, ParaMEDIC is quite generic with respect to the metadata processing required by the different processes. For many applications, it is possible to tune the amount of post-processing performed on the output data, with the general trend being—the more the post-processing computation, the better the reduction in the meta-data size. That is, an application plugin can perform more processing of the output data to reduce the I/O cost.

3.2 *Integration with mpiBLAST*

In a *cluster* environment, most of the mpiBLAST execution time is spent on the search itself as the BLAST string-matching algorithm is computationally intensive, compared to which the cost of formatting and writing the results is minimal, especially when many advanced clusters are configured with high-performance parallel file-systems. However, in a *distributed* environment, the output typically needs to be written over a wide-area network to a remote file-system. Hence, the cost of writing the results can easily dominate the execution profile of mpiBLAST and become a severe performance bottleneck. By replacing the traditional distributed I/O framework with ParaMEDIC (as shown at the top of Figure 3), we can provide large reduction in the amount of data I/O performed. For example, as we will see in Section 5, a mpiBLAST-specific instance of ParaMEDIC reduces the volume of data written across a wide-area network by *more than two orders of magnitude*.

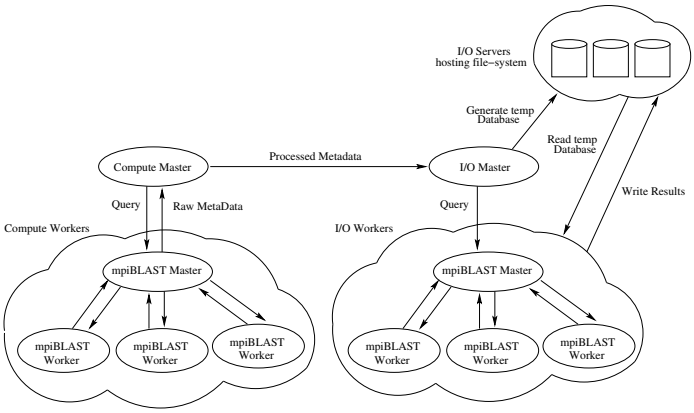

Figure 3: ParaMEDIC and mpiBLAST Integration

Figure 3 depicts how mpiBLAST is integrated with ParaMEDIC. First, on the compute site (the left cloud in Figure 3), once the output is generated by mpiBLAST, the mpiBLAST application plugin for ParaMEDIC processes this output to generate orders-of-magnitude smaller metadata. ParaMEDIC transfers this metadata to the storage site. At the storage site, 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 storage site by taking as input the same query sequence and the temporary database to generate and write output to the storage system. The overhead in rerunning mpiBLAST at the storage site is quite small as the temporary database that is searched is substantially smaller, with only about 250 sequences in it, compared to the several millions of sequences in large DNA databases.

4 *ParaMEDIC on a World-wide Supercomputer*

In this paper, to accommodate the compute and storage requirements of the computational biology applications discussed in Section 2, we utilize a world-wide supercomputer that, in aggregate, provides the required compute power and storage resources. The worldwide supercomputer comprises nine high-performance computing systems at seven different sites across the U.S. and a large-scale storage facility in Japan, to create a single high-performance *distributed* computing system. The specifics of each individual system are in Table 2. In the next subsections, we address the issues with working on such a large-scale distributed system that are not immediately apparent on smaller-scale systems.

Name	Location	Cores	Architecture	Memory (GB)	Network	Storage (TB)	Operating System	Distance from Storage (km)
SystemX	Virginia Tech	2200	IBM PowerPC 970FX	4	InfiniBand	NFS (30)	Linux 2.6.21	11,000
Breadboard Compute Cluster	Argonne National Lab	128	AMD Opteron	4	10-Gigabit Ethernet	NFS (5)	Linux 2.6.20	10,000
BlueGene/L Supercomputer	Argonne National Lab	2048	IBM PowerPC 440	1	Proprietary	PVFS (14)	Linux CNK	10,000
SiCortex Supercomputer	Argonne National Lab	5832	SiCortex SC5832	3	Proprietary	NFS (4)	Linux 2.6.15	10,000
Jazz Compute Cluster	Argonne National Lab	700	Intel Xeon	1-2	Gigabit Ethernet	GFS (10) PVFS (10)	Linux 2.4.29	10,000
NSF TeraGrid Distributed System	University of Chicago	320	Intel Itanium2	4	Myrinet 2000	NFS (4)	Linux 2.4.21	10,000
NSF TeraGrid Distributed System	San Diego Super-computer Center	60	Intel Itanium2	4	Myrinet 2000	GPFS (50)	Linux 2.4.21	9,000
Oliver Compute Cluster	Louisiana State University	512	Intel Xeon	4	InfiniBand	Lustre (12)	Linux 2.6.9	11,000
Open Science Grid	United States	200	AMD Opteron Intel Xeon	1-2	Gigabit Ethernet	-	Linux 2.4/2.6	11,000
TSUBAME Storage Cluster	Tokyo Institute of Technology	72	AMD Opteron	16	Gigabit Ethernet	Lustre (350)	Linux 2.6	0

Table 2: Specification of the Different Systems that Formed our Worldwide Supercomputer

4.1 *Dealing with Dynamic Availability of Compute Clients and Other Faults*

Several systems in our worldwide supercomputer operate in batch-mode. Users submit jobs to system queues and are scheduled to execute on the available resources. That is, compute resources are not available in a dedicated manner, but become available when our job is scheduled for execution and become unavailable when our job terminates.

To handle this issue, we segment the overall work to be performed into small tasks that can be performed independently (i.e., sequentially, concurrently or out-of-order). The management of tasks is done by a centralized server running on a dedicated resource. As each job is executed, it contacts this server for the next task, computes the task, transfers the output to metadata and transmits the metadata to the storage site. This approach has two benefits. First, the clients are completely stateless. That is, if a client job terminates before it has finished its computation or metadata transmission to the storage site, the servers handle this failure by reassigning the task to a different compute client. The second advantage is if the metadata corresponding to a tasks is either not received completely or is corrupted, the server just discards the data and reassigns the task to another compute node. Thus, I/O failures are never catastrophic.

4.2 *Architectural Heterogeneity*

One of the key impediments to large-scale distributed systems is system heterogeneity. Many distributed systems, such as the one used in this paper, cannot obtain a homogeneous environment in either hardware or software and efficient use of the system requires overcoming this obstacle. The worldwide supercomputer used in this paper contains six different CPU architectures (e.g., IBM PowerPC 970FX, IBM PowerPC 440, AMD Opteron, SiCortex MIPS64, Intel Xeon, Intel Itanium2), five different network interconnects (e.g., Gigabit Ethernet, 10-Gigabit Ethernet, InfiniBand, IBM proprietary 3D toroidal network, and SiCortex Kautz graph), and eight variations of the Linux operating system.

To deal with this issue, all data being transferred over the network has to be converted to an architecture-independent format. Since the total amount of data that is generated and has to be moved to the storage site in enormous, this can significantly impact on traditional distributed I/O. However, with ParaMEDIC, only metadata generated by processing the actual output is transferred across the wire. Since this metadata is orders-of-magnitude smaller as compared to the actual output, such byte manipulation to deal with heterogeneity has minimal impact on overall performance.

4.3 *Utilizing the Parallelism in Compute Nodes*

In traditional file I/O, there are two levels of parallelism. First, multiple I/O servers are aggregated into a parallel file-system to take advantage of the aggregate storage bandwidth of these servers. Second, multiple compute clients, that process different tasks, write data to such file-systems in parallel as well. Most parallel file-systems are optimized for such access to give the best performance.

With ParaMEDIC, there are three I/O components: (i) compute clients that perform I/O, (ii) post-processing servers that process the metadata to regenerate the original output and (iii) I/O servers which host the file-system. Similar to the traditional I/O model, the first and third components are already parallelized. That is, multiple streams of data being written in parallel by different compute clients and the I/O servers parallelize each stream of data that is being written to them. However, in order to achieve the best performance, it is important that the second component, post-processing servers, be parallelized as well.

Parallelizing the post-processing servers adds its own overhead and complexity mainly with respect to synchronization between the different parallel processes. To avoid this, we utilize an embarrassingly parallel approach for these servers. Each incoming stream of data is allocated to a separate process till a maximum number of processes is reached, after which the incoming data requests are queued till a process becomes available again. Thus, different processes do not have to share any information and can proceed independently. The advantage of this approach is its simplicity and lack of synchronization required between different parallel post-processing servers. The disadvantage, however, is that the number of data streams generated from the post-processing servers is equal to the number of incoming data streams. That is, if only two tasks are active at one time, only two streams of data are written to the actual storage system. Thus, the performance might not be optimal. However, in most cases, we expect the number of incoming streams to be sufficiently large to not face such performance issues.

4.4 *Handling Large Communication Latencies with Disconnected I/O*

As seen in Table 2, the computational sites are between 9,000 and 11,000 kilometers away from the storage site. At these distance, communication latencies are in tens of milliseconds. Such large latencies can severely limit the effectiveness of a synchronous remote file-systems that can be used for distributed I/O, since each synchronization operation has to make round-trip hops on the network. To overcome the bottleneck incurred by such high-latency, our worldwide supercomputer utilizes a lazy asynchronous I/O approach. By caching the output data locally before performing the actual output, clients can perform their computations while disconnected from the remote file-system. After a substantial amount of data is generated a bulk transfer of the metadata occurs, thereby maximally utilizing the bandwidth available between the sites and mitigating the effect of high-latency.

An issue with this approach of disconnected computation is fault-tolerance. Once a task is assigned to a compute client, the server is completely disconnected from this client. After the computation is complete, the client reconnects and sends the generated meta-data. Though, this two-phase synchronization model is more error prone and requires additional state information in the server, it makes the compute clients *truly stateless* even when the actual computation is going on.

5 *Experiments, Measurements and Analysis*

In this paper, we use ParaMEDIC to search the entire microbial genome database against itself. Several supercomputing centers within the U.S. perform the computation, while the data generated is stored at a large storage resource in Tokyo. This section

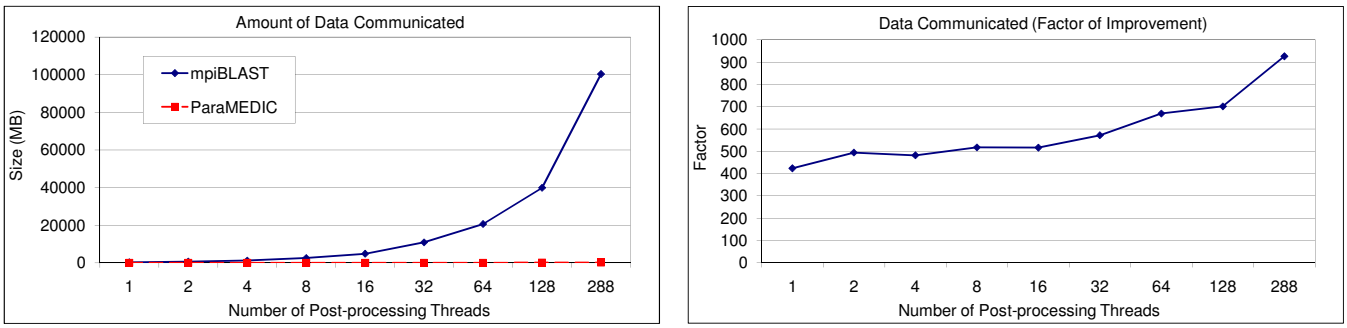

Figure 4: Data I/O Overheads: (a) Total amount of data communicated and (b) Factor of Improvement

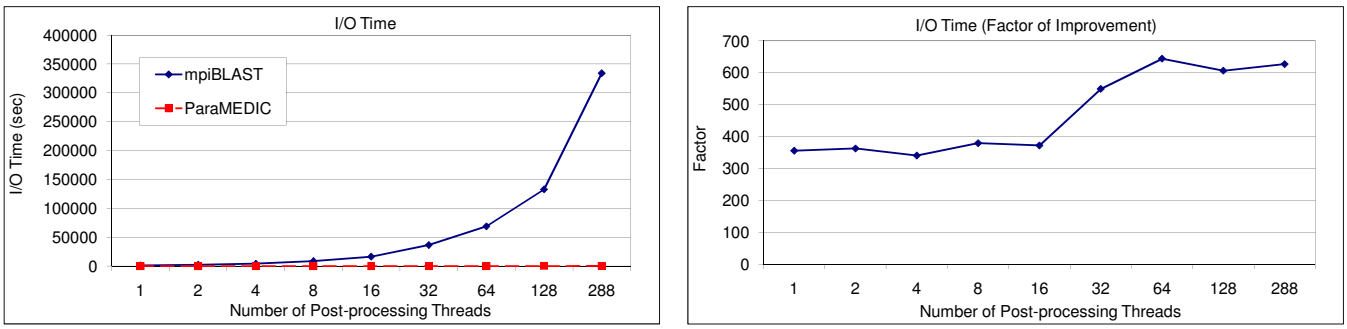

Figure 5: I/O Time Measurements: (a) Total I/O time and (b) Factor of Improvement

compares the performance of ParaMEDIC with vanilla mpiBLAST with respect to the amount of data communicated and the I/O time taken, as well as the storage bandwidth utilization. Several other results with respect to different output formats and scalability studies have been dropped due to space constraints and can be included in the final version if needed.

5.1 *Data I/O Overheads*

Figure 4 illustrates the amount of data transmitted between the compute and the storage sites for different number of post-processing threads, and the factor of reduction in the amount of data. Each post-processing thread processes one segment of data which has the output for 10,000 query sequences. Most segments have approximately similar output sizes, so the amount of data communicated over the distributed network increases linearly with the number of segments, and hence the number of post-processing threads. ParaMEDIC, on the other hand, processes the generated data and converts it into metadata before performing the actual transfer. Thus, the actual data that is transferred over the network is much smaller. For example, with 288 threads, mpiBLAST communicates about 100GB of data, while ParaMEDIC only communicates about 108MB—a *900-fold* reduction.

We also illustrate the I/O time in Figure 5. As shown, ParaMEDIC outperforms mpiBLAST by more than two orders-of-magnitude. This result is attributed to multiple aspects. First, given the shared network connection between the two sites, the achievable network performance is usually much lower than within the cluster. Thus, with mpiBLAST transferring the entire output over this network, its performance would be heavily impacted by the network performance. Second, the distance between the two sites causes the communication latency to be high. Thus, file-system control messages tend to take a long time to be exchanged, resulting in further loss in performance. Third, for mpiBLAST, since the wide-area network is a bottleneck, the number simultaneously transmitted data streams does not matter; communication is serialized in the network. However, with ParaMEDIC, since the wide-area network is no longer a bottleneck, it can more effectively utilize the parallelism in the data streams to better take advantage of the storage bandwidth available at the storage site, as desribed in more detail in Section 5.2.

5.2 *Storage Bandwidth Utilization*

Figure 6(a) illustrates the storage bandwidth utilization achieved by mpiBLAST, ParaMEDIC, and the MPI-IO-Test benchmark, that is used as an indication of the peak performance capability of the I/O subsystem. We notice that the storage utilization of mpiBLAST is very poor compared to ParaMEDIC. This is because, for mpiBLAST, the I/O is limited by the wide-area network bandwidth. Thus, though more than 10,000 processors are performing the compute part of the task, the network connecting the

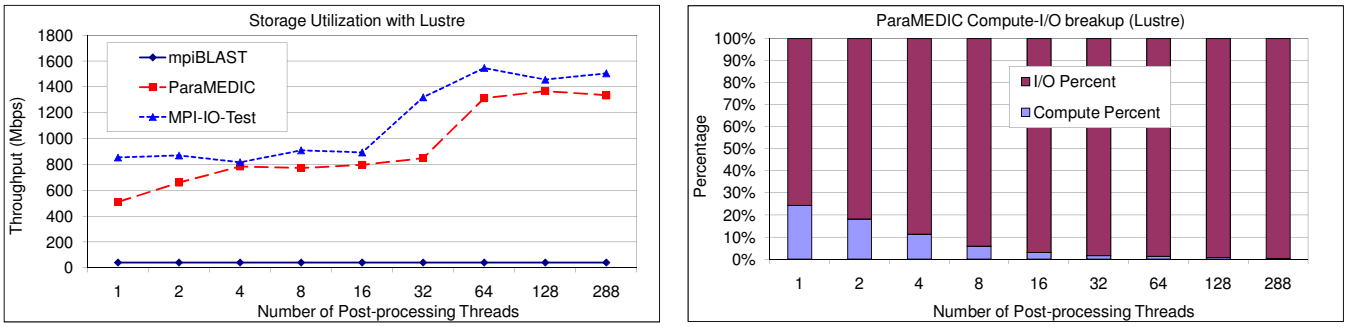

Figure 6: Storage Bandwidth Utilization using Lustre: (i) Storage Utilization Improvement and (ii) Computation and I/O time

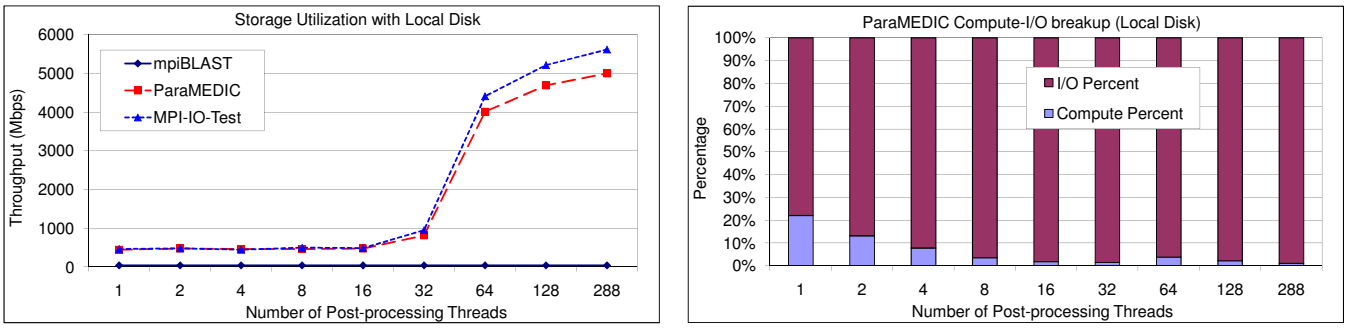

Figure 7: Storage Utilization using Local File-system: (i) Storage Utilization Improvement and (ii) Computation and I/O time

compute servers in the U.S. and the storage system in Tokyo becomes the bottleneck.

On the other hand, ParaMEDIC uses more than 90% of the storage system capability (shown by MPI-IO-test). When the number of processing threads is low (x-axis in figure), ParaMEDIC uses about half the storage capability. However, as the number of processing threads increases, the I/O utilization of ParaMEDIC increases as well.

Figure 6(b) illustrates the percentage breakup of the time spent in ParaMEDIC's post-processing phase. A significant portion of the time spent is in the I/O part. This shows that in spite of using a fast parallel file system such as Lustre, ParaMEDIC is still bottlenecked by the I/O subsystem. In fact, our analysis has shown that in this case the bottleneck lies in the 1-Gigabit Ethernet network subsystem connecting the storage nodes. Thus, we expect that even for systems with faster I/O subsystems, ParaMEDIC will further scale up and continue to use a significant portion of the I/O bandwidth provided.

In Figure 7(a), we remove the file-system network bottleneck and directly perform I/O on the local nodes. Storage utilization achieved in this case is significantly higher than going over the network. Even in this case, ParaMEDIC completely uses the storage capability with more than 90% efficiency. Figure 7(b) shows the percentage breakup of the time spent. Similar to the case with the Lustre file-system, a significant portion of the time is still spent on I/O. Thus, again, ParaMEDIC can be expected to scale and fully use even faster storage resources.

6 *Discussion*

Though this paper only deals with enhancing the mpiBLAST application through ParaMEDIC, the idea is relevant for many other applications as well. For example, applications that have natively been built for distributed environments such as SETI@home [35] and other BOINC applications [1] can easily use similar ideas, and can benefit aspects in which such techniques are possible. In the field of communication profiling with MPE [4], we have also done some preliminary work that utilizes metadata transformation of profiled data through ParaMEDIC. Specifically, based on the observation that most scientific applications have a very uniform and periodic communication pattern, we perform a fourier transform on the data to identify this periodicity and use this as an abstract block. The metadata comprises one complete abstract block and just the differences for all other blocks. Our preliminary numbers in this field have demonstrated between 2 and 5-fold reduction in the I/O time using ParaMEDIC. Work on other application fields including earthquake modeling and remote visualization is ongoing as well, with promising preliminary results.

7 *Related Work*

Efficient I/O access for scientific applications in distributed environments has been an ongoing subject of research for various parallel and distributed file-systems [14,22,32,34]. There has also been work on explicit data transfer protocols such as GridFTP [6]. Other efforts include providing remote data access through MPI-IO [29]. RIO [19] introduced a proof-of-concept implementation that allows applications to access remote files though ROMIO [37]. RFS [25] enhanced the remote write performance with *active buffering*, by optimizing overlap between applications computation and I/O. Studies have also been done in translating MPI-IO calls into operations of lower level data protocols such as Logistic Network [26]. However, all these approaches deal with data as a byte-stream. Conversely, our approach focuses on aggressively reducing the amount of I/O data to be communicated by taking advantage of application semantics and dealing with data as high-level abstract units, rather than a stream of bytes.

Semantic-based data transformation is not new. Several semantic compression algorithms have been investigated in compressing relational databases [10,23,24]. Leveraging the table semantics, these algorithms first build descriptive models of the database using data mining techniques such as clustering, and strip out data that can be regenerated. In the multimedia field, context-based coding techniques (similar to semantics-based approaches) have been widely used in various video compression standards [15, 30, 31]. With aid of context modeling, these techniques efficiently identify redundant information in the media. Although sharing the same goal of reducing data to store or transfer with ParaMEDIC, these data compression studies do not address the remote I/O issue.

Thus, ParaMEDIC utilizes ideas from different fields to provide a novel approach for distributed I/O.

8 *Conclusion*

Rapid growth of computational power is enabling computational biology to tackle increasingly large problems such as discovering missing genes and providing structure to genetic sequence databases. However, as the problems grow larger, so does the data consumed and produced by the applications. For many applications, the required compute power and storage resources cannot be found at a single location, precipitating the transfer of large amounts of data across the wide-area network. ParaMEDIC mitigates this issue by pursuing a non-traditional approach to distributed I/O. By trading computation for I/O, ParaMEDIC utilizes application semantics information to transform the output to orders-of-magnitude smaller metadata. In this paper, we presented our experiences in solving large-scale computational biology problems by utilizing nine different high-performance compute sites within the U.S. to generate a petabyte of data, that was transferred to a large-scale storage facility in Tokyo using ParaMEDIC's distributed I/O capability. We demonstrated that ParaMEDIC can achieve a performance improvement of several orders of magnitude compared to traditional I/O. In future, we plan to evaluate semantic-based compression for other applications.

9 *Acknowledgments*

We would like to thank the following people for their technical help in managing the large-scale run and other experiments associated with this paper: (i) R. Kettimuthu, M. Papka and J. Insley from the University of Chicago, (ii) N. Desai and R. Bradshaw from Argonne National Laboratory, (iii) G. Zelenka, J. Lockhart, N. Ramakrishnan, L. Zhang, L. Heath, and C. Ribbens from Virginia Tech, (iv) M. Rynge and J. McGee from Renaissance Computing Institute, (v) R. Fukushima, T. Nishikawa, T. Kujiraoka and S. Ihara from Tokyo Institute of Technology, (vi) S. Vail, S. Cochrane, C. Kingwood, B. Cauthen, S. See, J. Fragalla, J. Bates, R. Cagle, R. Gaines and C. Bohm from Sun Microsystems, and (vii) H. Liu from Louisiana State University.

References

[1] BOINC: Berkeley Open Infrastructure for Network Computing. http://boinc.berkeley.edu.

[2] CANARIE. http://www.canarie.ca.

[3] Ethernet Everywhere: Price/Performance as the Key to Cluster & Grid Interconnects. http://www.cse.scitech.ac.uk/disco/mew14-cd/Talks/Force10_VanCampen.pdf.

[4] MPE : MPI Parallel Environment. http://www-unix.mcs.anl.gov/perfvis/download/index.htm.

[5] SURFnet. http://www.surfnet.nl.

[6] B. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus Striped GridFTP Framework and Server. In *SC*, 2005.

[7] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.

[8] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI–BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25:3389–3402, 1997.

[9] S.F. Altshul, M.S. Boguski, W. Gish, and J.C. Wootton. Issues in searching molecular sequence databases. *Nat Genet*, 6(2):119–29, 1994.

[10] S. Babu, M. Garofalakis, and R. Rastogi. Spartan: a model-based semantic compression system for massive data tables. *SIGMOD Rec.*, 30(2), 2001.

[11] P. Balaji, W. Feng, J. Archuleta, H. Lin, R. Kettimuthu, R. Thakur, and X. Ma. Semantics-Based Distributed I/O for mpiBLAST. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, Salt Lake City, Utah, February 2008.

[12] P. Balaji, W. Feng, and H. Lin. Semantics-Based Distributed I/O with the ParaMEDIC Framework. Technical Report ANL/MCS-P1477-0108, Argonne National Laboratory, Argonne, IL, 2008.

[13] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler. Genbank. *Nucleic Acids Res*, 36(Database issue), January 2008.

[14] P. Carns, W. Ligon III, R. Ross, and R. Thakur. PVFS: A Parallel File System For Linux Clusters. In *LSC*, 2000.

[15] T. Wiegand D. Marpe, H. Schwarz. Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7), 2003.

[16] A. E. Darling, L. Carey, and W. Feng. The Design, Implementation, and Evaluation of mpiBLAST. In *ClusterWorld Conference & Expo and the 4th International Conference on Linux Cluster: The HPC Revolution*, 2003.

[17] A. L. Delcher, K. A. Bratke, E. C. Powers, and S. L. Salzberg. Identifying Bacterial Genes and Endosymbiont DNA with Glimmer. *Bioinformatics*, January 2007.

[18] A.C. Driskell, C. Ané, J.G. Burleigh, M.M. McMahon, B.C. O'Meara, and M. J. Sanderson. Prospects for building the tree of life from large sequence databases. *Science*, 306(5699):1172–1174, November 2004.

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

[20] J. D. Gans, W. Feng, and M. Wolinsky. Whole Genome, Physics-Based Sequence Alignment for Pathogen Signature Design. In *12th SIAM Conference on Parallel Processing for Scientific Computing*, San Francisco, California, February 2006. (electronic version unavailable).

[21] S.L. Havre, B.-J. Webb-Robertson, A. Shah, C. Posse, B. Gopalan, and F.J. Brockma. Bioinformatic insights from metagenomics through visualization. *Computational Systems Bioinformatics Conference, 2005. Proceedings. 2005 IEEE*, pages 341–350, 8-11 Aug. 2005.

[22] Howard. An overview of the andrew file system. In *USENIX Winter Technical Conference*, February 1988.

[23] H. Jagadish, J. Madar, and R. Ng. Semantic compression and pattern extraction with fascicles. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[24] H. Jagadish, R. Ng, B. Ooi, and A. Tung. Itcompress: an iterative semantic compression algorithm.

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

[26] 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.

[27] H. Lin, X. Ma, P. Chandramohan, A. Geist, and N. Samatova. Efficient Data Access for Parallel BLAST. In *IPDPS*, 2005.

[28] A. Lukashin and M. Borodovsky. Genemark. hmm: new solutions for gene finding, 1998.

[29] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Standard*, July 1997.

[30] Generic Coding of Moving Pictures and Associated Audio Information - Part 2: Video. ITU-T and ISO/IEC JTC1, ITU-T Recommendation H.262 ISO/IEC 13 818-2 (MPEG-2), 1994.

[31] Coding of Audio-Visual Objects - Part 2: Visual. ISO/IEC JTC1, ISO/IEC 14 496-2 (MPEG-4 Visual version 1), Apr. 1999; Amendment1 (version 2), Feb. 2000; Amendment 4 (streaming profile), Jan. 2001.

[32] B. Nowicki. *NFS: Network File System Protocol Specification*. Network Working Group RFC1094, 1989.

[33] D. A. Reed. Grids, the teragrid, and beyond. *Computer*, 36(1):62–68, 2003.

[34] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *FAST*, 2002.

[35] Seti@home: The search for extraterrestrial intelligence. http://setiathome.ssl.berkeley.edu/.

[36] S. Simms, G. Pike, and D. Balog. Wide Area Filesystem Performance using Lustre on the TeraGrid. In *TeraGrid Conference*, 2007.

[37] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO. In *Frontiers of Massively Parallel Computation*, 1999.

[38] O. Thorsen, B. Smith, C. P. Sosa, K. Jiang, H. Lin, A. Peters, and W. Feng. Parallel Genomic Sequence-Search on a Massively Parallel System. In *ACM International Conference on Computing Frontiers*, Ischia, Italy, May 2007.