ParaMEDIC: Parallel Metadata Environment for Distributed I/O and Computing

(Storage Challenge, Supercomputing 2007)

P. Balaji[§]
[§]Math. and Computer Science, Argonne National Laboratory
balaji@mcs.anl.gov

W. Feng[¶] J. Archuleta[¶]
[¶]Dept. of Computer Science, Virginia Tech
{feng, jsarch}@cs.vt.edu

H. Lin[*]
[*]Dept. of Computer Science, North Carolina State Univ.
hlin2@ncsu.edu

I. Team Information

The ParaMEDIC team consists of four individuals across three institutions: Pavan Balaji (Argonne National Laboratory), Wu-chun Feng and Jeremy Archuleta (Virginia Tech) and Heshan Lin (North Carolina State University). ParaMEDIC is a Parallel Metadata Environment for Distributed I/O and Computing that utilizes the the MPICH2 communication library from Argonne National Laboratory and the mpiBLAST parallel sequence-searching application from Prof. Feng's group at Virginia Tech. Partner institutions that supported this effort by providing equipment resources include the Tokyo Institute of Technology with support from Sun Microsystems, the Center for Computation and Technology at Louisiana State University, the Renaissance Computing Institute, the University of Chicago, and the San Diego Supercomputing Center.

Short biographies of the ParaMEDIC team members are provided below.

Pavan Balaji: Pavan Balaji holds a joint appointment as a post-doctoral researcher at the Argonne National Laboratory and as a fellow of the Computation Institute at the University of Chicago. Dr. Balaji received his Ph.D. from the Computer Science and Engineering department at the Ohio State University. His research interests include high-speed interconnects, efficient IP-based protocols, parallel programming models and middleware, and job scheduling and resource management. He has nearly 30 publications and has given several talks and tutorials in the above areas. Dr. Balaji has also served as a Program Committee Member and Technical Referee for several international conferences and journals. He is a member of the IEEE and ACM. More details about Dr. Balaji, including a comprehensive CV, are available at (http://www.mcs.anl.gov/∼balaji).

Wu-chun Feng: Wu-chun Feng is an associate professor of Computer Science and Electrical and Computer Engineering at Virginia Tech, where he directs the Synergy Laboratory. His research interests span many areas of high-performance networking and computing from the systems level to the applications level, e.g., advanced computer architecture, green supercomputing, network protocols, grid computing, virtual computing, and bioinformatics. He has amassed over 100 peer-reviewed publications, given more than 20 invited talks (including two keynote talks), and served on 10 conference panels as well as 30 program committees for international conferences and workshops in high-performance computing. In addition to having his work recognized in the New York Times, CNN, and BBC News, he was also the recipient of three R&D 100 Awards and was named as one of HPCwire's Top People to Watch in 2004. He holds a B.S. and an M.S. in computer engineering and a B.S. Honors in music, all from Penn State University, and a Ph.D. in computer science from the University of Illinois at Urbana-Champaign. He is a senior member of the IEEE and a member of the ACM.

Jeremy Archuleta: Jeremy Archuleta is an Institute for Critical Technologies and Applied Science Doctoral Scholar and Ph.D. candidate in the Department of Computer Science at Virginia Tech. His research areas include high-performance computational science and software engineering. Mr. Archuleta has a M.S. in Computer Science from the University of Utah and a B.S. in Electrical Engineering and Computer Science from the University of California at Berkeley.

Heshan Lin: Heshan Lin is a Ph.D. student in the Department of Computer Science at North Carolina State University. His research focuses on high-performance computing, particularly parallel I/O and distributed computing. Mr. Lin has several publications in ACM and IEEE-sponsored conference proceedings. He received a M.S. in Computer Science from Temple University and

a B.S. in Applied Mathematics from the South China University of Technology.

II. Abstract

The ParaMEDIC team created a worldwide supercomputer that aggregates supercomputers around the world in order to tackle large-scale bioinformatics problems that cannot easily be solved in a traditional supercomputing environment. The bioinformatics problems of significance are as follows:

- Sequence-search all the 567 microbial genomes (that have been completed to date) against each other in order to discover *missing genes* via mpiBLAST sequence-similarity computations [5], [19].
- Generate a complete genome sequence-similarity tree, based on the above results, in order to structure the sequence databases so as to enable pruning of the sequence-search space and, thus, accelerate the sequence-search process.

In order to solve the above problems, we used more more than 10,000 processors distributed across several different supercomputing centers in the U.S. and generated a petabyte of uncompressed data in one month and wrote a compressed version of this dataset to the 0.5-petabyte filesystem at the Tokyo Institute of Technology.

Given the daunting computational and I/O scale of the above problems, solving them required more than a brute-force approach. It required a new framework that we call *ParaMEDIC (Parallel Metadata Environment for Distributed I/O and Computing)*. ParaMEDIC leverages MPICH2 and mpiBLAST to provide an environment that decouples computation and I/O in applications such as mpiBLAST and drastically reduces I/O overhead through metadata processing. ParaMEDIC trades a small amount of additional computation (in the form of metadata processing) with a significant reduction in the amount of I/O to achieve high performance. Preliminary studies demonstrated that ParaMEDIC improves the performance of mpiBLAST by 5-fold over the Teragrid infrastructure and by 25-fold on regular Internet2 connectivity between Virginia Tech and the Argonne National Laboratory. *For this SC|07 Storage Challenge, ParaMEDIC improved the I/O time of mpiBLAST by more than three orders of magnitude, over our worldwide supercomputer, which consisted of 10,000+ computational cores located in the U.S. and on the order of a petabyte of storage in Japan.*

III. Problem Statement

In the past decade, there has been an increasing research focus on computational biology as a means to provide sophisticated mechanisms to allow biologists to interact with and analyze existing information about well-known biological entities and utilize it to improve insights into the behavior of newer entities. Nucleotide and protein sequence-searches are common examples for such interactions, where biologists search an unknown sequence (from a new organism) in a large database of sequences corresponding to known biological entities [1] in order to find genetic similarities [16] between the organisms. For example, in 2003, sequence matching helped biologists to identify the similarities between the recent SARS virus and the more well-studied coronaviruses, thus enhancing the biologists' ability to combat the new virus [14]. The ubiquitous tool that is used to perform these sequence searches is BLAST [1].

With the size of the GenBank database doubling every 12 months [2], [7], and the computational horsepower of a single processor doubling only every 18-24 months, the growth rate of the databases has been fast outstripping the ability of a single processor to keep up. For instance, in 2002, searching a 300-KB **portion** of the *E. chrysanthemi* genome against the NT database took 22.4 hours (80,775 seconds); by 2005, the same search but with an updated NT database took 49.1 hours (176,880 seconds) to complete.

Given the importance of biosequence searching using BLAST, researchers have designed a number of tools to perform these searches in an efficient manner. mpiBLAST [5], [6] is a freely available, open-source parallelization of the popular BLAST sequence search library. By segmenting the query and fragmenting the database, along with many other advanced parallel techniques, mpiBLAST gives super-linear speedups to BLAST searches and has become an indispensable tool for computational biologists.

Like most other parallel sequence search applications, the overall software architecture of mpiBLAST follows a Master-Worker model. At the core of mpiBLAST resides database segmentation, which fragments a sequence database across multiple nodes so that each fragment fits in memory. Each worker process then searches its unique portion of the database independently of the other workers. Once the search is complete, the results of the search are merged at the master process and written out to a central file for later examination or post-processing by the biologist.

While a Master-Worker model provides the required compute horsepower to perform the actual sequence search, as the scale of the problem and the number of processes involved in the search increases, it is not always possible for all the processes to have high-speed access to the file-system. It forces the different compute processes to utilize a potentially distant and/or slow

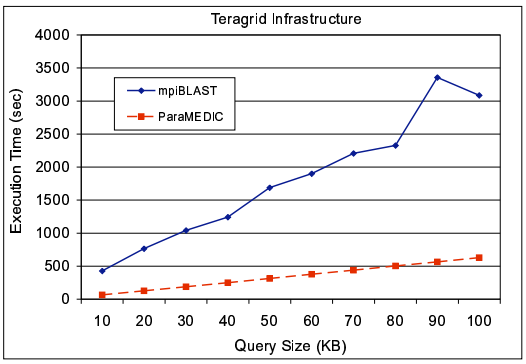


Fig. 1. Teragrid Infrastructure using University of Chicago and the San Diego Supercomputing Center

distributed filesystem for writing the final output to. As the database size grows, the search output generated also grows proportionally. Thus, such distributed I/O creates a significant bottleneck which severely limits the performance of the application.

For example, on the Teragrid infrastructure, compute servers are distributed across multiple locations in the U.S. (including University of Chicago, Texas Advanced Computing Center and Oak Ridge National Laboratory), with a common distributed filesystem being hosted at San Diego Supercomputer Center (SDSC). Our evaluations, shown in Figure 1, demonstrated that the I/O time of mpiBLAST when the query was computed at SDSC was more than 5-fold faster as compared to when the same query was computed at a different location, in spite of having a 30Gbps network link between the different locations. This essentially limits the utility of the large number of compute resources available by converting the sequence-search problem from a compute-bound problem to an I/O-bound problem.

In this paper, we propose to sequence-search all of the 567 microbial genomes that have been completed to date, against each other, and build a similarity tree based on these results. Given the scale and complexity of this problem, the number of compute and storage resources required to solve it is gigantic. For example, the 567 microbial genomes together consist of more than 16 million individual sequences, and require more than 263 trillion sequence comparisons. Further, the output generated from all these searches and the similarity tree is expected to be more than one petabyte. Given that very few institutes possess the capability to perform such large computations *AND* store such a large amount of data, only through the collective resources of a worldwide computer can such multi-genome searches be undertaken. This, however, means that the final storage is *not* local for any of the compute resources, leading to the distributed I/O bottleneck described above.

To handle this problem, we propose *ParaMEDIC (Parallel Metadata Environment for Distributed I/O and Computing)*, a novel framework that decouples computation and I/O through metadata processing. Specifically, ParaMEDIC converts the output generated to application-specific metadata at the compute site, moves the metadata to the storage site, and reconverts the metadata to the actual output again at the storage site. In other words, ParaMEDIC trades a small amount of additional computation (in terms of metadata processing) for significantly lesser I/O in distributed environments. This allows the distributed computational resources to be effectively utilized without having to *stall* for long periods on I/O.

IV. Hardware Configuration

In order to solve the problems described in Section III, we used more than 10,000 processors distributed across several different supercomputing centers in the U.S., and a large petabyte scale storage facility at the Tokyo Institute of Technology at Japan. The overall hardware configuration is broken down into two categories—compute resources and I/O resources.

Compute Resources: Following the software architecture of ParaMEDIC, the compute resources are responsible for completing the required 263 trillion searches and generating the resultant metadata. The following resources were distributed around the U.S. and performed the bulk of the computation, but minimal I/O, in the application:

1) 2200-processor *System X* supercomputer at Virginia Tech.
2) 2048-processor *BG/L* supercomputer at Argonne National Laboratory.
3) 5832-processor *Sicortex* supercomputer at Argonne National Laboratory.
4) 700-processor Intel *Jazz* supercomputer at the Argonne National Laboratory.
5) A few hundred processors on the Teragrid system located at the San Diego Supercomputing Center and University of Chicago.
6) A few hundred processors located the Center for Computation and Technology located at Louisiana State University.
7) A few hundred processors on the Open Science Grid located at the Renaissance Computing Institute.

8) A few hundred processors on the *Breadboard* system at the Argonne National Laboratory.

I/O Resources: The final output was stored at the Tokyo Institute of Technology with support from Sun Microsystems. The details of this storage system are:

1) Clients: 10 quad-core SunFire X4200 and 2 16-core SunFire X4500 systems
2) Object Storage Servers (OSS): 20 SunFire X4500
3) Object Storage Targets (OST): 140 SunFire X4500 (each OSS has 7 OST)
4) RAID configuration for OST: RAID5 with 6 drives
5) Network: 1 Gigabit Ethernet
6) Kernel: 2.6
7) Lustre Version: 1.6.2

V. Data and Storage Layout

As described earlier, the goal of this project is two fold—(i) Sequence-search all the 567 microbial genomes (that have been completed to date) against each other in order to discover *missing genes* via mpiBLAST sequence-similarity computations, and (ii) Generate a complete genome sequence-similarity tree, based on the above sequence-searching, in order to structure the sequence databases so as to enable pruning of the sequence-search space and, thus, accelerate the sequence-search process.

With respect to the first goal, the output data generated by mpiBLAST is a list of matching sequences with the relevant details about the matching portions of the sequence, significance of the match, and others. Typically, a separate output file is generated for each query – a query itself is comprised of multiple query sequences. We took all 567 microbial genomes from NCBI (ftp.ncbi.nih.gov/genomes/Bacteria) and split each replicon (a chromosome or plasmid) into open reading frames (i.e., potential genes) of length 99 nucleotides or greater. A minimum length is generally used to prevent "useless" sequences from being perceived to be functional genes. This resulted in more than 16 million query sequences (or potential genes) being created, comprising more than 5 gigabytes of data. These 16 million sequences were split into 16,242 queries with each query generating a single output file.

With respect to the second goal, for each sequence in the database, we further utilized this generated output to form a *sequence-similarity tree* of matching sequences at different depths. That is, if a sequence s1 directly matches sequence s2 during a search, it is considered a first-level match in the tree. If s2 directly matches s3, then s3 is considered to be a second-level match in the tree for s1. This allows the database to have a more well-defined structure that relates each sequence to the remaining sequences in the database.

For the size of the output, the division of each microbial genome into its respective potential genes and the subsequent *all-to-all* comparison has never been performed before; so there was no real expected size of the amount of I/O that this would generate. However, from preliminary testing, we estimated the final output including the matches as well as a similarity restructuring of the database to be around one petabyte. Due to the large storage requirement, we utilized the storage facility available at the Tokyo Institute of Technology where data on the order of a petabyte could be stored.

Though generating or storing one petabyte is probably not a major step by itself, the number of compute resources required for generating this data force the compute resources to be globally distributed (since very few institutes hold as many compute resources in a single site). Furthermore, the large geographical distances between these resources and the final storage location makes this scientific study extremely challenging and the solution innovative.

VI. Software Used

We used three primary software stacks for this challenge – (i) ParaMEDIC, a parallel metadata environment for distributed I/O and computing from Argonne National Laboratory and Virginia Tech, (ii) MPICH2, an implementation of the MPI standard from Argonne National Laboratory and (iii) mpiBLAST, a parallel sequence-search application from Virginia Tech.

Of note, all three primary applications used in this work (ParaMEDIC, MPICH2 and mpiBLAST) are *home-grown* by the authors. ParaMEDIC and mpiBLAST were developed by the authors to address and resolve issues in high-performance bioinformatics applications, while MPICH2 was developed as a high-performance and portable implementation of the Message Passing Interface (MPI). Also of note, the version of mpiBLAST used in this experiment is mpiBLAST-PIO, an optimized version of mpiBLAST that utilizes parallel I/O techniques to exploit the tightly-coupled computational resources, i.e., clusters, at the different sites.

Other than these three applications, only standard Linux and Mac OS X tools were used on the corresponding platforms.

VII. Description of Solution to the Problem

In this section, we present a detailed description of our approach, namely ParaMEDIC (short for *Para*llel *M*etadata *E*nvironment for *D*istributed *I*/O and *C*omputation). We first describe the overall framework

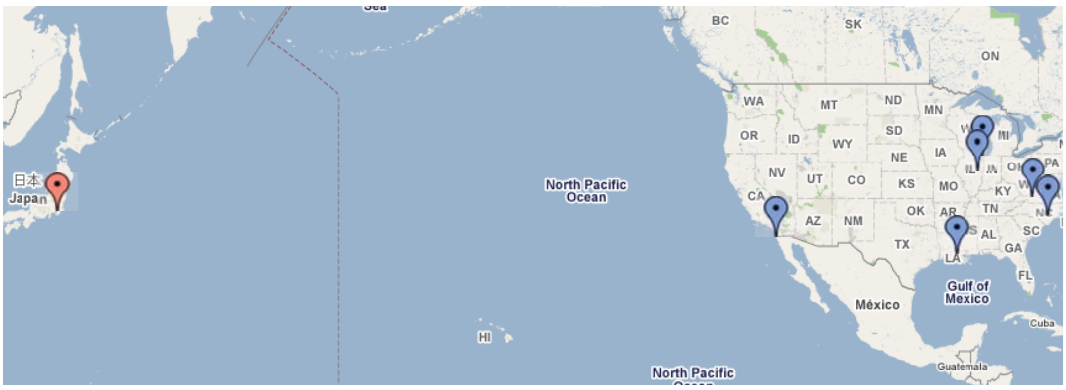


Fig. 2. World-wide supercomputer

in section VII-A, followed by details of the generation of the metadata by the compute workers in section VII-B. We then conclude the section with a description of how the post-processing is performed by the I/O workers to regenerate the final output in section VII-C.

A. The ParaMEDIC Framework

As seen in Figure 3, ParaMEDIC utilizes a two-tiered hierarchical framework for decoupling computation and I/O in mpiBLAST. The upper tier consists of two processes – *compute manager* and *I/O manager* — while the lower tier consists of two groups of processes – *compute workers* and *I/O workers*. The actual searching of the query sequences in the database is handled by the compute workers. These workers, however, do not generate their regular sequence output; they instead generate a raw version of metadata that represents the actual output. Once the raw metadata is generated, the compute master processes this metadata, compresses it, writes it to the file-system, and sends a signal to the I/O master. The I/O master, upon receiving a signal from the compute master, utilizes the I/O workers to process the metadata and generate the final output.

1) Trading Computation with I/O cost: The amount of computation required in ParaMEDIC is higher than what is required by the original mpiBLAST implementation. For example, after the output is generated by the compute workers, it has to be processed to generate the metadata, sent to the I/O master, and again processed by the I/O workers to re-generate the final output. However, the I/O cost can potentially be significantly reduced by this framework. In other words, ParaMEDIC aims at trading additional computation for reduced I/O cost. Figure 4 shows the execution breakup of running mpiBLAST and ParaMEDIC over a WAN with 100ms network latency. As can be seen, although paying a little extra computation (post-processing) overhead, ParaMEDIC can reduce I/O costs by orders of magnitudes.

With respect to the additional computational cost incurred, ParaMEDIC is quite generic with respect to the metadata processing that is required by the different processes. For example, the metadata can be a simple and basic compression of the actual output or a complex application-specific data structure that can be utilized to regenerate the final output. In a simple compression scheme, no application-specific information is needed, thus allowing new applications to utilize the framework quickly and easily. However, the post-processing required is high and the I/O savings might not be that significant either. On the other hand, with a complex application-specific data structure generation scheme, the amount of additional post-processing required to regenerate the final output can be low, and the I/O savings significant.

2) Managing Compute and I/O Worker Processes: Managing the compute and I/O worker processes in ParaMEDIC essentially determines the tradeoff in the amount of time spent in computation versus the amount of time saved in I/O. In general, since the I/O worker processes are restricted to the cluster that hosts the file-system, the number of I/O workers that are available is restricted. For example, in a distributed environment hosting 10,000 processors, only 100 processors might reside on the same cluster that hosts the file-system. Thus, in this case, it is ideal to maintain a 100:1 ratio between the number of compute workers and the number of I/O workers.

This forces ParaMEDIC to utilize algorithms that can generate metadata from the result and re-generate the result from the metadata with minimal processing requirements. In other words, the framework ensures that the post-processing cost required to process the metadata

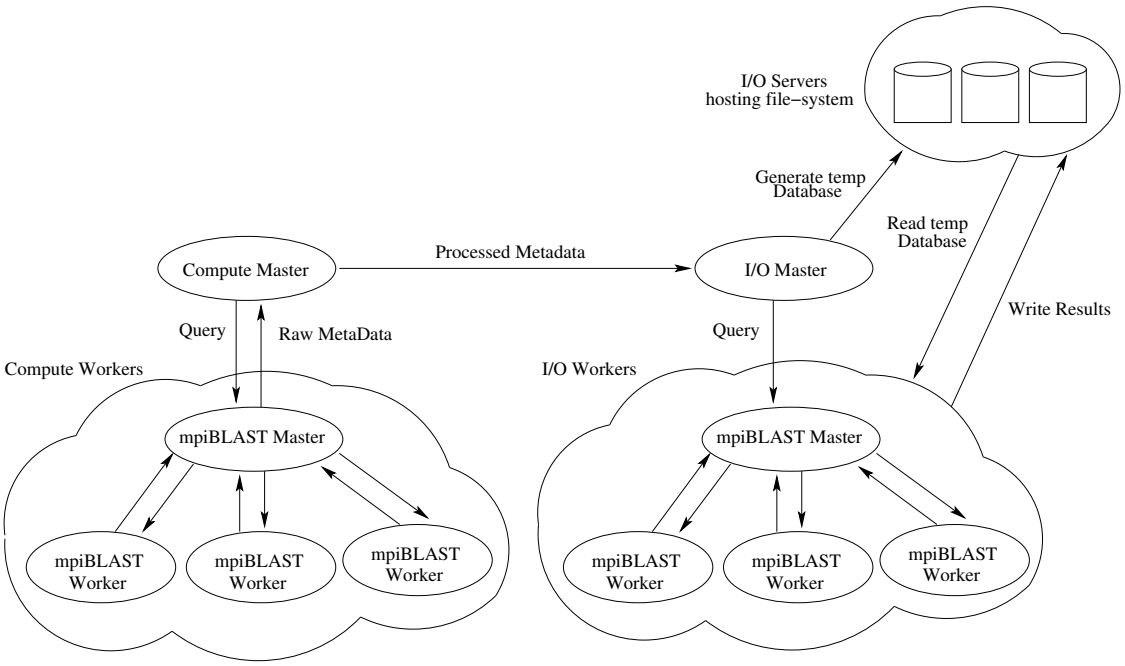


Fig. 3. The ParaMEDIC Framework

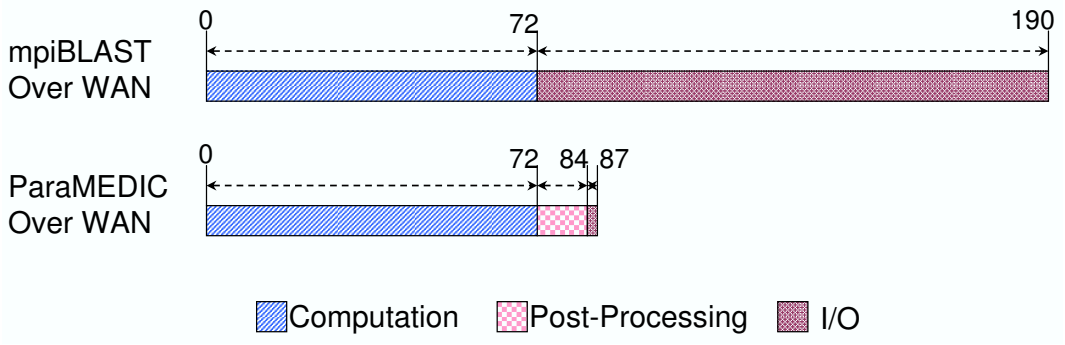


Fig. 4. Execution breakup for running mpiBLAST and ParaMEDIC over WAN with 100ms network latency. The performance numbers are measured in seconds.

and generate the final output is significantly less than the actual computation needed to search for the query sequence within the database.

B. Metadata Generation

As described in section VII-A1, the metadata generated by ParaMEDIC can be as simple as a basic compression of the result, or as complex as an application-specific data structure that can be used to regenerate the result. In this section, we describe the metadata generation algorithm used in mpiBLAST.

Several databases utilize unique identifiers for each sequence in the database. These identifiers, though unique for each sequence, are not always ordered in the database. The result that is generated from mpiBLAST typically consists of the sequences themselves, together with a significant amount of additional information such as details about the software package, additional information about each sequence, pattern matches that BLAST discovered for each sequence, the e-value representing how close the match was for each sequence, and various others statistics.

ParaMEDIC parses through the results generated by the mpiBLAST compute workers and extracts the sequence identifier information for each matching sequence in the database. Once the sequence identifier information for all the matching sequences in the database

for each query sequence is identified, this data is compressed (e.g., by discarding duplicate GI values) and sent to the I/O workers at the storage site.

We note that as the number of query sequences increase or as the number of result sequences that are required by the user increases, the metadata generated as well as the time for generating and processing the metadata increases too. Thus, at some point, the additional cost of metadata processing becomes greater than the saving in I/O time that is achievable. In order to address this, ParaMEDIC finds the size of the metadata that is created and uses a tunable threshold parameter to determine whether decoupled computation and I/O can provide any benefit. If the metadata size is larger than the threshold, the scheme falls back to the regular mpiBLAST implementation. However, with the large genome database that we used in our current work (containing 16 million sequences), to reach this threshold, the user would have to request for unrealistically large number of matches per sequence. Thus, in our experiments, this threshold is never reached.

C. I/O Post-Processing

I/O post-processing is handled by the I/O workers in ParaMEDIC. The post-processing comprises of two primary components, viz., database creation and query search. In the first component, the I/O master creates a new temporary database based on the *matched segments* that were found by the compute workers. This temporary database is typically *much* smaller than the original database. For example, each query sequence generates at most 250 matched sequences by default (unless the user requests for more matches) which is only 0.001% of the original database. Once the temporary database is created, the I/O workers re-search the original query sequences against this temporary database to generate the final output. Since the temporary database is typically very small, this search time is minimal.

The size of the metadata generated directly impacts the size of the temporary database. Thus, as the number of unique sequences that match the query sequences increases (either as the number of query sequences increases or the number of matches requested by the user increases), the size of the temporary database, and consequently, the I/O post-processing time increases. Again, the threshold (mentioned in section VII-B) ensures that the metadata size in this scheme is small and thus the ParaMEDIC scheme beneficial.

VIII. Experiments, Measurements and Quantitative Results of Solution

The section presents the performance evaluation of ParaMEDIC-enhanced mpiBLAST (hereafter referred to as simply ParaMEDIC) and compares it with the existing mpiBLAST implementation. In Section VIII-A, we first present preliminary performance studies which demonstrate the capabilities of ParaMEDIC in sample environments, including a distributed testbed between Argonne National Laboratory and Virginia Tech over the Internet2 network connection and the Teragrid infrastructure with nodes from University of Chicago and San Diego Supercomputing Center participating in the experiment. Then, in Section VIII-B, we present the detailed quantitative analysis of our experiments directly pertinent to the storage challenge, in a world-wide distributed supercomputing environment.

A. ParaMEDIC Evaluation in Sample Environments

Figure 5 illustrates the performance of 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.

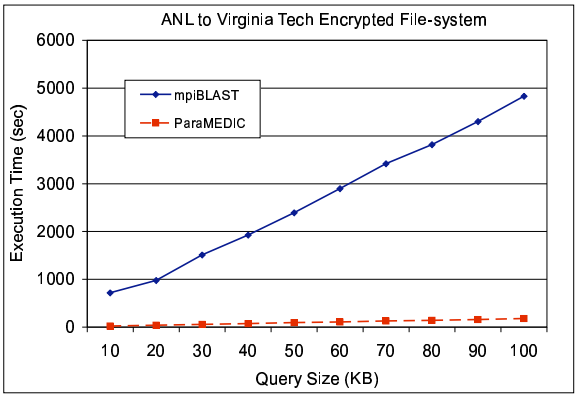


Fig. 5. Argonne National Laboratory to Virginia Tech encrypted file-system

As shown in the figure, ParaMEDIC outperforms mpiBLAST by more than *25-fold* in this environment. This is attributed to multiple aspects. First, given that the network connection between the two sites is shared by other users, the effective network performance 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

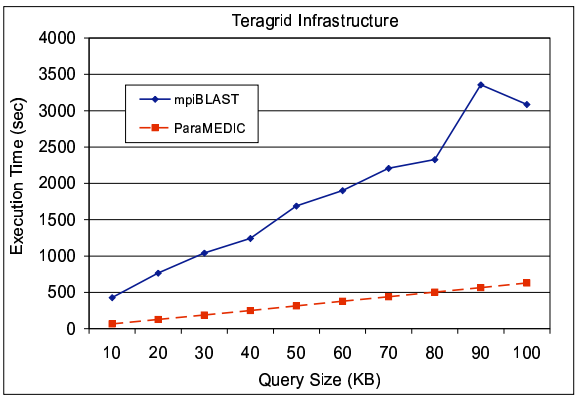


Fig. 6. Teragrid Infrastructure using University of Chicago and the San Diego Supercomputing Center

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 control messages tend to take a long time to be exchanged resulting in further loss of performance.

Figure 6 illustrates the performance of mpiBLAST and ParaMEDIC on the Teragrid infrastructure. Teragrid represents a widely used distributed environment for several compute- and I/O-intensive applications including mpiBLAST. A GPFS-based distributed file-system 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. Nodes at the University of Chicago were configured as compute resources while nodes at SDSC were configured as I/O resources.

For both mpiBLAST and ParaMEDIC, while the final output is written to the same global file-system in both cases, mpiBLAST suffers from the fact that the compute workers are performing the I/O for the output results. Since they reside on a remote cluster as compared to the actual file-system, their I/O performance is limited resulting in an overall degradation in execution time. For ParaMEDIC, on the other hand, since the I/O workers 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 a factor of about *five times*.

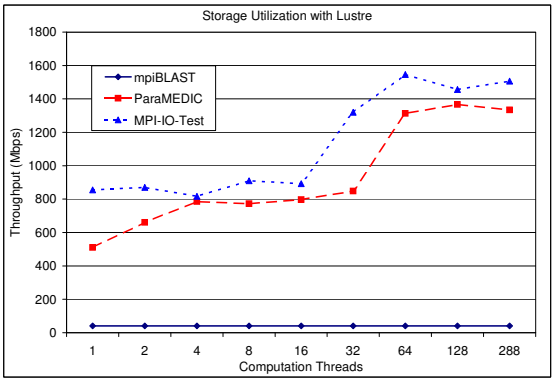
B. Quantitative Analysis on a Worldwide Supercomputer

As described earlier, in this project, we utilize ParaMEDIC to sequence the entire microbial genome database against itself. Given the enormous compute and storage requirements of this project, we utilized a worldwide distributed supercomputer that is powered by our ParaMEDIC framework, to sequence the entire database. Several supercomputing centers (described in Section IV) within the United States were utilized for performing the computation, while the data generated was stored at a large storage resource in Tokyo. In this section, we primarily present data that describes the improvement in storage utilization that we could achieve through ParaMEDIC.

Figure 7(a) illustrates the storage bandwidth utilization achieved by mpiBLAST, ParaMEDIC and the MPI-IO-Test benchmark which 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 as compared to ParaMEDIC. The reason for this is that mpiBLAST requires all of the generated output to be transferred to the remote storage subsystem. In case where the network connectivity is limited (such as across the Internet as it is in this case), the storage utilization suffers. It is to be noted that, though more than 10,000 processors are performing the compute part of the task, the network connecting the compute servers in the United States and the storage system in Tokyo is the bottleneck.

On the other hand, ParaMEDIC is able to utilize close to 100% of the capability provided by the storage subsystem (as demonstrated by the MPI-IO-test benchmark). When the number of compute processes performing post-processing is low (x-axis in the figure), ParaMEDIC utilizes about half of the storage capability. However, as the number of compute processes increases, the I/O utilization of ParaMEDIC increases as well. In fact, with 288 compute threads, ParaMEDIC utilizes more than 90% of the capability of the storage system.

Figure 7(b) illustrates the percentage breakup of the time spent in ParaMEDIC's post-processing phase. As shown in the figure, a significant portion of the time spent is in the I/O part. This shows that inspite 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 utilize a significant portion of the I/O bandwidth provided.



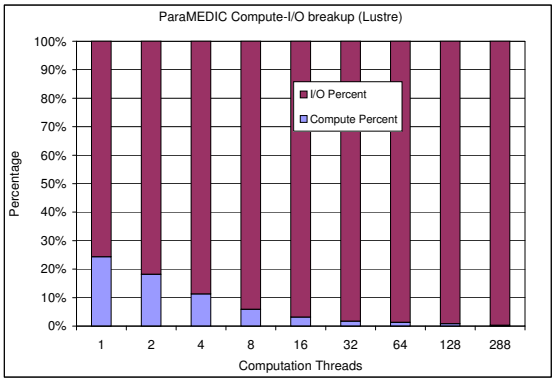
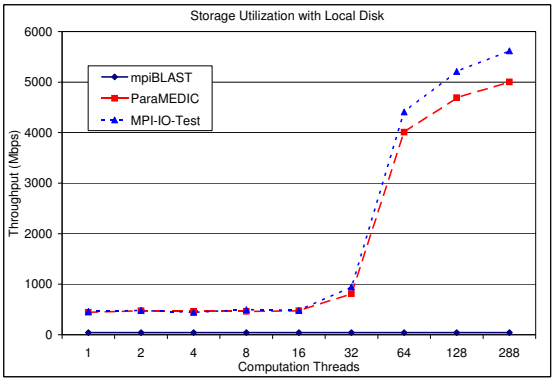


Fig. 7. Storage Bandwidth Utilization using the Parallel Lustre Filesystem: (i) Improvement in Storage Utilization and (ii) Computation and I/O time breakup



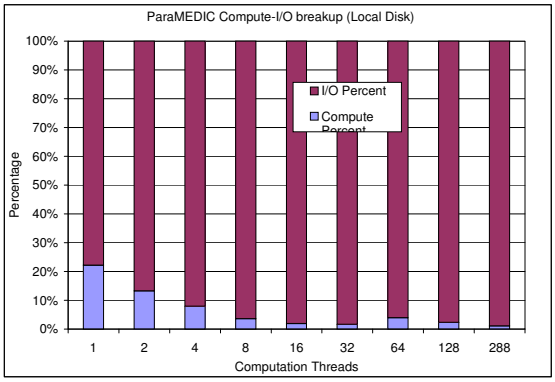


Fig. 8. Storage Utilization using the Local Filesystem: (i) Improvement in Storage Utilization and (ii) Computation and I/O time breakup

Since the performance of networked file-system such as Lustre heavily depends on the capability of the network, in Figure 8(a), we remove the network from the equation and directly perform I/O on the local nodes. We notice that the storage utilization achieved in this case is significantly higher than going over the network. However, even in this case, ParaMEDIC completely utilizes the storage capability provided. Even with a small number of compute threads, we notice that ParaMEDIC is able to utilize more than 90% of the storage capability.

Figure 8(b) shows the percentage breakup of the time spent in ParaMEDIC's post-processing phase. Similar to the case with the Lustre file-system, even in this case, a significant portion of the time is still spent on I/O. Thus, again, ParaMEDIC can be expected to scale and fully utilize even significantly more capable storage resources.

Figure 9 shows the storage utilization of ParaMEDIC with different output formats. Each output format generates a different amount of data; these sizes are sometimes more than a factor of ten apart. However, the amount of computation needed for each format is about the same. Thus, depending on the kind of output format, it is possible that the performance of ParaMEDIC varies. As shown in the figure, while there is some amount of variation, the storage utilization is comparable for all formats. Further, for all formats, ParaMEDIC performs significantly better than the current mpiBLAST imple-

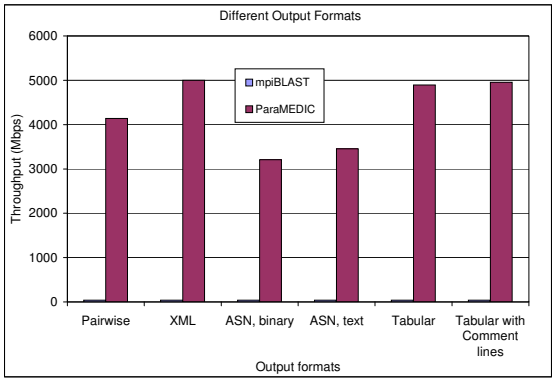


Fig. 9. Storage Utilization of mpiBLAST and ParaMEDIC for different types of BLAST Output formats

mentation can perform.

IX. Conclusions Based on the Experiments

The goal of this project was two fold—(i) Sequence-search all the 567 microbial genomes (that have been completed to date) against each other in order to discover *missing genes* via mpiBLAST sequence-similarity computations, and (ii) Generate a complete genome sequence-similarity tree, based on the above sequence-searching, in order to structure the sequence databases so as to enable pruning of the sequence-search space and, thus, accelerate the sequence-search process. Given the scale and complexity of these problems, a regular brute-force method would simultaneously require a large amount of compute and storage resources together to solve them. While most supercomputing centers provide one of these two requirements (either large compute power or storage), very few manage to provide both. Thus, the entire data generated at one site has to be moved to another site for storage. This is clearly inefficient, especially in distributed environments.

We presented *ParaMEDIC: (Parallel Metadata Environment for Distributed I/O and Computing)*, a novel framework that decouples computation and I/O in applications such as mpiBLAST and drastically reduces I/O overhead through metadata processing. ParaMEDIC trades a small amount of additional computation (in the form of metadata processing) with a significant reduction in the amount of I/O to achieve high performance. We utilized ParaMEDIC to create a worldwide supercomputer with more than 10,000 processors distributed across six supercomputing centers in the U.S. and a storage subsystem in Tokyo. The project generated a petabyte of uncompressed data in one month and wrote a compressed version of this dataset to the 0.5-petabyte filesystem at the Tokyo Institute of Technology. Through ParaMEDIC, we showed that we can increase the utilization of the storage subsystem by several orders of magnitude, since we are no longer limited by the distributed infrastructure network connectivity. In many cases, ParaMEDIC could utilize more than 90% of the capability of the storage subsystem.

X. Claims: Addressing the Judging Criteria

We believe this work has several contributions in the following areas:

Application Performance: The primary goal of the ParaMEDIC framework is to allow mpiBLAST to be relieved off being I/O bound in distributed environments and effectively utilize the available compute resources. Thus, in distributed environments, especially where the compute resources are separated from the file-system hosting server by slow network connectivity, we expect to see a large improvement (several orders of magnitude) in performance.

System and/or Application: While the system utilized is not a large single cluster, it utilizes a large number of globally separated moderate to large-sized clusters. Thus, our approach utilizes multiple independent clusters as a loosely coupled cluster-of-clusters environment, or a worldwide supercomputer.

With respect to the application, over the past three years, mpiBLAST has become an integral component of many high-performance cluster distributions [3], [4], [8], [9], [11], [12], [13], [15] and is an officially supported application at various high-performance computing facilities [17], [18]. With more than 40,000 direct downloads (not including downloads through other packaged distributions such as NPACI Rocks Cluster distribution [10], BioBrew Linux [3] and iNquiry [9]), mpiBLAST has become one of the most popular parallel sequence-search tools used in the community.

Scalability: The primary goal of the ParaMEDIC framework is to remove the I/O bottleneck of mpiBLAST in distributed environments and allow it to scale with the number of compute resources. Though the scalability is not expected to be perfect, due to the additional metadata computation and I/O that needs to be performed, we expect it to be very close to perfect. In fact, as the number of processors grows very large, we expect to close to virtually perfect scalability.

Storage Resource Utilization: Since mpiBLAST is primarily a compute-intensive application, the computation time can be expected to be significantly larger than the I/O time when performed on a local file-system. In the distributed environment, however, the I/O time can increase significantly causing storage under-utilization. To avoid this, in our experimentation, we over-subscribed the number of compute resources to I/O resources by a 100:1 factor. From our tests, about 25:1 ratio seemed to be sufficient to completely utilize the storage bandwidth in our environment. A 100:1 factor not only guarantees a complete storage resource utilization, but also provides enough *extra* buffered data to handle cases where the remote resources become unavailable for some periods of time (e.g., when our allocation of the remote resource has completed).

Innovation: To the best of our knowledge, there is no existing mechanism which utilizes such a metadata based approach to decouple computation and I/O in order to alleviate the I/O bottleneck in distributed systems. Thus, we believe that our approach is a novel contribution in this regard.

Effectiveness: Given the popularity of mpiBLAST and

the current inability of I/O resources to scale equally with compute resources, we believe that ParaMEDIC might be an effective approach for scientists to utilize. Even for local-area environments which provide limited I/O capability (e.g., a BG/L system can provide one I/O node for every 32 or 64 compute nodes), such a framework can help relieve the I/O performance bottlenecks significantly.

XI. Deviations from the Project Proposal

Throughout the life of this experiment several obstacles forced us to make small deviations from the original proposal. These deviations were:

1) Using 567 microbial genomes instead of the NT database.
2) Creating a sequence similarity tree in addition to the BLAST output.
3) Using the storage system at Tokyo Institute of Technology instead of at Argonne National Laboratory.

While there are a few deviations from the original proposal, in reality, these changes are in fact quite small and more importantly do not change the overall spirit of the experiment. Just as we proposed, we have successfully used ParaMEDIC to improve performance when running mpiBLAST on a large-scale bioinformatics problem. The benefit of using ParaMEDIC allowed us to not only perform an "all-to-all" sequence search, but also allowed us to pursue the search for missing genes. Both experiments are unprecedented undertakings requiring computational and storage resources on a scale unfathomable to normal bioinformatics scientists and impossible without a worldwide supercomputer.

XII. Acknowledgments

We would like to thank Rajeev Thakur and Rajkumar Kettimuttu from the Argonne National Laboratory, and Xiaosong Ma from the North Carolina State University for their help and technical support in creating ParaMEDIC. We would also like to thank several members at the Argonne National Laboratory and Virginia Tech for allowing us access on several compute systems for this storage challenge.

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.
[2] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. GenBank. *Nucleic Acids Res.*, 30:17–20, 2002.
[3] BioBrew / NPACI Rocks. http://bioinformatics.org/biobrew/.
[4] Cray. http://www.cray.com/solutions/life/applications.html.
[5] 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.
[6] W. Feng. Green destiny + mpiblast = bioinfomagic. In *International Conference on Parallel Computing (ParCo)*, 2003.
[7] Gold - Genomes Online Database. http://www.genomesonline.org/.
[8] IBM BlueGene. http://researchcomp.stanford.edu/hpc/archives/BlueGene.pdf.
[9] iNquiry. http://www.bioteam.net/.
[10] NPACI Rocks. http://www.rocksclusters.org.
[11] Orion Multisystems. http://www.orionmulti.com/support/faq_mpiblast.
[12] Penguin Computing / Scyld. http://bioinformatics.org/biobrew/.
[13] Rocketcalc. http://www.rocketcalc.com/package.php?Key=15.
[14] SARS Coronavirus: New Insights from Genome Sequence Analysis. http://infectious-diseases.jwatch.org/cgi/content/full/2003/523/1, 2003.
[15] Scalable Informatics. http://www.scalableinformatics.com.
[16] T. F. Smith and M. S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
[17] Terascale Computing Facility. http://www.tcf.vt.edu/.
[18] Teragrid. http://www.teragrid.org/.
[19] O. Thorsen, K. Jiang, A. Peters, B. Smith, H. Lin, W. Feng, and C. Sosa. Parallel Genomic Sequence-Search on a Massively Parallel System. In *ACM International Conference on Computing Frontiers*, May 2007.