Mendel: A Distributed Storage Framework for Similarity Searching over Sequencing Data

Cameron Tolooee, Sangmi Lee Pallickara, Asa Ben-Hur
Department of Computer Science
Colorado State University
Fort Collins, USA
ctolooee@colostate.edu

Abstract—**Rapid advances in genomic sequencing technology have resulted in a data deluge in biology and bioinformatics. This increase in data volumes has introduced computational challenges for frequently performed sequence analytics routines such as DNA and protein homology searches; these must also preferably be done in real-time. In this paper, we propose a scalable and similarity-aware distributed storage framework, Mendel, that enables retrieval of biologically significant DNA and protein alignments against a voluminous genomic sequence database. Mendel fragments the sequence data and generates an inverted-index, which is then dispersed over a distributed collection of machines using a locality aware distributed hash table. A novel distributed nearest neighbor search algorithm identifies sequence segments with high similarity and extending them to find an alignment. This paper includes an empirical evaluation of the performance, sensitivity, and scalability of the proposed system versus the National Center for Biotechnology Information's non-redundant protein dataset. Mendel demonstrates higher sensitivity and faster query evaluations when compared to other modern frameworks.**

Index Terms—**Mendel; sequence alignment; distributed storage system;**

I. Introduction

The emergence of next-generation sequencing technologies has contributed to a dramatic increase in genomic data volumes. The variety of biological analyses such as SNP discovery, genotyping, and personal genomics have posed significant I/O workload challenges. Genomic sequence alignment and homology searching are critical components in genomic analysis. We investigate this problem in the context of similarity-aware distributed hash tables (DHTs) with nearest neighbor searches. DHTs provide efficient, scalable, and robust scale-out architectures where commodity hardware can be added incrementally if there is demand for additional storage or processing.

Sequence alignment is the process of identifying regions in deoxyribonucleic acid (DNA) or protein sequences that are similar as a result of a some biological relationship between the sequences. The similarity between sequences, or lack thereof, can often provide important clues about the functionality and evolutionary origins of genes and other genomic elements. To be able to account for evolutionary changes and sequencing errors, an alignment method needs to perform *inexact* matching. Efficient sequence alignment methods have been actively explored [1]–[3]. These approaches use an algorithmic technique called *seed-and-extend* alignment. However, these tools are designed to run on a single computer, where it may result in prolonged response times or limited sensitivity in the alignments that are found [4]. To improve performance, efforts in parallel and distributed computing setting have targeted the use of message passing interfaces (MPI) [3] and MapReduce frameworks [4]–[7].

We have designed and developed a scalable, similarity-aware distributed storage framework, Mendel, for large-scale genomic sequence analyses. Mendel provides a similarity aware sequence alignment over a voluminous collection of reference sequences using locality sensitive DHTs and an efficient distributed nearest neighbor search (NNS) algorithm. Our sliding window style exhaustive indexing scheme reduces the probability of missing relevant sequences due to variations within the sequences. Our algorithms are tailored particularly for distributed clusters to retain the ability to harness the datacenter (or cloud) storage and computing environments.

A. Usage Scenarios

Metagenomics, also known as environmental genomics, is a powerful tool for analyzing microbial communities in their natural environment without requiring a laboratory culture of the member organisms. The extracted DNA is mapped to known sequences within a database. Next-generation sequencers are capable of producing large quantities of sequence data that current homology search tools, such as BLAST, struggle to process in sufficient time. Our framework can identify significant alignments of the large sampled DNA in an extensive database of sequences. The large volume of data sequencers produce is processed in parallel to produce results faster than BLAST while maintaining high sensitivity.

B. Research Challenges

We consider the problem of scalable, fast, and sensitive search of genomic sequence alignment queries over a large collection of reference sequences. The challenges involved in doing so include:

1) The collection of reference sequences may be voluminous and continues to grow rapidly.
2) Algorithms used in existing systems are not particularly applicable for the cluster computing environment.

3) The queries we consider need to support both DNA and protein sequence data.
4) Existing systems compensate similarity sensitive search for better performance.

C. Research Questions

Research questions that we explore in this paper include:

1) How can we enable scalable indexing over a collection of reference sequences while preserving similarity among the sequences?
2) How can the distributed cluster environment be harnessed to achieve fast query evaluations over voluminous sequencing datasets?
3) How can similarity queries evaluations, rather than exact matching, be performed at scale?
4) Can we achieve these goals while being timely and minimizing user-intervention?

D. Paper Contributions

Here, we present our framework, Mendel, and alignment algorithm for searching and aligning sequences over a large collection of reference sequences that are indexed and dispersed over a distributed cluster. We have extended the NNS data structure vantage point tree (vp-tree) to a distributed storage environment to support similar sequence search at scale. We have designed a inverted indexing scheme to index sequence segments to a DHT while preserving similarity within the vp-tree structure. We also include a refinement of our algorithm to balance the vp-tree to ensure fast traversals over the tree structure during query evaluations.

We propose an alignment algorithm for decomposing the original query into a set of independent sub-queries the results of which are then combined to produce the final results.

E. Paper Organization

The remainder of this paper is organized as follows: section II a description of related works. Section III provides background info on vantage point trees and how they can be adapted to be used for locality sensitive hashing. Section IV describes an architectural overview of the proposed framework. Section V describes data indexing and the query evaluation process. We report on our performance evaluations in section VI. The paper is brought to a close with our conclusions and future work in section VII.

II. Related Work

A. Locality Sensitive Distributed Hash Tables

Locality sensitive hashing in the context of distributed hash tables aim hash similar data items to the same or near by nodes in the DHT indexing space. Hamming DHT [8] leverages work showing similarity between items can be represented by the Hamming distance between their Random Hyperplane Hashing (RHH) identifiers. The Hamming DHT provides a systems that maintains a structure that establishes connections between nodes according the the Hamming distance between their RHH identifiers. This creates a system where small groups of machines hold similar data thus reducing the hops in the decentralized system compared to a traditional DHT network overlay like as Chord.

Other work has be done in effectively distributing multidimensional data using LSH techniques [9], [10]. Many challenges arise when combining these two concepts. There are numerous LSH functions each with their respective abilities and shortcomings, there is no "silver bullet" technique for applying them to a distributed setting. Furthermore, load balancing across a cluster of machines becomes a significant challenge. Because data is now being grouped by similarity, the attributes of the data play a role in their location. If a dataset has high similarity the node(s) assigned to that similar subset may be overworked.

B. Sequence Alignment and Homology Searching

1) Basic local alignment search tool (BLAST): The Basic Local Alignment Tool (BLAST) [1] is one of the most popular tools for homology searching DNA and proteomic sequences. BLAST allows for similarity searches bounded by a threshold value to determine when a sequence does not have sufficient similarity to the query. BLAST uses a word-based heuristic that finds short matches between sequences and extends them to create High-scoring Segment Pairs (HSP) to be used to find an alignment. First, the query sequence is tokenized into k-letter words. Probably variants for each word are generated and BLAST then searches the whole database for exact matches to the generated tokens. Each match is extended in both directions until the accumulated score begins to decrease. HSPs having high enough score are kept; the rest are discarded. The significance of each HSP is evaluated. High scoring HSPs are further extended to find gapped alignments. Because BLAST requires, to some extent, a complete search when looking for exact matches, large numbers of sequences result in poor running times.

2) Other Alignment Tools: Many tools have been developed to improve upon the performance of BLAST [2], [11]. mpiBLAST [3] utilizes the Message Passing Interface (MPI) to parallelize the BLAST algorithm across multiple processes. The BLAST database is distributed onto each of the processing nodes. BLAST searches are then run on each segment in parallel and subsequently aggregating results. While this solution provided superlinear speedups in some cases, its applicability falls short in the context of cloud resources. MPI, in general, performs worse in environments with shared memory over distributed systems. Even more challenges arise when considering the elastic infrastructure that cloud resources provide.

The BLAST Like Alignment Tool (BLAT) is one of the more famous tools that improve on BLAST. By utilizing the lookup speeds of hash tables, BLAT observers speed ups about 50 times faster than BLAST. In doing so, however, it sacrifices sensitivity due to the inherent matching restrictions that hash tables impose.

Ghostx [12] is an alignment tool that utilizes suffix arrays for both the database and queries. It follows the same seed-

and-extend strategy as BLAST: search for seeds of the query in the database, extend the seeds first without gaps, then finally perform a gapped extension. It differs from BLAST in its technique to identify seeds. Ghostx uses suffix arrays with heuristics to prune the searching space. While Ghostx showed substantial performance improvement versus BLAST with similar sensitivity, their approach is designed for a single machine and thus is very memory heavy.

Locality sensitive hashing has also been explored in the bioinformatics community. The LSH-ALL-PAIRS algorithms developed by Jeremy Buhler [13] was one of the first LSH algorithms for finding similarities in genomic databases. LSH-ALL-PAIRS is a randomized search algorithm for ungapped local DNA alignments. A LSH function, $h(X)$, chooses k indices from the sequence at random to form a k-tuple. There is a high probability that two similar sequences will produce the same k-tuple from $h(x)$. This drastically reduces the number of comparisons required to confidently infer similarity between sequences.

C. Bioinformatics in the Cloud

Moving bioinformatics applications to the cloud has been a challenge [14]. There have been efforts to implement the BLAST algorithm in the cloud via MapReduce. CloudBLAST [5] and Biodoop [6] provide the parallelization, deployment, and management of the BLAST algorithm in a distributed environment. CloudBLAST utilizes Apache Hadoop, an open-source implementation of the MapReduce paradigm, to parallelize the execution of BLAST. The approach entailed segmenting the query sequences and running multiple instances of BLAST on each segment. Biodoop takes an opposing approach: distribute the data among computing resources, rather than the computation, and individually take reference sequences to produce alignments with the query sequences. However, both methods see sublinear speedup as the number of compute resources grow.

Our system differs from other relevant methods previously discussed as it targets elastic cloud infrastructures without the dependency on MapReduce implementations such as Hadoop. With the use of LSH and inverted indexing over a distributed hash table, we achieve higher performance with the ability to scale with the rapid growth of sequenced genomic data. Other methods presented in this section either are not designed to scale or scale their solutions by forcing the computation into MapReduce.

III. Locality Sensitive Hashing with Vantage Point Trees

Nearest neighbor search problems are found in many scientific disciplines. NNSs are formulated as an optimization problem for finding objects similar to a target within a set and are typically computationally expensive. They can be used to locate and align target sequences against a reference by searching a single segment versus other sequences. Vantage point trees (vp-tree) [15] provide a method for finding nearest neighbors with logarithmic time bounds on creation and operations with linear space.

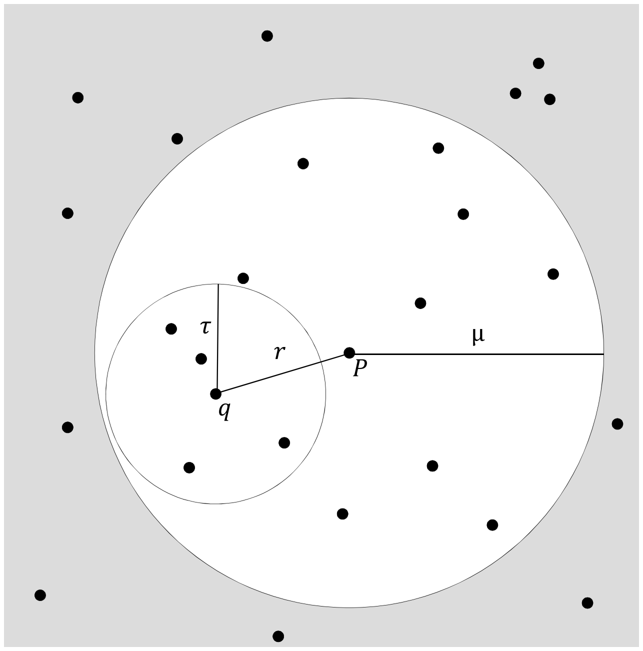


Fig. 1: A visual representation of a vertex, P, and a query, q, in a vp-tree in relation to points in P's left and right subtrees. Black dots in the shaded region represent elements in P's right subtree. While black dots in the non-shaded regions reside in P's left subtree.

A. Background: Vantage Point Trees

A vp-tree is a binary partitioning tree over data in a metric space. The fundamental concept is quite simple: given a set of data and a central data element (vantage point), recursively partition the data points into two divisions: those that are close to the vantage point and those that are not. In other words, elements that are near the parent will be in the left subtree and elements that are far from the parent will be in the right subtree. This creates a binary tree in which neighboring vertices are likely to be close in the metric space.

Each vertex in a vp-tree maintains four values: an center value, a radius μ, a left child, and a right child. Figure 1 shows a graphical representation of a node, P, and a query, q, within a vp-tree. The non-shaded circle, whose radius is labeled μ, represents the distance threshold of the parent node P. All of the elements within the non-shaded circle have a distance to the parent that is less than μ, and thus belong in the left subtree. Conversely, the elements in the shaded region will reside in the right subtree as they have distances to the parent that are greater than μ. The radius of P must encompass roughly half of the data points in order to maintain a balanced vp-tree.

B. Distance Functions

Defining a distance function between genomic sequences has been heavily studied [1], [16], [17]. The vp-tree's requirement for a metric space distance function eliminates many of the prominent scoring techniques used to define the similarity between protein sequences. Amino acid scoring matrices such as PAM [16] and BLOSUM [17] effectively evaluate the quality of alignments, but do not meet the metric space requirements.

In the case of DNA sequences, Mendel uses a simple metric of Hamming distance. Hamming distance [18] is defined as the number of positions between two equal length strings at which the characters differ. Hamming distance satisfies the metric space prerequisites. While trivial to compute, this distance function has some inherent weaknesses. Substitution errors between sequences are effectively captured in the distance, but errors that produce shifts, e.g. insertions and deletions (indels), produce inaccurate distances. Mendel overcomes this challenge with the use of sliding windows to account for shifts; this topic is further discussed in section V.

Finding a metric space distance function for protein sequences is a much greater challenge. Comparing the similarity of amino acids is much more complex. The variance of the average amino acid residues distribution with protein sequences invalidates Hamming distance as a quality measure of distance, even without indels. The most frequently occurring amino acid, Leucine (Leu), appears almost nine times more frequently than Tryptophan (Trp), the most infrequent, according to the September 2015 UniProtKB/Swiss-Prot protein knowledgebase statistics [19]. More specifically, a Trp-Trp match is much stronger than a Leu-Leu match since it is significantly less likely to occur by chance.

Furthermore, non-uniform mutation rates between amino acids create a gradient of possible pairwise similarity scores for mismatches. In comparison to DNA sequences, where bases are classified as a match or mismatch, amino acid mismatches can vary in strength. Point accepted mutations (PAM), are the replacement of an amino acid within a protein sequence that is accepted by natural selection. The PAM matrix, used to score protein sequence alignments, indicates the likelihood of a certain amino acid replacing another [16]. Similarly, the **BLO**cks **SU**bstitution **M**atrix (BLOSUM) is another, arguably more, popular scoring matrix that takes into account the similar factors as PAM, but uses an implicit model of evolution.The BLOSUM62 matrix is a common default scoring matrix in modern alignment applications including BLAST.

Mendel uses the absolute value of the difference between characters as the distance. For instance, for each entry in the BLOSUM62 matrix, $B_{i,j}$, we apply the following element-wise operation to compute the corresponding Mendel distance matrix entry, $M_{i,j}$:

$$M_{i,j} = \left|B_{i,j} - B_{i,i}\right|$$

This operation transforms each column in the lower triangle matrix with respect to the diagonal entry such that each diagonal is zero. This new matrix can be used to define the distance between protein sequences in a metric space with higher accuracy than the Hamming distance function. Because each column is corrected independently, the mismatches retain the same amplitude of penalty versus the exact match. The major trade-off here is that some degree of accuracy is lost in the case of exact matches. All diagonal entries being zero, a requirement for reflexivity, means that the average amino acid composition is not represented in the distance between exact matches. It is important to note that this distance matrix is *not* used to score the actual alignments, instead it is used as a distance function to identify similar sequences in the vp-tree. The matrix used to score the alignments is a user defined parameter.

C. Vantage Point Tree Similarity Search

Searching a vp-tree for the nearest neighbors of some target requires a single traversal. Let q be the query's input point and let τ be a radius around q that will contain q's n nearest neighbors. Initially τ encompasses all points in the tree. At each step of the traversal, we redefine $\tau = min_{s \in S}(d(q, vp), \tau)$. This redefinition allows τ to shrink to a radius around q's nearest neighbors. We observe three possible cases of how τ can relate to the current vantage point in the vp-tree:

1) The area of τ lies *completely* inside of the area of μ;
2) the area of τ lies *completely* outside of the area of μ;
3) the areas of τ and μ intersect.

In the first case, depicted by the point q in Figure 1, all of q's nearest neighbors are guaranteed to be within the area defined by μ, thus the right subtree does not contain any of the nearest n neighbors and can safely be omitted in the search. The second case is just the opposite: q's n nearest neighbors would lie outside the area defined by μ. Therefore, for the same reason, the left subtree can be omitted in the search. Finally, in the worst case, if τ and μ's areas intersect, q's nearest neighbors can potentially be in both subtrees and, thus, the search space is not reduced. The computational complexity of searching for nearest neighbors is $O(log(n))$ in the average case since each search is ultimately the traversal of a path from root to leaf in a binary tree.

D. Performance Improvements

The original vp-tree can be altered to achieve better performance in terms of memory usage and execution time [15]. Two major optimizations can be made: (1) add buckets at each leaf to increase the tree's capacity and (2) creating upper and lower bounds at internal nodes on the subspaces as seen by the ancestral vantage point. Adding large buckets to the leaves of the vp-tree, contrast to each leaf maintaining only one element, vastly reduces the total number of vertices, especially with large number of elements.

One major challenge with genomic datasets and vp-trees we discovered is that the dataset in its entirety must be present and inserted at the time of creation. The original data structure did not support single element insertions. Naïvely inserting subsequences one-at-a-time quickly leads to an unbalanced tree. When data volumes grow large this imbalance resulted in linear running times which impacted performance substantially. The dynamic indexing problem for vp-trees breaks down into four cases when updating the tree [20]:

1) Leaf node bucket is *not* full:
   - Add to bucket
2) Leaf node bucket is full, but sibling node has room:
   - Redistributed all values under the common parent
3) Leaf and sibling nodes are full, but there exists an ancestor node whose subtree has room:
   - redistribute all values under the common ancestor
4) Completely full tree:
   - Split the root into two
   - Apply case (2) or (3) as needed

To help alleviate the added complexity of element insertions we strike a middle ground by adding elements in large batches, instead of individually, which maintains an acceptable performance while maintaining an optimized, balanced vp-tree to use as an NNS data structure.

E. Vantage Point Tree as a LSH Function

Utilizing a vp-tree as the data structure for voluminous datasets presents new challenges. Biological datasets can contain billions of items to act as elements in a vp-tree. Storing all of the elements in a single, memory resident data structure is not feasible when the datasets grow large. The vp-tree can be augmented by adding a binary prefix to each node within the tree. The value of the prefix of a given node is computed as follows: the root has a prefix of 1 and child vertices will left shift its parent's prefix by one, and add 1 if it is a right child. This small modification gives nodes an integral value that uniquely represents the path taken to get there. This creates some degree of integral relationship between node prefixes and the metric distance between them.

F. Vantage-Point Prefix Tree Hashing

The vp-prefix tree does not alone create a good hashing function as there are many problems with the proposed hashing scheme. Maintaining a vp-tree for the entire dataset at this scale is non-trivial. Also, searching over a large vp-tree creates a memory intensive task that causes a severe bottleneck when hashing numerous items.

A cutoff threshold depth is imposed to coarsely index data into similar groups. After the threshold depth has been reached, the traversal stops and the hash value is computed from there based on the prefix. This will create a hash function that produces collisions when two data points are similar. The hierarchical two-tiered DHT utilizes collisions as a way to group similar data for query evaluation. Figure 2 shows a small example of how a vp-prefix tree might hash data into a groups.

IV. System Architecture

A. Distributed Hash Tables

Prominent distributed storage systems such as Amazon Dynamo [21] and Apache Cassandra [22] both utilize the DHT paradigm as their underlying infrastructure. As the name suggests, DHTs employ similar insertion and retrieval mechanisms to that of a hash table: key-value storage and lookup. Each node is partitioned onto a logical keyspace typically using a flat hashing scheme. Subsequently, data points are hashed using a unique key to the same keyspace in order to determine its storage node.

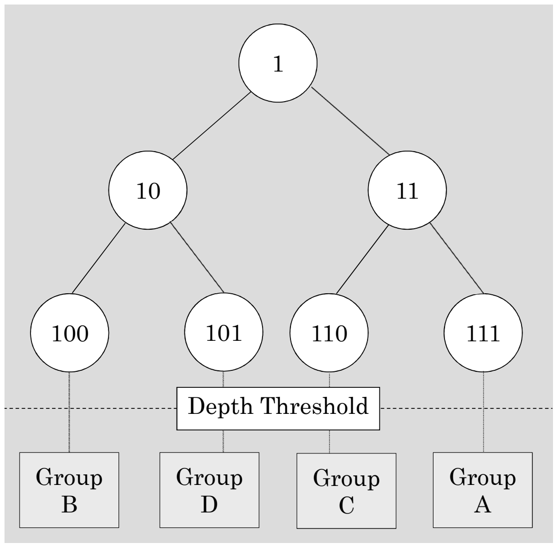


Fig. 2: A vp-prefix tree being used as a group hash function with a depth threshold of 3. The depth of the threshold effectively determines the resolution of similarity that each group maintains.

DHTs do not come without a slew of their own problems and challenges. Like a hash table, lookups are inherently limited to exact match queries. Data cannot be retrieved without the unique key the data was indexed with. Expressive queries such as wild card, range-based, or approximate queries are not possible with the basic DHT design. There have been many attempts on overcoming this challenge with the uses of locally-sensitive hashing or hierarchical DHTs [8], [23]. Also, the decentralization requirements increase the complexity of routing requests. Having each node maintain locations for all nodes in the cluster, introduces challenges when nodes leave and join. Conversely, maintaining relationships to only portions of the cluster adds complexity via routing protocols thus increasing request latency.

B. Inverted Indexing

Many widely used large-scale data storage systems utilize inverted indexing as a central component to achieve timely query results. An inverted index is a data structure used to locate content quickly by mapping content to its location in a database or documents. This is contrary to the traditional forward index which records the content of each document. Inverted indexing is ideal for data that has content disproportional to the number of documents containing it and in scenarios where data is inserted infrequently and queried often.

Figure 3 shows a simple example of an inverted index over text documents. In this toy example, the database contains

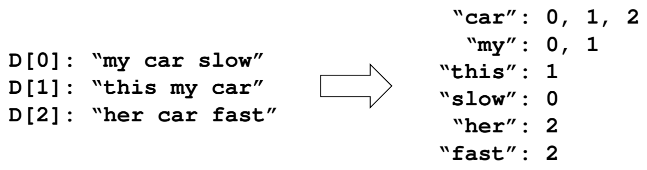


Fig. 3: A small demonstration of an inverted index over three documents. Each word is indexed by the document(s) it is found in.

three documents. A lookup query for the search terms "fast car," for example, would compute the intersection between the individual queries "fast" (D[2]) and "car" (D[0], D[1], D[2]) to return location D[2]. Without inverted indexing, the same query would require a sequential iteration of all three documents to find documents matching all search terms.

In the context of sequence alignment, treating segments of the reference sequences as the content and treating the segment's location in the sequence analogous to the database location, an inverted index can be used to find an alignment of the query segment to its position in the reference sequence. Since query sequences are short in comparison to the genomes being searched over, this creates an optimal environment to apply inverted indexing.

There are a few significant shortcomings of utilizing an inverted index alone to find alignments. Most notably, inverted indices mandate perfect matches between the target and the reference. If even one character in the sequence differs from the indexed segment, there will not be an initial match during the lookup and no results will be found. This also severely limits the expressive capabilities of a query as the exact match requirement constrains it to a specific length. Accounting for both sequencing errors and genomic structural variation are essential to a sequence similarity search tool. Mendel resolves these issues with the use of sliding windows and NNSs over vp-trees to allow for variable length queries without the exact match limitation.

C. Network Topology

Mendel's network overlay topology is organized as a zero-hop DHT. The class of zero-hop DHT's, such as Amazon Dynamo, provide enough state at each node to allow for direct routing of requests to their destination without the need for intermediate hops.

Mendel deviates from the standard DHT in that it employs a hierarchical partitioning scheme. Each storage node within the system is placed in a group. The size and quantity of groups are a user-configurable parameter that can be adjusted to best fit the data stored. This scheme leverages the vp-prefix tree to coarsely hash data elements to groupings of nodes. A second flat hash will index the data among its group evenly to maintain a good load balance to avoid data hotspots. The two-tiered partitioning structure, where data is first placed in groups among similar data, then hashed within that group, increases the efficiency of retrieval operations by reducing the search space to only similar data.

V. Indexing and Query Evaluation

A. Inverted Indexing Blocks

To combat the exact match challenges of inverted indexing, a series of sliding windows and locality sensitive hashes are used to index sequences in a manner that can be queried without the equality restriction. Each sequence to be inserted into the system follows three steps to be successfully indexed: (1) inverted index block creation, (2) vp-prefix tree sequence dispersion, and (3) local vp-tree indexing.

1) Inverted Index Block Creation: In the first phase, segments of the sequence are created from the input data. The sequences are iterated with a k-length sliding window producing $L-k$ segments per sequence, where L is the sequence length. These segments, called *inverted index blocks*, are the basic unit of computation and storage in the system. By analyzing these blocks with NNS data structures, queries can be accurately evaluated even if they are of variable lengths or contain mismatches. Metadata, including sequence ID, start/end positions, and references to the previous/next blocks, is obtained here to be used during query evaluation. Batches of inverted indexing blocks are accumulated as the input data is parsed and are submitted in sets to the vp-prefix hash tree for distribution among the cluster.

2) Vp-Prefix Tree Sequence Dispersion: Each block is hashed independently using the vp-prefix tree indexing scheme to determine the its storage group. Using this group hashing system, sequences with similar structures will be collocated within the same group. The depth threshold is set to half the tree's depth to strike a balance between timely calculation of hash values and achieving a balanced distribution of data over the cluster.

When blocks arrive at a storage group the individual storage node must still be calculated. Employing a second-tier vp-prefix hashing tree at this level proved to be ineffective. Load balancing became significantly harder to achieve with a finer grain vp-prefix tree hash. During large insertions the indexing tree was frequently updated and redistributed requiring a choice between trade-offs: relocating data between nodes during updates to maintain a balanced tree, versus keeping an unbalanced tree but creating hotspots within the groups. Neither option yields promising performance. Furthermore, we want to exploit the inherent parallelism during large, computationally expensive queries. Grouping similar blocks onto the same node drastically reduces the amount of parallelism thus hindering performance.

Instead, Mendel use a tried-and-true flat hashing scheme, SHA-1, to disperse the blocks *within* a group. The trade-off being queries must be replicated to all nodes within a group since any node may have a matching block. Load balancing within groups will be near optimal with a flat hashing system. Because of this, it is highly likely that all nodes within a group contain relevant blocks to any query assigned to that group, optimizing the group-wide parallelism during large queries.

3) Local vp-Tree Indexing: Finally, once an inverted index block reaches its destination storage node within its storage

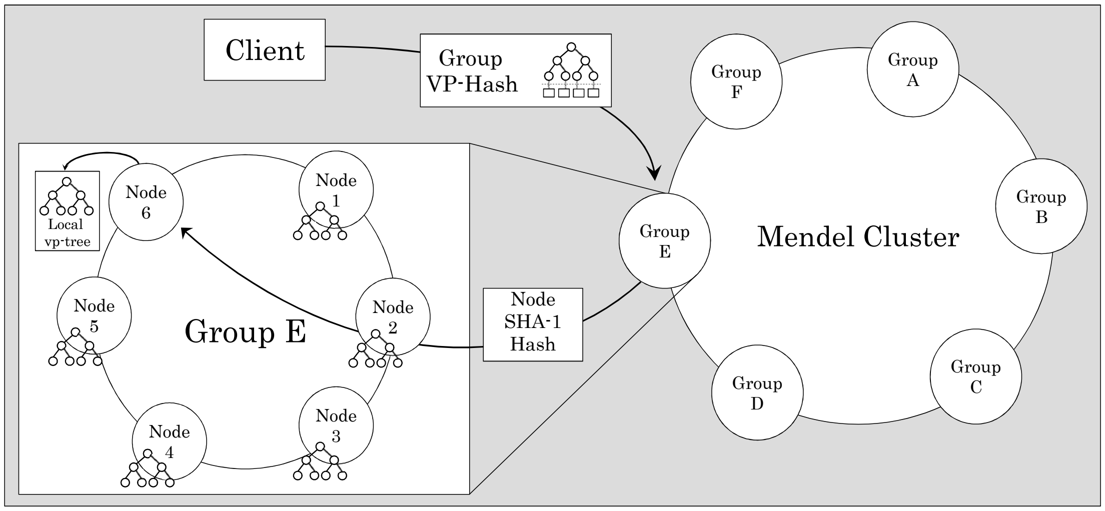


Fig. 4: An illustration of the network topology and data flow of a Mendel cluster. Each inverted index block is hashed to a predefined group of storage nodes using the vp-prefix tree hash. Within its group, the data is hashed a second time using a SHA-1 hash to distribute data among the group evenly.

group, it will be indexed in a regular local vp-tree that contains all blocks the storage node maintains locally. This vp-tree is implemented using dynamic update balancing, thus further optimizing query performance in exchange for additional preprocessing. This memory-resident NNS structure serves as a starting point for queries to find high similarity segments to begin the sequence alignment analysis.

B. Query Evaluation

Mendel strives to emulate the prompt responsiveness that DHTs provide along with the ability to conduct robust queries. During query evaluations, the target query sequence(s) will pass through a series of steps similar to data insertions to determine storage node groups that are likely to have relevant results.

Initially, when a query enters the system, the storage node that receives the query will be tracked as the query's entry point. This framework supports a symmetric architecture: any node in the cluster can perform as a query's entry point and generates identical results. Query entry points, at both the system and group levels, are utilized as query coordinators for result aggregation checkpoints. Much like the indexing stage, a sliding window process is performed over the query sequence. This normalizes the query into subqueries that are the same length as the indexed data. The sliding window here, however, steps over the query sequence in larger intervals of size k, rather than of size one, to reduce the amplification of the subqueries. Using vp-prefix tree hash function, each target query subsequence is hashed to determine the groups within the system that may contain relevant subsequences. Notably, multiple groups can be selected from the vp-hash tree if the path branches while traversing the tree. In this case, the subquery is replicated to both groups.

Each group receiving a subquery will be tracked as the query's group entry point. Since the data blocks within the group were distributed using a flat hash, any node has the possibility of having a high scoring match. Thus, the query block is replicated to all nodes within a group in parallel. For each segment of the query that reaches an individual storage node, a local vp-tree lookup is performed. Using parameters defined within the query, the n nearest neighbors to the subsequence are added to a candidate list of possible matches. Two measures are computed for each candidate: (1) a percent identity score, computed as $\frac{hamming(subsequence, candidate)}{length(candidate)}$ and (2) a consecutivity score, *c-score*, that calculates from the existing matches the percent of those matches that are in succession. The c-score provides a metric to identify strong partial matches. For protein sequences, substitutions to which the BLOSUM62 matrix gives a positive score are considered as successive. The query specifies minimum c-scores to be considered. Candidates with a score lower than that threshold are dropped from the candidate list. The remaining matches are used as anchors to be extended.

Each inverted index block maintains references to its neighboring blocks. This allows the expansion of the anchors in both directions to lengthen them. Starting with the segment previous to the match, the sequence is incrementally extended until the extension deteriorates the score of a match below the threshold. This expansion is done on both sides of the match to create an anchor for the alignment. The diagonal of the anchor (the difference between the starting positions of the database and query sequences) is recorded and each anchor is then categorized by its sequence ID; binning matches with other anchors from the same sequence. The bins are sorted by the anchor start position to create a set of categorized anchors.

The first aggregation stage occurs at each query group entry point. All nodes in the group send their expanded anchor set to the group entry point to combine overlapping anchors on the same diagonal. A similar step is repeated at the system entry point: all group coordinators send their matching segments to the system coordinator. Again, any overlapping anchors on the same diagonal are combined and scored. Finally, to identify potential gapped alignments from a bin of extended anchors, we follow a similar approach to that of Gapped BLAST [11]. For each anchor having a normalized score greater than some threshold S, a gapped extension is performed. The gapped extension considers all anchors from the same sequence within l diagonals in either direction. If the resulting gapped extension has an expectation value, E, low enough to be of interest, it will be included in the final report of alignments. Finally, all results are scored according to the specified scoring matrix, ranked by expectation value, and returned to the client. All the different query parameters with brief descriptions are outlined in table I.

TABLE I: Query Parameters

Parameter	Description	Type
k	Sliding window step	int(1..∞)
n	No. of nearest neighbors to find	int(1..∞)
i	Identity threshold	float(0..1)
c	Consecutivity score threshold	float(0..1)
M	Scoring Matrix	string
S	Score threshold for gapped extension	float(0..∞)
l	Gapped alignment band width	int(0..∞)
E	Expectation value threshold	float(0..∞)

VI. Performance Evaluation

To benchmark the effectiveness of Mendel, we ran several tests to simulate application usage on a heterogeneous cluster. We targeted four main aspects in our tests: (1) the performance of the vantage point prefix tree as a LSH function, (2) query turnaround time, (3) the sensitivity achieved, and (4) the scalability of our system with respect to data volume and number of nodes.

A. Experiment Environment

1) Cluster Setup: The testing environment consisted of a 50-node heterogeneous cluster composed of 25 HP DL160 servers (Xeon E5620 12 GB RAM, 15,000 RPM disk) and 25 Sun SunFire X4100 servers (Opteron 254, 8 GB RAM, 10,000 RPM disk) connected over a LAN. All machines are running Fedora 21 (Twenty One) and OpenJDK 1.8.0.

2) Datasets: Protein sequences were sourced from the National Center for Biotechnology Information (NCBI) genomic database. The datasets included the non-redundant protein (`nr`) containing 73,021,224 reference sequences and two smaller whole genome query sets (`s_aureus` and `e_coli`).

B. Data Distribution and Load Balancing Evaluation

Our first benchmark aims to test the load distribution of Mendel. We indexed the 100 GB of genomic data over the 50-node cluster. The percentage of total system data being stored at each node was recorded. Figure 5 shows the load balance using our hierarchical hashing topology in comparison to a standard flat hash. While this data distribution is not as balanced as the SHA-1, the difference between single nodes never exceeds 1% of the total data volume stored. The load balancing within groups maintains a near perfect distribution since a SHA-1 hash is used for the inter-group data dispersion. This is also observed in the clustering of groups; the group configuration of size five, is evident in the figure.

C. Query Performance

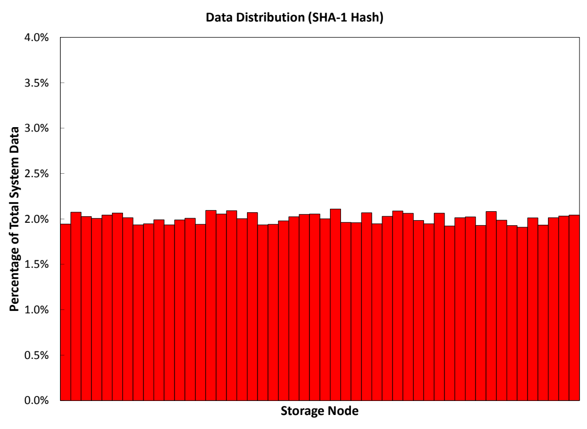
We looked at two different aspects of the data and its impact on the performance. First, the length of a query plays an important role in the overall performance of sequence similarity searches. Large query lengths create substantially more processing than that of a smaller ones. We carried out an experiment to measure the impact query length has on Mendel versus BLAST. We ran NCBI's BLAST+ version 2.2.31 for these benchmarks. According to an analysis by George Coulouris of several hundred thousand BLAST queries run at the National Institutes of Health, 90% of BLAST protein sequence queries are less than 1000 amino acid residues in length [24]. We executed queries from the `s_aureus` query set with target sequence lengths ranging from 500 to 3000 residues over the `nr` dataset. Figure 6a shows average turnaround times for the various queries. The length of an alignment query has little effect on the overall performance in Mendel. Another essential component of performance is the volume of the data being searched over. We conducted an experiment to test this aspect by fixing the length of the queries to 1000 residues and incrementally increasing the database size; measuring the average query response times. Figure 6b shows the results of this benchmark. Database size has a less impact on the performance of the system in comparison to BLAST. We observe nearly constant average turnaround times The DHT design can accommodate very large volumes of data before the impact of performance is observed. While BLAST can maintain sufficient performance when the database is memory resident, progress comes to a halt when the data volumes grow large. The support for incremental scalability allows users to tailor the cluster to their specific needs.

D. Scalability

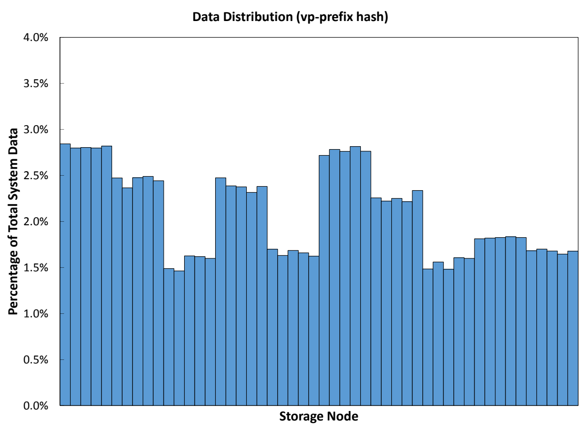
The scalability of Mendel is essential to be able to cope with growing rates of genomic data. The system should be able to cope with large volumes of data while maintaining acceptable performance. Figure 6b shows the hash-table like query performance as the data volume grows. Performance improvements should also be observed as the amount of resources increase. To test how well the Mendel scales with resources, we indexed the `nr` dataset over clusters of varying sizes and measured the average turnaround time for the `e_coli` query set for each cluster size. Figure 6c shows a sufficient scalability with respect to the size of the cluster.

E. Query Sensitivity

The final experiments we conducted concern the sensitivity of our system. Sensitivity is a pivotal component to sequence alignment. Fast results are near useless if they are inaccurate. Sensitivity in this context can be defined as the likelihood of finding high scoring alignments providing they exists. Finding

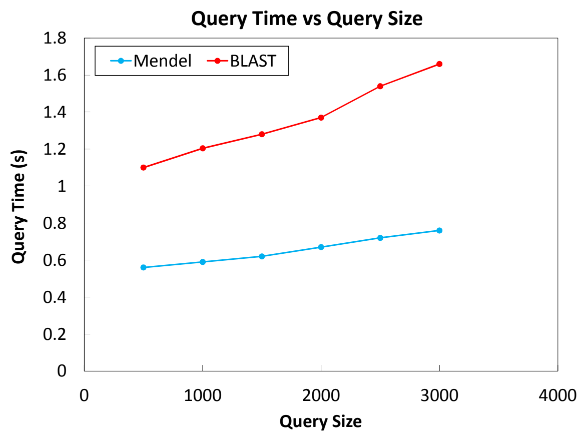


(a)

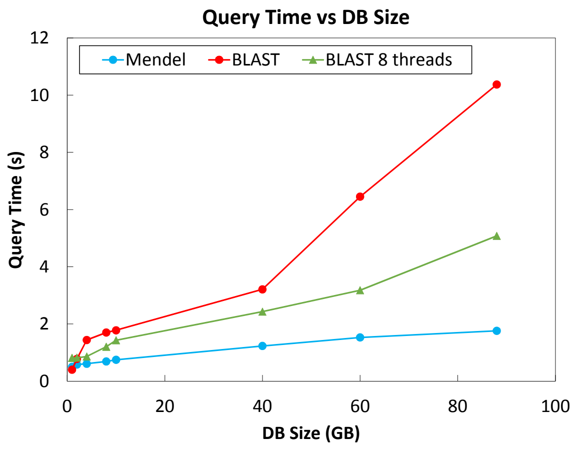


(b)

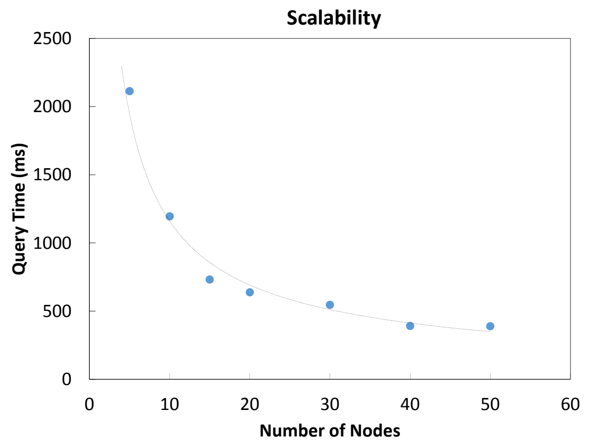
Fig. 5: A comparison between the load balancing of a standard hash function (a) versus our two-tiered vantage point LSH hashing scheme (b)



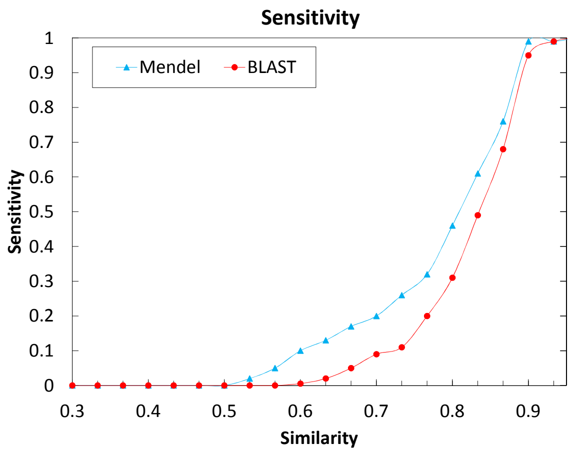
(a)



(b)



(c)



(d)

Fig. 6: Various performance benchmarks of our proposal versus BLAST. (a) and (b) show how the performance doesn't degrade as the different inputs grow large. Plot (c) shows the scalability of the system as nodes are added to the cluster. The sensitivity is shown in plot (d) in comparison with BLAST.

the balance between performance and sensitivity is a key issue in sequence similarity searching. The final benchmark we conducted involved finding the sensitivity limits of our solution. We generated a 1000 amino acid residue target sequence to be the starting point in the sensitivity measure. At decreasing similarity levels, groups of sequences are generated by randomly mutating residues from the original sequence corresponding to the desired similarity level. For each similarity level, an all versus all query is conducted and the percentage of matches found was recorded. Figure 6d displays the results of the experiment. The NNS algorithm overcomes the challenge of finding alignment when the similarity is low. Since the NNS is able to identify larger seeds that may be missed in other systems it can better identify lower similarity matches.

VII. Conclusions and Future Work

A. Conclusions

We have proposed a novel distributed system, Mendel, aimed at efficiently conducting similarity searches of DNA and protein sequences versus a large genomic database. We approached this problem with a distributed systems mindset to tackle the computational challenges associated with sequence alignment at scale. Inverted indexing is a known solution to the genre of indexing problems where there is a disproportion between content and the locations that hold it. By applying and inverted index over the sequence data in a distributed hash table, we can efficiently identify small similar segments. We modified a NNS data structure, the vantage point tree, as a way to create a locality sensitive hash function over inverted indexing sequence segments into a distributed hash table. Grouping similar inverted indexing blocks into the same cluster group allows substantial reduction in the search space needed to find matching segments for alignment queries. The same base NNS data structure is used to find the local data on each individual storage node that is matching to a certain threshold. By using these matching segments as an anchor for extension, similar subsequences can be identified. Our benchmarks exhibit performance improvements in runtime, sensitivity, and scalability over other modern sequence alignment tools.

B. Future Work

Currently, many aspects of the system configuration require user intervention with an in-depth knowledge of the Mendel framework and are difficult to change on-the-fly. Also, indexing times for exceedingly large datasets can be inhibitive. Adding the ability to save pre-indexed data for popular large datasets, such as the non-redundant protein (nr) or reference sequence (refseq_protein), for various cluster sizes would save researchers a lot of time. There are also a few aspects of the distributed environment that are left unchecked. Providing a fault tolerant system, in terms of data integrity as well as jobs completion, is a key part that warrants our attention. With the growing popularity of personal genomics security concerns become more prevalent; especially in a public cloud settings.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[2] W. J. Kent, "Blatthe blast-like alignment tool," *Genome research*, vol. 12, no. 4, pp. 656–664, 2002.

[3] A. Darling, L. Carey, and W.-c. Feng, "The design, implementation, and evaluation of mpiblast," *Proceedings of ClusterWorld*, vol. 2003, pp. 13–15, 2003.

[4] M. C. Schatz, "Cloudburst: highly sensitive read mapping with mapreduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[5] A. Matsunaga, M. Tsugawa, and J. Fortes, "Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 222–229.

[6] S. Leo, F. Santoni, and G. Zanetti, "Biodoop: bioinformatics on hadoop," in *Parallel Processing Workshops, 2009. ICPPW'09. International Conference on*. IEEE, 2009, pp. 415–422.

[7] M. Brock and A. Goscinski, "Execution of compute intensive applications on hybrid clouds (case study with mpiblast)," in *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 995–1000.

[8] R. da Silva Villaca, L. B. de Paula, R. Pasquini, and M. F. Magalhães, "Hamming dht: Taming the similarity search," in *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*. IEEE, 2013, pp. 7–12.

[9] P. Haghani, S. Michel, P. Cudré-Mauroux, and K. Aberer, "Lsh at large-distributed knn search in high dimensions."

[10] B. Bahmani, A. Goel, and R. Shinde, "Efficient distributed locality sensitive hashing," in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 2174–2178.

[11] S. F. Altschul, T. L. Madden, A. A. Schäffer, 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 research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[12] S. Suzuki, M. Kakuta, T. Ishida, and Y. Akiyama, "Ghostx: An improved sequence homology search algorithm using a query suffix array and a database suffix array," *PLoS ONE*, vol. 9, no. 8, p. e103833, 2014.

[13] V. Satuluri and S. Parthasarathy, "Bayesian locality sensitive hashing for fast similarity search," *Proceedings of the VLDB Endowment*, vol. 5, no. 5, pp. 430–441, 2012.

[14] C.-H. Hsu, C.-Y. Lin, M. Ouyang, and Y. K. Guo, "Biocloud: cloud computing for biological, genomics, and drug design," *BioMed research international*, vol. 2013, 2013.

[15] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *SODA*, vol. 93, no. 194, 1993, pp. 311–321.

[16] M. O. Dayhoff and R. M. Schwartz, "A model of evolutionary change in proteins," in *In Atlas of protein sequence and structure*. Citeseer, 1978.

[17] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proceedings of the National Academy of Sciences*, vol. 89, no. 22, pp. 10 915–10 919, 1992.

[18] R. W. Hamming, "Error detecting and error correcting codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[19] U. Consortium *et al.*, "Uniprot/swiss-prot release 2015_09 statistics," 2015, accessed: 09-19-2015.

[20] A. W. chee Fu, P. M. S. Chan, Y. ling Cheung, and Y. S. Moon, "Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances," *VLDB Journal*, vol. 9, pp. 154–173, 2000.

[21] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," vol. 41, no. 6, pp. 205–220, 2007.

[22] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[23] M. Malensek, S. L. Pallickara, and S. Pallickara, "Expressive query support for multidimensional data in distributed hash tables," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*. IEEE, 2012, pp. 31–38.

[24] G. Coulouris. (2013) Blast benchmark. [Online]. Available: http://fiehnlab.ucdavis.edu/staff/kind/Collector/Benchmark/Blast_Benchmark