THESIS

ON THE USE OF LOCALITY AWARE DISTRIBUTED HASH TABLES FOR HOMOLOGY SEARCHES OVER VOLUMINOUS BIOLOGICAL SEQUENCE DATA

Submitted by

Cameron Tolooee

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2015

Master's Committee:

Advisor: Sangmi Pallickara

Asa Ben-Hur
Joseph von Fischer

ABSTRACT

ON THE USE OF LOCALITY AWARE DISTRIBUTED HASH TABLES FOR HOMOLOGY SEARCHES OVER VOLUMINOUS BIOLOGICAL SEQUENCE DATA

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. This thesis proposes 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 splices them together to form an alignment. This paper includes an empirical evaluation of the performance, sensitivity, and scalability of the proposed system over the NCBI's non-redundant protein dataset. In these benchmarks, Mendel demonstrates higher sensitivity and faster query evaluations when compared to other modern frameworks.

TABLE OF CONTENTS

Chapter 1

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. This chapter will introduce the problem of sequence alignment and discuss the research challenges and contributions, as well as outline the rest of the chapters for the thesis.

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, 2, 3]. These approaches use an algorithmic technique called *seed-and-extend* alignment. Seed-and-extend mapping starts by finding a small seed that matches a substring in both the query and reference sequences, and then extends the matching seed to allow mismatches or gaps within thresholds. There are various methods for finding and extending the seeds. 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]. Other algorithmic approaches have relied on using either a suffix tree or an enhanced suffix

array [5]. To improve performance, efforts in parallel and distributed computing setting have targeted the use of message passing interfaces (MPI) [3] and MapReduce frameworks [4, 6, 7, 8].

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. All sequence alignments used in our approach rely on similarity-sensitive alignment and 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.

1.1 Usage Scenarios

Metagenomics, also known as environmental genomics, is a powerful tool to analyze microbial communities in their natural environment without requiring a laboratory culture of the member organisms. 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. Gene prediction is a process in comparative metagenomics that uses homology searches versus known sequences to help identify genes. Our framework can identify significant alignments of the voluminous DNA samples in an extensive database of sequences. The large volume data sequencers produce are processed in parallel to produce results faster than BLAST while maintaining high sensitivity.

1.1.1 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, RNA, and protein sequence data.
4. Existing systems compensate similarity sensitive search for better performance.
5. Different sequencing methods have different typical types of errors.

1.1.2 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?

1.2 Thesis 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.

1.3 Thesis Organization

The remainder of this thesis is organized as follows: The following chapter describes other related works. Chapter 3 provides background info on vantage point trees and how they can be adapted to be used for locality sensitive hashing. Chapter 4 reviews the principles of distributed hash tables and inverted indexing, then describes an architectural overview of the proposed framework. Chapter 5 describes data indexing and the query evaluation process. We report on our performance evaluations in chapter 6. The thesis is brought to a close with our conclusions and future work in chapter 7.

Chapter 2

Related Work

2.1 Locality Sensitive Distributed Hash Tables

Locality sensitive hashing in the context of distributed hash tables aim to hash similar data items to the same or near by nodes in the DHT indexing space. Hamming DHT [9] 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 been done in effectively distributing multidimensional data using LSH techniques [10, 11]. 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.

2.2 Sequence Alignment and Homology Searching

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. Probable 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.

Other Alignment Tools

Many tools have been developed to improve upon the performance of BLAST [2, 12]. 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 [5] 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.

2.3 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 [6] and Biodoop [7] 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.

Mendel 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.

Chapter 3

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 small segments of the target sequence versus small segments of the reference sequences to find similar pairs. Vantage point trees (vp-tree) [15] provide a method for finding nearest neighbors with logarithmic time bounds on the data structure creation and operations with linear space. In this chapter, we will review the original vantage point tree data structure, including metric space requirements and how they apply to sequence data. We will also discuss the adaptations made to use this NNS data structure as an efficient LSH function.

3.1 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 sets: those points 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 in which they are embedded.

Each vertex in a vp-tree maintains four values: a center value, a radius μ, a left child, and a right child. Figure 3.1 shows a graphical representation of a node P, and a query q

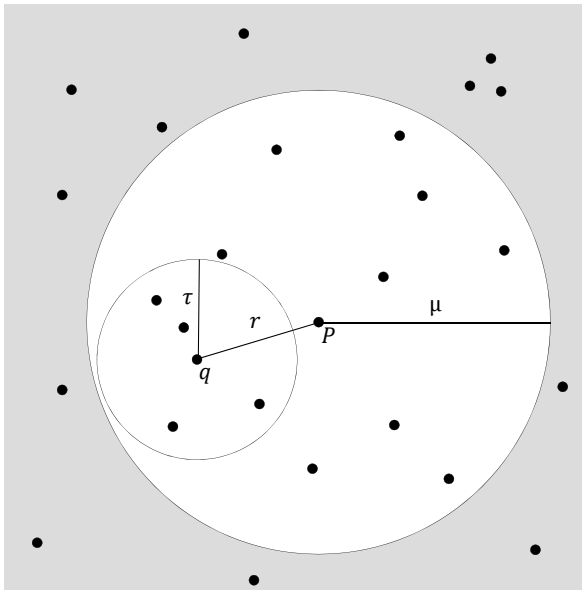


Figure 3.1: A visual representation of a vertex P, and a query q, in a vp-tree in relation to points in P's left and right subtrees. Black dots in the shaded region represent elements in P's right subtree. While black dots in the non-shaded region reside in P's left subtree.

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 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.

3.1.1 Metric Spaces

A metric space, informally, is defined by a set of objects and a metric, or distance function, between them such that all objects in the set have a distance between each other. More formally, in order for a set to occupy a metric space, the following properties must be satisfied. Given a metric space (S, d), where S is a set and d is the distance function, for any elements x, y, z in S:

1. **Reflexivity**: $\forall x \in S, d(x, x) = 0$

2. **Symmetry**: $\forall x, y \in S, d(x, y) = d(y, x)$

3. **Triangle Inequality**: $\forall x, y, z \in S, d(x, y) + d(y, z) \geq d(x, z)$

4. **Non-negativity**: $\forall x, y \in S, d(x, y) \geq 0$

The go-to example for metric spaces is Euclidean space. For example, points in a two-dimensional plane. The distance between two points in Euclidean space is the length of the line segment connecting them. This distance, which can be computed using the Pythagorean formula, satisfies all four of aforementioned properties of a metric space.

3.1.2 Distance Functions for Sequence Data

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 because they are measuring similarity instead of distance. For example, in both matrices identical sequences produce scores greater than zero thus violating the reflexivity requirement. Sequence similarity, however, can be converted to a distance.

Complexity differences between protein and DNA sequences mandate different distance functions. In the case of DNA sequences, Mendel uses a simple metric which is the Hamming distance. The Hamming distance [18] is defined as the number of positions between two equal length strings at which the characters differ. The 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. More intricate distance functions, such as the Levenshtein distance [19] or Jaccard indexing [20], can handle indels at the cost of more expensive processing; Hamming distance, however, provides a low complexity distance measure. Mendel overcomes this challenge with the use of sliding windows to account for insertions and deletions; this topic is further discussed in chapter 5.

Finding a 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 the 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 [21]. 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. One common approach to evaluate the similarity between protein sequences is to use a *scoring matrix*. Scoring matrices score every possible amino acid reside pair according to many factors. 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 more popular scoring matrix that takes into account similar factors as PAM, but uses an implicit model of evolution. BLOSUM is calculated from only highly conserved regions of protein families and thus is better suited for detecting distant similarities. The BLOSUM62 matrix is a common default scoring matrix in modern alignment applications including BLAST.

These scoring matrices are not suitable for a distance function in a vp-tree. 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, $M_{i,j}$:

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

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	**9**																				C
S	-1	**4**																			S
T	-1	1	**5**																		T
P	-3	-1	-1	**7**																	P
A	0	1	0	-1	**4**																A
G	-3	0	-2	-2	0	**6**															G
N	-3	1	0	-2	-2	0	**6**														N
D	-3	0	-1	-1	-2	-1	1	**6**													D
E	-4	0	-1	-1	-1	-2	0	2	**5**												E
Q	-3	0	-1	-1	-1	-2	0	0	2	**5**											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	**8**										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	**5**									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	**5**								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	**5**							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	**4**						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	**4**					L
V	-1	-1	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	**4**				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	**6**			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	**7**		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	**11**	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

(a) The unaltered BLOSUM62 scoring matrix.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	**0**																				C
S	10	**0**																			S
T	10	3	**0**																		T
P	12	5	6	**0**																	P
A	9	3	5	8	**0**																A
G	12	4	7	9	4	**0**															G
N	12	3	5	9	6	6	**0**														N
D	12	4	6	8	6	7	5	**0**													D
E	13	4	6	8	5	8	6	4	**0**												E
Q	12	4	6	8	5	8	6	6	3	**0**											Q
H	12	5	7	9	6	8	5	7	5	5	**0**										H
R	12	5	6	9	5	8	6	8	5	4	8	**0**									R
K	12	4	6	8	5	8	6	7	4	4	9	3	**0**								K
M	10	5	6	9	5	9	8	9	7	5	10	6	6	**0**							M
I	10	6	6	10	5	10	9	9	8	8	11	8	8	4	**0**						I
L	10	6	6	10	5	10	9	10	8	7	11	7	7	3	2	**0**					L
V	10	5	5	9	4	9	9	9	7	7	11	8	7	4	1	3	**0**				V
F	11	6	7	11	6	9	9	9	8	8	9	8	8	5	4	4	5	**0**			F
Y	11	6	7	10	6	9	8	9	7	6	6	7	7	6	5	5	5	3	**0**		Y
W	11	7	7	11	7	8	10	10	8	7	10	8	8	6	7	6	7	5	5	**0**	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

(b) The Mendel indexing matrix for protein sequences.

Figure 3.2: BLOSUM62 scoring matrix and the Mendel metric space translation matrix. The x and y axes contain the 20 amino acid residues and their correspond cells indicate the score (a) or distance (b) of the residue pair.

This operation transforms each column in the lower triangle matrix with respect to the diagonal entry such that each diagonal element is zero. Figure 3.2 shows the original BLOSUM62 matrix and the resulting Mendel protein distance matrix after the operation. 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.

3.1.3 Vantage Point Tree Construction

Constructing a vp-tree can be boiled down to partitioning an array of elements. The selected root of the tree corresponds to the entire space the data occupies. From this *vantage point*, the data is partitioned into left and right subspaces based on μ, calculated as the median distance between the vantage point and all other points. This is done by using the *quickselect* algorithm to partition the array around the median distance from the vantage point, *vp*. This process is repeated recursively until the branch contains one element, or all of the elements have the same distance from its parent. The construction of a vp-tree requires $O(n\ log(n))$ time and linear space.

Algorithm 1 Construct vp-tree

function BUILD_VP_TREE(S[])
    $root \leftarrow$ PARTITION$(S[], 0, S.length)$ **return** *root*
end function

function PARTITION(S, lower, upper)
    if $lower - upper < 2$ **then return**
    end if
    $\mu \leftarrow Median_{s \in S} d(vp, s)$

$$middle \leftarrow \frac{lower + upper}{2}$$

    QUICKSELECT$(S, lower, upper, middle, \mu)$
    $left\_child \leftarrow$ PARTITION$(S, lower, middle)$
    $right\_child \leftarrow$ PARTITION$(S, middle + 1, upper)$
    return *middle*
end function

3.1.4 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 $\tau = \min(d(q, vp), \tau)$. This redefinition allows τ to shrink to a radius around q's

nearest neighbors. By creating a circle around the query with a radius of τ, we observe three possible cases of how that area can relate to the current vantage point in the vp-tree:

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

In the first case, depicted by the point q in Figure 3.1, 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, but you may need to traverse multiple subtrees.

3.2 Performance Improvements

As Yianilos explained in his initial work with vp-trees, the implementation previously described can be altered slightly in order to achieve better performance in terms of memory usage and execution time [15]. He proposed two major optimizations: (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 voluminous datasets. Using upper and lower bounds to calculate a *middle* distance proved to be an effective, cost efficient estimate for the true median distance to other points in the subspace as seen by a vantage point.

One major challenge with genomic datasets and vp-trees we discovered is that in Yianilos' proposal, the dataset in its entirety must be present and inserted at the time of creation.

Algorithm 2 Find k Nearest Neighbors

```
1: function K_Nearest_Neihgbors(root, q, k)
2:     τ ← ∞
3:     nodes[] ← root
4:     neighbors ← BoundedPriorityQueue(k)
5:     while nodes.length() ¿ 0 do
6:         node ← nodes.pop()
7:         d ← distance(q, node)
8:         if d < τ then
9:             neighbors.add(node)
10:
11:            // Shrink τ to the farthest nearest neighbor
12:            τ ← min(τ, distance(q, neighbor.tail) )
13:
14:            // Check which branches need to be searched
15:            if d < node.μ then
16:                if d >= node.μ + τ then
17:                    nodes.add(node.left_child)
18:                end if
19:                if d >= node.μ − τ then
20:                    nodes.add(node.right_child)
21:                end if
22:            else
23:                if d < node.μ + τ then
24:                    nodes.add(node.left_child)
25:                end if
26:                if d >= node.μ − τ then
27:                    nodes.add(node.right_child)
28:                end if
29:            end if
30:        end if
31:    end while
32:    return neighbors
33: end function
```

The original data structure did not support insertions to an previously constructed vp-tree. Since datasets must be able to be added to an existing database we needed to find an effective way to do so.

Naïvely inserting data points one-at-a-time, passing through the tree and inserting the element into its appropriate bucket, 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 essentially breaks down into four cases when updating the tree [22]:

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:
    - Find nearest ancestor and redistribute all values under it
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 acceptable performance while maintaining an optimized, balanced vp-tree to use as an NNS data structure.

3.3 Vantage Point Tree as a LSH Function

Utilizing a vp-tree as the data structure for voluminous datasets presents new challenges. Initially, this data structure was used as a similarity image retrieval method over a library of 604 $\times$ 468 pixel images [15]. 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. In this section we introduce a heuristic to the vp-tree that allows it to be leveraged as a similarity based hashing function. 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:

$$\text{prefix}_{\text{curr}} = \begin{cases} 1, & \text{if current is the root} \\ \text{prefix}_{\text{parent}} \cdot 2, & \text{is a left child} \\ \text{prefix}_{\text{parent}} \cdot 2 + 1, & \text{is a right child} \end{cases}$$

In other words, 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. Given that (a) child nodes in the left subtrees will be closer to the root, in metric space, than the child nodes in the right subtrees and (b) leaves towards the left will have smaller values than those toward the right, this creates some degree of integral relationship between node prefixes and the metric distance between them.

3.3.1 Vantage-Point Prefix Tree Hashing

To use the vp-prefix tree as a hash function, a single traversal from root to leaf, without branching, is required. Along the way, each node prefix is accumulated to create a hash value from the traversal. 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.

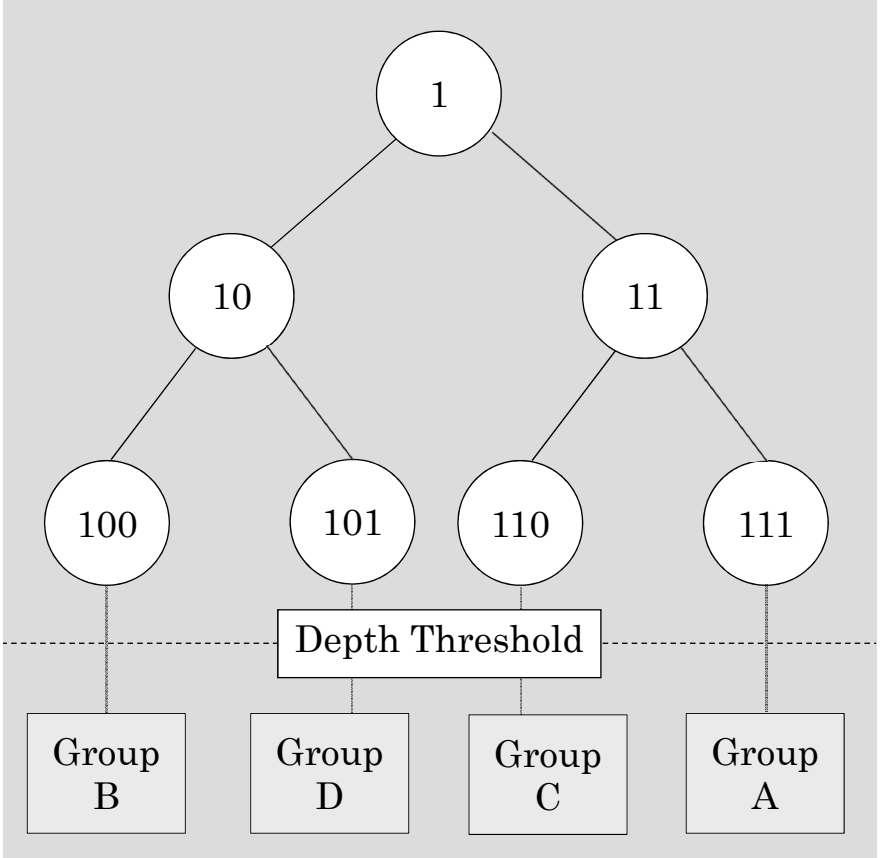


Figure 3.3: A vp-prefix tree being used as a group hash function with a depth threshold of 3. The depth of the threshold effectively determines the resolution of similarity that each group maintains. A deeper depth threshold will have groups with a higher level of similarity.

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. While in a normal DHT this may sound less than desirable, The hierarchical two-tiered DHT not only tolerates these collisions, but utilizes collisions as a way to group similar data for query evaluation. Figure 3.3 shows a small example of how a vp-prefix tree might hash data into groups.

Chapter 4

System Architecture

4.1 Distributed Hash Tables

Distributed hash tables have been an essential part of the big data era and the NoSQL system movement. Prominent distributed storage systems such as Amazon Dynamo [23] and Apache Cassandra [24] 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. In a distributed setting, 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. In order to perform a lookup, the unique key must be provided and is hashed to find the storage node to route the lookup request towards. Incremental scalability can be achieved because this storage scheme allows for nodes to join and leave the system in a decentralized fashion. Thus, aligning to the current big data movement and shift towards cloud computing. This has paved the way to its current popularity.

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 of overcoming this challenge with the use of locally-sensitive hashing or hierarchical DHTs [9, 25]. In addition to the lack of robust queries, 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

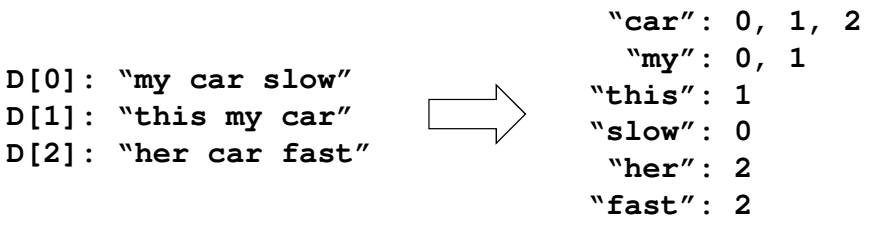


Figure 4.1: A small demonstration of an inverted index over three documents. Each word is indexed by the document(s) it is found in.

relationships to only portions of the cluster adds complexity via routing protocols thus increasing request latency.

4.2 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 4.1 shows a simple example of an inverted index over text documents. In this toy example, the database contains three documents. After the inverted index is applied to the data, each word maintains a list of the locations of the documents it occurred in. 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, the desired result is an alignment of a target sequence versus some set of reference sequences. By 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 sequencing errors, such as substitutions, insertions, and/or deletions, and genomic structural variation, such as single nucleotide polymorphisms, and evolution 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.

4.2.1 Network Topology

Mendel's network overlay topology is organized as a zero-hop DHT. DHTs provide a decentralized, highly scalable overlay network that allow for insertion and retrieval similar to that of a hash table; e.g. *put(key, value)*, and *get(key*). 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. The similarity

hashing function also expands data interactions beyond the *put(key, value)/get(key)* type, into nearest neighbor queries to find alignments of queries to sequences stored in the database.

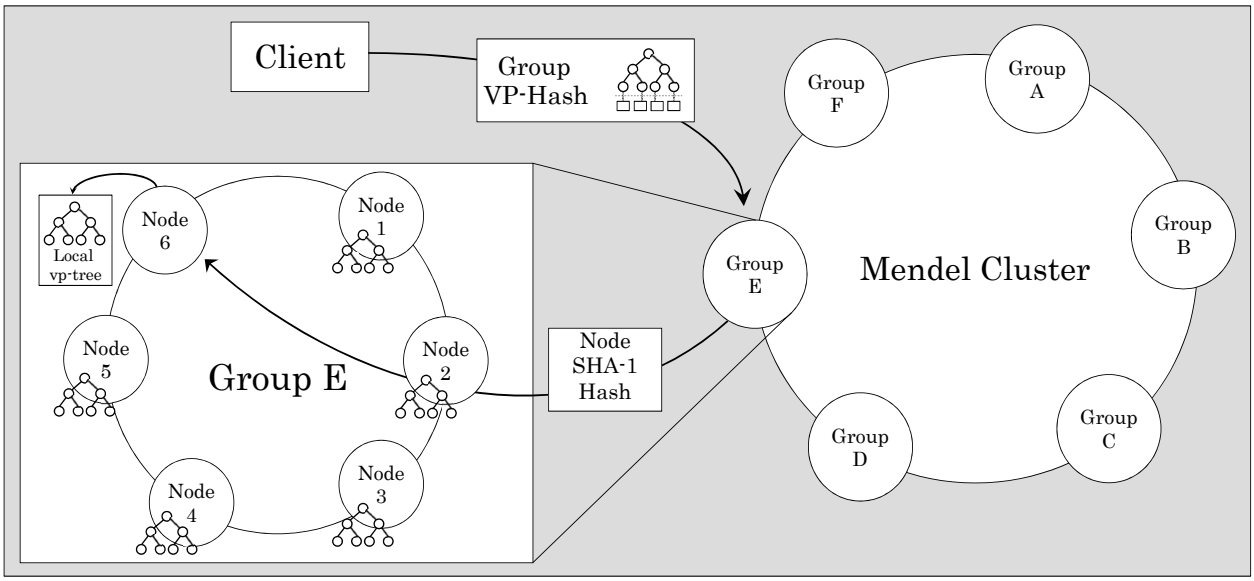


Figure 4.2: An illustration of the network topology and data flow of a Mendel cluster. Each inverted index block is hashed to a predefined group of storage nodes using the vp-prefix tree hash. Within its group, the data is hashed a second time using a SHA-1 hash to distribute data among the group evenly. Finally, once the storage node has been determined for the

Chapter 5

Indexing and Query Evaluation

5.1 Sequence Indexing and Storage

5.1.1 Inverted Index Blocks

In general, when querying an inverted index structure the query must *exactly* match the indexed content to retrieve relevant results. In the context of sequence alignment, exact matches are easily invalidated by substitutions, insertions, or deletions within a sequence. A problem exacerbated by the distribution of the inverted indices over the cluster. 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 exact match 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.

5.1.2 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, at increments of 1, producing $L - k + 1$ 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.

5.1.3 Vp-Prefix Tree Sequence Dispersion

Each block is hashed independently using the vp-prefix tree indexing scheme, previously outlined in chapter IV, to determine its storage group. Using this group hashing system, sequences with similar structures will be collocated within the same group. During query evaluation, a similar process is conducted to determine relevant storage nodes; thus, query sequences will be routed to storage groups that contain inverted index blocks that are similar. 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 good 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 that 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.

5.1.4 Local vp-Tree Indexing

Finally, once an inverted index block reaches its destination storage node within its storage 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. Figure 5.1 visualizes each of the three steps in the indexing phase.

5.2 Query Evaluation

Mendel strives to emulate the prompt responsiveness that DHTs provide along with the ability to conduct robust queries. Queries can be sent to any storage node in the cluster due to its decentralized design. 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 serve 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 the vp-prefix tree hash function, each target query segment is hashed to determine the groups within the system that may contain relevant segments. 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,

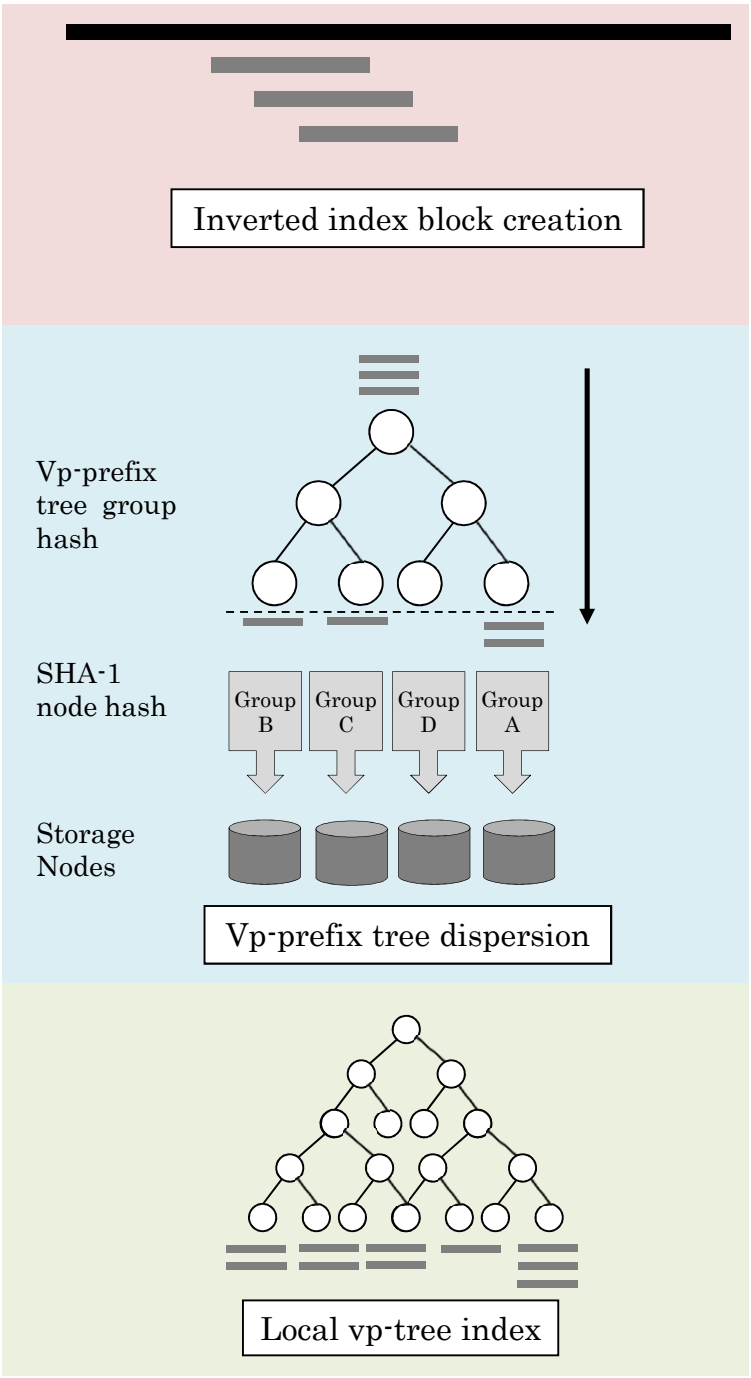


Figure 5.1: An overview of the three stages of indexing sequence data into Mendel.

default 10, nearest neighbors to the segment are added to a candidate list of possible matches. The result after all the NNSs are performed at a node is a list of candidate inverted index blocks allocated to that node. Two measures are computed for each candidate: (1) a percent identity score, computed as $\frac{hamming(segment, candidate)}{length(candidate)}$ 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 then moving to the next segment, 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 [12]. 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 5.1.

Table 5.1: 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..∞)

5.3 Expectation Values

Measuring the similarity between two sequences effectively started with Needleman and Wunsch's *global* sequence alignment algorithm [26]. Global alignments attempt to align *every* residue in both sequences to one another and are best suited for sequences that are approximately equal length. Because of this, aligning dissimilar sequences that may contain small regions of similarity are challenging with global alignments. Conversely, *local* alignments aim to identify *regions* of similarity between sequences. Local alignments are more commonly used for homology detection because they allow one to distinguish conserved domains which may only inhabit a small portion of the sequence.

Determining the significance of an alignment is pivotal in assessing if there is evidence of homology. Understanding the relationship between *statistical significance* and *biological significance* allows researchers to make stronger conclusions over the resulting alignments. In stating that two sequences are homologous, we are concluding the sequences diverged from some common ancestor. Homology in the context of protein sequences implies that the two sequences are similar in structure. In nearly all cases, statistically significant similarity

between sequences identifies significant structural similarity [27]. The opposite, however, does not hold true; dissimilar sequences may share significant structural similarity [28]. Regardless of the similarity scoring method, the question with regards to alignment scores remains the same: "What is the probability that a particular score can be obtained by chance in a database of non-homologous sequences?"

To represent the statistical significance of an obtained score, many systems use *p-values* and/or *e-values*. A p-value, represented by $P(x, n)$, is the probability of observing one or more scores greater than or equal to a given score x in a database search of n sequences [29]. An e-value, or expectation value, represents the number of scores greater than or equal to x expected to be found in a database search of n sequences; represented by $E(x, n)$. We denote the probability of a *single* score being greater than or equal to x between non-homologous sequences to be $P(S \geq x)$. $E(x, n)$ and $P(x, n)$ are given by the following [30]:

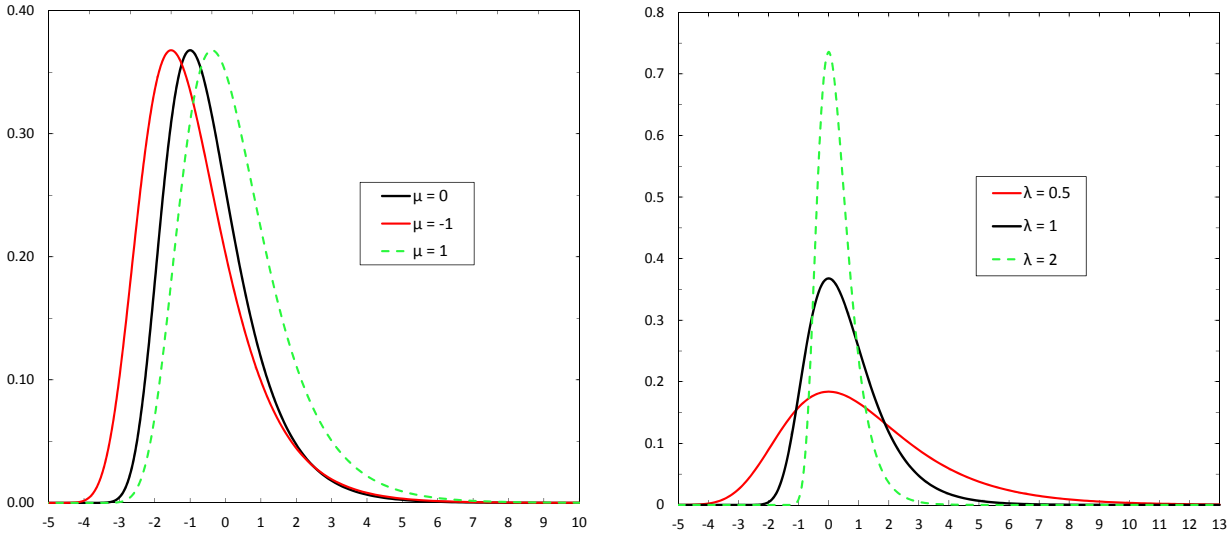
$$P(x, n) = 1 - e^{1-nP(S \geq x)} \tag{5.1}$$

$$E(x, n) = nP(S \geq x) \tag{5.2}$$

As shown in (5.2)–(5.1), calculating p-values and e-values is elementary once $P(S \geq x)$ is obtained. Karlin and Altschul achieved a substantial breakthrough in biostatistics in their analysis of local alignment scores *without* gaps [30]. They demonstrated that ungapped alignment scores could be modeled by an *extreme-value distribution* (EVD). An analytical solution for the distribution of $P(S \geq x)$ for gapped alignments is yet to be found. However, it has been empirically verified through numerous simulations that the distribution of optimal gapped similarity scores can also be approximated by the EVD [31]. Many approaches, including BLAST and several other Smith/Waterman algorithm variants, use curve fitting to fit alignment scores to an EVD.

The EVD is given by the probability density function (pdf):

$$P(x) = \lambda \exp[-\lambda(x - \mu) - e^{-\lambda(x-\mu)}] \tag{5.3}$$



(a) Effect of the location parameter μ

(b) Effect of the scale parameter λ

Figure 5.2: The effects of parameters λ and μ on the probability density function of an extreme-value distribution

An EDV is described by parameters μ, location, and λ, scale. Figure 5.2 shows the effects of μ and λ on the location and scale of the curve respectively.

Using maximum likelihood estimation to evaluate model parameters λ and μ is a widely accepted approach [32]. The goal of this approach is to find estimates of λ and μ that maximize the log likelihood of drawing n samples from an EVD with parameters λ and μ. In Lawless' work on maximum likelihood fitting, he was able to derive μ in terms of λ, thus simplifying the process to a single parameter shown in (5.4) [32].

$$\mu = -\frac{1}{\lambda} \log \left[\frac{1}{n} \sum_{i=1}^{n} e^{-\lambda x_i} \right] \tag{5.4}$$

Following the approach outlined in [29], the estimate for the parameter λ is the root of the log likelihood with respect to λ:

$$0 = \frac{1}{\lambda} - \frac{1}{n} \sum_{i=1}^{n} x_i + \frac{\sum_{i=1}^{n} x_i e^{-\lambda x_i}}{\sum_{i=1}^{n} e^{-\lambda x_i}} \tag{5.5}$$

Using Newton's method we can find the root, thus the whole process can be summarized as follows. First, begin with a small conjecture for λ and plug the conjecture into 5.5. If the result is sufficiently close to zero, you are done and can plug in the conjectured value of λ into

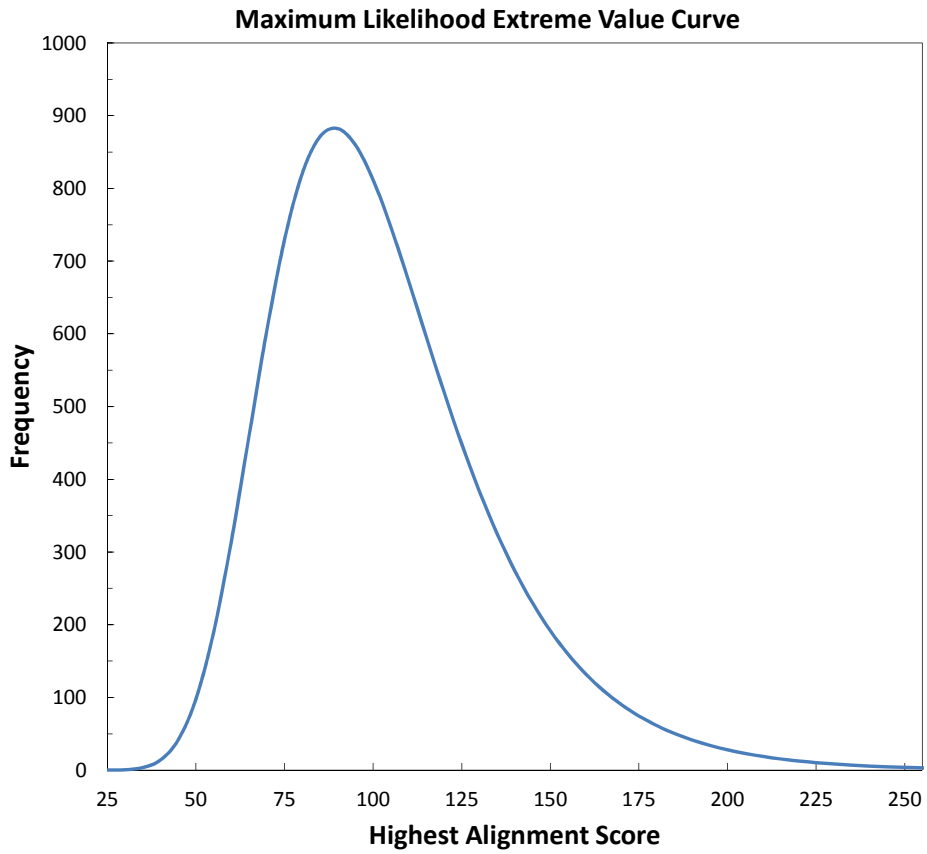


Figure 5.3: The fitted plot of the optimal local alignment scores generated by calculated 10,000 alignments with Mendel over a database of 10,000 randomly generated protein sequences

5.4 to obtain μ. Otherwise, we iterate over Newton's method to get a better approximation and try again; repeating this process until a satisfactory value of λ is found.

Rather than estimating λ and μ based on actual individual database searches like FASTA [33], Mendel takes an approach similar to BLAST [1] by estimating the values beforehand according to the implemented scoring scheme. To do this, 10,000 protein sequences of length 1,000 were generated according to the average amino acid composition in the UniProtKB/Swiss-Prot data bank statistics [21] and stored in Mendel. From this, we calculated the distribution of 10,000 optimal local alignment scores to be used in the maximum likelihood calculation described earlier. Figure 5.3 shows the results of this experiment resulting in $\lambda = 0.04$ and $mu = 89$.

Chapter 6

Performance Evaluation

To benchmark the effectiveness of Mendel, we ran several tests to simulate application usage on a heterogeneous cluster. This chapter will discuss the various benchmarks performed and outline the results found. We targeted four main aspects in our tests: (1) the performance of the vantage point prefix tree as an 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.

6.1 Experiment Environment

6.1.1 Cluster Setup

The testing environment consisted of a 50-node heterogeneous cluster connected over a LAN. Details about the specifications of the individual machines are outlined in table 6.1. All machines are running Fedora 21 (Twenty One) and OpenJDK 1.8.0.

Table 6.1: 50-node Cluster Configuration

Count	Model	CPU	Memory	Disk Speed
25	HP DL160	Xeon E5620	12 GB	15000 RPM
25	Sun SunFire X4100	Opteron 254	8 GB	10000 RPM

Datasets

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

and `e_coli`. We also used the Astral SCOPe 2.05 dataset composed from protein domain sequences with less than 40% identity to each other. Tables 6.2 – 6.3 summarize the query sets and datasets.

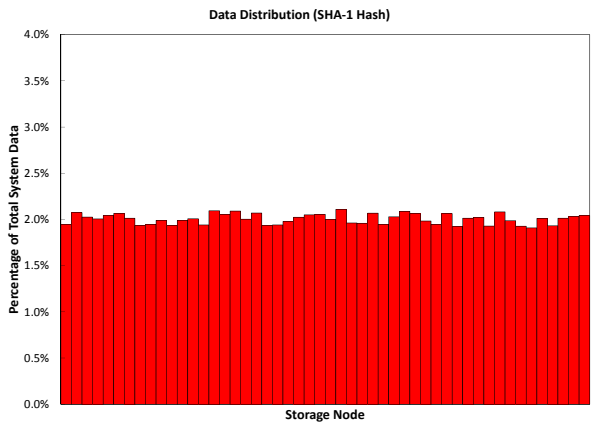
Table 6.2: Protein Query Sets

Dataset	No. Sequences	Size (MB)
`S_aureus`	1,964	0.90
`E_coli`	4,124	1.69

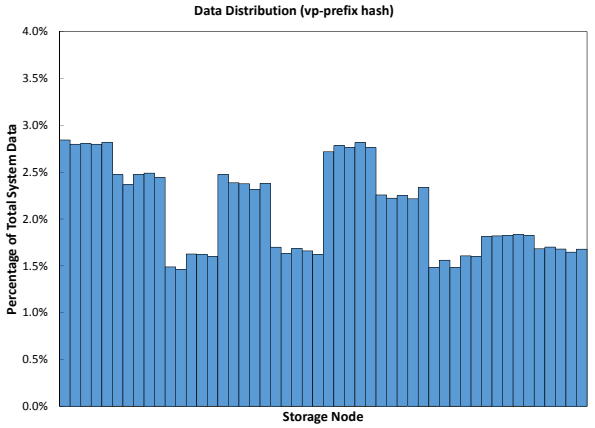
Table 6.3: Protein Data Sets

Dataset	No. Sequences	Size (MB)
`astral40`	13,365	3.54
`nr`	64,057,457	35,958

6.2 Data Distribution and Load Balancing Evaluation



(a) The distribution of data using a flat SHA-1 hash to determine the storage node a data point belongs to.



(b) The distribution of data using the hierarchical similarity hashing scheme.

Figure 6.1: A comparison between the balance of content distribution of a standard hash function versus our two-tiered hierarchical hashing scheme

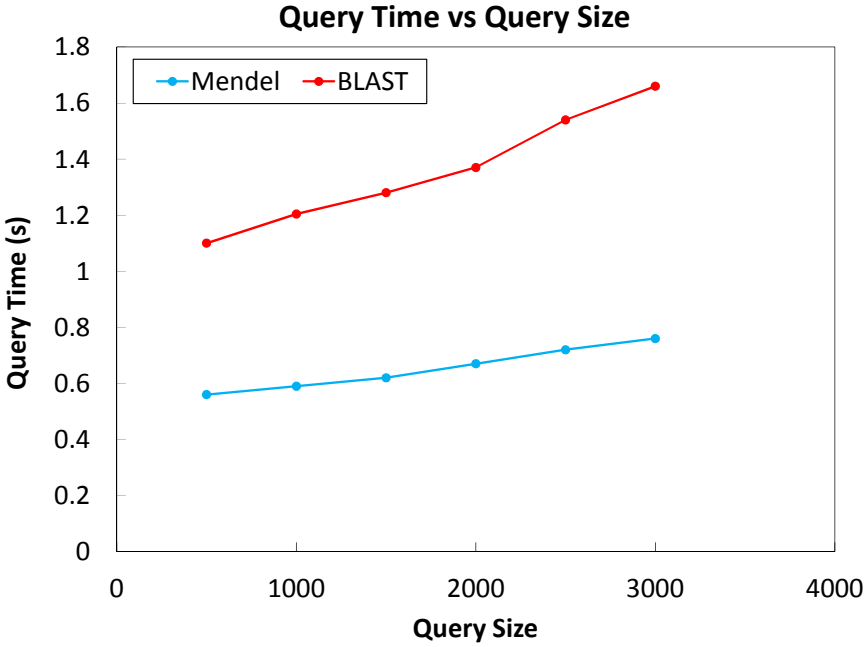
Our first benchmark aims to test the load distribution of Mendel. We indexed the 100 GB of genomic data over the 50-node cluster. The percentage of total system data being

stored at each node was recorded. Figure 6.1 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.

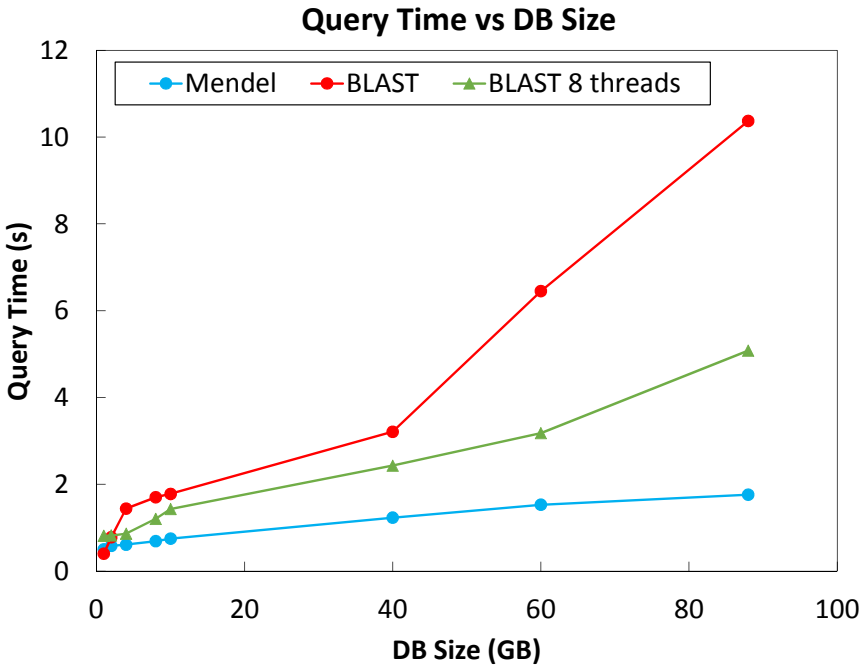
6.3 Query Performance

As stated earlier, Mendel aims to emulate the performance of a hash table. In these benchmarks, we compare our results versus BLAST [1]. 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 [34]. We executed queries from the `S_aureus` query set with target sequence lengths ranging from 500 to 3000 residues over the `nr` dataset. Figure 6.2a 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 6.2b 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



(a) Plot of the execution time versus the length of the alignment query.



(b) Plot of query times as a function of the total size of the database

Figure 6.2: Various performance benchmarks of our proposal versus BLAST. The figures show how the performance doesn't degrade as the different inputs grow large.

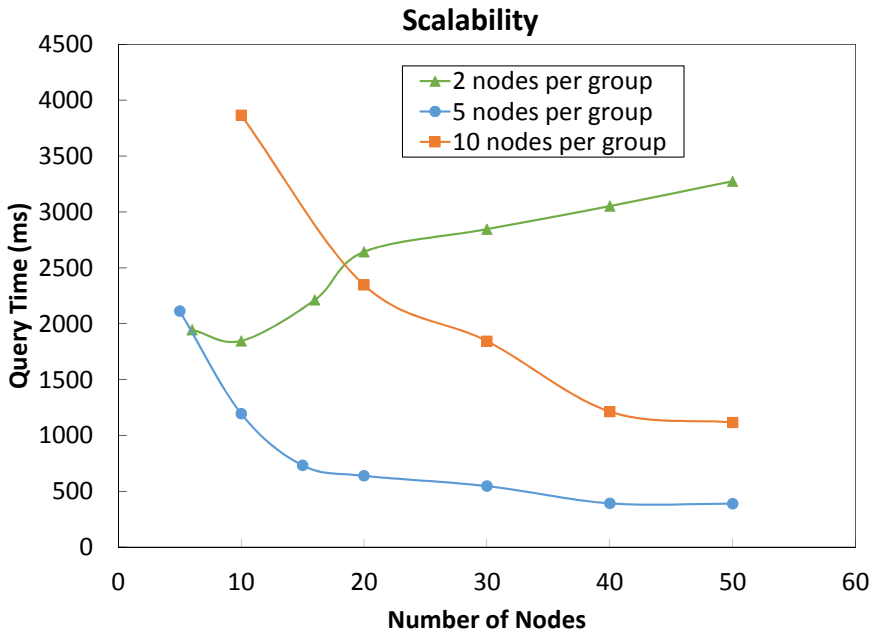


Figure 6.3: Plot of query times versus the number of nodes being used in the system.

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.

6.4 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 6.2b 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. We performed this benchmark with varying numbers of groups sizes. Figure 6.3 shows a sufficient scalability with respect to the size of the cluster for group sizes greater than 5.

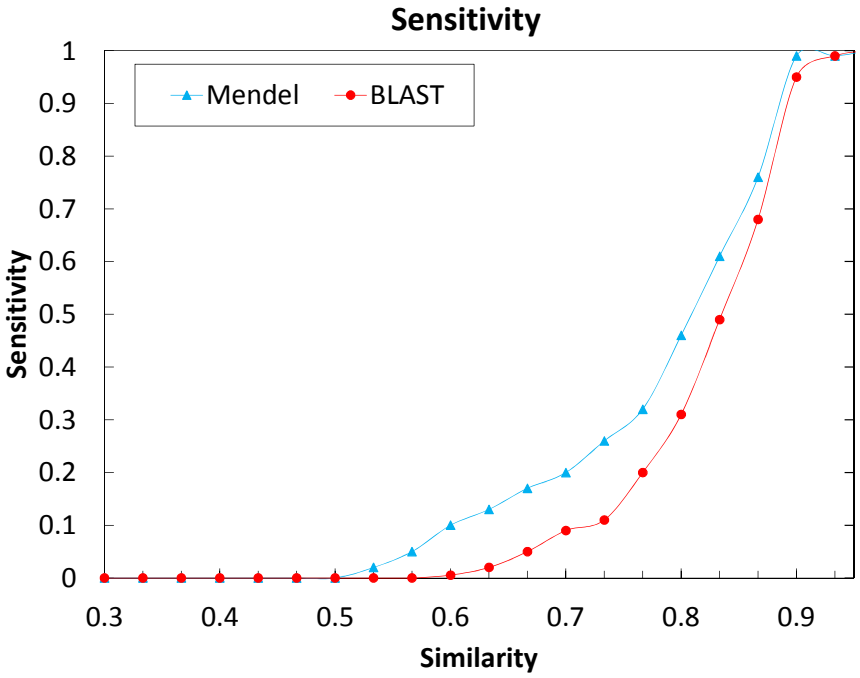


Figure 6.4: Plot of sensitivity versus query similarity of Mendel versus BLAST.

6.5 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 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 1,000 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 6.4 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.

6.5.1 SCOPe Homology Search

The **S**tructural **C**lassification **o**f **P**roteins **e**xtended (SCOPe) is a database of proteins classified by structural domains [35]. SCoPe classifies proteins into a 7 level hierarchical classification:

1. Class – Types of folds
2. Fold – General shape of the domains
3. Superfamily – Distant common ancestor (structural similarity)
4. Family – More recent common ancestor (sequence similarity)
5. Protein domain – Protein class
6. Species – Protein domains grouped by species
7. Domain – Conserved parts of the protein sequence

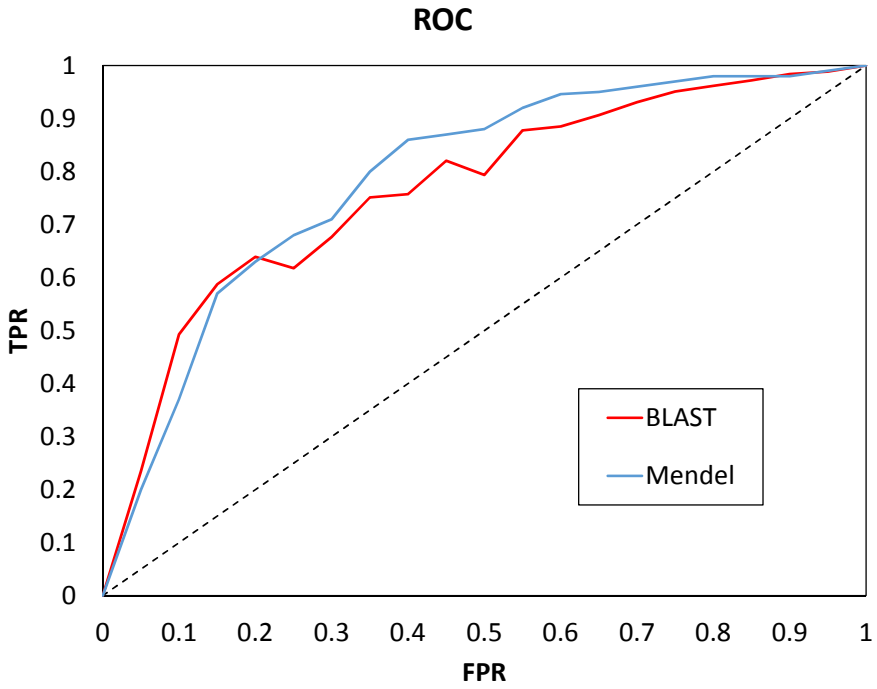


Figure 6.5: The ROC curve showing the accuracy of Mendel versus BLAST over the `astral40` dataset as classified by SCOPe

We used SCOPe to find known homologs to use as an oracle in a similarity search. For each sequence in the `astral40` dataset, we conducted all versus all queries and used SCOPe to verify the family classification for each query. Since the `astral40` dataset, a dataset derived from SCOPe, contains proteins that have already been classified, we can measure the number of true positive and false positives for each query. From this we created a receiving operating characteristic (ROC) curve by averaging the true positive rates (TPR) for false positive rates (FPR) of all the queries. The ROC curves shown in figure 6.5 show the results of the experiment.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

We have proposed a novel distributed system, Mendel, aimed at efficiently computing 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 efficiently identify small similar segments. We modified a nearest neighbor search 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 anchor alignments for a query. 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 segments can be identified. Our benchmarks exhibit performance improvements in runtime, sensitivity, and scalability over other modern sequence alignment tools.

7.2 Future Work

There are many improvements and extensions of the presented work to head towards. Some components in our system, for example the depth threshold for the vp-prefix hash tree, would benefit from further investigation or even automated tuning. 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. 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.

The vast majority of query execution time is spent at the individual node level processing NNSs in its local vp-tree. Better performance could be achieved by applying the excluded middle vantage point forest algorithm [36] to exploit the inherent parallelism of the vp-tree search. This would allow better utilization of resources at a single node and improve the overall time spent searching in the vp-tree. Further more, using spaced seeds for the segment creation could also been pivotal to not only the query execution time, but also the accuracy of the system. Since a sliding window is being used to create the segments including every other character would be a way to reduce the total number of bases/residues per segment, thus reducing the time spent processing each segment. Spaced seeds have been shown to be capable of identifying high scoring alignments [13] and could contribute a lot to Mendel.

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] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[2] W James Kent. Blatthe blast-like alignment tool. *Genome research*, 12(4):656–664, 2002.

[3] Aaron Darling, Lucas Carey, and Wu-chun Feng. The design, implementation, and evaluation of mpiblast. *Proceedings of ClusterWorld*, 2003:13–15, 2003.

[4] Michael C Schatz. Cloudburst: highly sensitive read mapping with mapreduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[5] Shuji Suzuki, Masanori Kakuta, Takashi Ishida, and Yutaka Akiyama. Ghostx: An improved sequence homology search algorithm using a query suffix array and a database suffix array. *PLoS ONE*, 9(8):e103833, 2014.

[6] Andréa Matsunaga, Maurício Tsugawa, and José Fortes. Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 222–229. IEEE, 2008.

[7] Simone Leo, Federico Santoni, and Gianluigi Zanetti. Biodoop: bioinformatics on hadoop. In *Parallel Processing Workshops, 2009. ICPPW'09. International Conference on*, pages 415–422. IEEE, 2009.

[8] Michael Brock and Andrzej 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*, pages 995–1000. IEEE, 2012.

[9] Rodolfo da Silva Villaca, Luciano Bernardes de Paula, Rafael Pasquini, and Maurício Ferreira Magalhães. Hamming dht: Taming the similarity search. In *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, pages 7–12. IEEE, 2013.

[10] Parisa Haghani, Sebastian Michel, Philippe Cudré-Mauroux, and Karl Aberer. Lsh at large-distributed knn search in high dimensions.

[11] Bahman Bahmani, Ashish Goel, and Rajendra Shinde. Efficient distributed locality sensitive hashing. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2174–2178. ACM, 2012.

[12] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[13] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *Proceedings of the VLDB Endowment*, 5(5):430–441, 2012.

[14] Ching-Hsien Hsu, Chun-Yuan Lin, Ming Ouyang, and Yi Ke Guo. Biocloud: cloud computing for biological, genomics, and drug design. *BioMed research international*, 2013, 2013.

[15] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–321, 1993.

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

[17] Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[18] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[19] Gregory V Bard. Spelling-error tolerant, order-independent pass-phrases via the damerau-levenshtein string-edit distance metric. In *Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68*, pages 117–124. Australian Computer Society, Inc., 2007.

[20] Paul Jaccard. *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz, 1901.

[21] UniProt Consortium et al. Uniprot/swiss-prot release 2015_09 statistics. http://web.expasy.org/docs/relnotes/relstat.html, 2015. Accessed: 09-19-2015.

[22] Ada Wai chee Fu, Polly M. S. Chan, Yin ling Cheung, and Y. S. Moon. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. *VLDB Journal*, 9:154–173, 2000.

[23] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. 41(6):205–220, 2007.

[24] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[25] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep 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*, pages 31–38. IEEE, 2012.

[26] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[27] JF Collins, AFW Coulson, and A Lyall. The significance of protein sequence similarities. *Computer applications in the biosciences: CABIOS*, 4(1):67–71, 1988.

[28] Stephan Lorenzen, Christoph Gille, Robert Preissner, and Cornelius Frömmel. Inverse sequence similarity of proteins does not imply structural similarity. *FEBS letters*, 545(2):105–109, 2003.

[29] Sean R Eddy. Maximum likelihood fitting of extreme value distributions. 1997.

[30] Samuel Karlin and Stephen F Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268, 1990.

[31] Richard Mott. Maximum-likelihood estimation of the statistical distribution of smith-waterman local sequence similarity scores. *Bulletin of Mathematical Biology*, 54(1):59–75, 1992.

[32] Jerald F Lawless. *Statistical models and methods for lifetime data*, volume 362. John Wiley & Sons, 2011.

[33] William R Pearson and David J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[34] George Coulouris. Blast benchmark, 2013.

[35] Naomi K Fox, Steven E Brenner, and John-Marc Chandonia. Scope: Structural classification of proteinsextended, integrating scop and astral data and classification of new structures. *Nucleic acids research*, 42(D1):D304–D309, 2014.

[36] Peter N Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *In DIMACS Implementation Challenge, ALENEX'99*. Citeseer, 1999.