Mendel: A Distributed Storage System for Efficient Similarity Searches and Sequence Alignment

Jan. 30, 2015

Cameron Tolooee

Outline

- Motivation
- Overview
- Vantage-Point Tree
- System Architecture
- Results
- Conclusion & Future Work
- Questions

Motivation

- Due to exponential growth of biological datasets, current similarity search tools are becoming less sufficient
  - BLAST, BLAT, YASS, FASTA, etc…
  - Algorithm centric, different heuristics on similar algorithm with different trade-offs
- Similarity between sequences, or lack thereof, can explain relationships between them
  - In some cases can provide important clues about common evolutionary roots of organisms

Basic Idea

- Inverted index
  - Map content to its location in the database
    - Rather than indexing what each location contains
  - Allows for efficient searches at the cost of additional processing for insertions
- DHTs provide extremely fast lookups for distributed datasets

Basic Idea

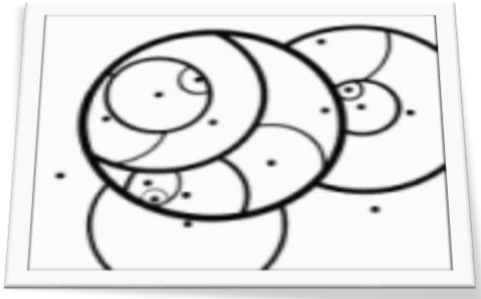
- Searching DNA sequences for subsequences is a challenging problem
  - Must consider partial matches, insertions/deletions (indels), repeated regions, etc..
- Sliding window over DNA sequence indexing on each substring
  - Sliding window can identify indels
- Store data with in a DHT with a nearest neighbor data structure
  - Nearest neighbor structure finds partial matches

Challenges

- How to locate matches for a non-exact match query in a DHT?
- How to balance content load on storage nodes?

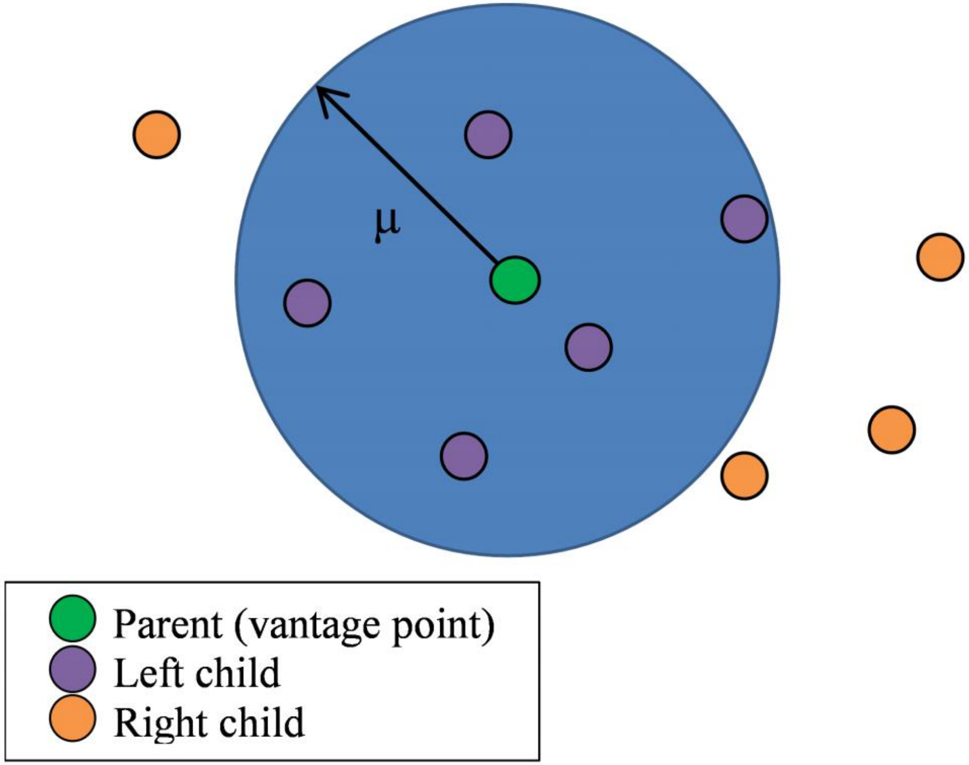
Vantage Point Tree

- Developed by Peter Yianilos and Jeffrey Uhlmann independently
- Data structure used for nearest neighbor searches in metric space
- Recursively partition data points into two divisions
  - Points that are within a threshold distance of the vantage point
  - Points that are outside the same threshold



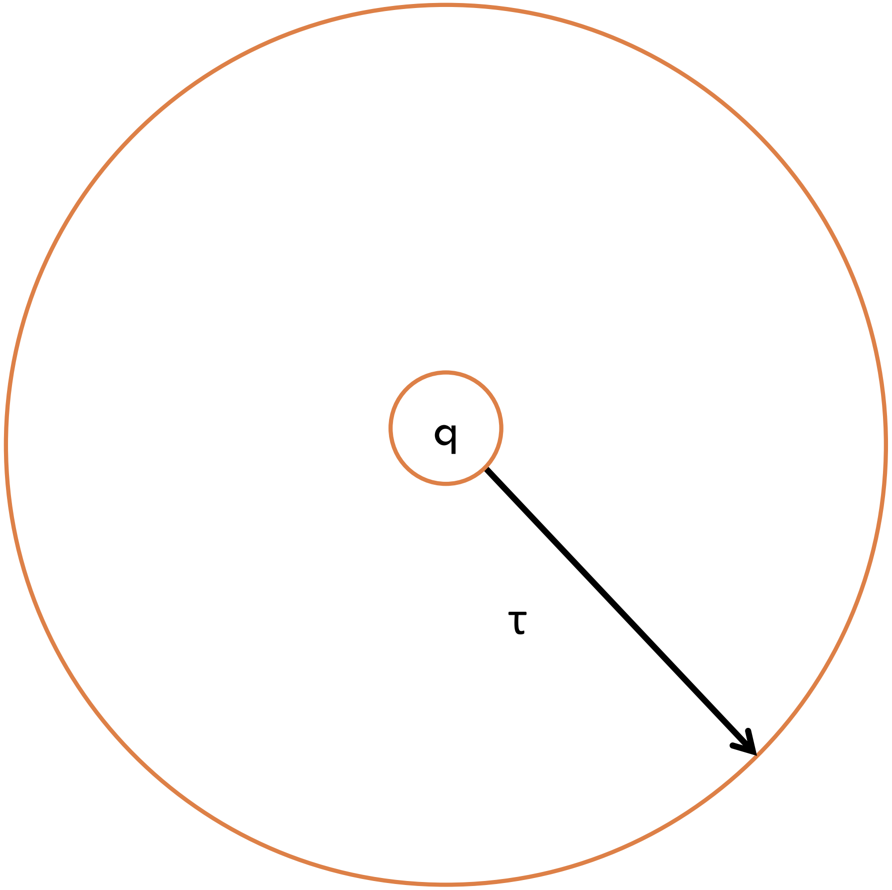
Vantage-Point Tree

- Each node in a vp-tree maintains four values:
  - Input value
  - Radius, μ
  - Left child
  - Right child



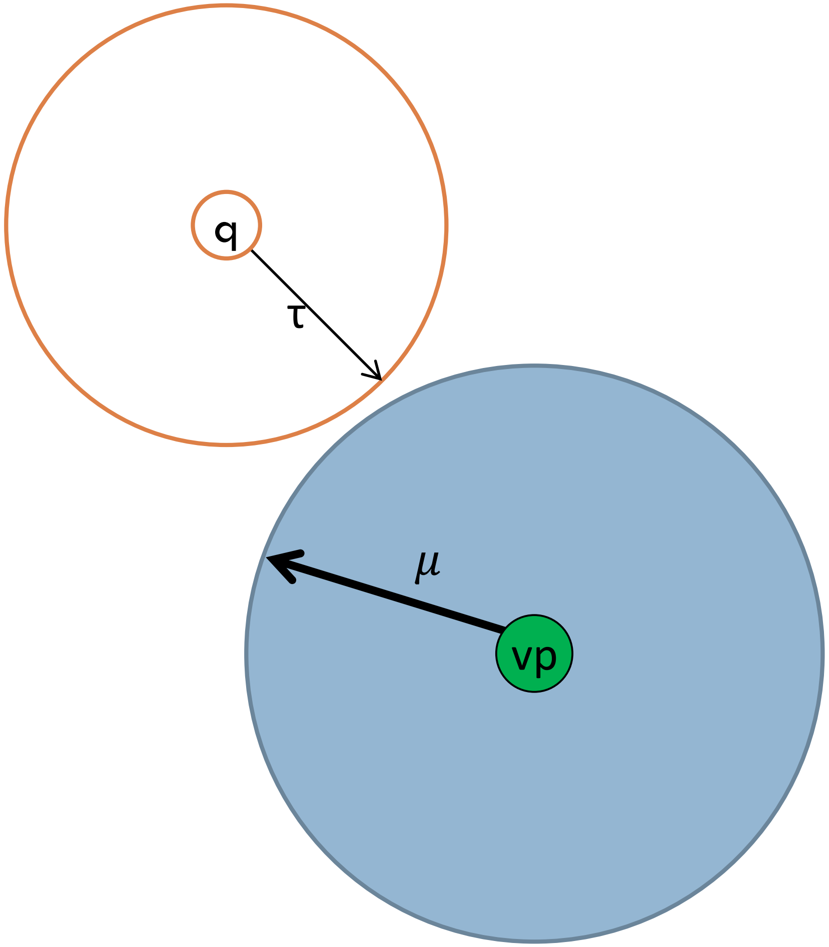
Searching vp-trees

- Let query be *q*
- Let radius of *q* be τ
- *k* nearest neighbors are contained within τ
- $\tau \ = \min(dist(q \rightarrow v, \tau))$
- 3 cases
  - τ lies completely within μ
  - τ lies completely outside μ
  - τ and μ intersect
- Stop recursing when leaves are reached



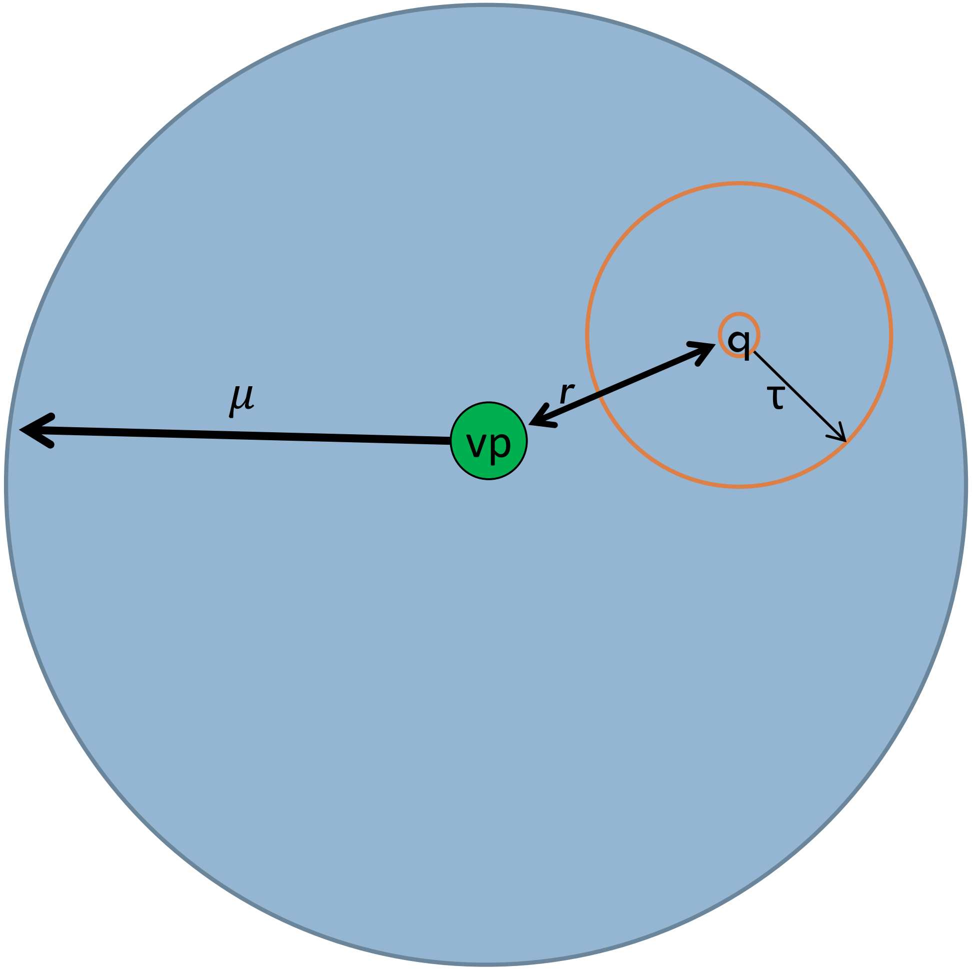
Searching vp-trees

- Case 1: τ *completely* outside of μ
  - Don't need to search left subtree
- $\tau = \min(dist(q \rightarrow vp), \tau)$
  - Recurse on right subtree



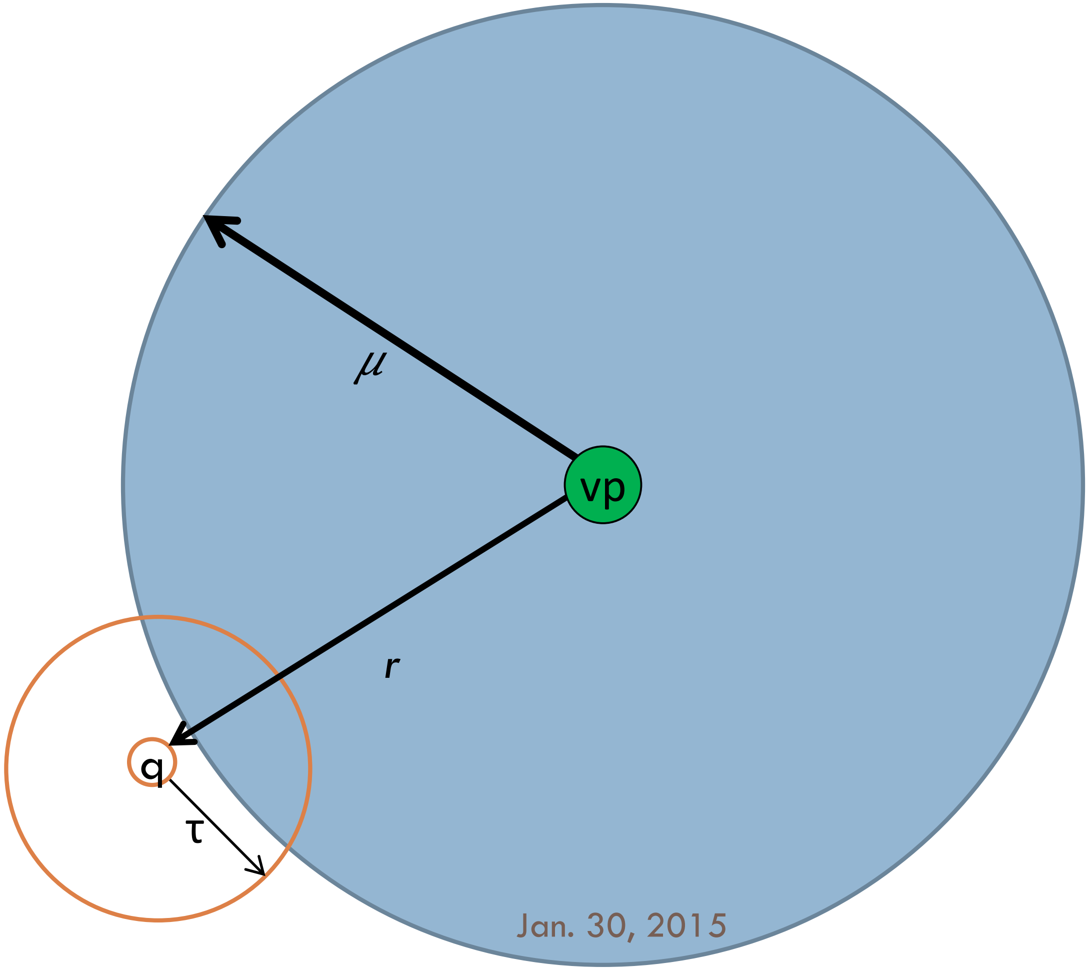
Searching vp-trees

- Case 2: τ *completely* inside μ
  - Don't need to search right subtree
- $\tau = \min(r, \tau)$
  - Recurse on left subtree



Searching vp-trees

- Case 3: worst case intersect
  - Must search both trees
- $\tau = \min(r, \tau)$
  - Recurse on both subtrees



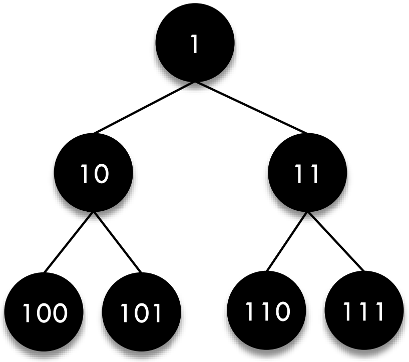
Jan. 30, 2015

Vantage-point prefix tree

- Global vp-tree as an index is not scalable
  - Utilize vp-tree as a similarity based hashing function
- Alter vp-tree node to contain a prefix

$$prefix_{left} = prefix_{parent} \ll 1$$
$$prefix_{right} = (prefix_{parent} \ll 1) + 1$$

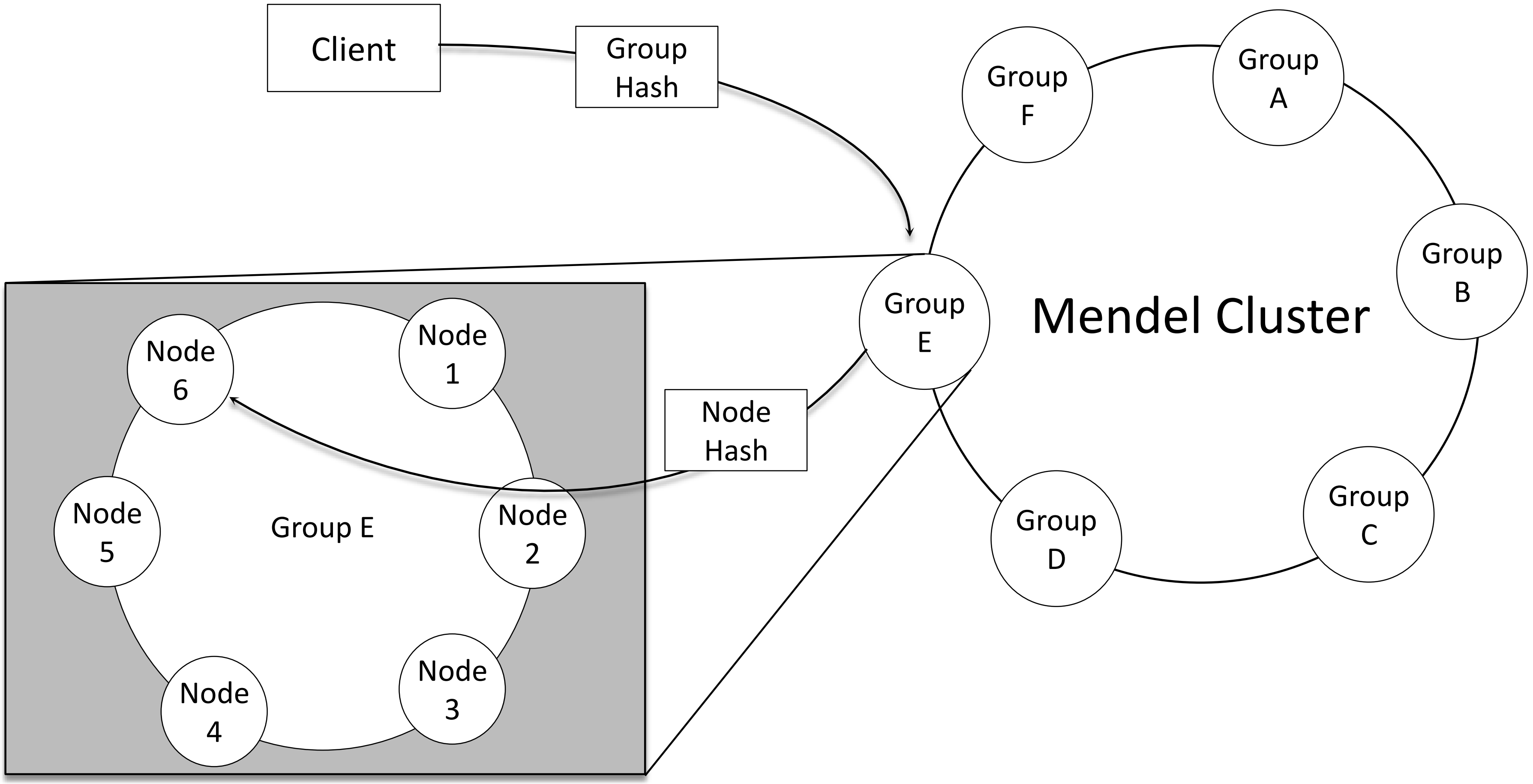
- Use as a group hash by assigning groups to subtrees
  - Requires a balanced vpp-tree



System Architecture

- Zero-hop distributed hash table
  - Such as Apache Cassandra and Amazon Dynamo
- Hierarchical, two-tier hashing scheme
- Each node belongs to a group
  - Groups are placed on the hashing ring
  - Two rounds of hashing required to place or retrieve data
    - Hashed to a group using the vpp-tree
    - Second hash among group nodes

System Architecture



Indexing Data

- 100bp sliding window over each contig
  - Each 100bp subsequence is individually indexed
- Passed through the vpp-tree to determine storage grouping
- Within the group, the subsequence is distributed using a SHA-1 hash to a storage node
- The subsequence block is maintained in a vp-tree local to its storage node

Query Evaluation

- Query is “hashed” in the vpp-prefix tree to find all subtrees that *may* have matching subsequences
- Each node in the selected group(s) performs a lookup in their vp-tree
  - Results are aggregated and filtered
- Results are send back to the client

Results

- Three benchmarks to test indexing speed, data distribution, and query speed
- Sourced real world data from the Genome Assembly Golden-standard Evaluation (GAGE)
  - Four genomes ranging from 2 Mbp to 3 Gbp
- Benchmark 1: index each of the genomes into the system and measure the time to complete

Results

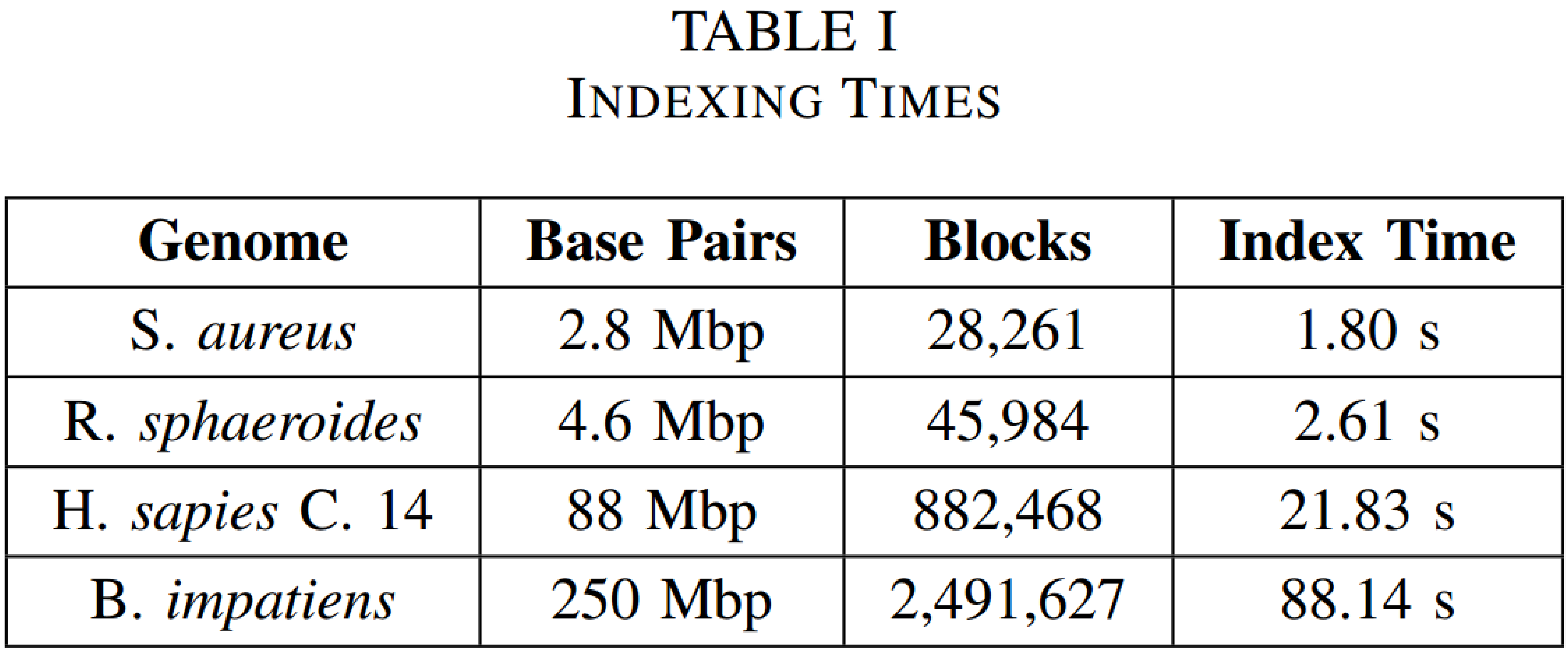
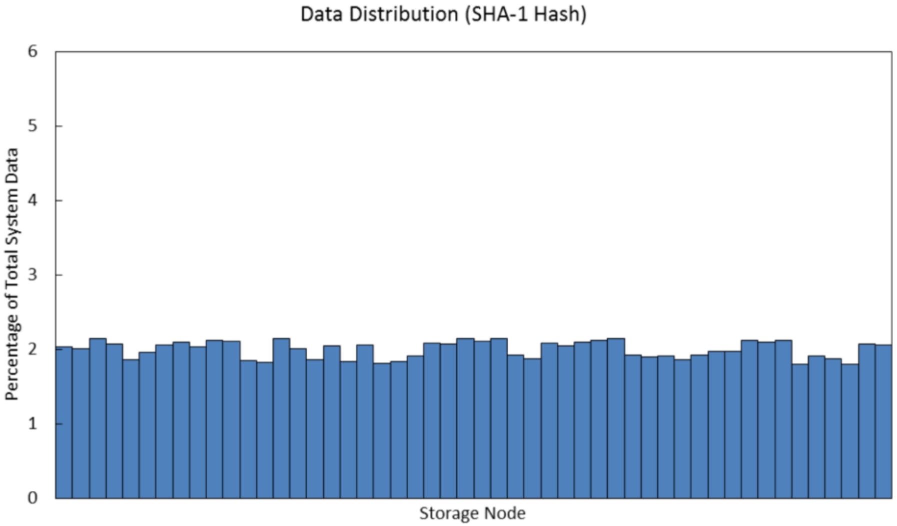


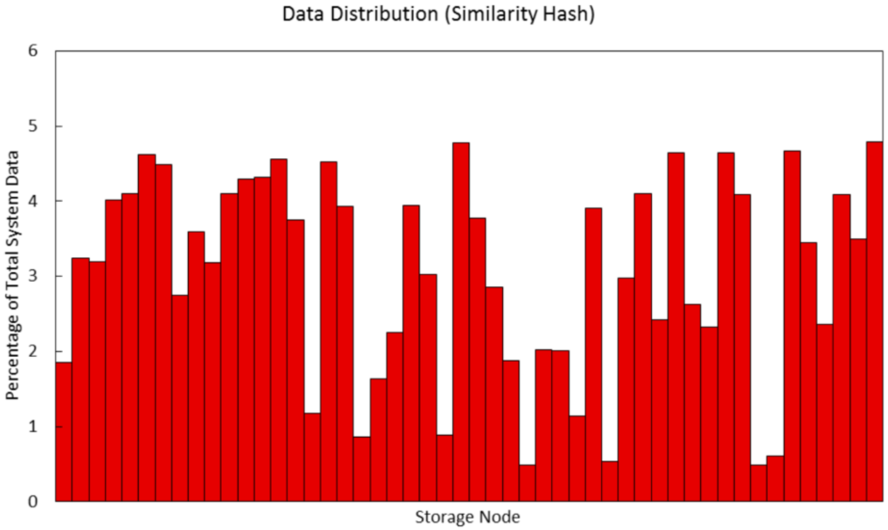
TABLE I
INDEXING TIMES

Genome	Base Pairs	Blocks	Index Time
S. *aureus*	2.8 Mbp	28,261	1.80 s
R. *sphaeroides*	4.6 Mbp	45,984	2.61 s
H. *sapies* C. 14	88 Mbp	882,468	21.83 s
B. *impatiens*	250 Mbp	2,491,627	88.14 s

Results

- Benchmark 2: Data distribution
  - After all datasets have been indexed count files per node
  - Compare versus flat SHA-1 hash





Results

- Benchmark 3: Issue a series of queries; measure response time and number of results
  - Exact match query whose target exists in the database
  - Exact match query whose target has a few errors to its match
  - Similarity query whose target exists in the database
  - Similarity whose target has a few errors to its match
  - Similarity whose target is randomly generated

Results

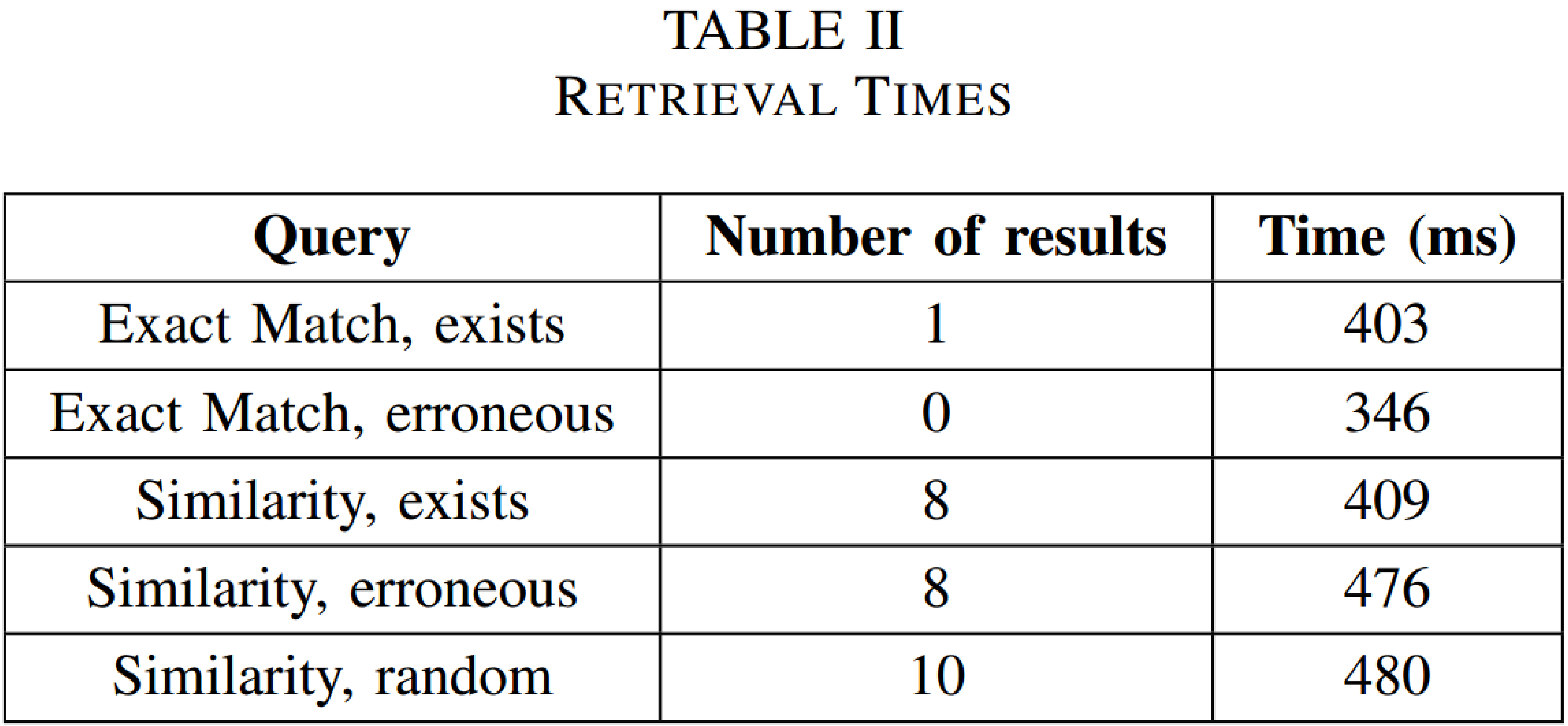


TABLE II
RETRIEVAL TIMES

Query	Number of results	Time (ms)
Exact Match, exists	1	403
Exact Match, erroneous	0	346
Similarity, exists	8	409
Similarity, erroneous	8	476
Similarity, random	10	480

Conclusion & Future Work

- The hashing scheme needs to be refined substantially in order to level the out the dispersion of the data
  - Data input one-by-one
  - Choosing initial vantage point (root)
- Currently queries must match the window they were indexed with
  - Sliding window over queries

Questions?

- Thanks!