DISTRIBUTED B+ TREE

An efficient key based distribution technique

Proneet Verma and Saurav Kumar

April 25, 2015

Indian Institute of Technology Kanpur

PROBLEM STATEMENT

PROBLEM STATEMENT

- To implement a B+ Tree index structure over a distributed multi-node network

PROBLEM STATEMENT

- To implement a B+ Tree index structure over a distributed multi-node network

- To devise an efficient distribution of nodes and analyze its performance

MOTIVATION

MOTIVATION

- In the age of ever increasing information collection and the need to look it up, there is a necessity to build systems which utilize the yet untapped and available compute resources.

MOTIVATION

- In the age of ever increasing information collection and the need to look it up, there is a necessity to build systems which utilize the yet untapped and available compute resources.

- The number of datasets being stored in a distributed manner, a scalable and efficient indexing approach is needed to locate the data.

APPROACH

Organization of index data

- Central Server:

Organization of index data

- Central Server:
  - Book-keeping jobs

Organization of index data

- **Central Server**:
  - Book-keeping jobs
  - Determines key distribution

Organization of index data

- **Central Server**:
  - Book-keeping jobs
  - Determines key distribution
  - Query response analysis

Organization of index data

- **Central Server**:
  - Book-keeping jobs
  - Determines key distribution
  - Query response analysis
  - Interface between index structure and user

Organization of index data

- Central Server:
  - Book-keeping jobs
  - Determines key distribution
  - Query response analysis
  - Interface between index structure and user
- Data Servers:

Organization of index data

- Central Server:
  - Book-keeping jobs
  - Determines key distribution
  - Query response analysis
  - Interface between index structure and user
- Data Servers:
  - Stores the index structure on disk

Organization of index data

- Central Server:
  - Book-keeping jobs
  - Determines key distribution
  - Query response analysis
  - Interface between index structure and user
- Data Servers:
  - Stores the index structure on disk
  - Each data server can process queries independently

Organization of index data

- Central Server:
  - Book-keeping jobs
  - Determines key distribution
  - Query response analysis
  - Interface between index structure and user
- Data Servers:
  - Stores the index structure on disk
  - Each data server can process queries independently

Organization of index data

- Central Server:
  - Book-keeping jobs
  - Determines key distribution
  - Query response analysis
  - Interface between index structure and user
- Data Servers:
  - Stores the index structure on disk
  - Each data server can process queries independently

Each server has its serverID, whose IP address and port are known.
Node pointers are pair of <serverID, fileName>.

Network calls

- Sockets: Using Python's *socket* library to handle network calls

Network calls

- **Sockets**: Using Python's socket library to handle network calls
- Threading: Deliver multiple node requests *parallely*

Network calls

- **Sockets**: Using Python's socket library to handle network calls
- **Threading**: Deliver multiple node requests parallely

Network calls

- **Sockets**: Using Python's socket library to handle network calls
- **Threading**: Deliver multiple node requests parallely

Invariant: Each request made to a server returns only on completion, with response.

Scoring Technique

- Score based system: Assign scores to the servers based on *network response time* and *hardware configurations* of data servers

Scoring Technique

- **Score based system**: Assign scores to the servers based on network response time and hardware configurations of data servers
- Likelihood: Assign scores to the keys based on their *probability of being queried*

Scoring Technique

- **Score based system**: Assign scores to the servers based on network response time and hardware configurations of data servers
- **Likelihood**: Assign scores to the keys based on their probability of being queried

Scoring Technique

- **Score based system**: Assign scores to the servers based on network response time and hardware configurations of data servers
- **Likelihood**: Assign scores to the keys based on their probability of being queried

Based on these two scores, we decide which data server to place this key on.

KEY DISTRIBUTION

When to decide?

When to decide?

Nodes are created when some other node splits. We use the first key in the node as the indicative key to determine which data server this new node should reside on.

When to decide?

Nodes are created when some other node splits. We use the first key in the node as the indicative key to determine which data server this new node should reside on.

How to decide?

When to decide?

Nodes are created when some other node splits. We use the first key in the node as the indicative key to determine which data server this new node should reside on.

How to decide?

- Randomly

When to decide?

Nodes are created when some other node splits. We use the first key in the node as the indicative key to determine which data server this new node should reside on.

How to decide?

- Randomly
- Equal segmentation of key ranges and randomly assign each sub-range to a data server

When to decide?

Nodes are created when some other node splits. We use the first key in the node as the indicative key to determine which data server this new node should reside on.

How to decide?

- Randomly
- Equal segmentation of key ranges and randomly assign each sub-range to a data server
- Using key and server scores to compute a mutual score

How the query is distributed?

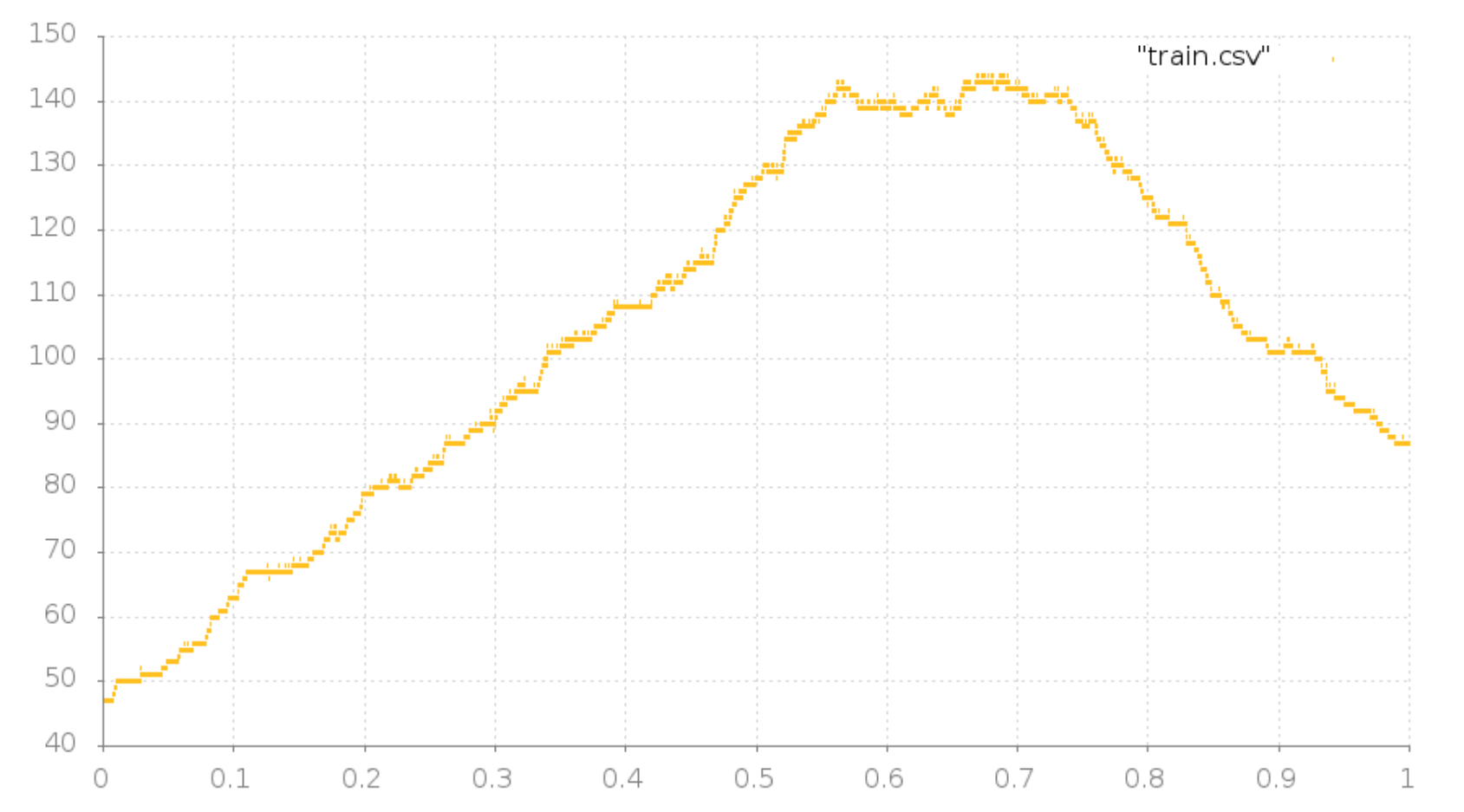

Figure: Plot showing the frequency of keys being returned

Random

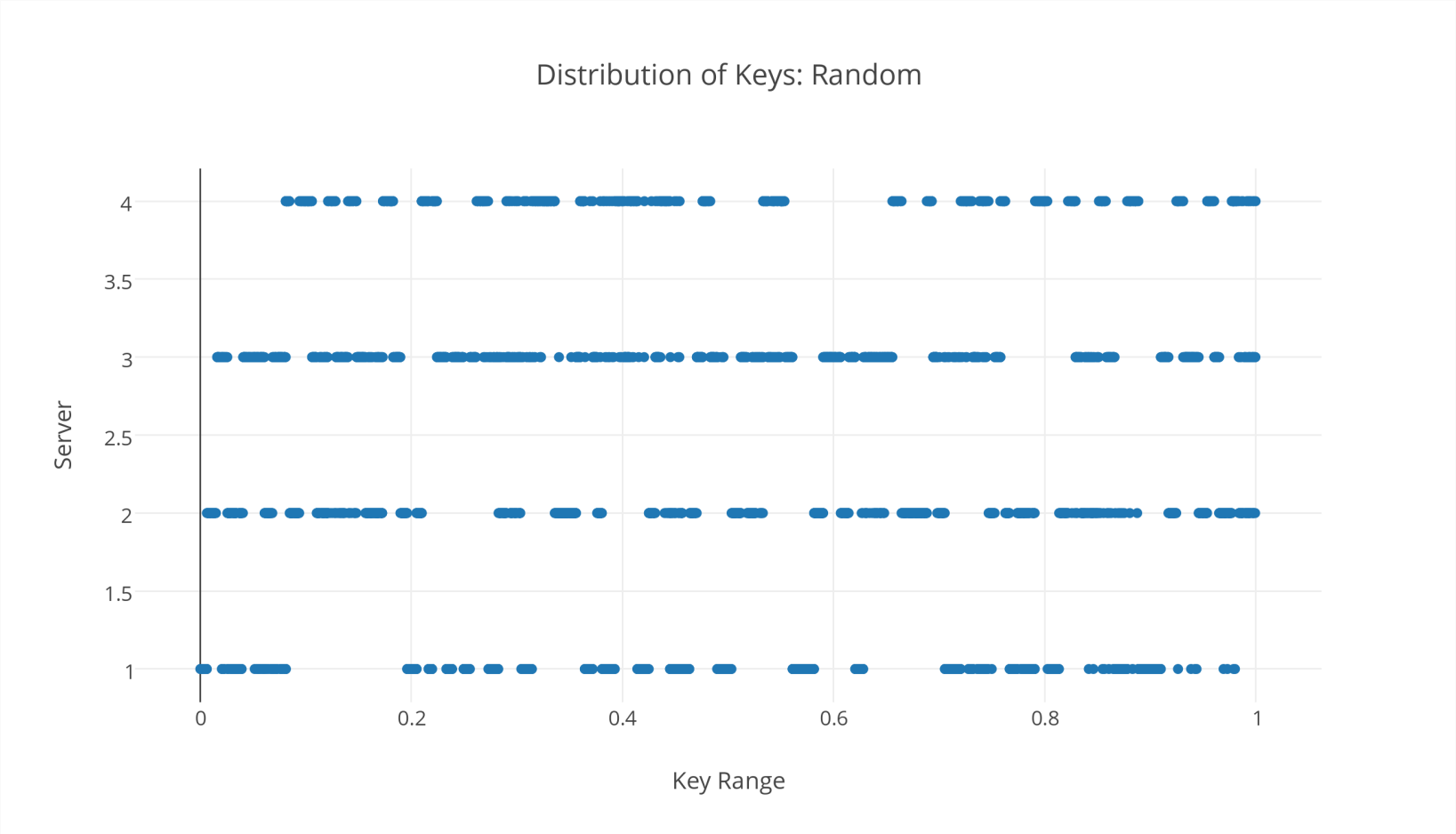

Figure: Distribution of keys among servers based on random strategy

Segmentation (Best Case)

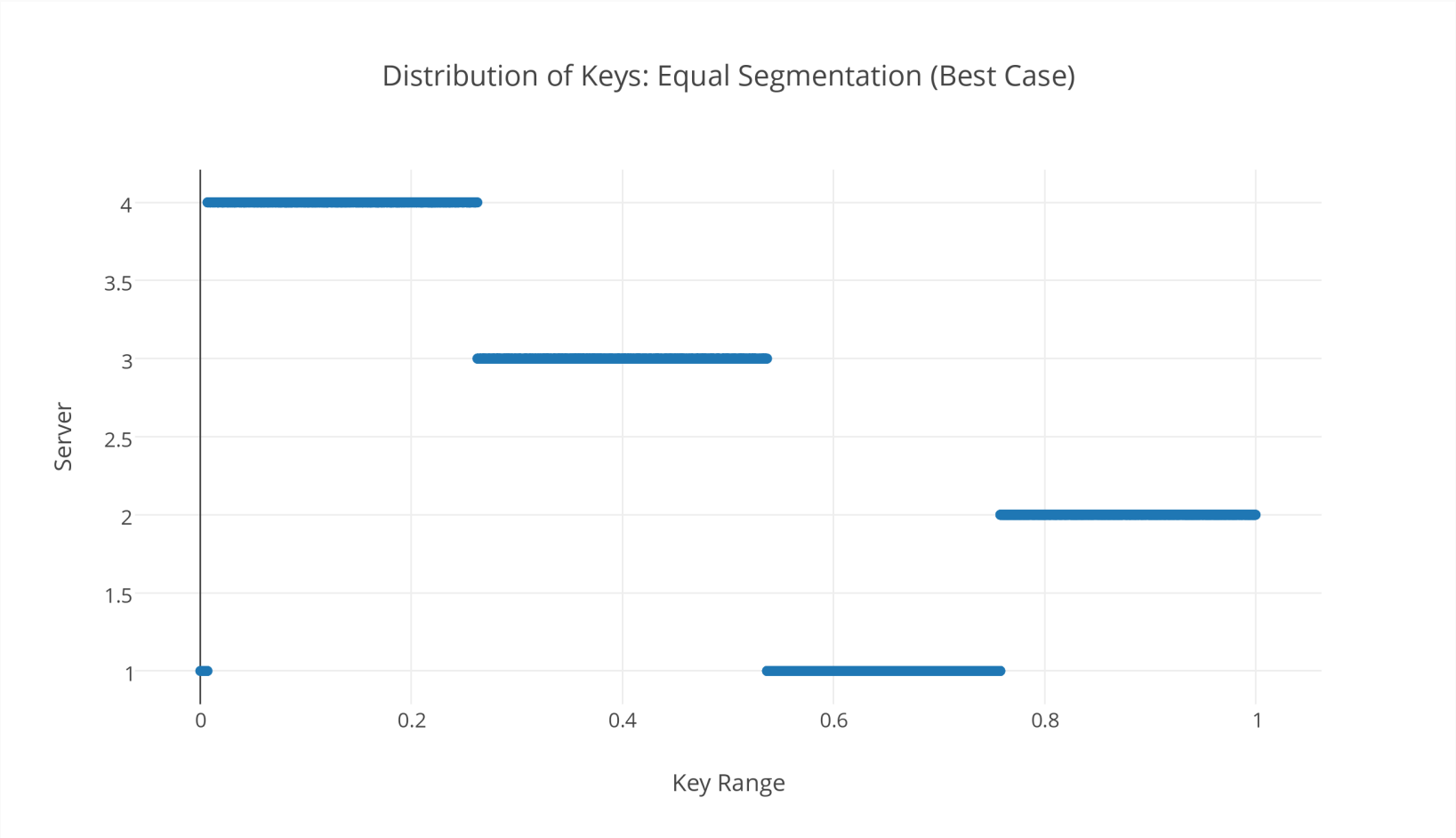

Figure: Distribution of keys among servers based on equal splitting of key range

Mutual Score Calculation

We need a function with these properties:

- Takes in two arguments both between 0 and 1

Mutual Score Calculation

We need a function with these properties:

- Takes in two arguments both between 0 and 1
- Returns a high value when key score and server score are both high (or low)

Mutual Score Calculation

We need a function with these properties:

- Takes in two arguments both between 0 and 1
- Returns a high value when key score and server score are both high (or low)
- Returns a low value when key score and server score are opposite.

Mutual Score Calculation

One such function is:

$$f(x, y) = c \times (x - 0.5) \times (y - 0.5)$$

Mutual Score Calculation

One such function is:

$$f(x, y) = c \times (x - 0.5) \times (y - 0.5)$$

Demerit: It works good only for two data server system because it ignores any other scores other than high and low.

Using

$$f(x, y) = c \times (x - 0.5) \times (y - 0.5)$$

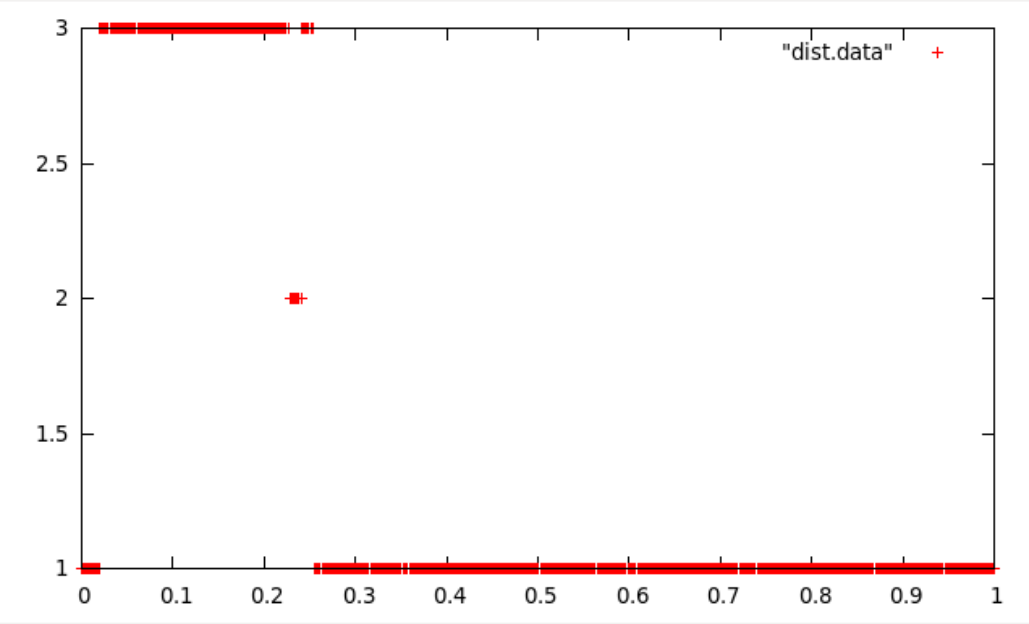

Figure: Skewed distribution of keys using this function

Mutual Score Calculation

Another function is:

$$f(x, y) = 1 - \text{abs}(x - y)$$

Merit: Very simple to compute and works very satisfactorily

RESULTS

Query Distribution

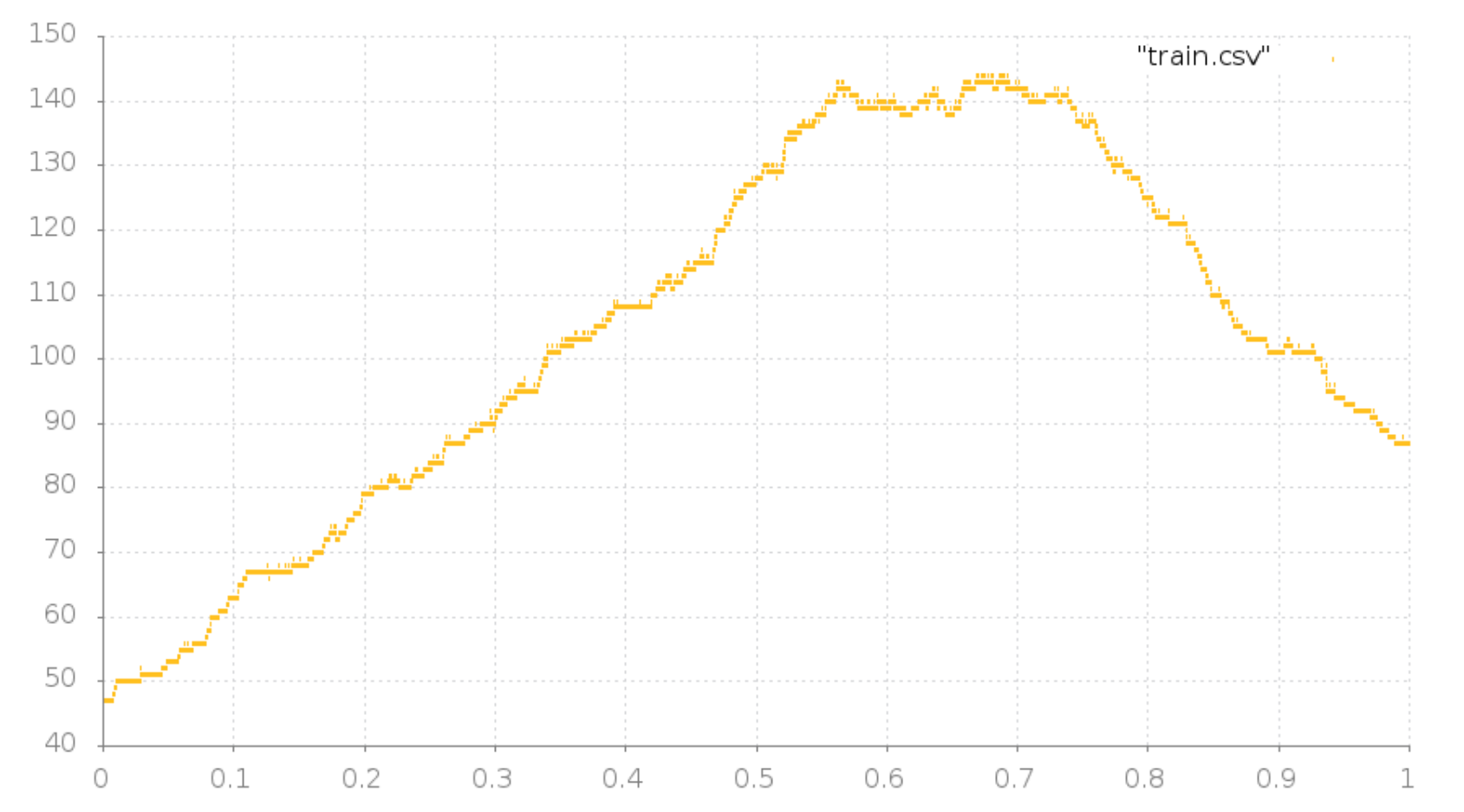

Figure: Plot showing the frequency of keys being returned

Key Distribution using scoring

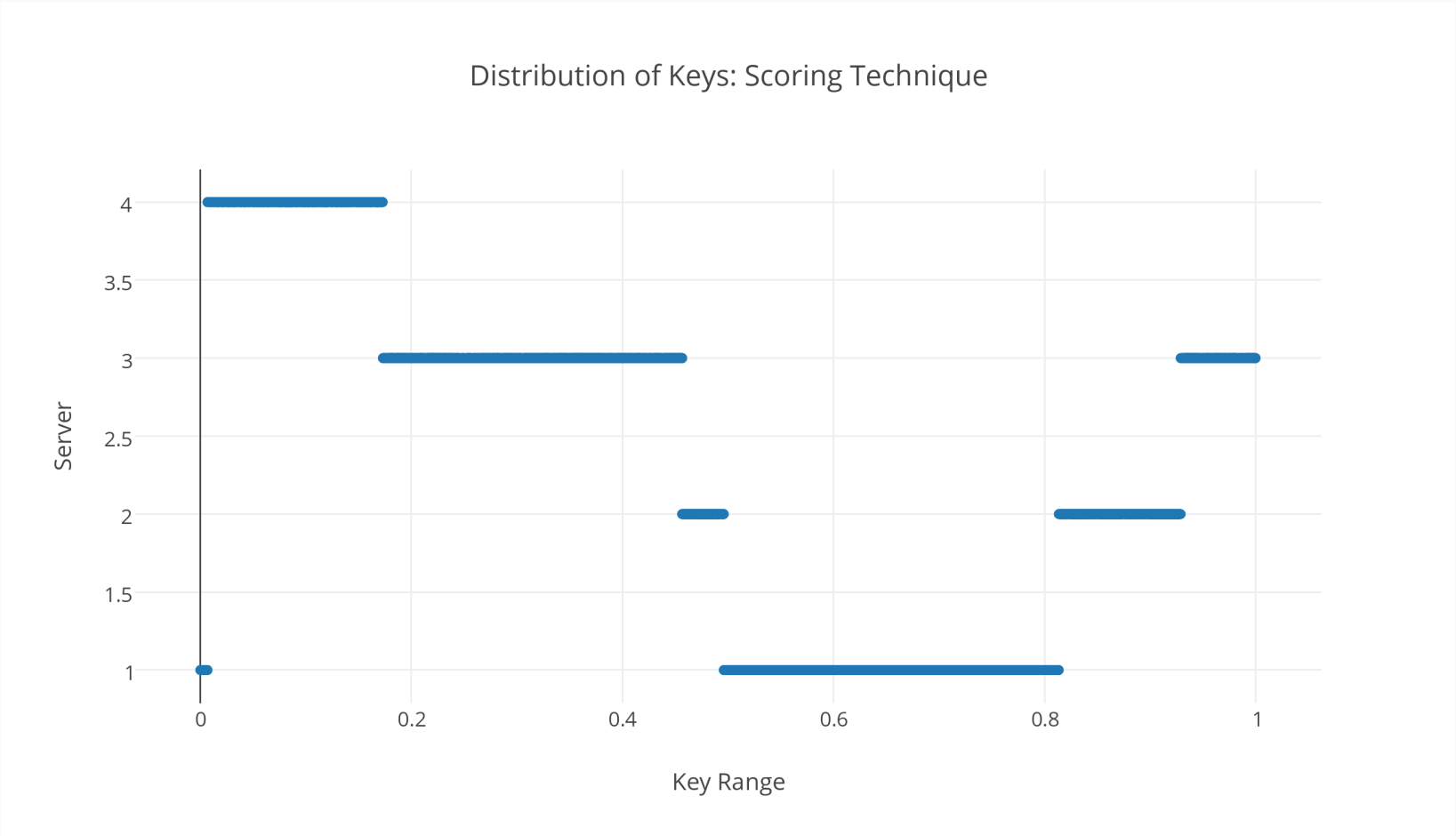

Figure: Distribution of keys among servers based on our scoring technique

Comparison

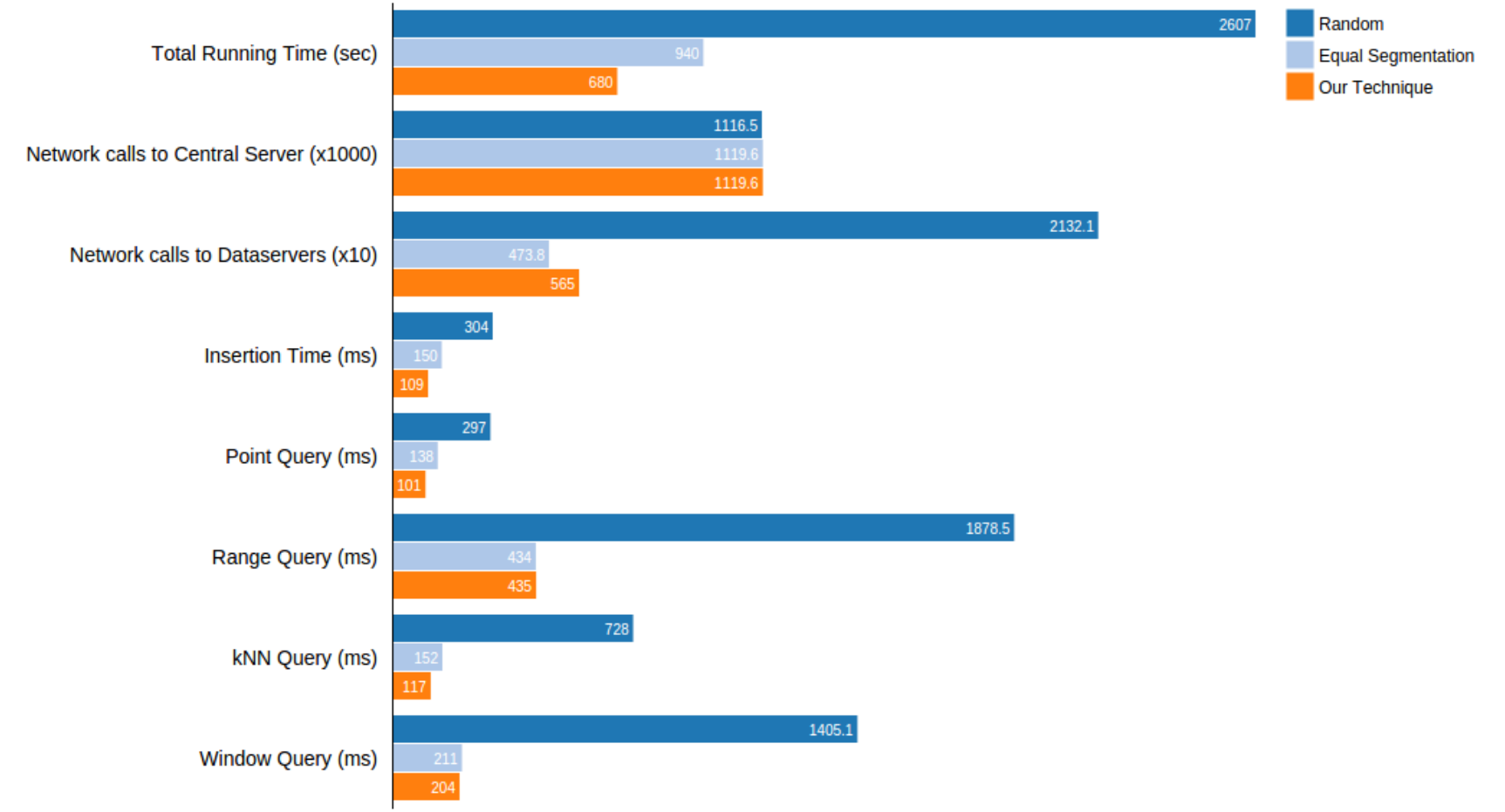

Figure: Comparison of key distribution techniques

Another Query Distribution

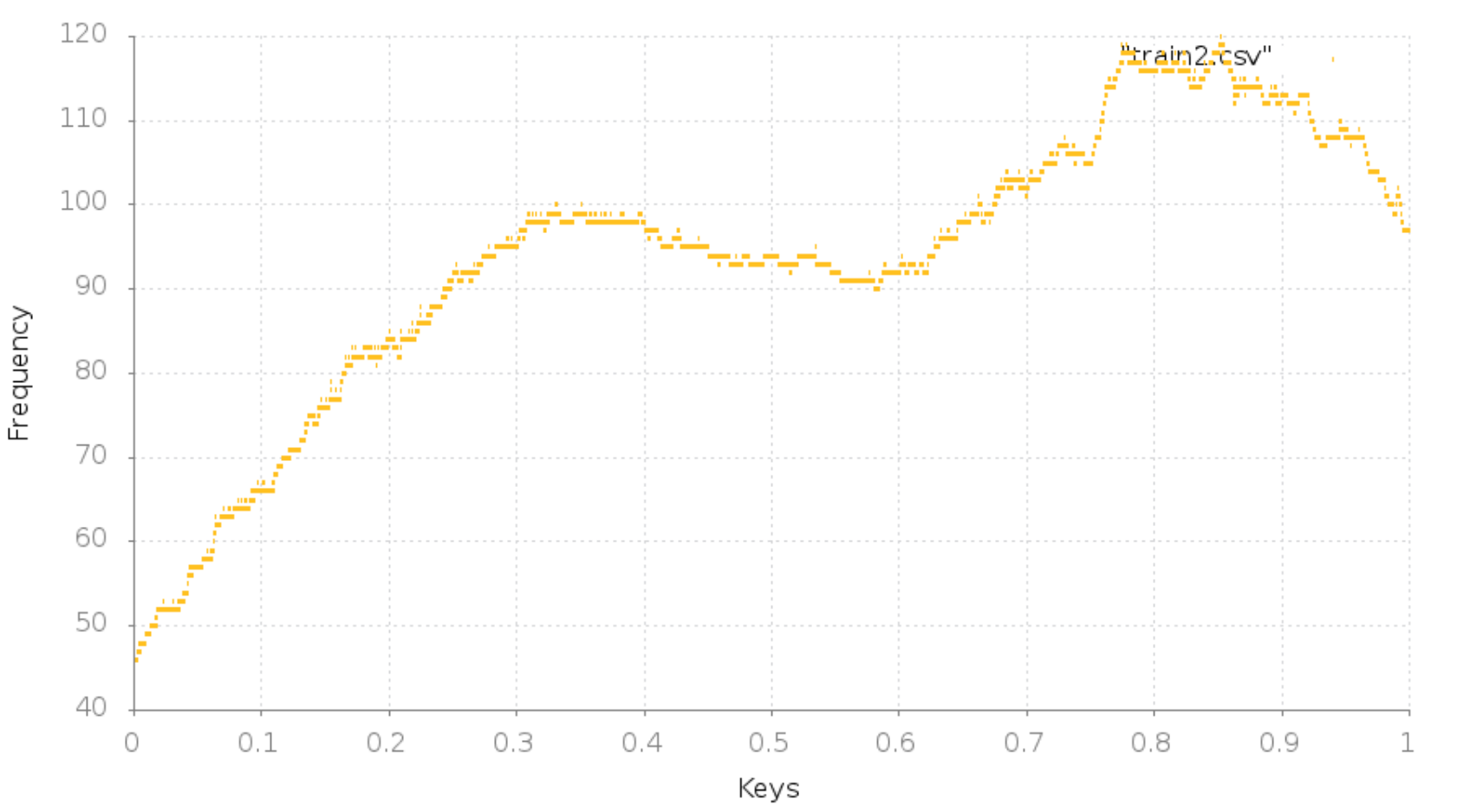

Figure: Plot showing the frequency of keys being returned

Random

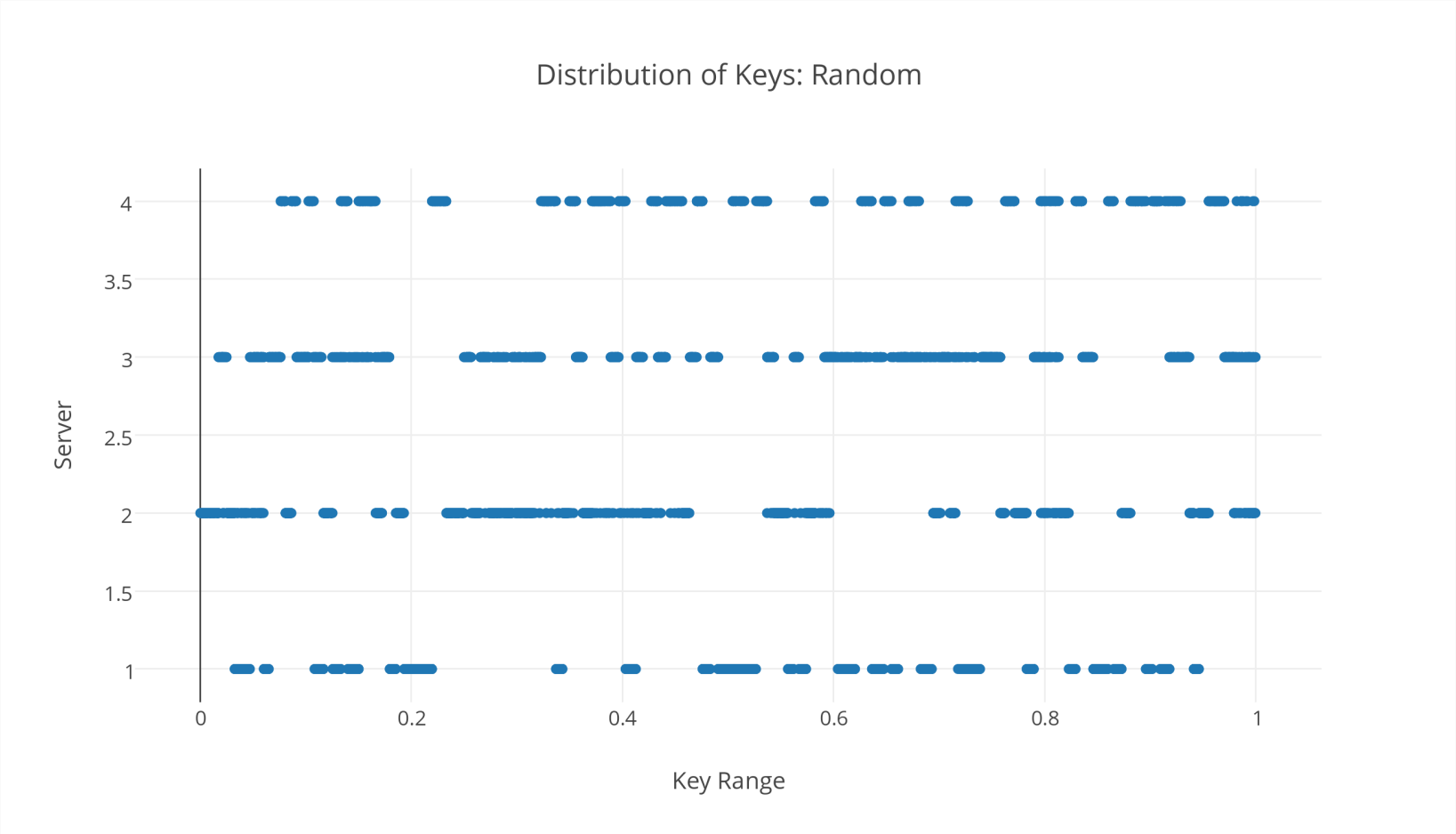

Figure: Distribution of keys among servers based on random strategy

Segmentation (Best Case)

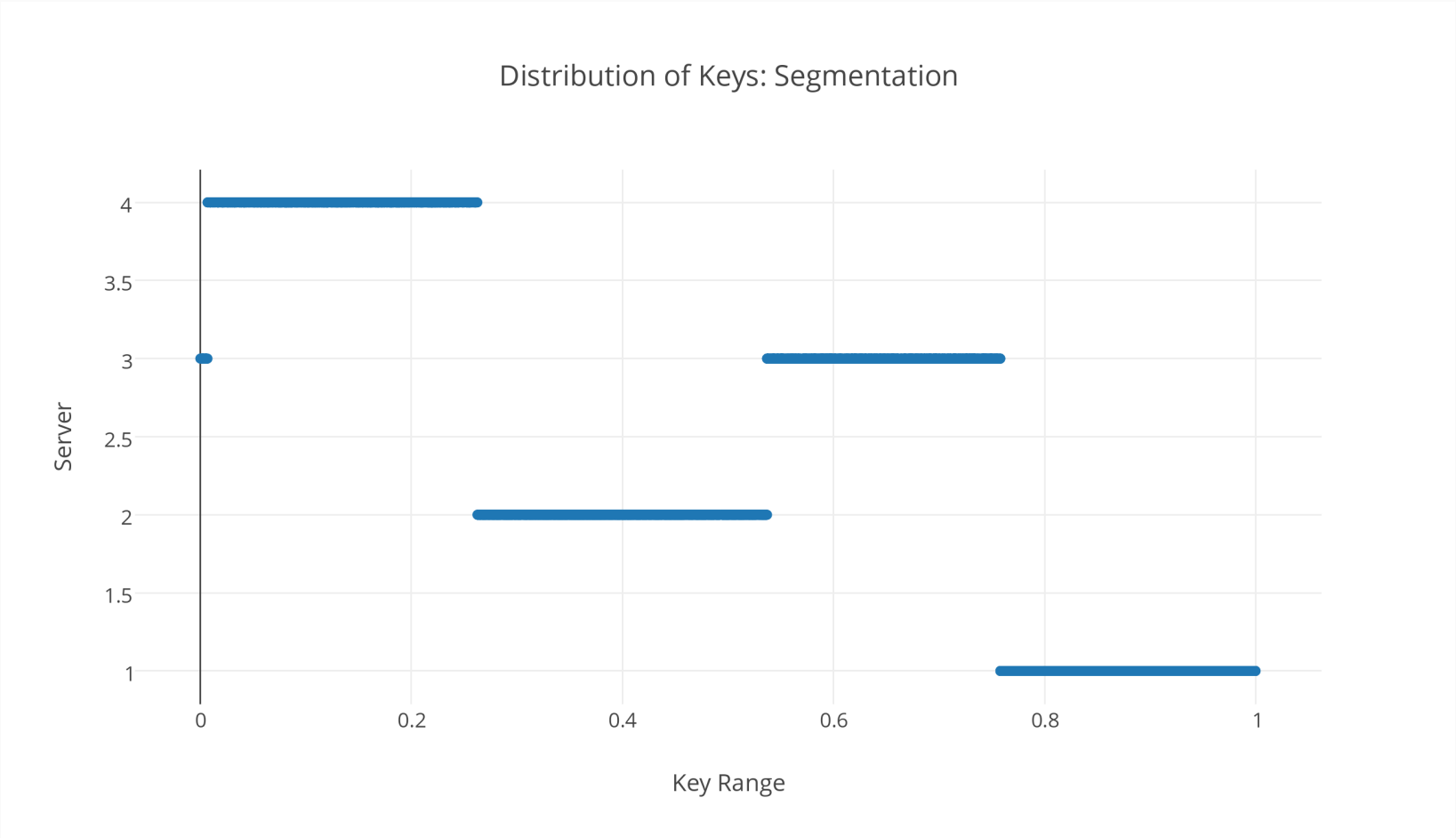

Figure: Distribution of keys among servers based on equal splitting of key range

Key Distribution using scoring

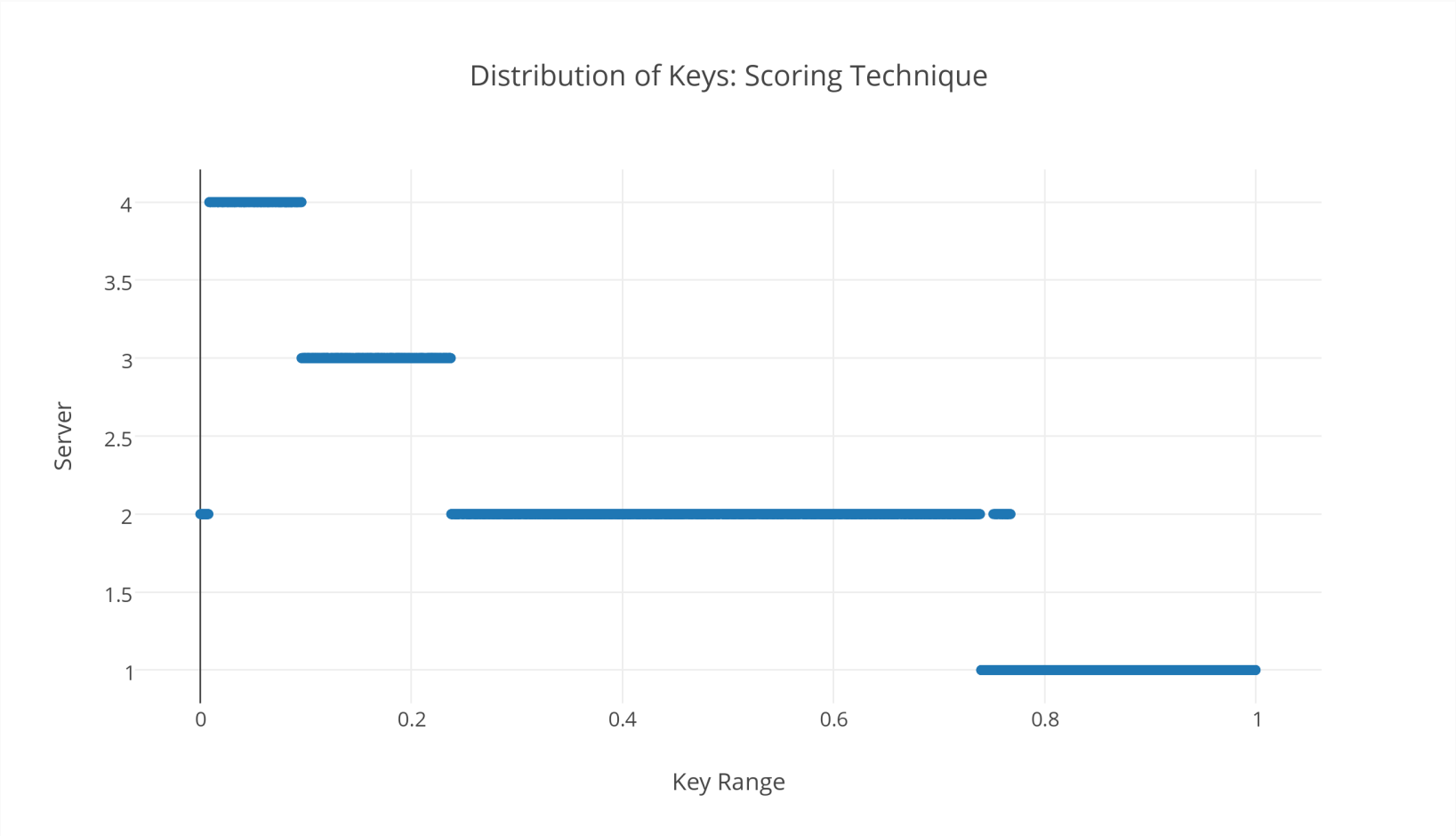

Figure: Distribution of keys among servers based on our scoring technique

Comparison

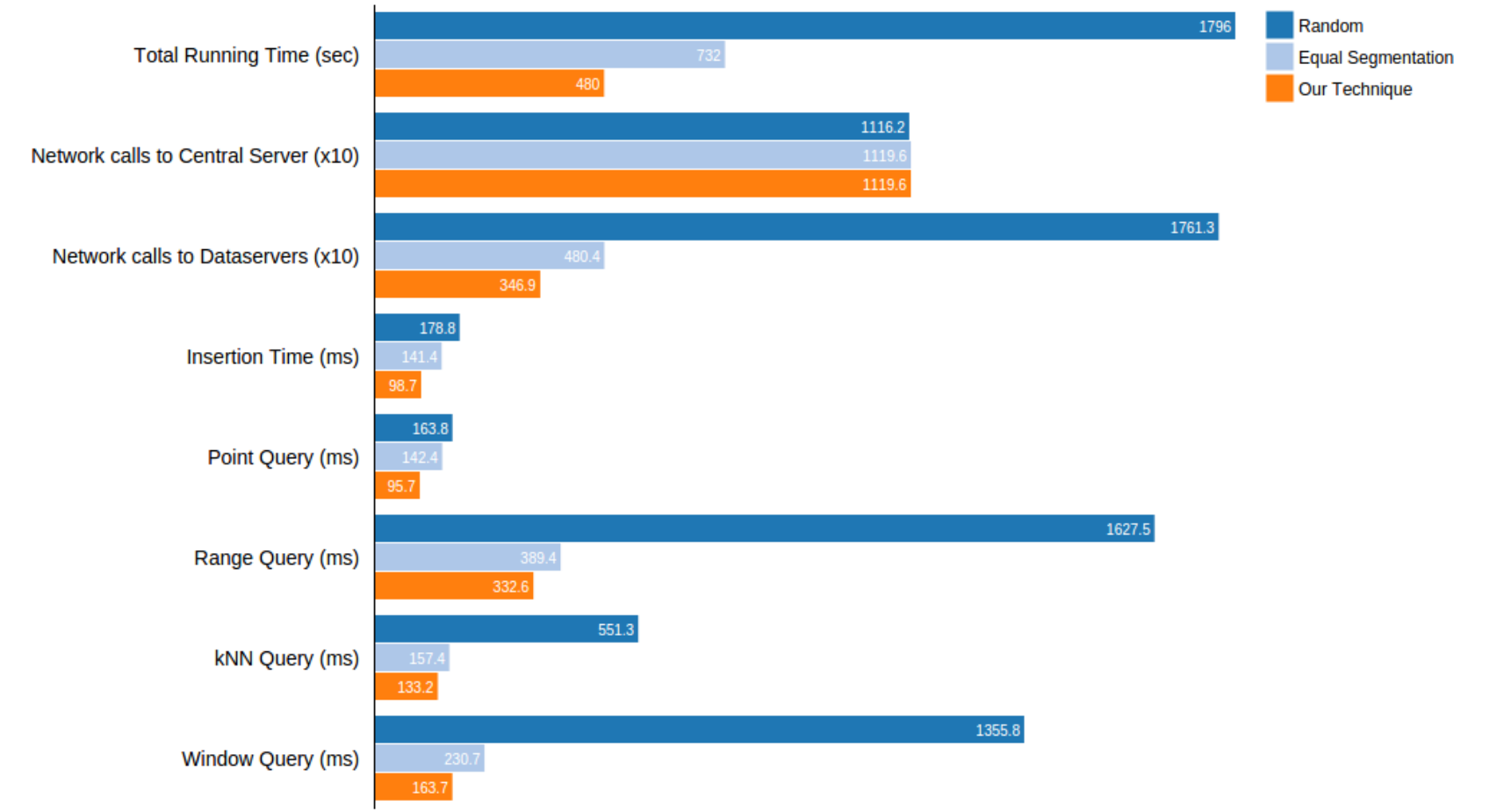

Figure: Comparison of key distribution techniques

Conclusion

- Our scoring technique outperformed the other two trivial approaches.

Conclusion

- Our scoring technique outperformed the other two trivial approaches.
- For queries which are more popular, there has been a considerable amount of reduction in their look up time.

Conclusion

- Our scoring technique outperformed the other two trivial approaches.
- For queries which are more popular, there has been a considerable amount of reduction in their look up time.
- With the analysis that we have put forward, we can infer that this indexing approach can be widely used for faster look up of globally important terms.

Questions?

Credits: Beamer(mtheme), GNUPlot, D3JS, ShareLaTeX

Thank You