An optimization-inspired approach to parallel sorting

Team Metropolis:
James Farzi, JJ Lay, Graham West

February 13, 2019

Outline

Data

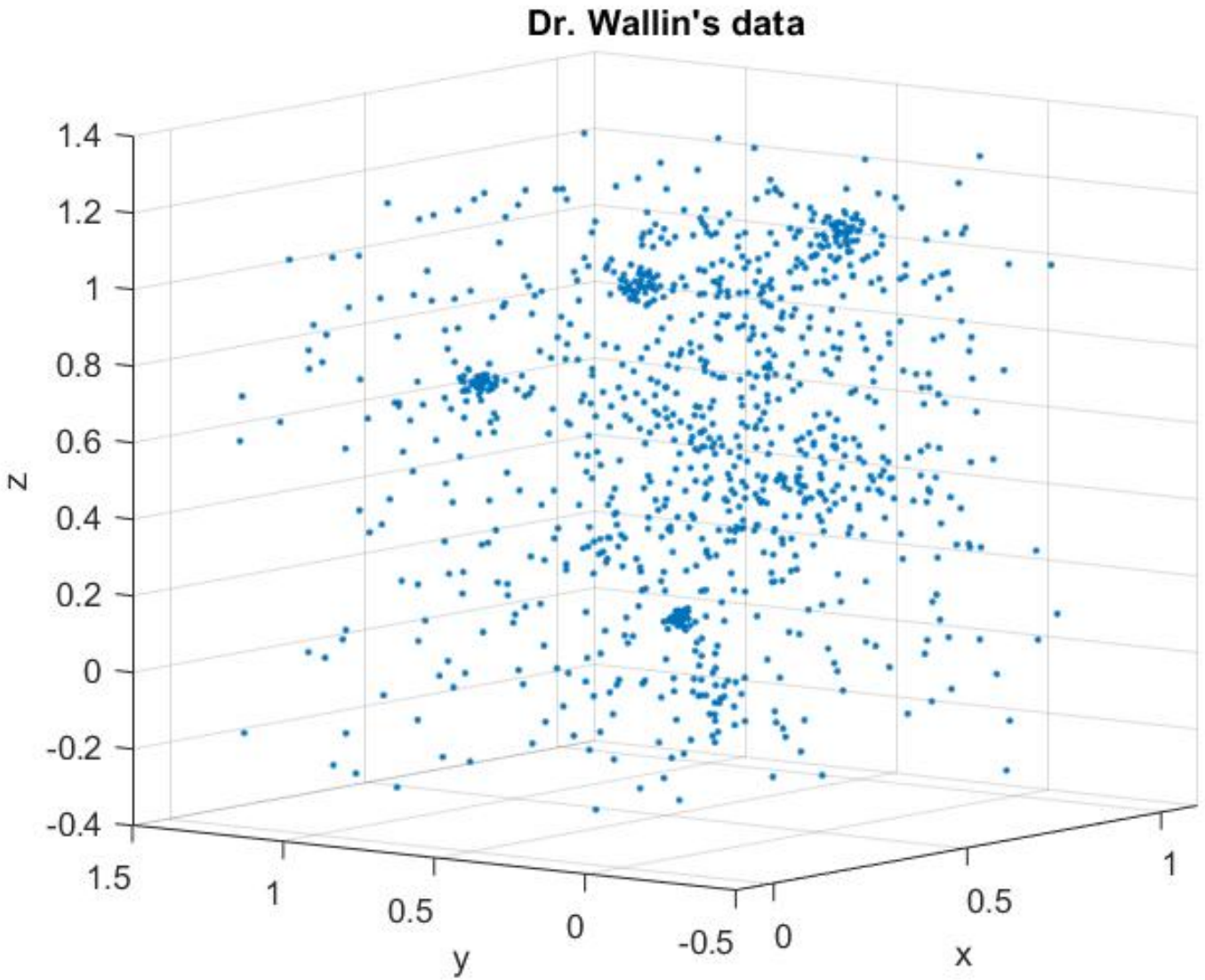

Figure 1: 1000 data points

The Algorithm

The Algorithm

Initial Phase

- Initialize MPI
- Read list of available data files
- Distribute available data files to workers

Work Phase

- Import data from files
- Perform initial sort
- Find uniform bins
- Exchange data
- Perform final sort

Distributing/Importing Files

distributeFiles

- Rank 0 reads files matching `datafile000000.txt`
- Files are distributed round robin to workers

importFiles

- Allocate 1D array of length:
  `NumFiles` $\times$ `MaxLinesPerFile` $\times$ 4 columns
- Each worker reads ASCII files and appends data to array

Sorting

We implemented Linked List Insertion Sort and Bubble Sort

Linked List Insertion Sort

- Best Case N, Worst Case N^2
- Selects items one at a time and inserts them in the appropriate place along a linked list
- Then replaces the original array with the elements from the linked list

Bubble Sort

- Use: time testing for efficiency
- Not to be used during normal operations

Binning

Binary search

- Since the data is sorted, we can use a binary search to find where the bin edges lie in index space
- We can then subtract successive edges' indices to find the number of elements in that bin

Initial bin edges

- As a first approximation, we assume that the bin edges are equally-spaced
- We then improve this over time

Binning

Adapting the bins

for interior bin edges (endpoint bins stay constant):

$$\begin{aligned} \Delta C &= 2.0(c_i^n - c_{i-1}^n)/(c_i^n + c_{i-1}^n) \\ \Delta B &= b_{i+1}^n - b_i^n \\ b_i^{n+1} &= b_i^n + \alpha \Delta C \Delta B \end{aligned} \tag{1}$$

where $0 < \alpha < 0.5$ and $b_i^n < b_{i+1}^n$ for all n

Uniformity metric

$$U^n = \max\left(\frac{c_{\max} - c_{\text{avg}}}{c_{\text{avg}}}, \frac{c_{\text{avg}} - c_{\min}}{c_{\text{avg}}}\right) \tag{2}$$

Binning

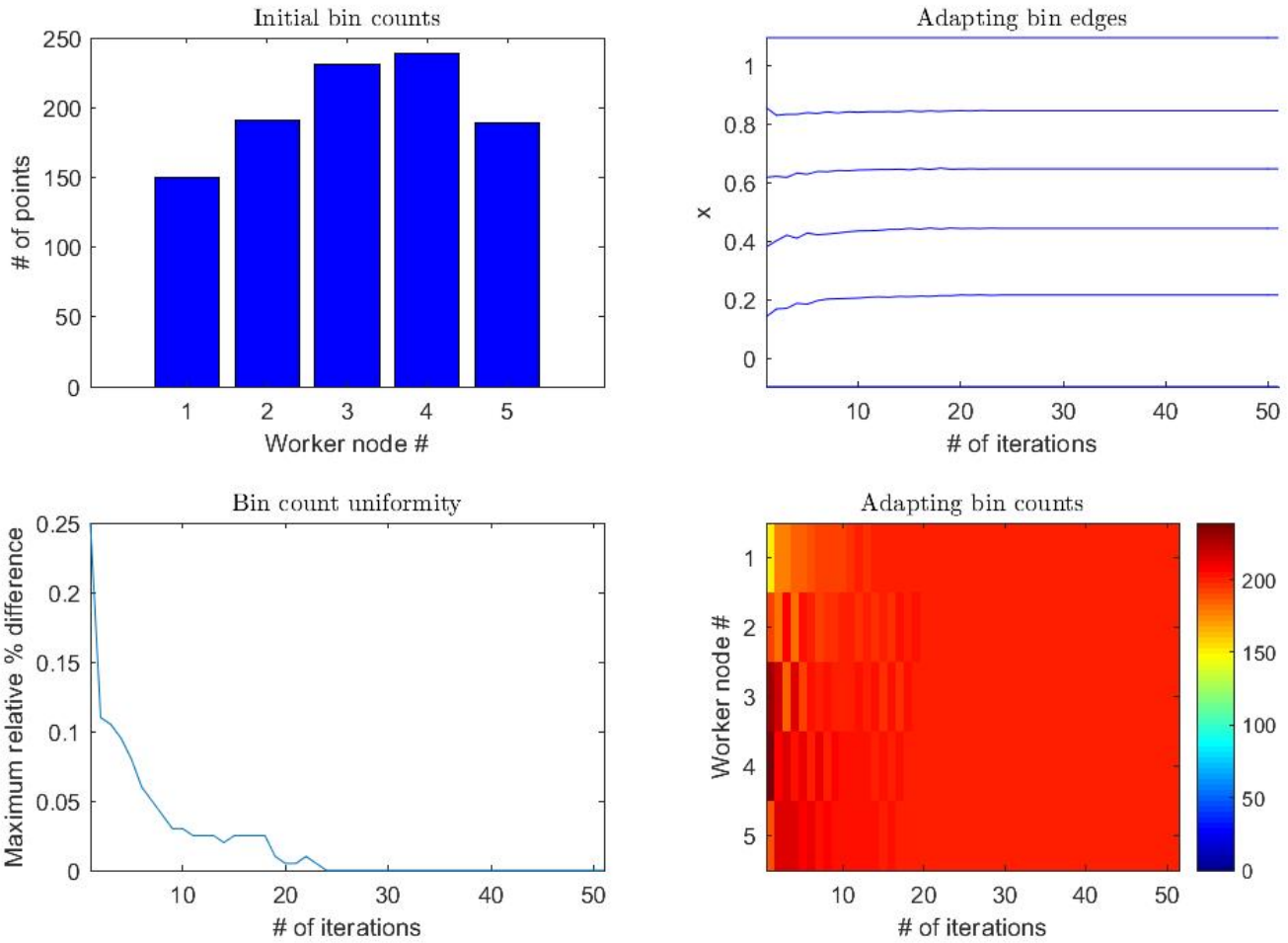

Figure 2: 5 nodes, 1000 data points, $\alpha = 0.475$

Binning

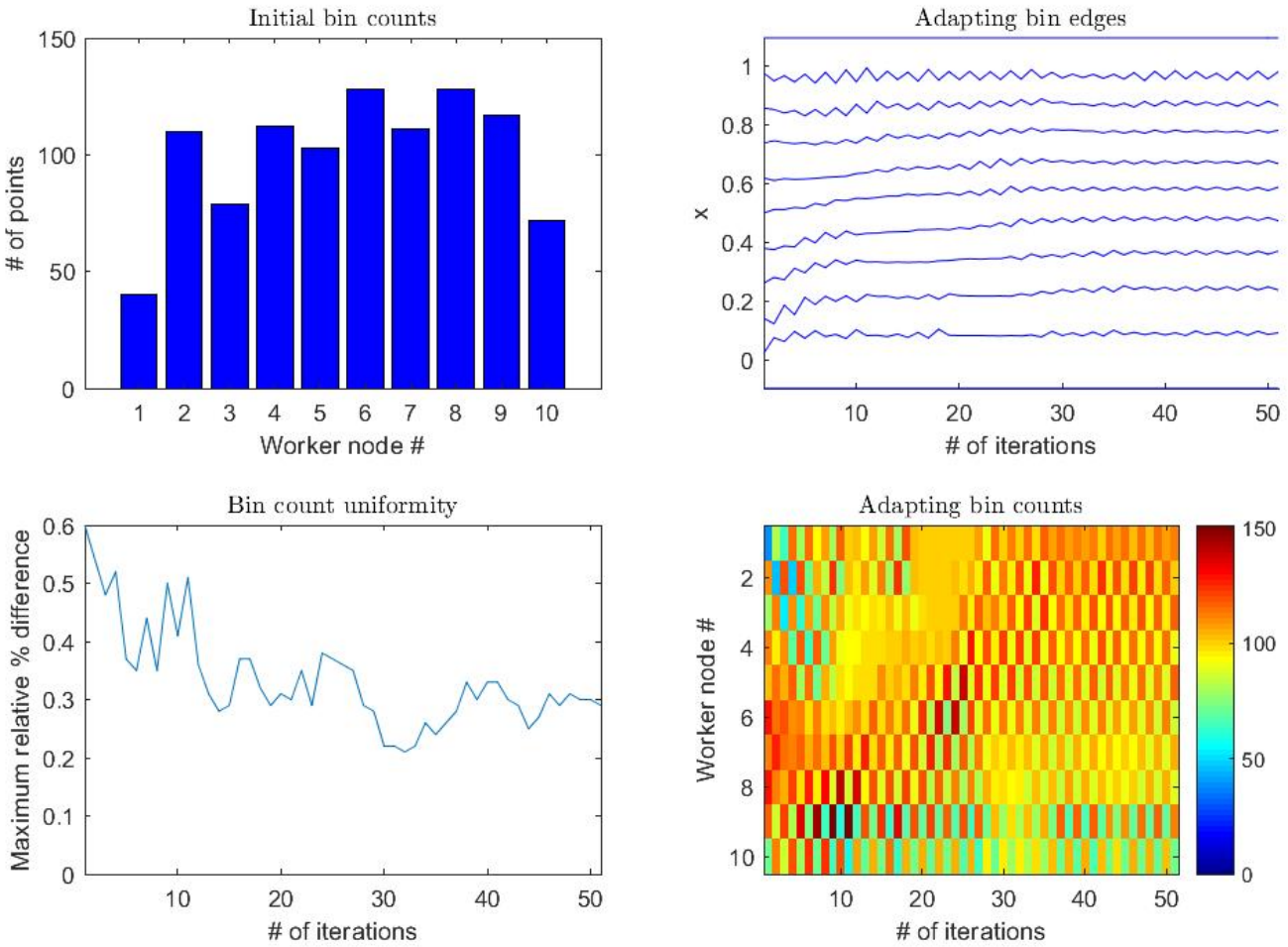

Figure 3: 10 nodes, 1000 data points, $\alpha = 0.475$

Binning

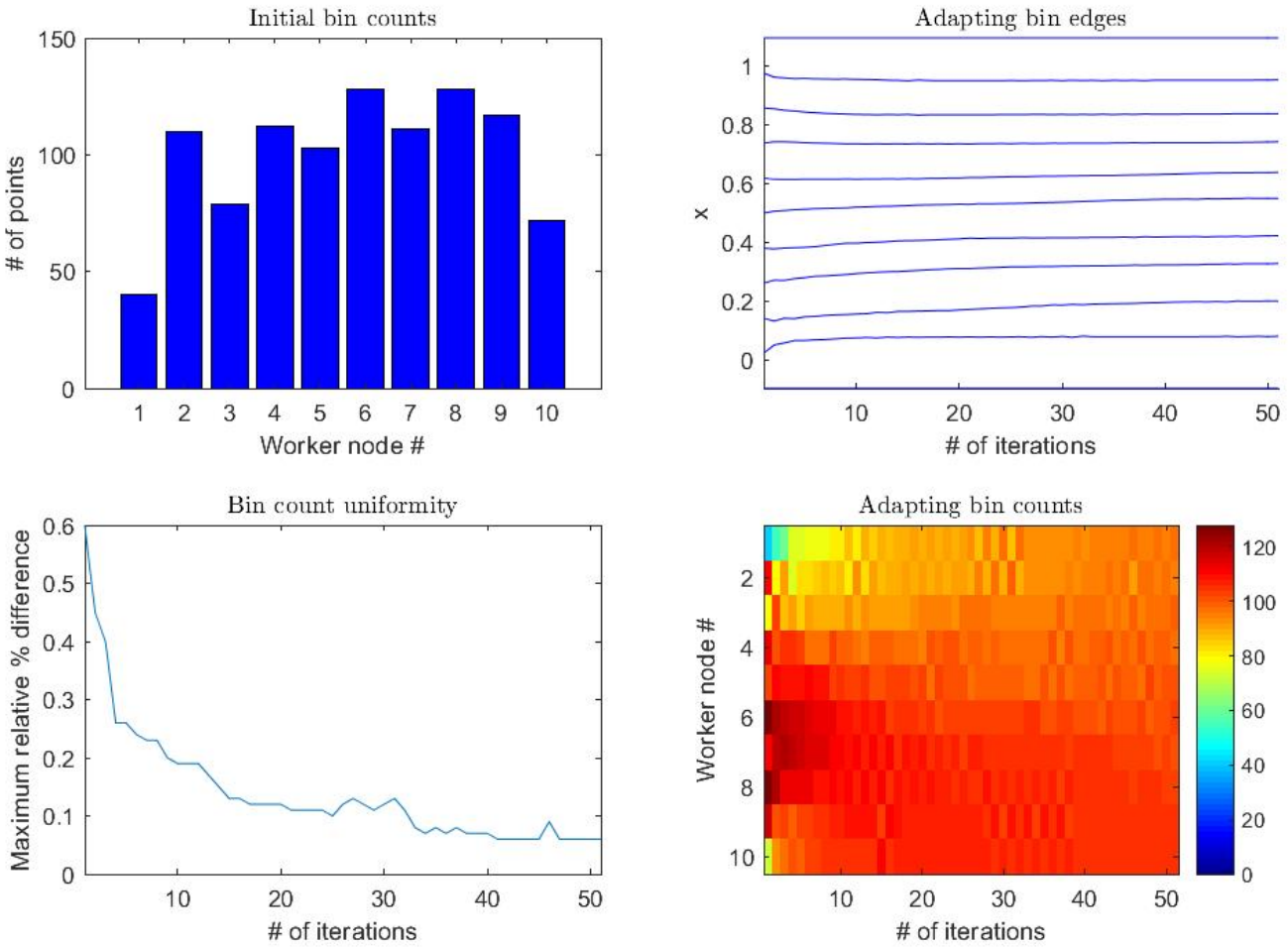

Figure 4: 10 nodes, 1000 data points, $\alpha = 0.25$

Exchanging data: using swapArrayParts then cleanUp

swapArrayParts

- Takes a specified amount of data from one node and appends it to the end of the existing data for the receiving array
- Inside the function: first transmits the length of data to be received then transfers the data
- No data is to be deleted at this time
- Allows for flexibility in implementation algorithm

cleanUp

- To be used after all nodes have performed a swapArrayParts to all other nodes
- Will go back through each nodes array removing data that had been transferred by swapArrayParts and resizing the array for each node

Testing

distributeFiles

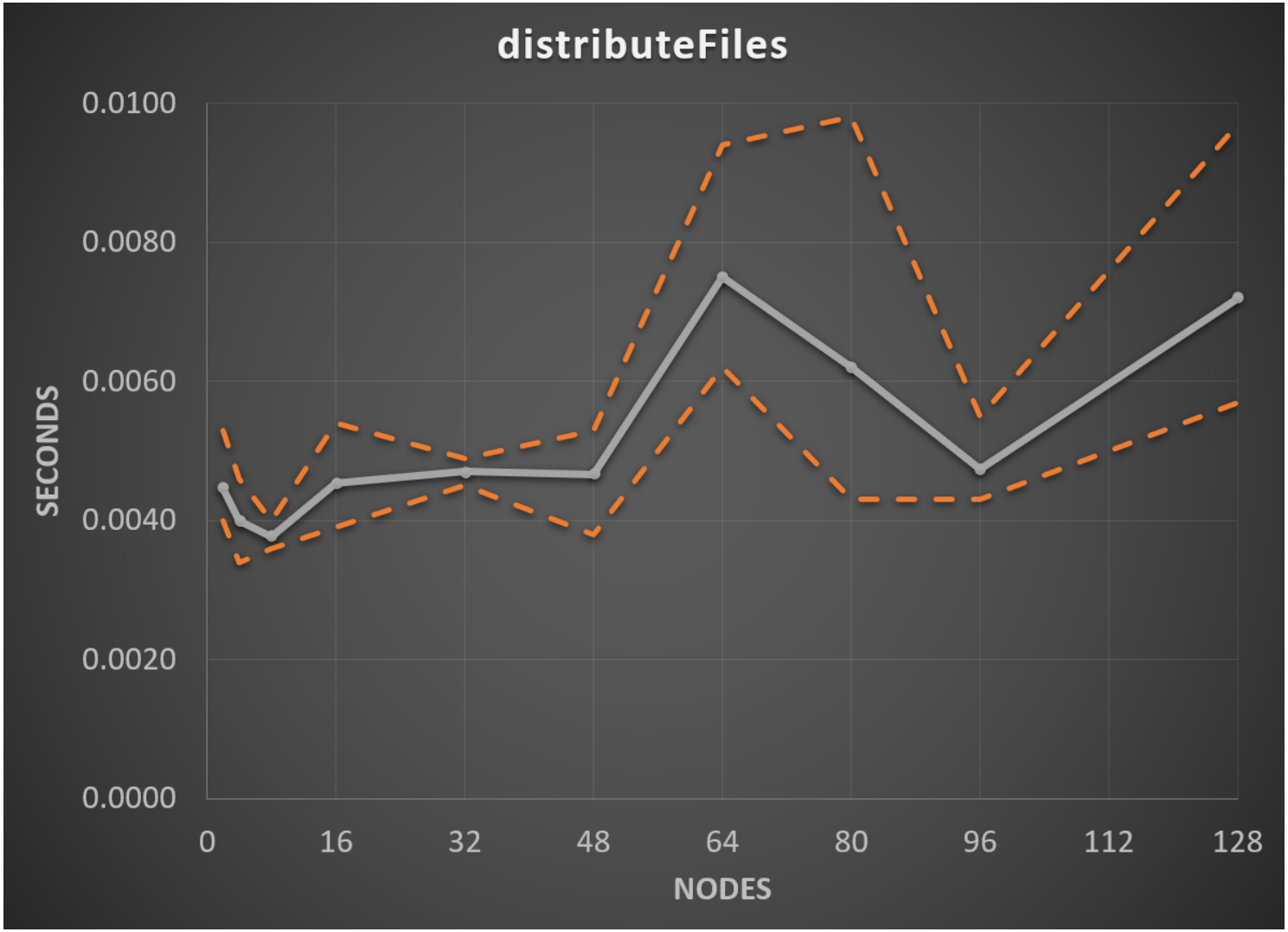

Figure 5: Time to read directory and distribute files to workers

importFiles

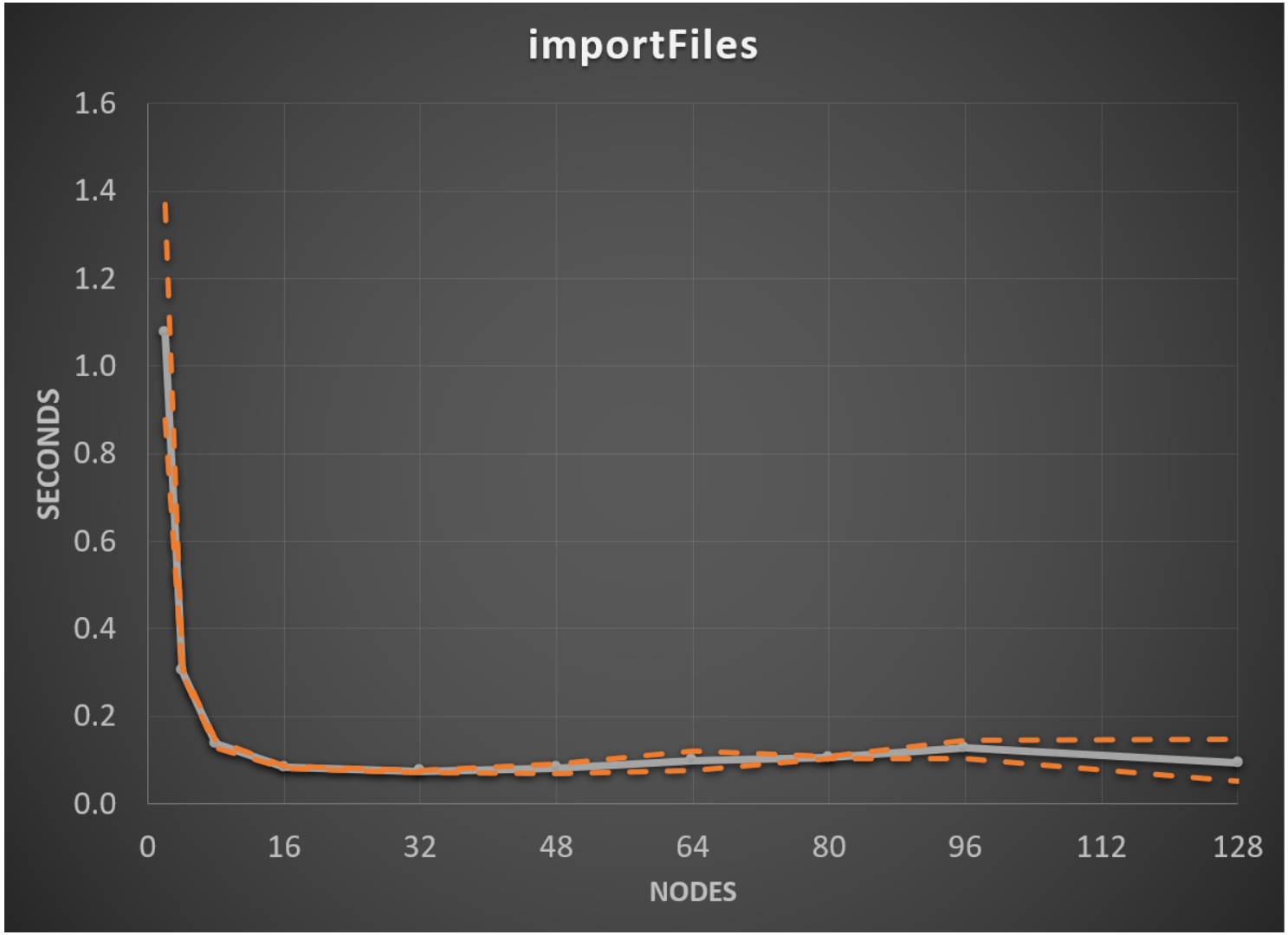

Figure 6: Time to import 1000 rows from each file

initialSort

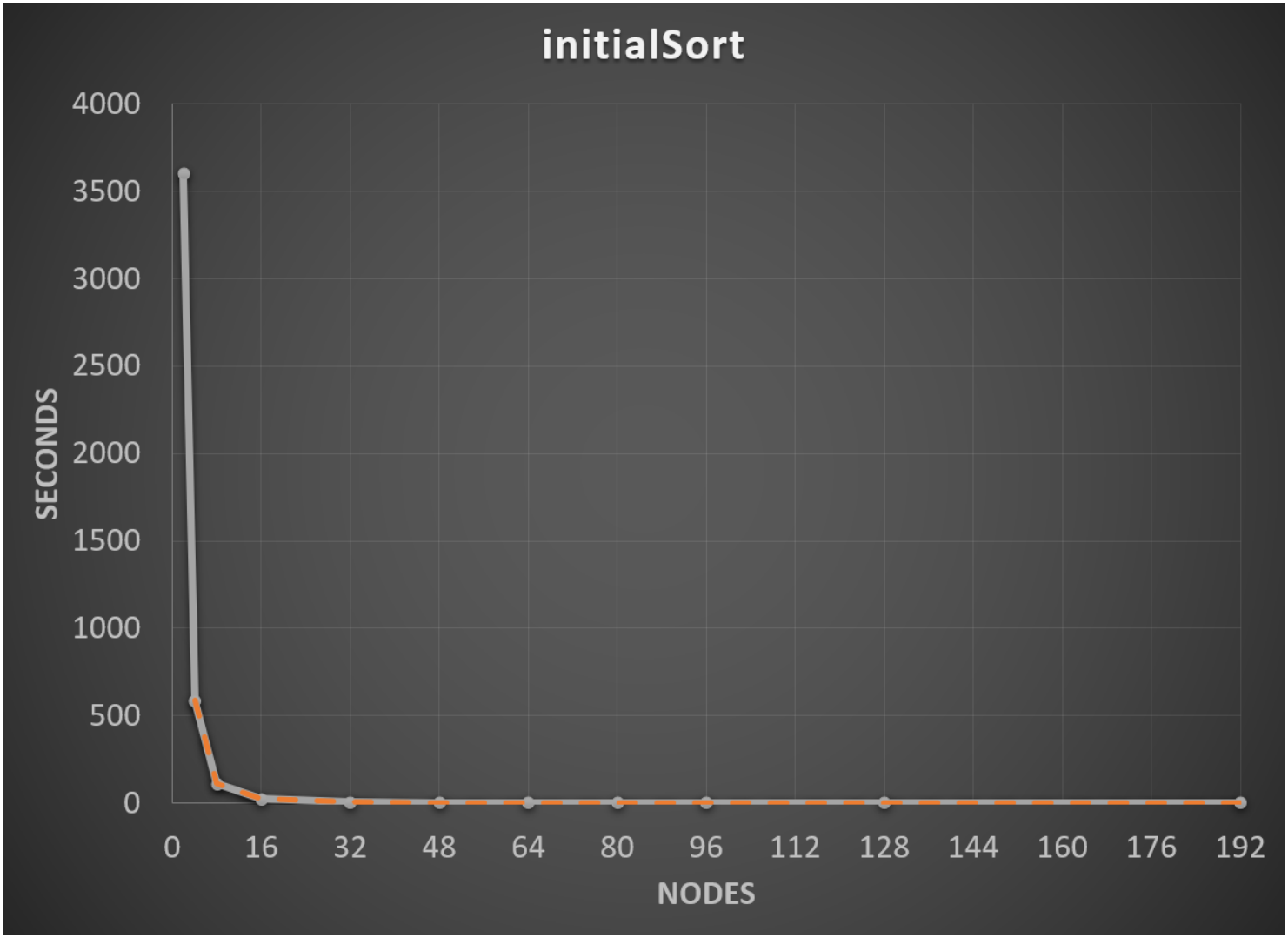

Figure 7: Time to sort 501000 rows

Conclusions

Challenges

Integration

- Identifying structure of parameters and return values
- Needed better planning of data types up front

Git

- Resolving merge conflicts
- Basic Git workflow

Coding

- Forgot to remove debug controls while collecting performance data

Testing

- Long runtimes when testing with full dataset
- Failures during long runs

Future Work

Reading the data

- Read binary datainstead of ASCII for speedup

Data types

- For simplicity, use vectors/structs instead of arrays

Finish data swap

- Resolve scaling issues (overflowing buffer)

Testing

- Easier collection of performance data
- Use loops in qsub files to automate multiple runs