Daylan Quinn

Professor Hare

CS 441

9 May 2021

Clojure Project Report

Text

Description automatically generated This report covers my findings in testing the execution of a mergesort algorithm in Clojure concurrently using multithreading. To recap, Clojure is a functional programming language and a dialect of lisp that originated in 2007. I ran the Clojure application though IntelliJ IDE, since the Eclipse plug-in seems to not be supported at the moment (maybe I just had issues on my machine).

To start the application, I created a function that will read in a file and create a list of the read in items from the file. This is for the list of unorganized numbers that we are to sort later. The next thing I did was created functions for each of the number of threads I will want to execute later. These functions partition the list of numbers into however many threads we want to execute over. For instance, if I want to run 16 concurrent threads, then I will split the list into 16 partitions to run over each thread.

Text

Description automatically generated The next thing the application does is creates the function for mergesort. To recap, mergesort splits a list into pieces and sorts the lists and merges them together recursively. It is a divide and conquer method to sorting lists. The mergesort method I used was found at <https://gist.github.com/alco/2135276> and was edited to run smoothly with the desired project requirements for multithreading.

With functions for placing the file contents into a list, the number of threads I want to use, and mergesort, the next thing I did was run them all together (with varying requirements). To calculate the time taken I used a built in “time” method that Clojure has that outputs the time taken for a function to execute. Below is the code for each thread execution.

Text

Description automatically generated with medium confidence

Text

Description automatically generatedWith the code ran we can see the displayed times for execution with the desired number of threads. I had a difficult time trying to place the time function as a variable as it seemed to not want to cooperate, so I just placed the 3 times for each thread amount execution in python and calculated the average times and then used matplotlib to create a bar graph of the results.

Graphical user interface, text, application

Description automatically generated

Chart, bar chart

Description automatically generated

The results of the multithreaded executions of mergesort in Clojure indicate that it ran the most optimally with 8 concurrent threads. With a number of threads lower than 8 it did not have enough threads to optimize the performance, with more than 8 threads we seem to have created a bottleneck effect that is overloading a queue that is not allowing the sorting function to run optimally. Overall, I was surprised that the mergesort function with a single thread ran as quick as it did. With a time-complexity of O(nlog(n)) it took roughly 5.3 seconds which seemed relatively quick to me and was a nice surprise. I think that sorting the list with 8 threads in roughly 3.3 seconds is a good enough improvement to call this project a success.