Concurrent Singly-Linked Lists

Amgad Rady

DisCoVeri Group
Department of Electrical Engineering and Computer Science
York University

December 3, 2015

Outline

1 Experimental Setup

2 Results

3 Conclusion & Future Work

Experimental Configuration

- Intel®Manycore Testing Lab.
- 40 processors on each node.
- Processor spec: Intel®Xeon®CPU E7-4860 @ 2.27 GHz
- 1GB heap allocated, run in server mode, 64-bit

Experiment Description

- Each thread (Task) is given two parameters: p and *range*
  - p is the probability the the task performs an insert ($1 - p$ is the probability that it performs a delete) on its next operation.
  - The Task inserts or deletes a random number modulo *range*.
- The main thread starts a collection of Tasks and performs a garbage collection. It then waits for the tasks to finish and estimates the runtime by computing a minimum start and maximum end time over all the threads.

Experiment Description

- Each thread (Task) is given two parameters: p and *range*
  - p is the probability the the task performs an insert ($1 - p$ is the probability that it performs a delete) on its next operation.
  - The Task inserts or deletes a random number modulo *range*.
- The main thread starts a collection of Tasks and performs a garbage collection. It then waits for the tasks to finish and estimates the runtime by computing a minimum start and maximum end time over all the threads.

Experiment Description

- Each thread (Task) is given two parameters: p and *range*
  - p is the probability the the task performs an insert ($1 - p$ is the probability that it performs a delete) on its next operation.
  - The Task inserts or deletes a random number modulo *range*.
- The main thread starts a collection of Tasks and performs a garbage collection. It then waits for the tasks to finish and estimates the runtime by computing a minimum start and maximum end time over all the threads.

Experiment Description

- Each thread (Task) is given two parameters: p and *range*
  - p is the probability the the task performs an insert ($1 - p$ is the probability that it performs a delete) on its next operation.
  - The Task inserts or deletes a random number modulo *range*.
- The main thread starts a collection of Tasks and performs a garbage collection. It then waits for the tasks to finish and estimates the runtime by computing a minimum start and maximum end time over all the threads.

Experiment Description, cont.

- The main thread performs 13 trials and ignores the first 3.
- It then computes the throughput by aggregating the number of operations (delete and insert) in each trial and returning the mean and standard deviation over 10 trials.

Experiment Description, cont.

- The main thread performs 13 trials and ignores the first 3.
- It then computes the throughput by aggregating the number of operations (delete and insert) in each trial and returning the mean and standard deviation over 10 trials.

Contention

- The *range* parameter is used as a proxy for contention.
- Intuitively, if several processes are operating using a restricted range of numbers (10) the probability of interference is higher than if the numbers were drawn from a much broader range (2^{32}).

Contention

- The *range* parameter is used as a proxy for contention.
- Intuitively, if several processes are operating using a restricted range of numbers (10) the probability of interference is higher than if the numbers were drawn from a much broader range (2^{32}).

Parameter p

- Under low contention, p has very little effect on the algorithm's throughput
  - This is due to a 'symmetry' in Harris's algorithm. Namely, both INSERT and DELETE call the SEARCH procedure.
  - If there is no contention, then the algorithm performs a constant number of operations for INSERT and DELETE
- Under high contention, this is no longer the case. In some sense, DELETE is selfish and INSERT is altruistic.

Parameter p

- Under low contention, p has very little effect on the algorithm's throughput
  - This is due to a 'symmetry' in Harris's algorithm. Namely, both INSERT and DELETE call the SEARCH procedure.
  - If there is no contention, then the algorithm performs a constant number of operations for INSERT and DELETE
- Under high contention, this is no longer the case. In some sense, DELETE is selfish and INSERT is altruistic.

Parameter p

- Under low contention, p has very little effect on the algorithm's throughput
  - This is due to a 'symmetry' in Harris's algorithm. Namely, both INSERT and DELETE call the SEARCH procedure.
  - If there is no contention, then the algorithm performs a constant number of operations for INSERT and DELETE
- Under high contention, this is no longer the case. In some sense, DELETE is selfish and INSERT is altruistic.

Parameter p

- Under low contention, p has very little effect on the algorithm's throughput
  - This is due to a 'symmetry' in Harris's algorithm. Namely, both INSERT and DELETE call the SEARCH procedure.
  - If there is no contention, then the algorithm performs a constant number of operations for INSERT and DELETE
- Under high contention, this is no longer the case. In some sense, DELETE is selfish and INSERT is altruistic.

Low contention, $p = 0.5$

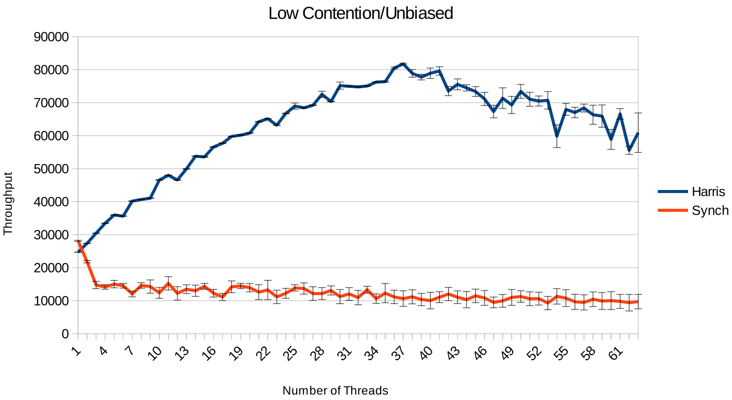
CSE6490A Presentation

Amgad Rady

Experimental Setup

Results

Conclusion & Future Work

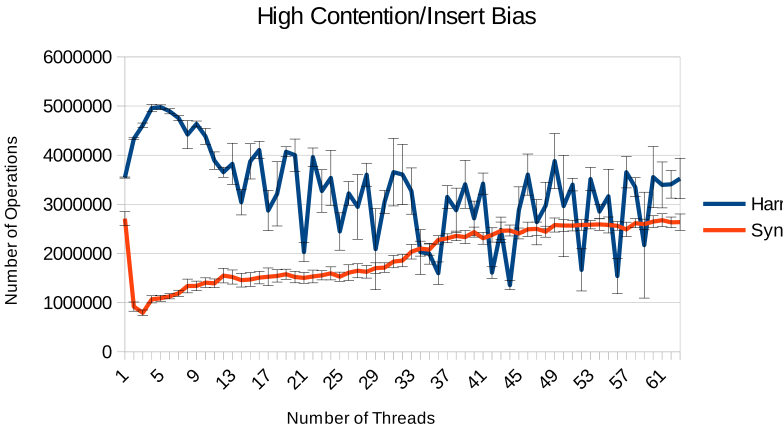


High contention, $p = 0.75$

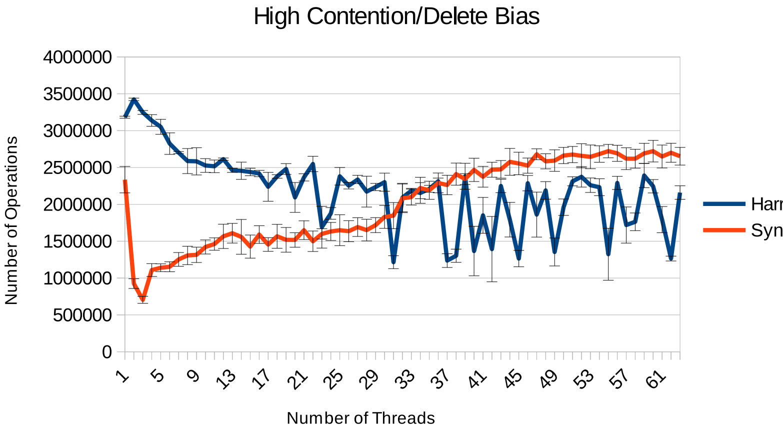
CSE6490A Presentation

Amgad Rady

- Experimental Setup
- Results
- Conclusion & Future Work



High contention, $p = 0.25$



Conclusion

- Performance of the synchronized algorithm is superior to the Harris algorithm under very high contention
  - The synchronized operations do not interfere with each other, so there is no possibility of failure and backtracking.
  - In this case, the data structure is very small, so locking it in its entirety is not very expensive
- Under low contention, Harris's algorithm has eight times the throughput of the most naïve locking algorithm.
- It scales well with physical resources.

Conclusion

- Performance of the synchronized algorithm is superior to the Harris algorithm under very high contention
  - The synchronized operations do not interfere with each other, so there is no possibility of failure and backtracking.
  - In this case, the data structure is very small, so locking it in its entirety is not very expensive
- Under low contention, Harris's algorithm has eight times the throughput of the most naïve locking algorithm.
- It scales well with physical resources.

Conclusion

- Performance of the synchronized algorithm is superior to the Harris algorithm under very high contention
  - The synchronized operations do not interfere with each other, so there is no possibility of failure and backtracking.
  - In this case, the data structure is very small, so locking it in its entirety is not very expensive
- Under low contention, Harris's algorithm has eight times the throughput of the most naïve locking algorithm.
- It scales well with physical resources.

Conclusion

- Performance of the synchronized algorithm is superior to the Harris algorithm under very high contention
  - The synchronized operations do not interfere with each other, so there is no possibility of failure and backtracking.
  - In this case, the data structure is very small, so locking it in its entirety is not very expensive
- Under low contention, Harris's algorithm has eight times the throughput of the most naïve locking algorithm.
- It scales well with physical resources.

Conclusion

- Performance of the synchronized algorithm is superior to the Harris algorithm under very high contention
  - The synchronized operations do not interfere with each other, so there is no possibility of failure and backtracking.
  - In this case, the data structure is very small, so locking it in its entirety is not very expensive
- Under low contention, Harris's algorithm has eight times the throughput of the most naïve locking algorithm.
- It scales well with physical resources.

Future Work

- Although Harris's algorithm's relative performance is good, baseline testing suggests that order(s) of magnitude improvement in absolute performance is possible with a lighter C&S primitive.
- Investigate the behaviour of the implementations further under higher contention and greater bias.
- Try to reduce variance of the results.

Future Work

- Although Harris's algorithm's relative performance is good, baseline testing suggests that order(s) of magnitude improvement in absolute performance is possible with a lighter C&S primitive.
- Investigate the behaviour of the implementations further under higher contention and greater bias.
- Try to reduce variance of the results.

Future Work

- Although Harris's algorithm's relative performance is good, baseline testing suggests that order(s) of magnitude improvement in absolute performance is possible with a lighter C&S primitive.
- Investigate the behaviour of the implementations further under higher contention and greater bias.
- Try to reduce variance of the results.

The End

Questions?