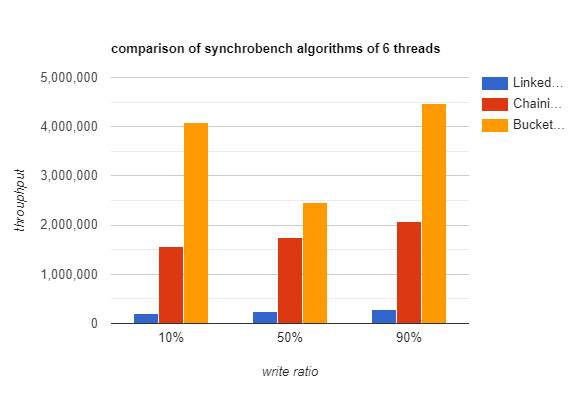
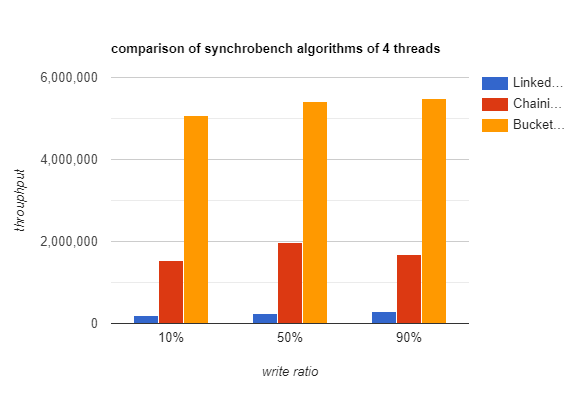
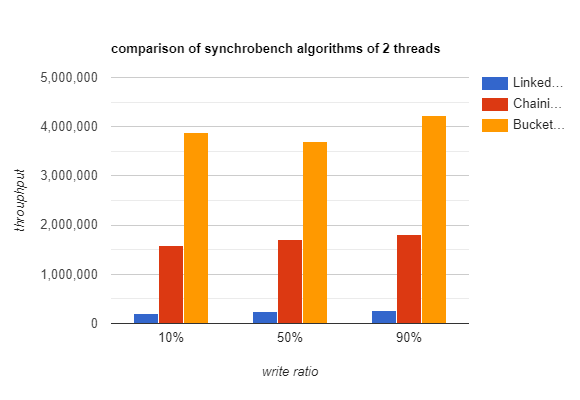
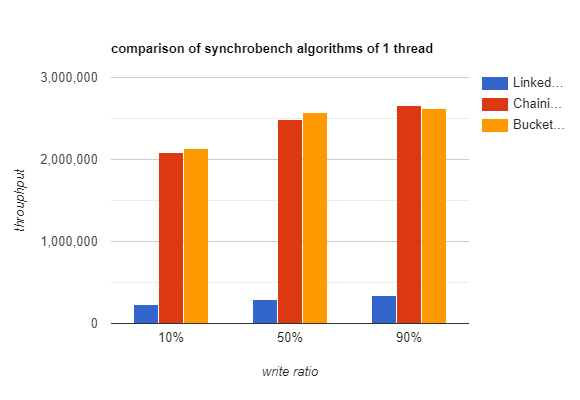
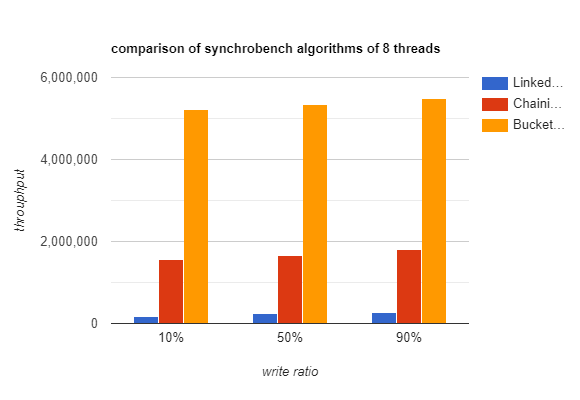
Soft3410 assignment 1 report

Shuwei Zhang

Observation on 3 different data structures:



Note: both hashset IntSets have 10 buckets hardcoded.

From the observation, we can summarize:

1. As the write ratio increases, the throughput (operations per second, performance) of all 3 data structures increases.
2. The performance of *CoarseGrainedLinkedListIntSet* is at about 200,000~300,000 operations per second. The performance of *CoarseGrainedChainingHashTableIntSet* is at about 2,000,000 to 2,400,000 when there is 1 thread, and about 1,600,000 to 2,000,000 when there are multiple threads(roughly 8~9 times of the linked list). Both data structures don’t scale well when there are multiple threads, thus having low scalability.
3. The performance of *BucketLockingChainingHashTableIntSet* is scaling very well with multiple threads. At 1 thread, it has roughly same performance with the *CoarseGrainedChainingHashTableIntSet*. But when there’re multiple threads, the performance increases drastically. It has about double performance at 2 threads executing concurrently than 1 thread, and keep increasing as the thread number goes up, but not at linear level.

Explanation of the observation:

1. The write ratio is the ratio of modifying operation(add / remove) compared to all operations. As modifying operations generally take less time to execute( it has lower time complexity in the context of linked list / hash table, compared to composite operations). Therefore, more operations can take place if there are more writing operations that takes shorter time.
2. When commencing write operation, the CoarseGrainedLinkedListIntSet class needs to traverse the whole list to find the right spot to insert then element. Therefore, the time complexity is O(n). The *CoarseGrainedChainingHashTableIntSet* has 10 buckets composed of *LinkedListIntSet* class. Therefore, when trying commence a write operation, it first determine which of 10 buckets the element should be at, then traverse that bucket. Therefore, the time complexity is O(n/10). So the latter one has roughly 8~9 times of performance to the first one since not all operations are optimistic or write operations. However, neither of them has a good scalability, since they all have a coarse grained lock. A coarse grained lock makes sure that no matter how many threads there are, only one thread can access the data structure at a time, to ensure the data safety. However, this also means that the rest of threads can do nothing but waiting when there is already a thread accessing the data structure. Therefore, they are unable to increase the performance by concurrency when there are multiple threads.
3. The *BucketLockingChainingHashTableIntSet* has a coarse grained lock on each bucket. This ensures a high scalability because when a thread gained a lock to one bucket, another thread is free to access another bucket. Therefore, there can be multiple threads accessing the data structure at the same time, resulting in a good concurrency operation. At one thread, since there is no concurrency, it has nearly the same performance as the *CoarseGrainedChainingHashTableIntSet*. But when there are multiple threads, it can utilize the benefits of concurrency. The performance is not increasing linearly as thread number goes up is also because not all operations are optimistic or write operations, there will always be threads waiting for another thread.