HOMEWORK 3

CIS4930/COP5618 Concurrent Programming

Anindya Nandy

80999501

Comparison of the semantics of CopyOnWriteHashMap with java.util.ConcurrentHashMap

CopyOnWriteHashMap: Copy On Write is an optimization technique for maintaining a copy of a collection of data. Copy On Write achieves this by copying only data that is modified immediately after the start of the replication process.

When a thread is created, the data is not copied, instead it is marked as read only. If either of the threads attempts to modify the data, a copy is made for that process. In order to allow each thread to have its own local values, all operations that change the Map are implemented by making a new copy of the underlying map. Therefore the operations that do not cause a change to this class occur quickly and concurrently.

There may be multiple threads that simultaneously try to modify the same data. In CopyOnWrite we let the new data override the old data. This is done by allowing read and write operations on different Maps separately .

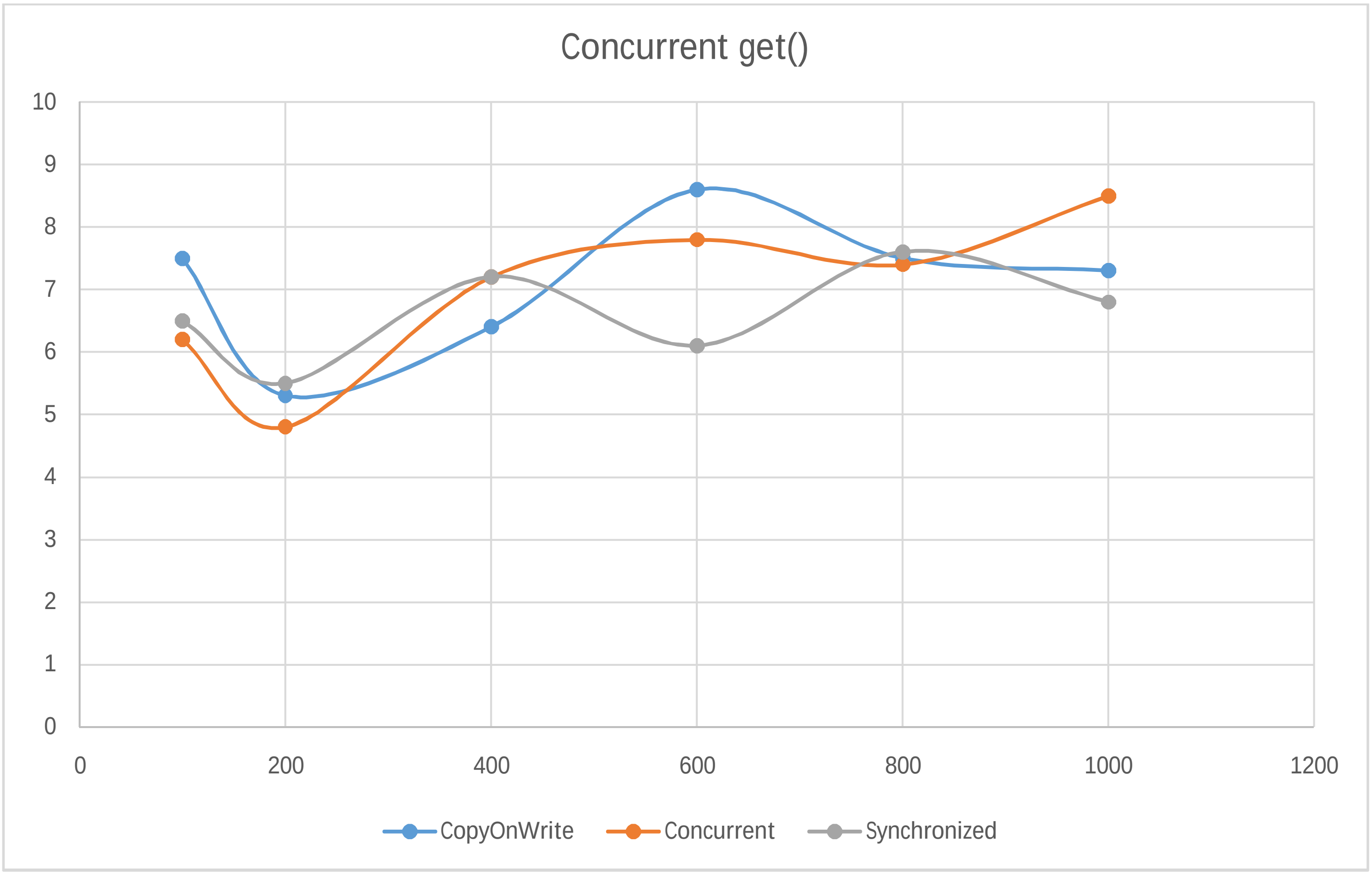
Java.util.ConcurrentHashMap: ConcurrentHashMap allows concurrent modification of the Map from several threads without the need to block them. It realises synchronization and allows parallel read access by multiple threads without synchronization and, more importantly, provides an Iterator that requires no synchronization and even allows the Map to be modified during the iteration. So, even though all operations are thread-safe, retrieval operations do not include locking, and there is no support for locking the entire hashtable in a way that prevents all access.

Retrieval operations do not block, so they can safely overlap with ‘put’ and ‘remove’ operations. The ‘get’ operations reflect the values after the most recently completed updated operations. The hashtable is internally partitioned to try to allow a specified number of concurrent update operations without conflict and modification loss. Multiple threads can read from and write to it without the chance of receiving out-of-date or corrupted data.

An empirical study comparing the performance of different types of concurrent implementations.

‘Concurrent Get’ operation:

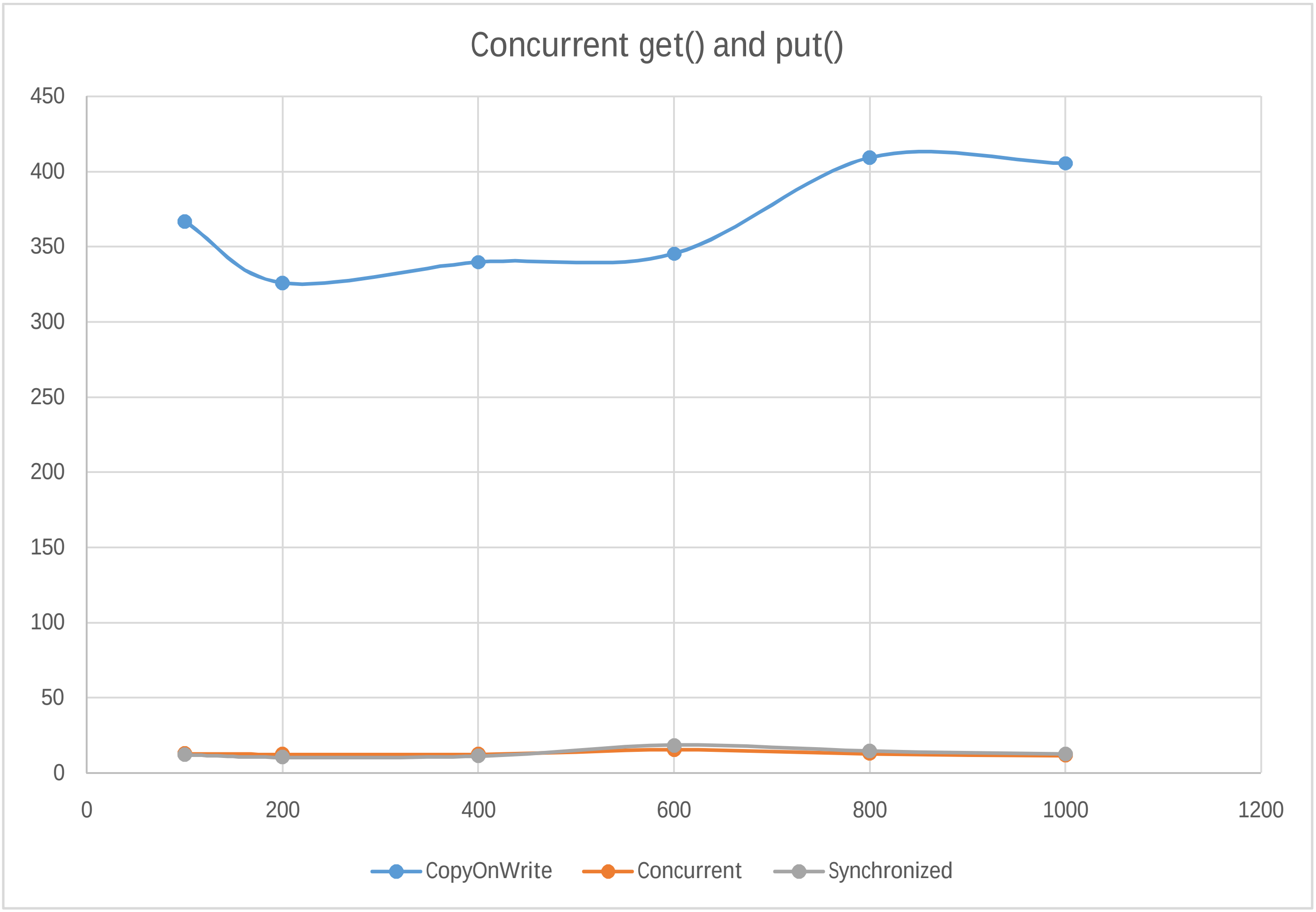
Size	CopyOnWriteHashMap	ConcurrentHashMap	SynchronizedMap
100	7.5	6.2	6.5
200	5.3	4.8	5.5
400	6.4	7.2	7.2
600	8.6	7.8	6.1
800	7.5	7.4	7.6
1000	7.3	8.5	6.8



From above, a ‘get’ operation leads to very similar results. So, this is evidence that simple ‘get’ operations even in the ‘CopyOnWrite’ implementation is as fast as the inbuilt concurrent implementations of the HashMap.

‘Concurrent Get and Put’ operation:

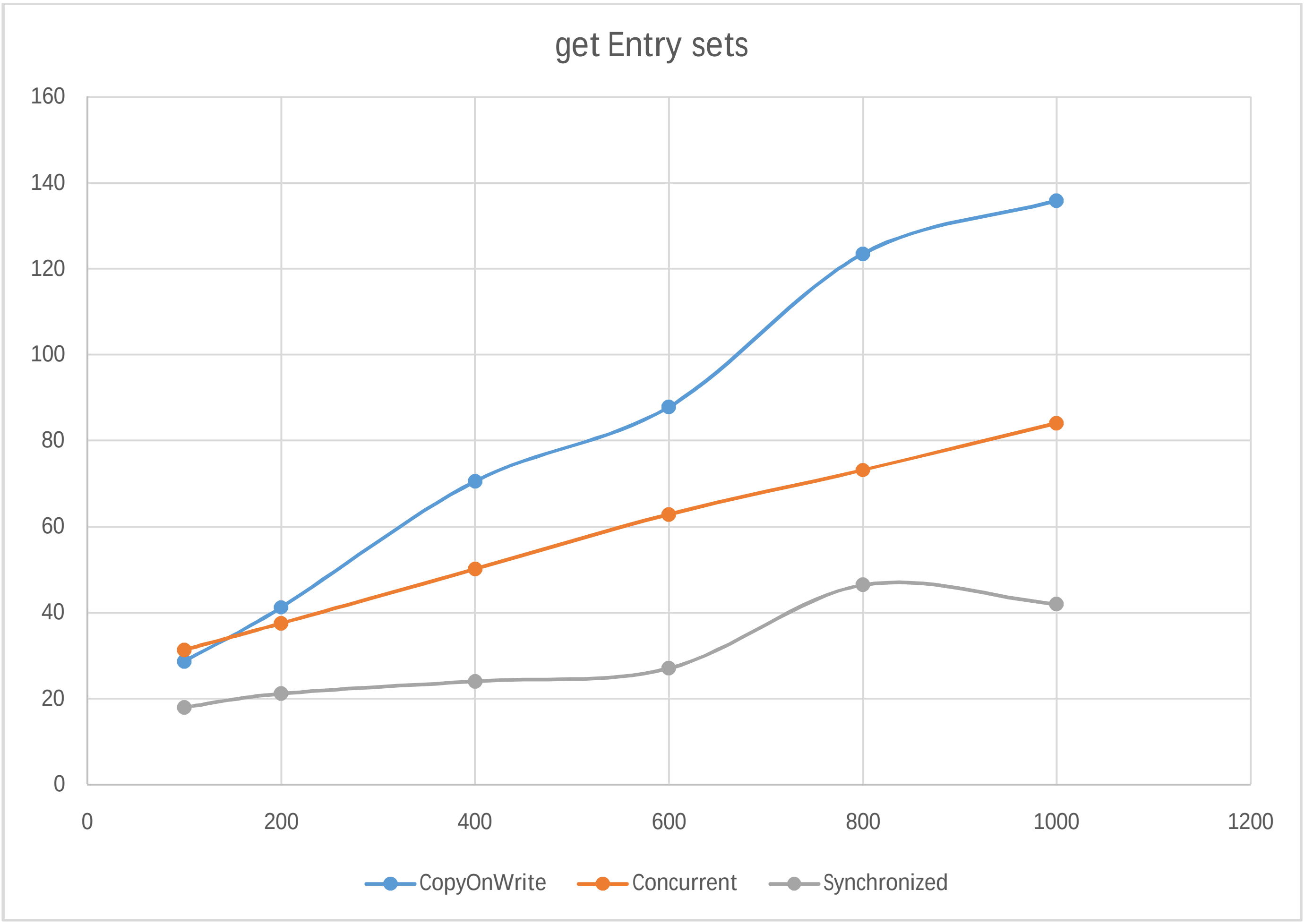
Size	CopyOnWriteHashMap	ConcurrentHashMap	SynchronizedMap
100	366.8	12.7	12.1
200	326	12.4	10.4
400	339.8	12.3	11.2
600	345.4	15.3	18.2
800	409.2	12.6	14.4
1000	405.4	11.5	12.4



From above ‘put’ operations in the CopyOnWrite implementation takes a much longer time than the other implementations. This shows that creation of a new Map for every thread that wants to perform a ‘put’ operation is very expensive which consumes time and system resources.

‘Get EntrySet’ operation:

Size	CopyOnWriteHashMap	ConcurrentHashMap	SynchronizedMap
100	28.7	31.4	18
200	41.3	37.6	21.2
400	70.6	50.2	24.1
600	87.9	62.9	27.1
800	123.5	73.2	46.5
1000	135.9	84.1	42



As seen here, iterating by using the EntrySet leads to the maximum time consumption in the CopyOnWrite implementation. EntrySet is a collection of all Map Entries and contains both Key and Value pairs.