This is the write-up for assignment # 6 (including extra credit)

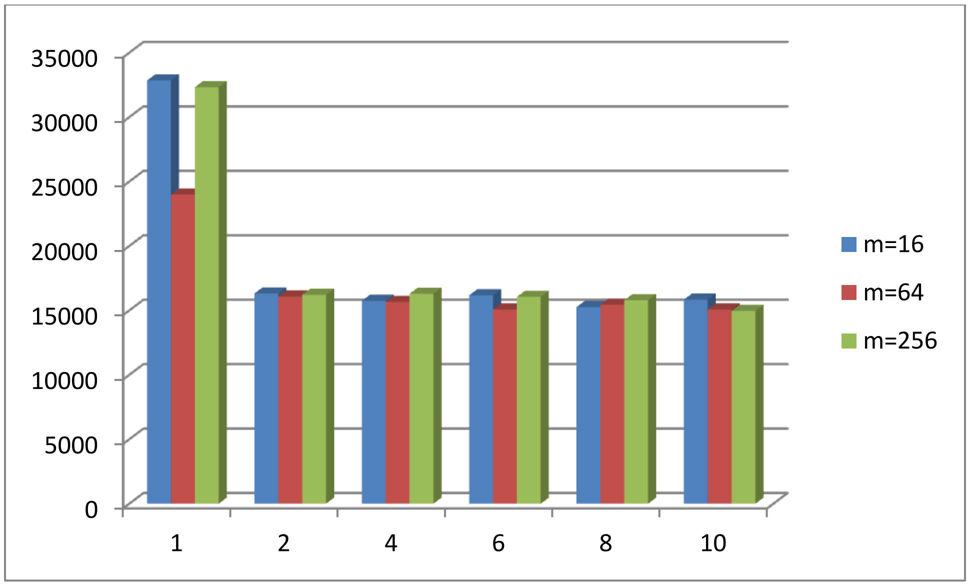
we designed the workload to contrast the performance of relaxed-transaction key/value store with our atomic key/value store. The workload is in file HashBench.cpp in both task_1 and task_3 folders. Briefly speaking, we select 5 random characters from printable ASCII character set to construct random strings. and select random operations to perform according to the parameter R (i.e,, the read/write ratio).

These strings are used as keys to be looked up in the hash table or inserted into or removed from the hash table.

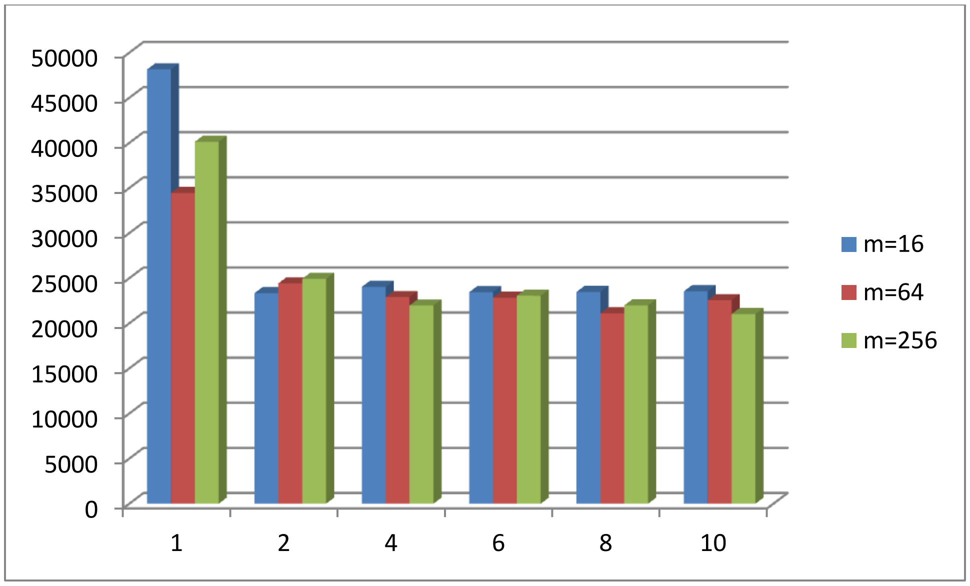
The results are shown in the following figures, notice that serie1 represents m=16, serie2 represents m=64, serie3 represents m=256. The y-axis represents throughput. The latency and throughput details are shown in write-up.txt

On sunlab machine:

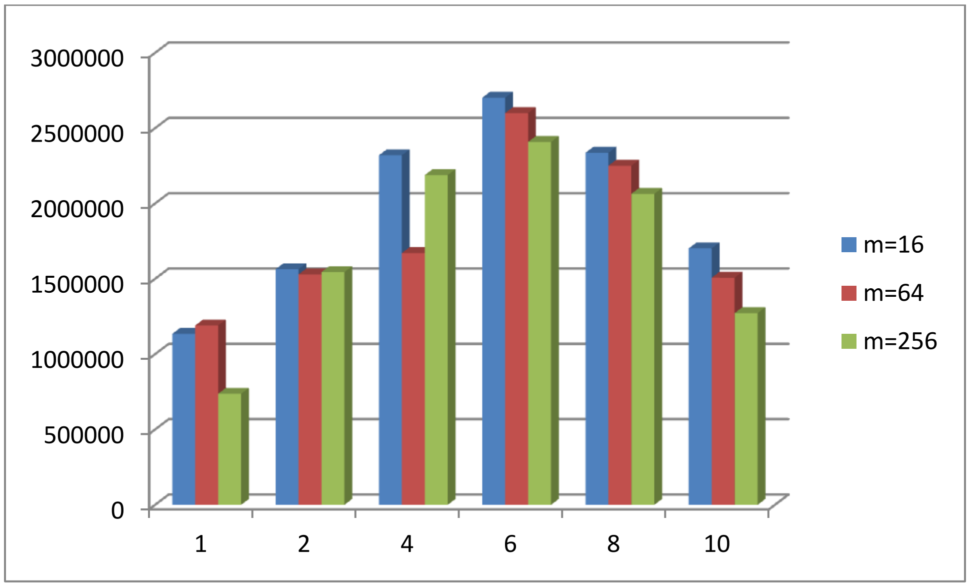
relaxed-transaction key/value store:



throughput when R=0

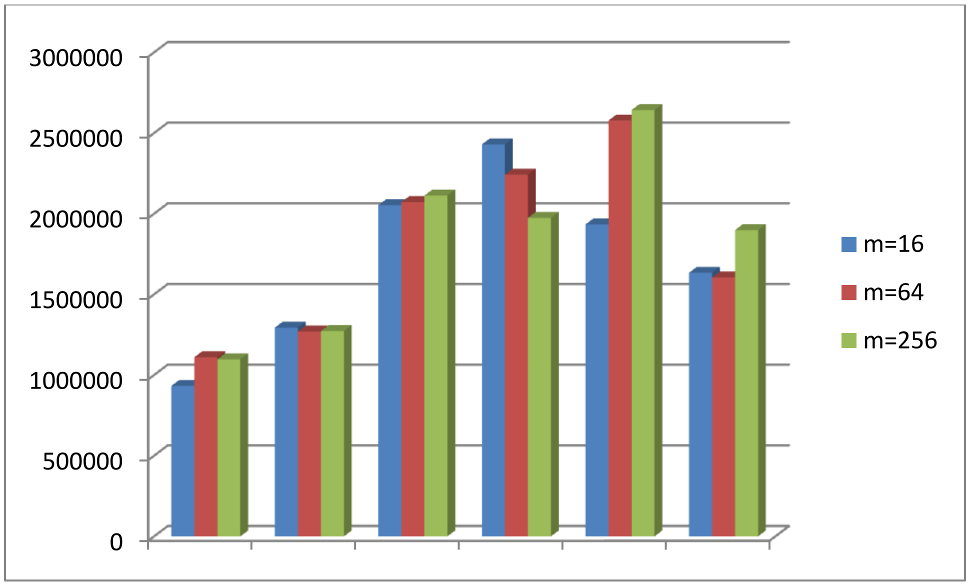


throughput when R=33

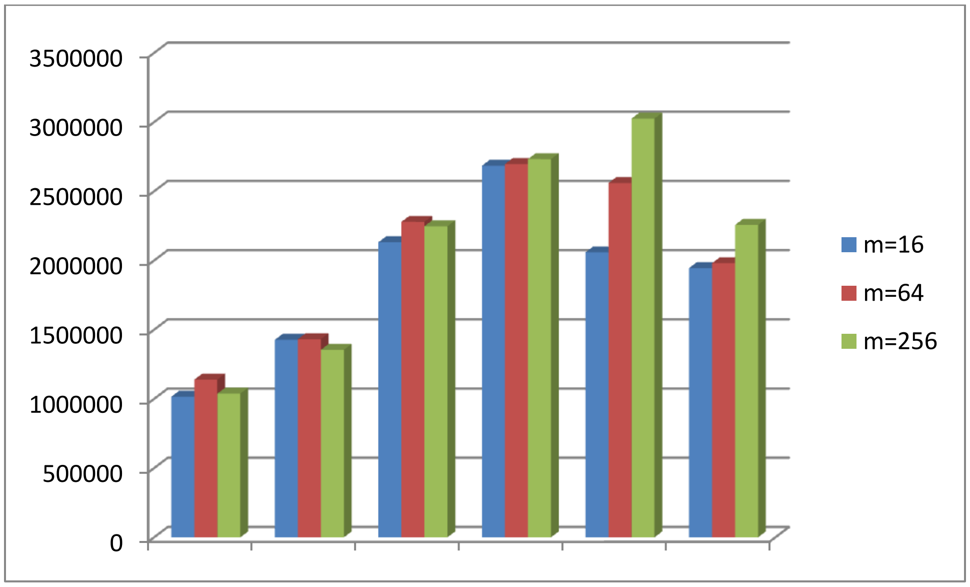


throughput when R=100

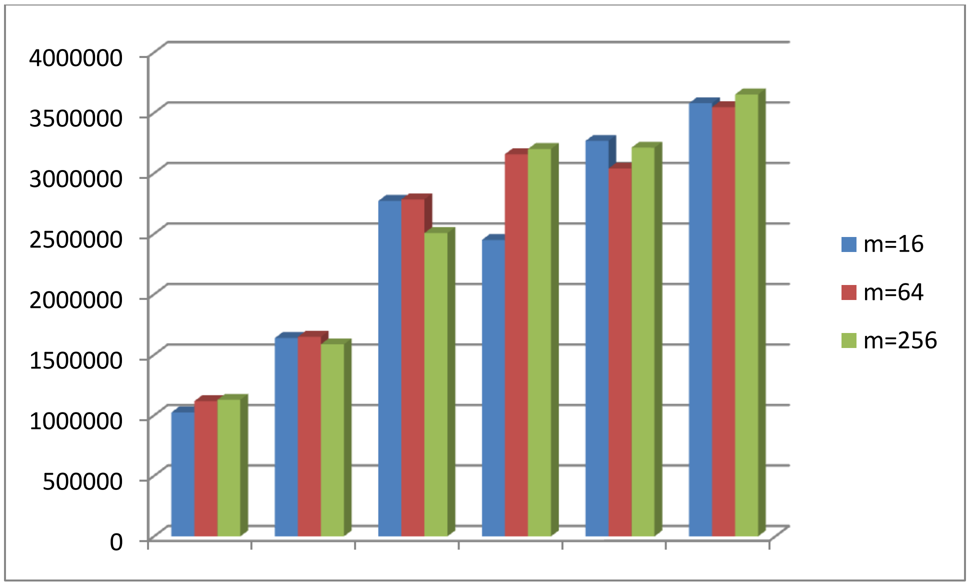
atomic-transaction key/value store:



throughput when R=0

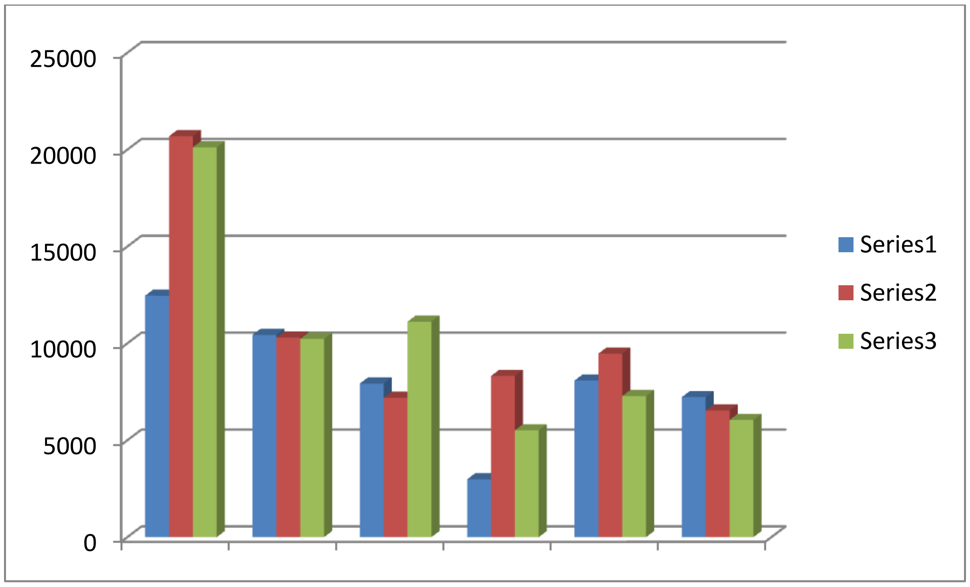


throughput when R=33

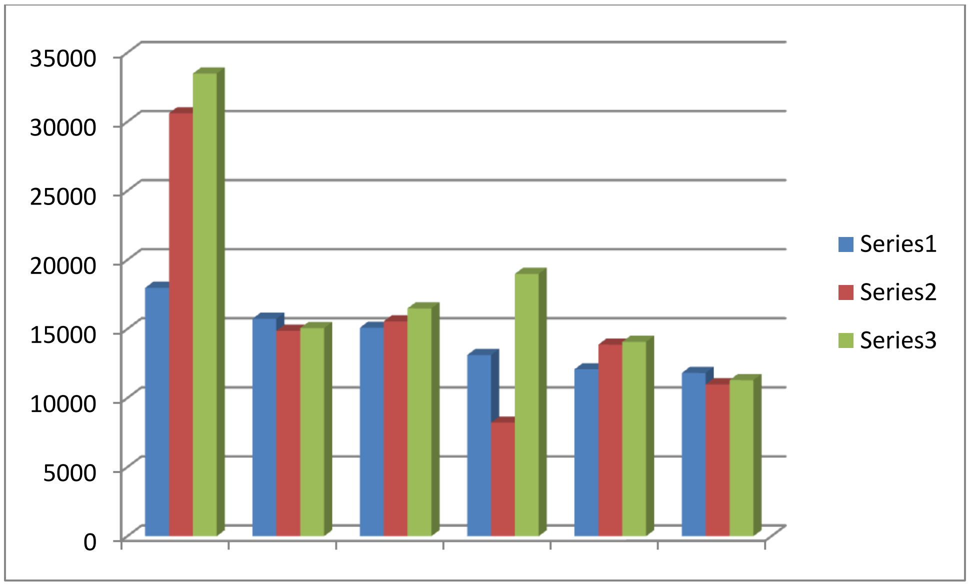


throughput when R=100

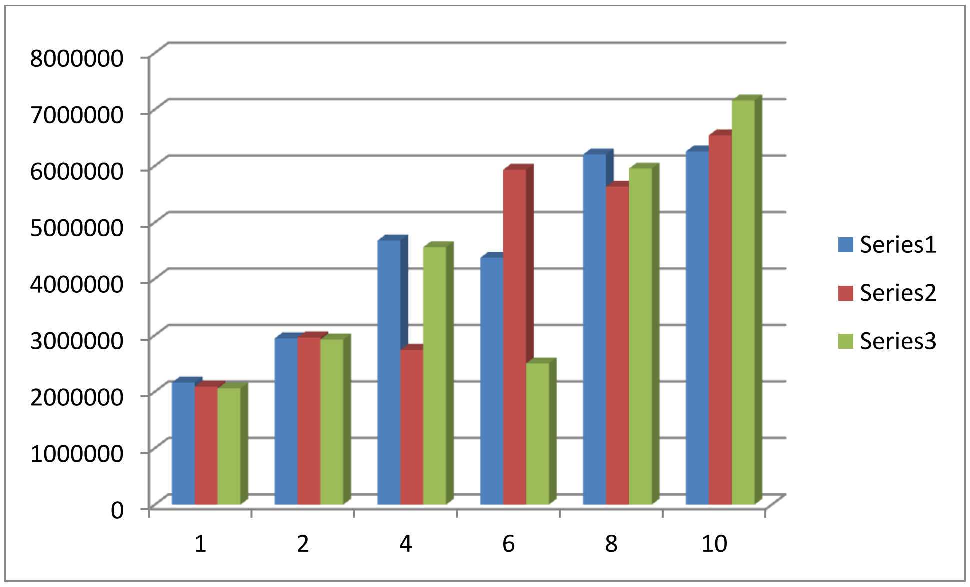
Transaction-safe strings:



throughput when R=0



throughput when R=33

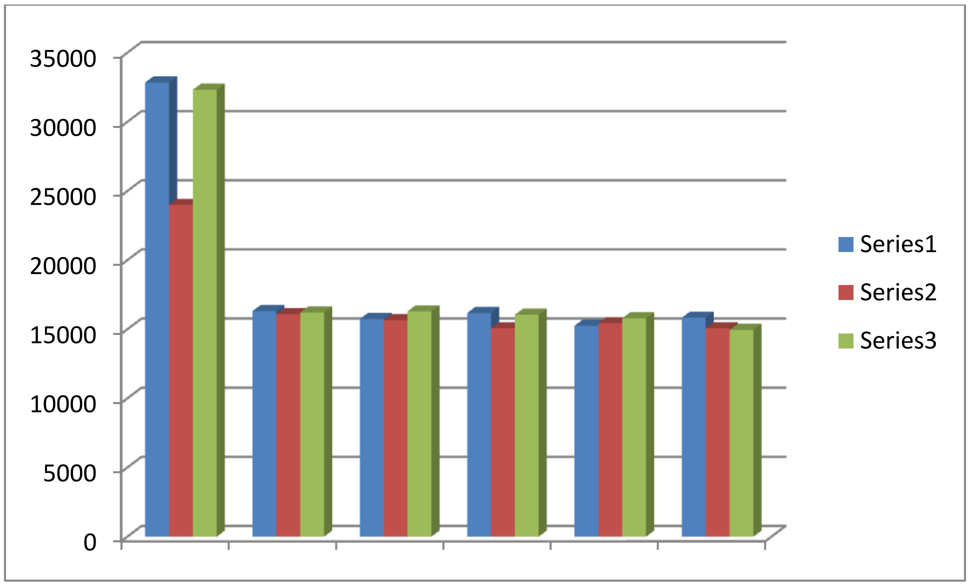


throughput when R=100

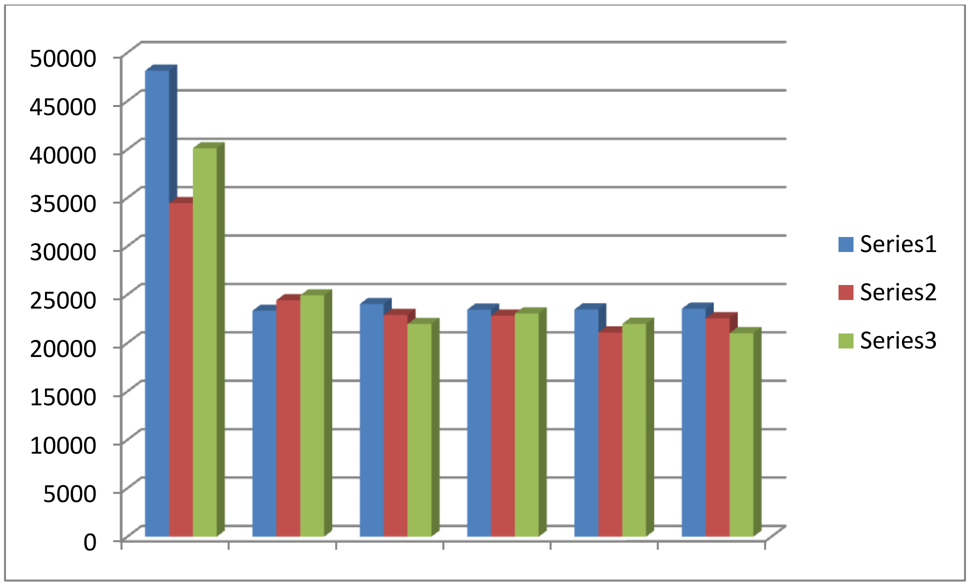
Our observation on sunlab is that when R is small, key/value store with transaction safe strings have the worst performance, especially according to throughput. atomic key/value store has the best performance. When R increases, the performance of relaxed key/value store improves significantly while the atomic one does not differ much. It is obvious that when number of threads increases, the performance is better, especially when R is large. When R equals to 100, it is even hard to notice the sudden drop from one thread to two threads. When R=0, performance of transaction-safe strings is the worst, however, when R=0, that of transaction-safe strings is the best. Atomic transaction key/value store performs in the opposite way. When R=0, it has the best performance, yet the improvement of performance with the increase of R is the lowest.

On 4pac machine:

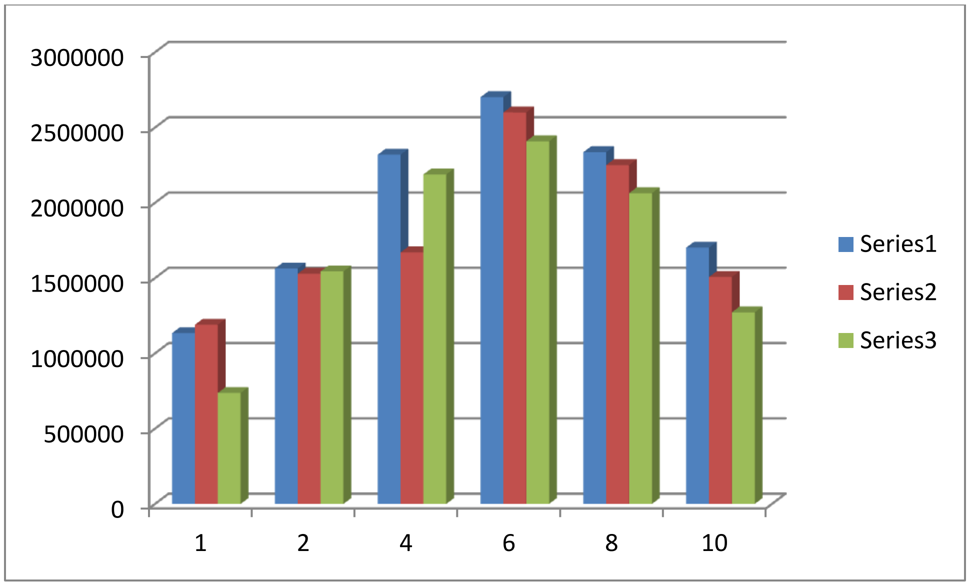
relaxed-transaction key/value store:



throughput when R=0

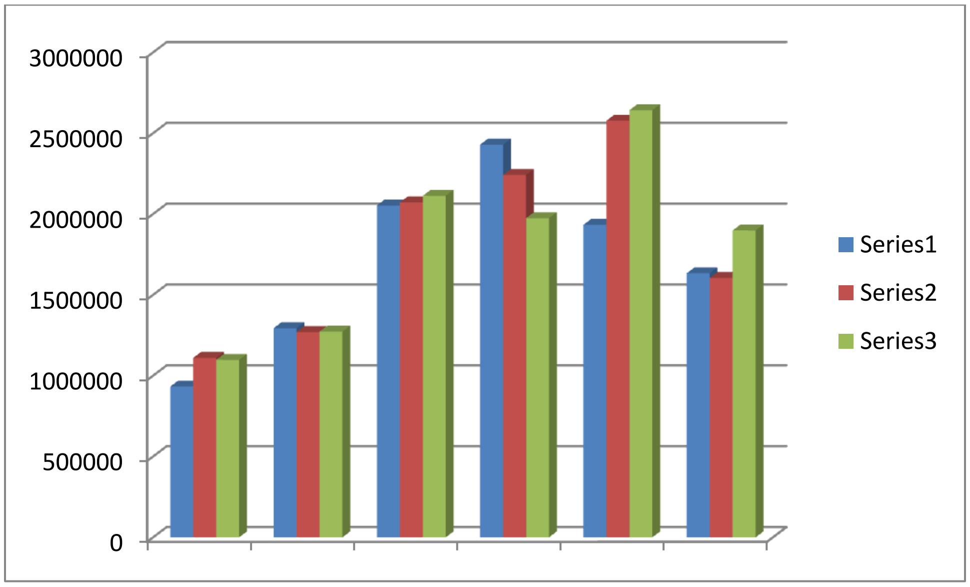


throughput when R=33

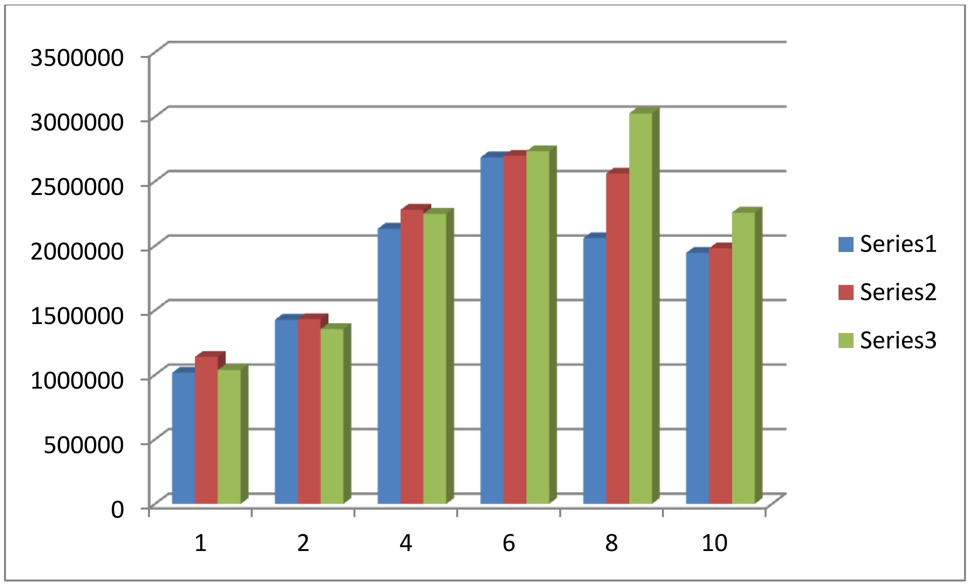


throughput when R=100

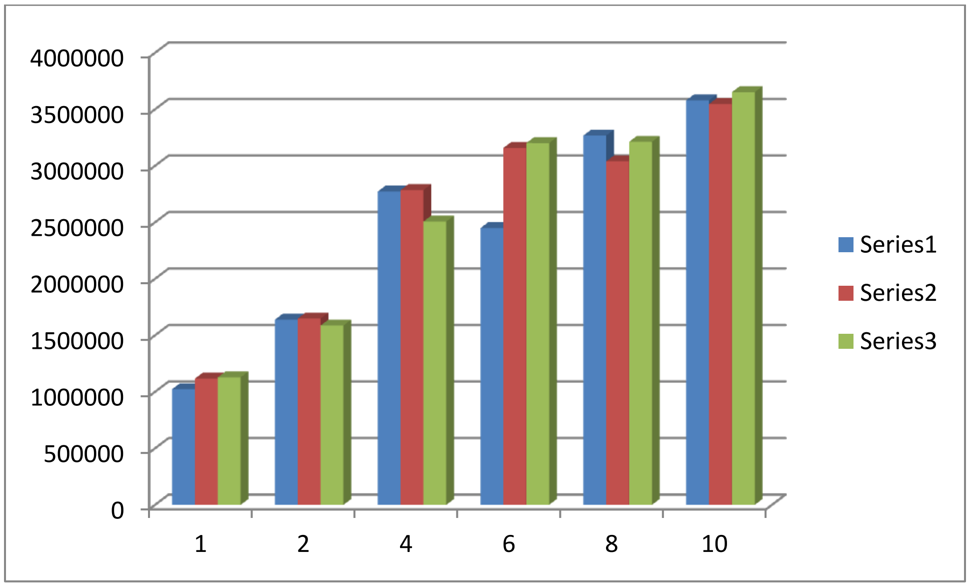
atomic-transaction key/value store:



throughput when R=0

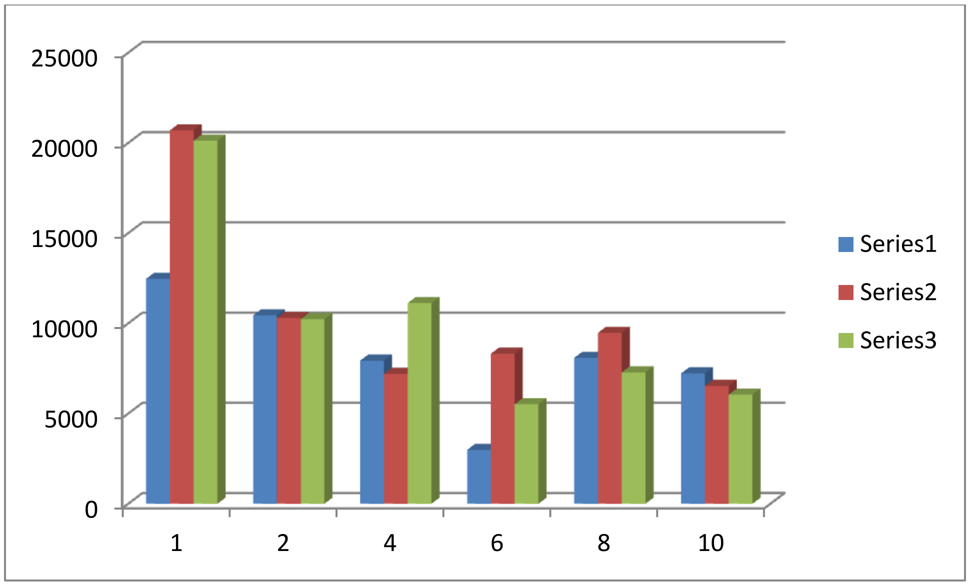


throughput when R=33

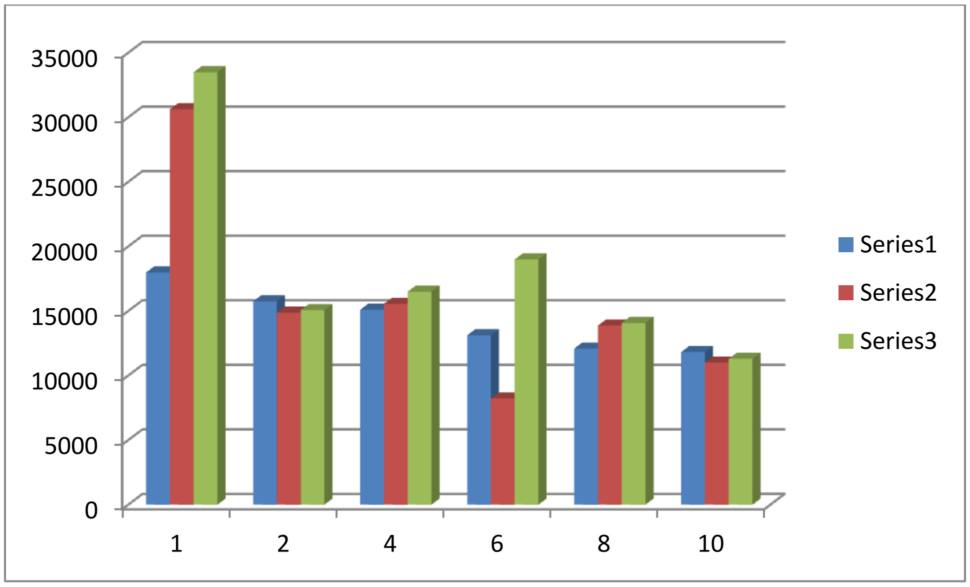


throughput when R=100

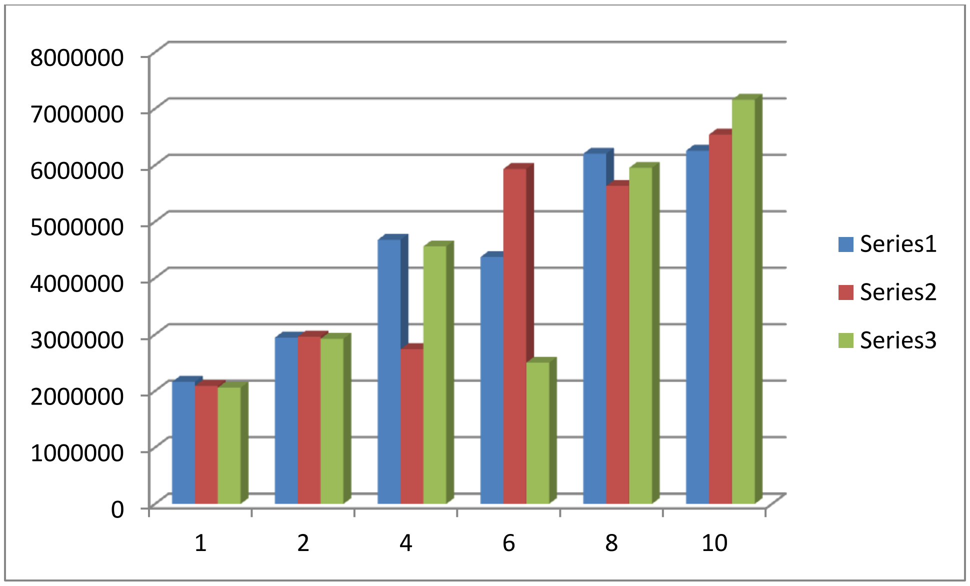
Transaction-safe strings:



throughput when R=0



throughput when R=33



throughput when R=100

Our observation is that the performance trend on haswell is not much different from the performance we have on sunlab. Our observation seems to prove the assumption that relaxed transactions are likely to serialize while atomic transactions should be able to scale unless there are conflicts.

From these tests, we can see that on sunlab machines, where software transaction algorithms are used in GCC TM, relaxed-transaction key/value store are consistently better than our atomic transaction key/value store in terms of both latency and throughput. So in terms of the use of strings inside of transactions, its better to use relaxed_transactions instead of using atomic ones.

Extra credit:

From previous tables, we can also find the contrast between the use of software transactions and the use of hardware transactions. When we use hardware transactions in HTM-support 4pac machines, and when workload is not long (i.e, the number of its operations are fewer than 10000), atomic-transaction key/value store plays better in terms of latency and performance only except the case where 8 threads are used. This indicates that HTM can substantially increase the performance of atomic key/value store.