Conclusion

KA Owned by Kashan Asim •••

Last updated: just a moment ago • 3 min read • See how many people viewed this page

Analysis of the Benchmark Results

Your benchmark results provide insights into how `Caffeine Cache` and `ConcurrentHashMap` perform for different operations (`Get`, `Put`, `Invalidate`, and `Remove`) and across varying data sizes (`100`, `1000`, `10000`). Below is a breakdown:

Benchmark	Data Size	Mode	Count	Score (ops/ms)	Error (ops/ms)	Units
`testCaffeineCacheGet`	100	Throughput	10	386,845.559	82,986.446	ops/ms
`testCaffeineCacheGet`	1000	Throughput	10	356,708.371	43,689.003	ops/ms
`testCaffeineCacheGet`	10000	Throughput	10	64,231.230	10,272.210	ops/ms
`testCaffeineCacheInvalidate`	100	Throughput	10	100,840.909	11,645.955	ops/ms
`testCaffeineCacheInvalidate`	1000	Throughput	10	101,813.685	3,833.503	ops/ms
`testCaffeineCacheInvalidate`	10000	Throughput	10	102,337.562	1,530.149	ops/ms
`testCaffeineCachePut`	100	Throughput	10	191,612.148	16,038.548	ops/ms
`testCaffeineCachePut`	1000	Throughput	10	129,892.854	71,495.725	ops/ms
`testCaffeineCachePut`	10000	Throughput	10	44,774.758	11,334.140	ops/ms

`testConcurrentHashMapGet`	1000	Throughput	10	634,243.141	31,602.562	ops/ms
`testConcurrentHashMapGet`	10000	Throughput	10	667,366.628	245,456.100	ops/ms
`testConcurrentHashMapPut`	100	Throughput	10	94,482.868	23,026.250	ops/ms
`testConcurrentHashMapPut`	1000	Throughput	10	187,624.238	3,569.353	ops/ms
`testConcurrentHashMapPut`	10000	Throughput	10	152,737.864	29,859.256	ops/ms
`testConcurrentHashMapRemove`	100	Throughput	10	965,401.198	254,343.061	ops/ms
`testConcurrentHashMapRemove`	1000	Throughput	10	166,302.550	6,028.536	ops/ms
`testConcurrentHashMapRemove`	10000	Throughput	10	172,465.163	22,285.585	ops/ms

1. Observations by Operation

Get Operations

- **ConcurrentHashMap** consistently outperforms **Caffeine Cache** in `Get` operations:
  - At smaller data sizes (100, 1000), the throughput for `ConcurrentHashMap` is nearly double that of `Caffeine Cache`.
  - Even at 10,000 entries, `ConcurrentHashMap` maintains higher throughput.

  Reason:
  - `ConcurrentHashMap` is designed for low-latency lookups and uses a simple data structure optimized for thread-safe reads.
  - Caffeine introduces additional complexity (e.g., eviction policies), which can add overhead.

Put Operations

- **Caffeine Cache** shows declining throughput as the data size increases, from `191,612 ops/ms` (100 entries) to `44,774 ops/ms` (10,000 entries).
- **ConcurrentHashMap**, on the other hand, performs more steadily and even surpasses `Caffeine Cache` at higher data sizes (e.g., at `10000` entries, `152,737 ops/ms` vs. `44,774 ops/ms`).

- `ConcurrentHashMap` lacks eviction mechanisms, making `Put` operations faster in larger maps.

Invalidate Operations (Caffeine Only)

- The performance of `Invalidate` in `Caffeine Cache` is consistent across all data sizes (`~100,000 ops/ms`), indicating that the cache invalidation mechanism is relatively independent of size.

Remove Operations (ConcurrentHashMap Only)

- Remove operations for `ConcurrentHashMap` show high throughput at small sizes (`965,401 ops/ms` for 100 entries) but drop significantly at 1,000 and 10,000 entries.

---

2. Observations by Data Size

Small Data Sizes (100 Entries)

- `ConcurrentHashMap` dominates in both `Get` and `Remove` operations.
- `Caffeine` performs well for `Put` operations at this size, but `ConcurrentHashMap` is still competitive.

Medium Data Sizes (1,000 Entries)

- The performance gap between `Caffeine` and `ConcurrentHashMap` widens for `Get` operations, with `ConcurrentHashMap` maintaining significantly higher throughput.
- `ConcurrentHashMap` shows improved `Put` performance compared to smaller sizes, likely due to JVM optimizations kicking in.

Large Data Sizes (10,000 Entries)

- `ConcurrentHashMap` maintains consistent performance for `Get` and `Put` operations, whereas `Caffeine` shows a significant drop in throughput, particularly for `Put` operations.
- Caffeine's overhead for managing eviction policies becomes more apparent at this size.

---

3. Error Margins

- The error margins (`±` values) are relatively high in some cases, particularly for `ConcurrentHashMap` operations with larger data sizes (e.g., `Get` at `10,000` entries has a margin of `± 245,456 ops/ms`).
  - This variability suggests contention or external factors (e.g., CPU scheduling, garbage collection).
  - Consider increasing `Forks` or `Measurement Iterations` for more stable results.

Key Takeaways

1. **ConcurrentHashMap** is generally faster for `Get` operations across all data sizes due to its simpler structure and optimized thread-safe design.
2. **Caffeine Cache** excels in `Invalidate` operations and small-scale `Put` operations but suffers at larger data sizes due to the overhead of eviction policies.
3. For workloads requiring frequent `Get` and `Put` operations on large datasets, **ConcurrentHashMap** might be a better choice unless cache eviction is critical.
4. If you require advanced cache features like eviction, `Caffeine` is a better option, albeit with some trade-offs in throughput.

- You prioritize throughput and do not need advanced caching features (e.g., eviction).

2. **Use Caffeine Cache** if:
   - Cache eviction or expiration policies are critical to your application.
3. **Optimize Benchmarking for Stability**:
   - Increase `@Fork(2)` and reduce `@Threads(2)` for more stable results if variability is an issue.
4. **Profile with Larger Data Sizes**:
   - Test with even larger datasets (e.g., `100,000` entries) if your use case involves large-scale data.