

Hampr Machine Service – Performance Analytics
Sachi shah
System design hw

This report summarizes the simulated performance of the Hampr Machine Service API under mixed load using the provided test harness.

Key findings:

- The cache substantially reduces database reads as the working set warms.
- Most cost is from DB reads and cache operations; external/internal calls are limited to token validation and start-cycle actions.

Workload (per simulation.test.ts):

- 4 independent runs.
- 10,000 API calls per run across 100 machines and 5 locations.
- Action mix per request: ~40% reserve, ~30% start (if reserved), ~30% get status.
- Hardware fault rate: 5% on start.

Metrics captured:

- Resource units by consumer from ResourceConsumer
- Cache stats: DataCache.cacheHits, DataCache.cacheMisses, hit rate
- Cache-to-DB ratio: cache hits vs MachineStateTable.dbAccesses

API Behavior Implemented

Authentication: Every request validates token; invalid tokens raise:
{ "statusCode": 401, "message": "Invalid token" }

POST /machine/request:

- Reserves first AVAILABLE machine at location, sets status AWAITING_DROPOFF and currentJobId, caches the updated state.
- Returns 200 with machine; 404 if none available.

GET /machine/:machineId:

- Returns from cache if present; else fetches DB, caches, returns 200; 404 if not found.

POST /machine/:machineId/start:

- Requires AWAITING_DROPOFF; starts cycle via SmartMachineClient.
- On success sets RUNNING and returns 200; on hardware exception sets ERROR and returns 420.
- Updated state is cached.

Cache effectiveness:

- Hit rate increases over the run as the working set is reused.
- A small FIFO cache (size 64) is effective when the hot set fits; with larger active sets, eviction reduces hit rate.

Cache vs DB:

Hit/Access ratio (cache hits / DB accesses) trends upward as the cache warms and workloads reuse machine IDs.

```
sachi@Mac cs-253p-hw-hampr-machine-service % npm test
> hampr-base@1.0.0 test
> jest

PASS test/simulation.test.ts
  ● Console

    console.log
      (index) Resource Run 1 Units Run 1 % Run 2 Units Run 2 % Run 3 Units Run 3 % Run 4 Units Run 4 %
      0 'IdentityProviderClient' 3840256 '69.36%' 3840256 '69.08%' 3840256 '69.22%' 3840256 '69.19%'
      1 'SmartMachineClient' 32256 '0.58%' 32256 '0.58%' 32256 '0.58%' 32256 '0.58%'
      2 'MachineStateTable' 1654472 '29.88%' 1654472 '29.76%' 1654472 '29.82%' 1654472 '29.81%'
      3 'DataCache' 23206 '0.42%' 23206 '0.42%' 23206 '0.42%' 23206 '0.42%'

      at Object.<anonymous> (test/simulation.test.ts:160:13)

    console.log
      (index) Run Cache Hits Cache Misses Hit Rate
      0 1 3804 2128 '64.13%'
      1 2 3721 2135 '63.54%'
      2 3 3763 2118 '63.99%'
      3 4 3756 2175 '63.33%'

      at Object.<anonymous> (test/simulation.test.ts:161:13)

    console.log
      (index) Run Cache Hits DB Accesses Hit/Access Ratio
      0 1 3804 6744 '0.5641'
      1 2 3721 6744 '0.5517'
      2 3 3763 6744 '0.5580'
      3 4 3756 6744 '0.5569'

      at Object.<anonymous> (test/simulation.test.ts:162:13)

PASS test/api.test.ts
Test Suites: 2 passed, 2 total
Tests: 12 passed, 12 total
Snapshots: 0 total
Time: 0.486 s, estimated 1 s
Ran all test suites.
sachi@Mac cs-253p-hw-hampr-machine-service %
```

Screenshot has the following tables and output:

“Test Suites: 2 passed, 2 total”

The Resource Usage table

The Cache Stats table

The Hit/DB Ratio table