Improving Performance and Scalability with Oracle Coherence

Aleksandar Seović
aleks@s4hc.com

Performance vs. Scalability

- **Performance**
  The amount of time an operation takes to complete
- **Scalability**
  The level of load an application can sustain before its performance decreases significantly

Performance vs. Scalability

- Architecting for performance can limit scalability
- Architecting for scalability often sacrifices absolute performance
- Need to consider availability and reliability as well

Performance vs. Scalability

- Absolute performance often does not matter
- What matters is that:
  - Performance remains within the defined boundaries as the load increases
  - The cost of supporting additional load is predictable

Coherence can help

- Scale the data tier effectively
- Bring data closer to the application
- Query and aggregate data in parallel
- Process data in parallel
- Implement Event Driven Architecture

But it is not a silver bullet

- You cannot simply plug-in Coherence into existing application and expect it to scale
- You need to architect for it!

Coherence is a distributed system

- The data often need to be serialized and moved across the network
- The laws of physics apply:
  - No matter how fast your network is, there is a limit to how much data you can move across it in a given unit of time

Coherence is a parallel system

- It allows you to query, aggregate and process data in parallel
- But it can be (ab)used sequentially
- Amdahl’s Law puts a limit on maximum performance

If you care about performance and scalability

- Reduce the amount of network calls and traffic as much as possible
- Reduce the amount of sequential processing as much as possible

Rule 1: Use optimal serialization format

- java.io.Serializable is easy to implement
- POF performs better and results in a much smaller serialized form

Rule 2:
Use putAll(), even for single objects

- Map.put() returns the old value
- Batch inserts/updates if possible

Rule 3:
Bring data in-process if possible

- Use Near Cache or CQC to improve read performance
- Use sticky load balancing to improve cache hit ratios for Near Cache

Rule 4:
Query using keySet()/getAll() idiom

- Ensures that Near Cache can satisfy at least some of the results

Rule 5:
Use aggregators

- Aggregations are performed in parallel
- Move the minimum amount of data across the wire

Rule 6:
Use key association

- Limits the scope for queries
- Can significantly improve query performance

Rule 7: Move processing where the data is

- Avoids data movement
- Allows processing to be performed in parallel

Q & A

Thank You!