Elastic Data

Schedule:	Timing	Topic
	20 minutes	Lecture
	00 minutes	Practice
	20 minutes	Total

Agenda

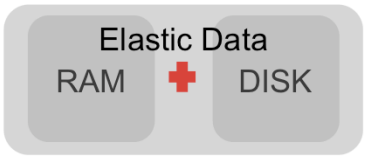
Elastic Data

What Is Elastic Data?

Elastic Data:

- Is a seamless, efficient, and transparent bridge between memory and disk
- Is tuned for Solid State Disks
- Supports massive storage capacity
- Shows virtually no performance difference between flash and memory
- Eliminates “out-of-memory” errors

A specific amount of memory is specified for a cache. The rest *spills* over into flash.



Simplifies configuration:

- Specify on-heap memory
- Specify storage location

ORACLE

Elastic data, introduced in Coherence 3.7, allows for the use of solid state devices, most typically flash drives, to provide spillover capacity for a cache. Solid state disks or SSDs use integrated circuit assemblies as memory to store data persistently rather than the spinning platter used in typical hard disk devices. SSDs are considerably faster, more resistant to shock, and silent in comparison to their HDD cousins. However, they are also considerably more expensive.

Using Elastic data, a cache is specified to use a specific type of backing map based on a RAM or DISK journal. After it is defined, the cache can support massive amounts of data per node, often as much as 100 GB. Elastic data has been extensively tested and is proven to have almost no performance difference with totally RAM-based solutions.

Additionally, Elastic data is simple to configure. In the base case, a caching scheme need only specify that the backing map be RAM journal based. However, a variety of configuration options exist for specifying how much memory to use for the backing map, what device to use, and others.

Benefits

Elastic Data:

- Stores up to four times as many objects in RAM compared to past default cache definitions
- Stores up to 100 GB of data per disk per cluster member while maintaining low latency and high throughput
- Self tunes
- Minimizes GC impact and performs on par or better than default schemes for pure in-memory datasets
- Very easy to configure

Elastic data is ideal for state-of-the-art SSDs such as those provided with Oracle Exalogic.

ORACLE

The benefits of elastic data versus a simple distributed scheme are many.

- Because of the use and nature of journaling, up to 4x objects may be stored in memory as compared to prior versions of coherence.
- With journaling and compression, datasets of 100 GB can be managed on single cluster members.
- Self-tuning – Elastic data self tunes and tracks object use, rebalancing the internal tree structures to maintain the most commonly used data in memory.
- Java garbage collection is minimized as a result of the tree-based journals used to manage data.
- It is very easy to use, based on several sample schemes provided.

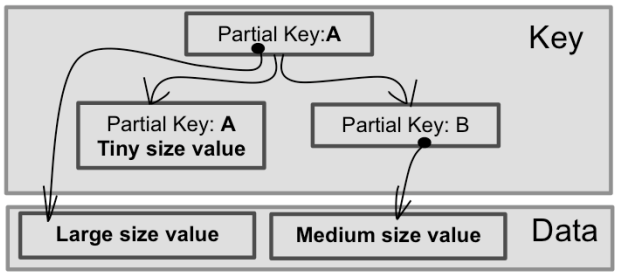
Elastic Data and Journaling

Elastic data is based on journals. Journals:

- Are a technique for storing object state changes
- Record values for a specific key
- Use in-memory trees to store the pointers to value for keys
- Are purged at regular intervals to remove stale data
- File writes are batched and based on device block size
- Are provided in two forms: RAM or flash journals

Tiny, less than seven bytes, values are stored directly in the tree.

```
put(A,large size value);
put(AA,tiny size value)
put(AB,medium size value)
```



ORACLE

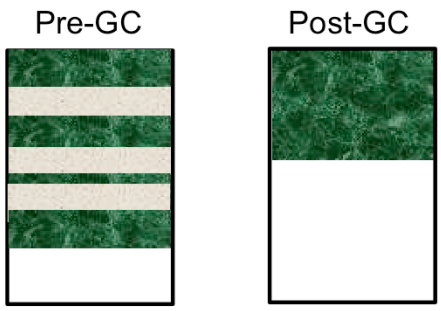
To enable elastic data, all you need to do is specify a backing map scheme that uses a flash or ram journal. Coherence provides two journal schemes, the RAM journal scheme and the flash journal scheme. Each of these schemes works in a similar way. In general, journals record the actual data object/data for a specific key. An in-memory map is then used to specify which journal contains the most recent entry for a given key. Over time, as mutations occur, journal elements become outdated, and are purged by Coherence.

In the example shown, three puts with small, medium, and large values, are done to a cache. As shown, the small value is stored directly in the tree, whereas the two other values are stored in a journal file.

Note that, when Journals are used, additional capacity is required for performing queries and aggregations. See the administrators' guide for details.

Journaling and Garbage Collection

- As values change, old values are invalidated and become garbage.
- When the garbage ratio threshold is reached, garbage collection occurs concurrently.
- Files are recycled during the garbage collection process.



ORACLE

As values change, garbage is created within a journal file. Coherence uses an internal garbage collection algorithm to track whether a garbage threshold is reached. When the threshold is reached, a garbage collection thread executes concurrently to remove old values. During this process, files are recycled and reused.

Configuring a Journaling Backing Map

Example:

```
<cache-config>
  <caching-scheme-mapping>
    <cache-mapping>
      <cache-name>Journaled</cache-name>
      <scheme-name>JournalScheme</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-schemes>
    <distributed-scheme>
      <scheme-name>JournalScheme</scheme-name>
      <service-name>DistributedCache</service-name>
      <backing-map-scheme>
         <[flash|ram]journal-scheme/>
      </backing-map-scheme>
      <autostart>true</autostart>
    </distributed-scheme></cache-config>
```

coherence-cache-confg.xml

ORACLE

The simplest form of an elastic cache is based on the RAM journal which can be added to the backing-map-scheme element to define an elastic data-based cache. When memory is exceeded, the RAM journal automatically delegates to a flash journal. Specifying a flash journal results in the initial use of a flash journal rather than being based on running out of memory.

Both flash and RAM journals can be controlled by using command line arguments and override files. The most common override is the size of the journal itself, typically some percentage of heap. However, a variety of other values can be controlled, such as how large a value to keep in memory, what directory to use for storage, and similar variables.

Controlling Journaling Behavior

Journal behavior is:

- Controlled by a journal manager
- Defined in an operational override or via properties

```
-Dtangosol.coherence.ramjournal.size={value}
```

Where `value` is %of heap or size. For example:

```
-Dtangosol.coherence.ramjournal.size=25%
-Dtangosol.coherence.ramjournal.size=10gb
```

`cache-server.[cm|sh]`

ORACLE

The most common variable to control with a journal is its size. Coherence supports defining the size of a RAM journal, which is inherited by a flash journal as well, as either a percentage of RAM or as a fixed size. The `tangosol.coherence.ramjournal.size` property can be used on the command line to define a size. However, a variety of other elements can also be defined when using an override file, such as the maximum size of a journal value entry.

Controlling Journaling Behavior

Journaling can be finely controlled by using the `journaling-config` element within a cluster configuration.

```
<?xml version='1.0'?>
<coherence >
  <cluster-config>
    . . .
    <ramjournal-manager>
        <maximum-value-size>64K</maximum-value-size>
        <maximum-size>2G</maximum-size>
    </ramjournal-manager>
    <flashjournal-manager>
        <block-size>256KB</block-size>
        <directory></directory>
        <async-limit>16MB</async-limit>
    </flashjournal-manager>
  </cluster-config>
. . .
</coherence>
```

Partial list

tangosol-coherence-override.xml

ORACLE

RAM and flash journals can be finely controlled by using an operational override file.

RAM journal manager subelements control the overall size as well as the maximum value size stored in memory. The two elements of a `ramjournal-manager` element are:

- **`maximum-value-size`** – The maximum size element that can be stored in memory. Default is 65 K, max 4 MB.
- **`maximum-size`** – The maximum amount of RAM to be used by the journal. The value can be specified as an actual value or a percentage. Values must be in the 16 MB to 64 GB range. Percentages are a fraction of max heap, typically specified using **`–Xmx`**.

Flash journal manager subelements include:

- **`maximum-value-size`** – The maximum size, in bytes, of binary values that are to be stored in the flash journal. The value cannot exceed 64 MB. The default value is 64 MB.
- **`block-size`** – It specifies the size of the write buffers in which writes to an underlying disk file occur. The size should match or be a multiple of the physical device's optimal block size and must be a power of two. The value must be between 4 KB and 1 MB. The default value is 256 KB.

- **`maximum-file-size`** – It specifies the maximum file size of the underlying journal files. The value must be a power of two and a multiple of the block size. The value must be between 1 MB and 4 GB. The default value is 2 GB.
- **`maximum-pool-size`** – It specifies the size, in bytes, for the buffer pool. The size does not limit the number of buffers that can be allocated or that can exist at any point in time. The size determines only the amount of buffers that are recycled. The pools size cannot exceed 1 GB. The default value is 16 MB.
- **`Directory`** – It specifies the directory where the journal files should be placed. The directory must exist and is not created at run time. If the directory does not exist or is not specified, the JVM/operating system's default temporary directory is used. The suggested location is a local flash (SSD) drive. Specifying a directory that is located on a drive which is shared by other applications or system operations increases the potential for unplanned space usage. Use a directory location on a nonshared disk partition to ensure a more predictable environment.
- **`async-limit`** – It specifies the maximum size, in bytes, of the backlog. The backlog is the amount of data that has yet to be persisted. Client threads are blocked if the configured limit is exceeded and remain blocked until the backlog recedes below the limit. This helps prevent out-of-memory conditions.

Note: The maximum amount of memory used by the backlog is at least twice the configured amount, because the data is in binary form and rendered to the write-behind buffers. The value must be between 4 KB and 1 GB. The default value is 16 MB.

Considerations for Elastic Data

Elastic data:

- Typically should be sized 1:10 memory to disk
- Uses files, but is not a persistence solution
- Indexes are stored in memory and must be considered in sizing
- Does not support Cache Eviction
- Should not be used with aggregations and entry processors