Coherence Monitoring

You know how, but what and why

David Whitmarsh

March 29, 2015

Outline

This talk is about monitoring. We have had several presentations on monitoring technologies at Coherence SIGs in the past, but they have been focussed on some particular product or tool. There are many of these on the market all with their own strengths and weaknesses. Rather than look at some particular tool and how we can use it, I'd like to talk instead about what, and why we monitor, and give some thought as to what we would like from our monitoring tools to help achieve our ends. Monitoring is more often a political and cultural challenge than a technical one. In a large organisation our choices are constrained and our requirements affected by corporate standards, by the organisational structure of the support and development teams, and by the relationships between teams responsible for interconnecting systems,

What?

- Operating System
- JMX
- Logs

operatings system metrics include details of memory, network and CPU usage, and in particular the status of running processes. JMX includes Coherence, platform and application MBeans. Log messages may derive from Coherence or our own application classes. It is advisable to maintain a separate log file for logs originating from Coherence, using Coherence's default log file format. Oracle have tools to extract information from standard format logs that might help in analysing a problem. You may have compliance problems sending logs to Oracle if they contain client confidential data from your own logging.

Development team culture can be a problem here. It is necessary to think about monitoring as part of the development process, rather than an afterthought. A little though about log message standards can make communication with L1 support teams very much easier, its in your own interest.

Why?

- Programmatic - orderly cluster startup/shutdown - *ms/seconds*
- Detection - *seconds/minutes*
- Analysis (incidents/debugging) - *hours/days*
- Planning - *months/years*

Why?

- Programmatic - orderly cluster startup/shutdown - *ms/seconds*
- Detection - *seconds/minutes*
- Analysis (incidents/debugging) - *hours/days*
- Planning - *months/years*

Programmatic monitoring is concerned with the orderly transition of system state. Have all the required storage nodes started and paritioning completed before starting feeds or web services that use them? Have all write-behind queues flushed before shutting down the cluster?
Detection is the identification of failures that have occurred or are in imminent danger of occurring leading to the generation of an alert so that appropriate action may be taken. The focus is on what is happening now or may soon happen.
Analysis is the examination of the history of system state over a particular period in order to ascertain the causes of a problem or to understand its behaviour under certain conditions. Requiring the tools to store, access, and visualise monitoring data.
Capacity planning is concerned with business events rather than cache events, the correlation between these will change as the business needs and the system implementation evolves. Monitoring should detect the long-term evolution of the system usage in terms of business events and provide

Resource Exhaustion

- Memory (JVM, OS)
- CPU
- i/o (network, disk)

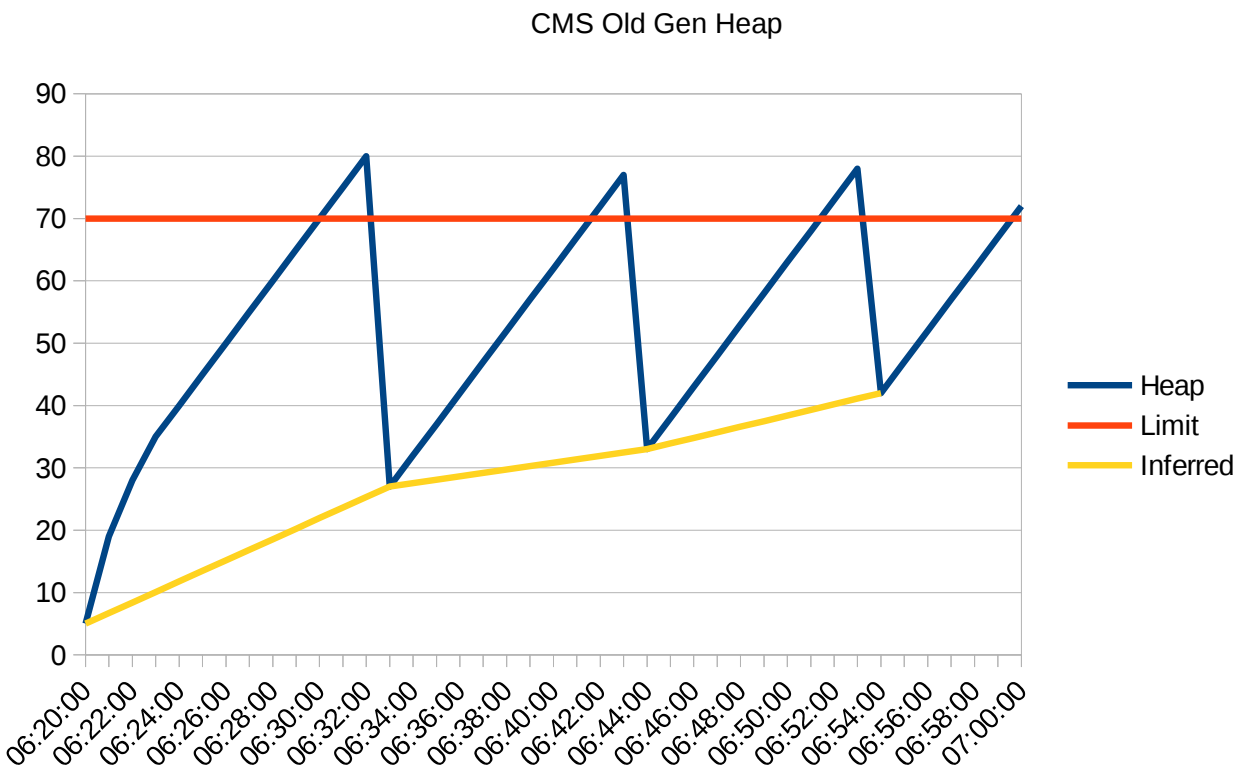
- Memory (JVM, OS)
- CPU
- i/o (network, disk)

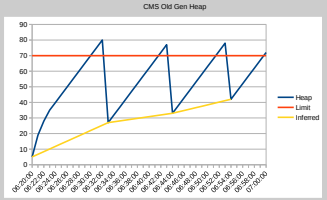
Alert on memory - your cluster will collapse. But a system working at capacity might be expected to use 100% CPU or i/o unless it is blocked on external resources. Of these conditions, only memory exhaustion(core or JVM) is a clear failure condition. Pinning JVMs in memory can mitigate problems where OS memory is exhausted by other processes but the CPU overhead imposed on a thrashing machine may still threaten cluster stability. Page faults obove zero (for un-pinned JVMS), or above some moderate threshold (for pinned JVMs) should generated an alert.

A saturated network for extended periods may threaten cluster stability. Maintaining all cluster members on the same subnet and on their own private switch can help in making network monitoring understandable. Visibility of load on external routers or network segments involved in cluster communications is harder. Similarly 100% CPU use may or may not impact cluster stability depending on various configuration and architecture choices.

Defining alert rules around CPU and network saturation can be problematic.

CMS garbage collection





CMS garbage collectors only tell you how much old gen heap is really being used immediately after a mark-sweep collection, At any other time, the figure includes some indterminate amount. Memory use will always increase until the collection threshold is reached, said threshold usually being well above the level at which you should be concerned if it were all really in use.

CMS Danger Signs

- Memory use *after GC* exceeds a configured limit
- PS MarkSweep collections become too frequent
- “Concurrent mode failure” in GC log

- Memory use *after GC* exceeds a configured limit
- PS MarkSweep collections become too frequent
- “Concurrent mode failure” in GC log

Concurrent mode failure happens when the rate at which new objects are being promoted to old gen is so fast that heap is exhasuted before the collection can complete. This triggers a "stop the world" GC which should always be considered an error. Investigate, analyse, tune the GC or modify application behaviour to prevent further occurrences. As real memory use increases, under a constant rate of promotions to old gen, collections will become more frequent. Consider setting an alert if they become too frequent. In the end-state the JVM may end up spending all its time performing GCs.

CMS Old gen heap after gc

MBean	Coherence:type=Platform, Domain=java.lang,subType=GarbageCollector, name=PS MarkSweep, nodeId=1
Attribute	LastGcInfo
attribute type	com.sun.management.GcInfo
Field	memoryUsageAfterGc
Field type	Map
Map Key	PS Old Gen
Map value type	java.lang.management.MemoryUsage
Field	Used

MBean	Coherence:type=Platform, Domain=java.lang,subType=GarbageCollector, name=PS MarkSweep, nodeId=1
Attribute	LastGcInfo
attribute type	com.sun.management.GcInfo
Field	memoryUsageAfterGc
Field type	Map
Map Key	PS Old Gen
Map value type	java.lang.management.MemoryUsage
Field	Used

Extracting memory use after last GC directly from the platform MBeans is possible but tricky. Not all monitoring tools can be configured to navigate this structure. Consider implementing an MBean in code that simply delegates to this one.

CPU

- Host CPU usage
- per JVM CPU usage
- Context Switches

Network problems

- OS interface stats
- MBeans: resent/failed packets
- Outgoing backlogs
- Cluster communication failures “probable remote gc”

Cluster Stability Log Messages

- Experienced a %n1 ms communication delay (probable remote GC) with Member %s
- validatePolls: This senior encountered an overdue poll, indicating a dead member, a significant network issue or an Operating System threading library bug (e.g. Linux NPTL): Poll
- Member(%s) left Cluster with senior member %n
- Assigned %n1 orphaned primary partitions

- Experienced a %n1 ms communication delay (probable remote GC) with Member %s
- validatePolls: This senior encountered an overdue poll, indicating a dead member, a significant network issue or an Operating System threading library bug (e.g. Linux NPTL): Poll
- Member(%s) left Cluster with senior member %n
- Assigned %n1 orphaned primary partitions

It is absolutely necessary to monitor Coherence logs in realtime to detect and alert on error conditions. These are are some of the messages that indicate potential or actual serious problems with the cluster. Alerts should be raised. The communication delay message can be evoked by many underlying issues: high CPU use and thread contention in one or more nodes, network congestion, network hardware failures, swapping, etc. Occasionally even by a remote GC.

validatePolls can be hard to pin down. Sometimes it is a transient condition, but it can result in a JVM that continues to run but becomes unresponsive, needing that node to be killed and restarted. The last two indicate that members have left the cluster, the last indicating specifically that data has been lost. Reliable detection of data loss can be tricky - monitoring logs for this message can be tricky. There are many other Coherence log messages that can be suggestive of, or diagnostic of problems, but be cautious in choosing to raise alerts - you don’t want too many false alarms caused by transient conditions.

Cluster Stability MBeans

- Cluster
  - ClusterSize
  - MembersDepartureCount
- Node
  - MemberName
  - RoleName
  - Service

Cluster Stability MBeans

- Cluster
  - ClusterSize
  - MembersDepartureCount
- Node
  - MemberName
  - RoleName
  - Service

Some MBeans to monitor to verify that the complete cluster is running. The Cluster MBean has a simple ClusterSize attribute, the total number of nodes in the cluster.

MembersDepartureCount attribute can be used to tell if any members have left. Like many attributes this is monotonically increasing so your monitoring solution should be able to compare current value to previous value for each poll and alert on change. Some MBeans (but not this one) have a reset statistics that you could use instead, but this introduces a race condition and prevents meaningful results being obtained by multiple tools.

A more detailed assertion that cluster composition is as expected can be performed by querying Node MBeans - are there the expected number of nodes of each RoleName?

More fine-grained still is to check the Service Mbeans - is each node of a given role running all expected services?

Monitoring should be configured with the expected cluster composition so that alerts can be raised on any deviation.

Network Statistics MBeans

- Node
  - PacketsRepeated
  - PacketsResent
  - PublisherSuccessRate, ReceiverSuccessRate - only useful with periodic reset
- ConnectionManagerMBean
  - OutgoingByteBaccklog
  - OutgoingMessageBaccklog
  - TotalBytesReceived
  - TotalBytesSent

Network Statistics MBeans

- Node
  - PacketsRepeated
  - PacketsResent
  - PublisherSuccessRate, ReceiverSuccessRate - only useful with periodic reset
- ConnectionManagerMBean
  - OutgoingByteBacklog
  - OutgoingMessageBacklog
  - TotalBytesReceived
  - TotalBytesSent

PublisherSuccessRate and ReceiverSuccessRate may at first glance look like useful attributes to monitor, but they aren't. They contain the average success rate since the service or node was started, or since statistics were reset. So they revert to a long term mean, over time since either of these. Packets repeated or resent or good indicators of network problems within the cluster. The connectionmanagermbean bytes received and sent can be useful in diagnosing network congestion when caused by increased traffic between cluster and extend clients. Backlogs indicate where outgoing traffic is being generated faster than it can be sent - which may or may not be a sign of network congestion. Alerts on thresholds on each of these may be useful early warnings.

Endangered Data MBeans

- Service
- PartitionAssignment
  - HAStatus, HATarget
  - ServiceNodeCount
  - RemainingDistributionCount
  - partition.lost notification

Endangered Data MBeans

- Service
- PartitionAssignment
  - HAStatus, HATarget
  - ServiceNodeCount
  - RemainingDistributionCount
  - partition.lost notification

(The Service MBean has an HAStatus field that can be used to check that each node service is in an expected, safe HAStatus value. The meaning of Expected and Safe may vary. A service with backup-count zero will never be better than NODE_SAFE. Other services may be MACHINE, RACK, or SITE safe depending on the configurations of the set of nodes. The ParitionAssignment MBean also provides a HATarget attribute identifying the best status that can be achieved with the current cluster configuration. Asserting that HAStatus = HATarget in conjunction with an assertion on the size and composition of the member set can be used to detect problems with explicitly configuring the target state.

These assertions can be used to check that data is in danger of being lost, but not whether any has been lost. To do that requires a canary cache, a custom MBean based on a PartitionLost listener, a check for orphaned partition messages in the log, or a JMX notification listener registered on partition.lost notifications)

Guardian timeouts

- You have set the guardian to logging?
- Do you want:
  - Heap dumps on slight excursions from normal behaviour and HUGE logs
  - Heap dumps only on extreme excursions and manageable logs

Guardian timeouts

- You have set the guardian to logging?
- Do you want:
  - Heap dumps on slight excursions from normal behaviour and HUGE logs
  - Heap dumps only on extreme excursions and manageable logs

Guardian timeouts set to anything other than logging are dangerous - arguably more so than the potential deadlock situations they are intended to guard against. The heap dumps provided by guardian timeout set to logging can be very useful in diagnosing problems, however if set too low, a poorly performing system can generate very large numbers of very large heap dumps in the logs. Which may itself cause or mask other problems.

Asynchronous Operations

- Cache MBean: StoreFailures
- Service MBean: EventInterceptorInfo.ExceptionCount
- StorageManager MBean: EventInterceptorInfo.ExceptionCount

We want to alert on failures of service or infrastructure, or risks to the stability of the entire system but we would normally expect that failures in individual workflows through the system would be handled by application logic. The exception is in the various asyncronous operations that can be configured within Coherence - actions taken by write-behind cachestores or post-commit interceptors cannot be easily reported back to the application. All such implementations should catch and log exceptions so that errors can be reported by log monitoring, but there are also useful MBean attributes. The Cache MBean reports the total number of cachestore failures in StoreFailures. The EventInterceptorInfo attribute of the Service MBean reports a count of transaction interceptor failures and the same attribute on StorageManager, the cache interceptor failures.

The problem of scale

- Many hosts
- Many JVMs
- High operation throughput

- We don’t want the management of monitoring data to become a problem on the same scale as the application we are monitoring
- Monitoring is not an afterthought

The problem of scale

- Many hosts
- Many JVMs
- High operation throughput
- We don't want the management of monitoring data to become a problem on the same scale as the application we are monitoring
- Monitoring is not an afterthought

A coherence cluster with many nodes can produce prodigious amounts of data. Monitoring a large cluster introduces challenges of scale. The overhead of collection large amounts of MBEan data and logs can be significant. Some guidelines: Report aggregated data where possible. Interest in maximum or average times to perform on operation can be gathered in-memory and reported via MBeans more efficiently and conveniently than logging every operation and analysing the data afterwards. See Yammer metrics or apache statistics libraries.

You will need a log monitoring solution that allows you to query and correlate logs from many nodes and machines. Logging into each machine separately is untenable in the long-term. Such a solution should not *cause* problems for the cluster, and should be as robust as possible against *existing* issues - e.g. network saturation.

Loggers should be asynchronous

Log to local desk, not to NAS or direct to network appenders

Aggregate log data to the central log monitoring solution asynchronously - if you have network problems, you will get the data eventually

Aggregation and Excursions

- MBeans for aggregate statistics
- Log excursions
- Log with context, cache, key, thread
- Consistency in log message structure
- Application Log at the right level
- Coherence log level 5 or 6?

.

- MBeans for aggregate statistics
- Log excursions
- Log with context, cache, key, thread
- Consistency in log message structure
- Application Log at the right level
- Coherence log level 5 or 6?

All common sense really. Too much volume becomes unmanageable. Increase log level selectively to further investigate specific issues as necessary. I've seen more problems caused by excessively verbose logging than solved by it

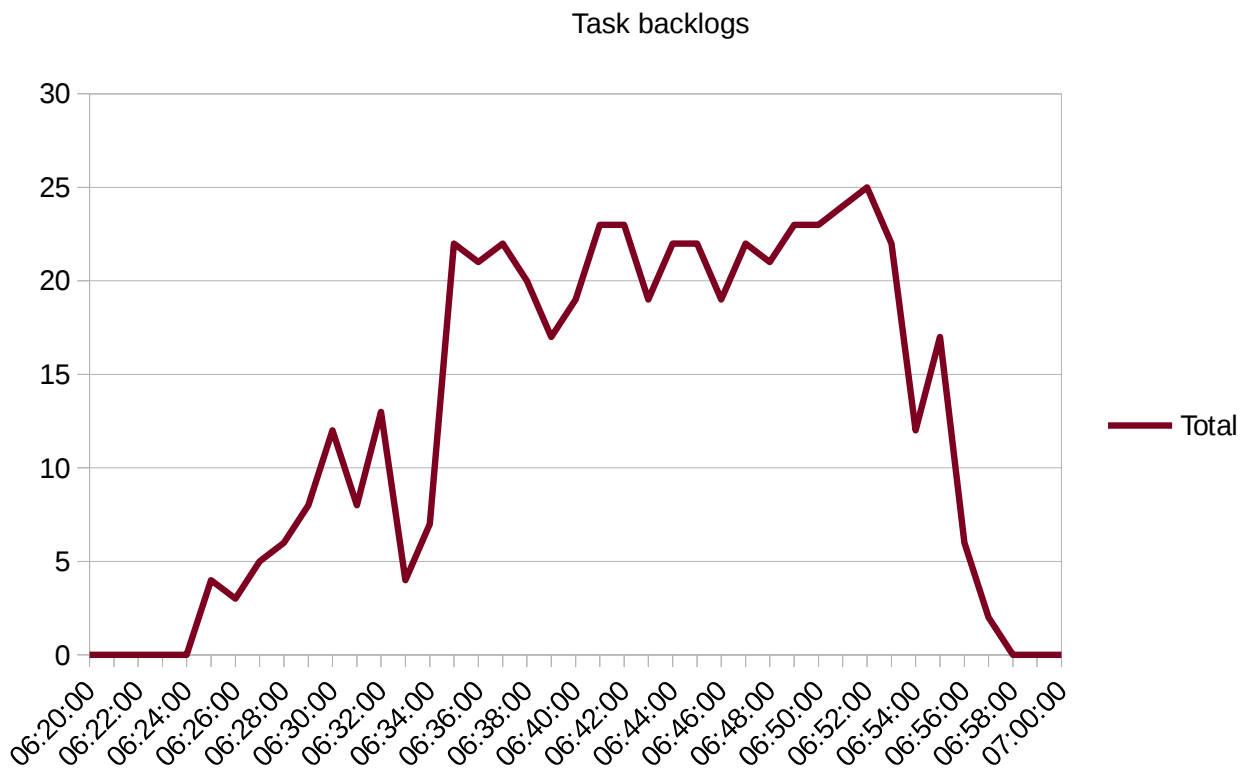
Analysing storage node service behaviour

- thread idle count
- task counts
- task backlog

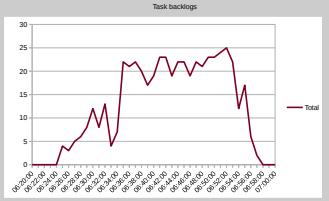
- thread idle count
- task counts
- task backlog

task counts tell you how much work is going on in the service, the backlog tells you if the service is unable to keep up with the workload.

Aggregated task backlog in a service

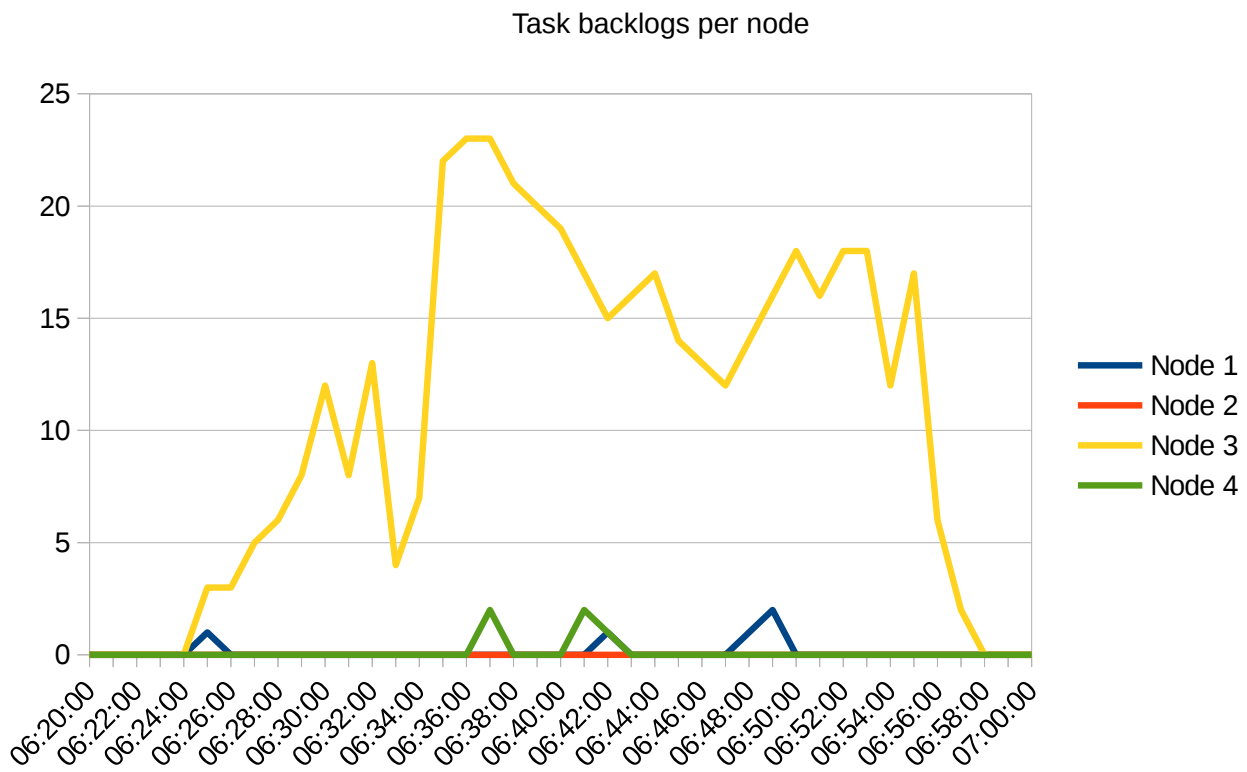


Aggregated task backlog in a service

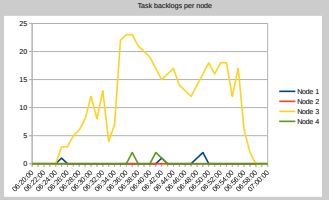


An aggregated count of backlog across the cluster tells you one thing

Task backlog per node



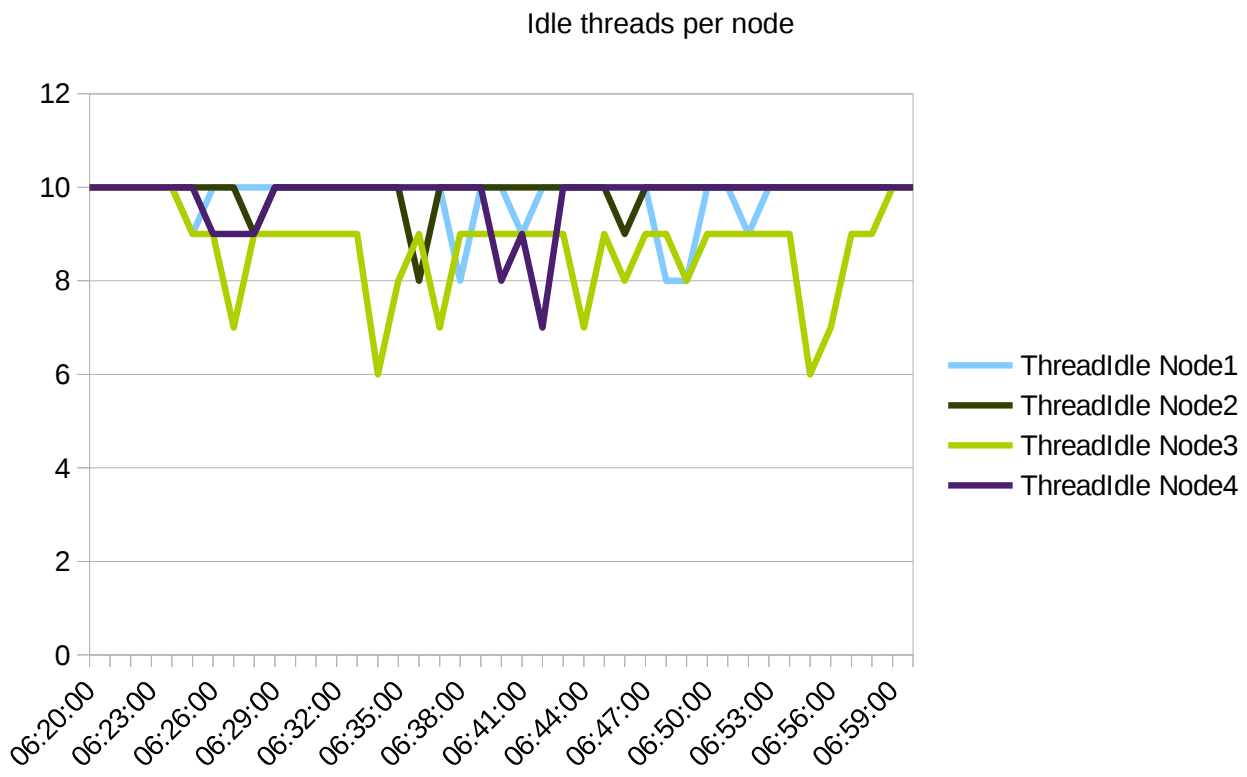
Task backlog per node

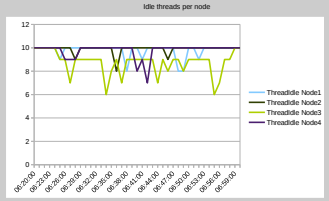


A breakdown by node can tell you more - in this case we have a hot node.

Aggregated stats can be very useful for the first look, but you need the drill down

Idle threads





Correlating with other metrics can also be informative. Our high backlog here correlates with low thread usage, so we infer that it isn't just a hot node, but a hot key.

Diagnosing long-running tasks

- High CPU use
- Operating on large object sets
- Degraded external resource
- Contention on external resource
- Contention on cache entries
- CPU contention
- Difficult to correlate with specific cache operations

- High CPU use
- Operating on large object sets
- Degraded external resource
- Contention on external resource
- Contention on cache entries
- CPU contention
- Difficult to correlate with specific cache operations

There are many potential causes of long-running tasks.

Client requests

- A request may be split into many tasks
- Task timeouts are seen in the client as exception with context stack trace
- Request timeout cause can be harder to diagnose

- A request may be split into many tasks
- Task timeouts are seen in the client as exception with context stack trace
- Request timeout cause can be harder to diagnose

A client request failing with a task timeout will give you some context to determine at least what the task was, a request timeout is much less informative. Make sure your request timeout period is significantly longer than your task timeout period to reduce the likelihood of this

Instrumentation

- Connection Pool and thread pool stats (JMX Mbeans)
- Timing statistics (JMX MBeans, max, rolling avg)
  - EP/Aggregator timings & set sizes
  - CacheStore operation timings & set sizes
  - Interceptor/backing map listener timings
  - Serialiser timings - Subclass/delegate POFContext
  - Query execution times - Filter decorator
- Log details of exceptional durations with context

- Connection Pool and thread pool stats (JMX Mbeans)
- Timing statistics (JMX MBeans, max, rolling avg)
  - EP/Aggregator timings & set sizes
  - CacheStore operation timings & set sizes
  - Interceptor/backing map listener timings
  - Serialiser timings - Subclass/delegate POFContext
  - Query execution times - Filter decorator
- Log details of exceptional durations with context

Really understanding the causes of latency issues requires instrumenting all of the various elements in which problems can arise. Some of these can be tricky - instrumenting an apache http client to get average times requests is tricky, but not impossible - it is important to separate out timings for obtaining a connection from a pool, creating a connection, and executing and fetching results from a request. For large objects frequently accessed or updated, serialisation times can be an issue.

Either register and MBean to publish rolling average and max times for particular operations, or log individual events that exceed a threshold, or maybe both.

Manage The Boundaries

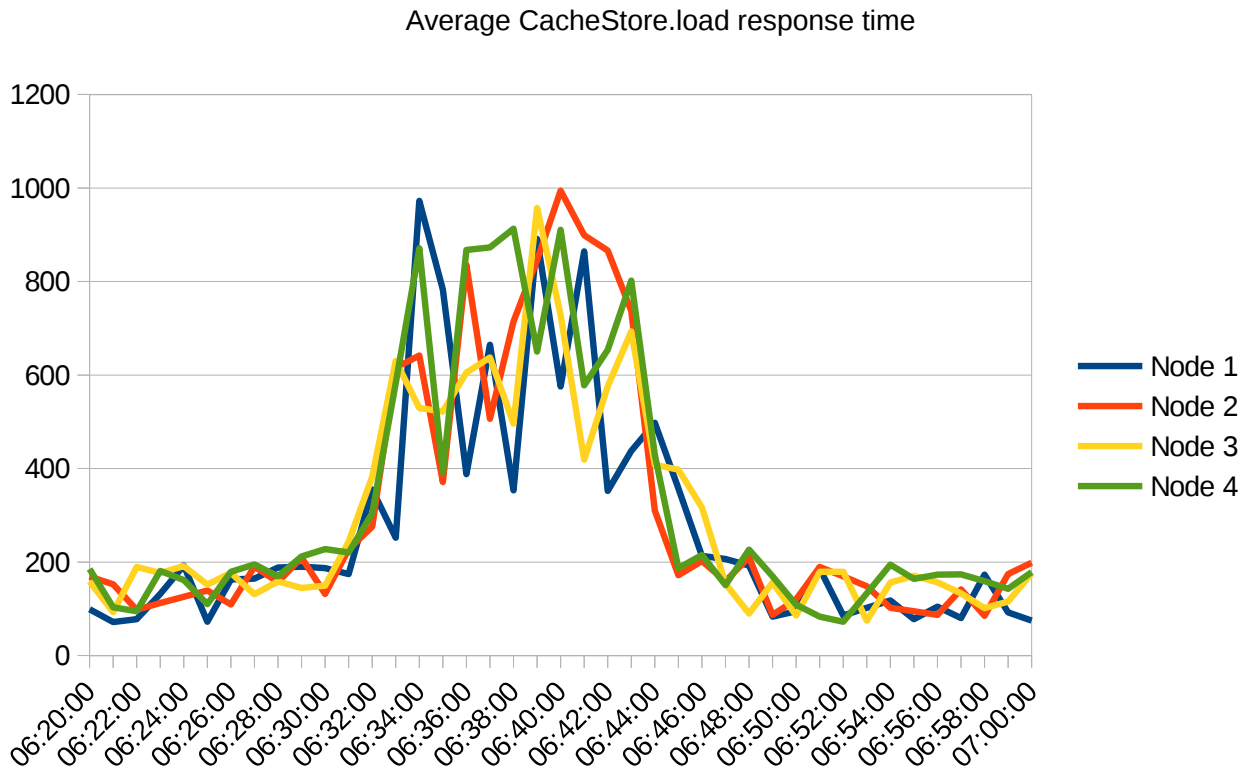
- Are you meeting your SLA for your clients?
- Are the systems you depend on meeting their SLAs for you?

Manage The Boundaries

- Are you meeting your SLA for your clients?
- Are the systems you depend on meeting their SLAs for you?

Instrumentation is particularly important at boundaries with other systems. It is much better to tell your clients that they may experience latency issues because your downstream reference data service is slow, than to have to investigate after they complain. The reference data team might even appreciate you telling them first.

Response Times



Monitoring Solution Wishlist

- Collate data from many sources (JMX, OS, logs)
- Handle composite JMX data, maps etc
- Easily configure for custom MBeans
- Summary and drill-down graphs
- Easily present different statistics on same timescale
- Parameterised configuration, source-controlled and released with code
- Continues to work when cluster stability is threatened.
- Calculate deltas on monotonically increasing quantities
- Handle JMX notifications
- Manage alert conditions during cluster startup/shutdown
- Aggregate repetitions of log messages by regex into single alert

About me

David Whitmarsh
david.whitmarsh@shadowmist.co.uk
http:/www.shadowmist.co.uk
http:/www.coherencecookbook.org