BigFix 11 Infrastructure Monitoring

Author: Mark Leitch @ HCL Software

BigFix provides several methods to monitor and understand the health of a deployment, including system health dashboards and relevance. However, at times deeper introspection is required into the infrastructure BigFix runs upon.

This document will describe the following infrastructure monitoring approaches.

1. System resource utilization.
2. The SQL package cache.
3. SQL deadlocks.
4. SQL storage performance.
5. SQL index fragmentation.
6. SQL statistics.
7. Miscellaneous SQL

Through the described monitors, it is possible to provide:

1. System monitoring results under load for the BigFix root and database servers.
    - If these servers are not collocated (i.e., they are on separate operating system instances), two invocations of the monitor will be required (i.e., one invocation for each system).
2. Package cache results under suitable workload for the database server.
3. SQL transaction deadlock event monitor information.
4. A report on the current database storage performance.
5. A report on the current database index fragmentation.
6. A report on the “freshness” of database statistics.
7. Various high level and low level monitoring approaches via SQL.

1 SYSTEM RESOURCE UTILIZATION

The BigFix Performance Toolkit (see the "Further Reading" section) provides a monitor wrapper utility known as MXPerfmon. This utility generates system specific monitoring commands that may be of value on any node in a BigFix deployment.

The following sample Windows monitor commands were generated by MXPerfmon and may be used directly to monitor system resources. Some notes on this monitor follow.

- You need to run an instance on each machine from an elevated shell.
- The sample monitor samples every 60 seconds for 1440 iterations, which is equivalent to 24 hours.
    - The monitor frequency may be adjusted, but it is recommended not to go below a five (5) second interval or have an excessive number of data points. For example, in the ballpark of 1000 data points is plenty.
- It will create a system monitor output BLG file in typically the C:\Perflogs directory.
- The monitor is named BFMon and is capped at 250MB.

```
logman delete BFMon

logman create counter BFMon -f bincirc -v mmddhhmm -max 250 -c "\LogicalDisk(*)\*"
"\Memory\*" "\Network Interface(*)\*" "\Paging File(*)\*" "\PhysicalDisk(*)\*"
"\Processor(*)\*" "\Process(*)\*" "\Redirector\*" "\Server\*" "\System\*"
"\Thread(*)\*" "\SQLServer:Databases(*)\*" "\SQLServer:Locks(*)\*"
"\SQLServer:Transactions(*)\*" "\SQLServer:Wait Statistics(*)\*"
"\SQLServer:Availability Replica(*)\*" "\SQLServer:Database Replica(*)\*" -si 60 -
sc 1440

logman start BFMon
```

Figure 1: MXPerfmon Windows Monitor

Additional notes:

- The SQL counters only apply to the SQL node, and the name of the SQL counters may not match your environment. It is best to inspect your local system to confirm and possibly update the counter names.
- It is possible to include other options, such as TCP/IP/UDP counters, using the provided MXPerfmon utility.
- Once the BLG file has been generated, it may be compressed and forwarded to product support. The file typically compresses well (e.g., an 8x compression rate).

2 THE SQL PACKAGE CACHE

The SQL package cache provides a snapshot of all currently "active" MS SQL transactions. For database experts, this snapshot can prove invaluable. The following query may be used to provide a current snapshot of the package cache. Note the "ORDER BY" predicate may be altered based on the desired ordering of results.

```
SELECT TOP 100 SUBSTRING(qt.TEXT, (qs.statement_start_offset/2)+1,
((CASE qs.statement_end_offset
WHEN -1 THEN DATALENGTH(qt.TEXT)
ELSE qs.statement_end_offset
END - qs.statement_start_offset)/2)+1) AS query_text,
qs.execution_count,
qs.total_rows, qs.last_rows, qs.total_rows / qs.execution_count avg_rows,
qs.total_dop, qs.last_dop, qs.total_dop / qs.execution_count avg_dop,
qs.total_logical_reads, qs.last_logical_reads,
qs.total_logical_writes, qs.last_logical_writes,
qs.total_physical_reads, qs.last_physical_reads,
qs.total_worker_time, qs.last_worker_time,
qs.total_worker_time / qs.execution_count avg_worker_time,
qs.total_elapsed_time / 1000 total_elapsed_time_in_ms,
qs.last_elapsed_time / 1000 last_elapsed_time_in_ms,
qs.total_elapsed_time / qs.execution_count / 1000 avg_elapsed_time_in_ms,
qs.creation_time, qs.last_execution_time, qp.query_plan
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle) qp
ORDER BY avg_elapsed_time_in_ms DESC -- CPU avg
```

Figure 2: SQL Package Cache

3 SQL DEADLOCKS

SQL deadlocks involve competing transactions from two or more applications. Deadlocks are rare events in BigFix and may be influenced by custom integrations and reporting tools. If a deadlock occurs, a victim is selected for rollback. Deadlock occurrences may be viewed as part of the SQL collector in the system resource monitor capture described in the first part of this guide. See the "Further Reading" sections for Microsoft resources that provide more detail on the definition and management of deadlocks.

To better understand the cause of a deadlock, MS SQL installs a deadlock monitor by default, and it may be inspected historically. Deadlocks may be inspected by the system_health session in SQL Server Management Studio (see "Further References"). The following query may be used to extract this information.

```
SELECT xdr.value('@timestamp', 'datetime') AS [Date],
       xdr.query('.') AS [Event_Data]
FROM (SELECT CAST ([target_data] AS XML) AS Target_Data
       FROM sys.dm_xe_session_targets AS xt
              INNER JOIN sys.dm_xe_sessions AS xs
              ON xs.address = xt.event_session_address
       WHERE xs.name = N'system_health'
              AND xt.target_name = N'ring_buffer') AS XML_Data
CROSS APPLY
Target_Data.nodes('RingBufferTarget/event[@name="xml_deadlock_report"]') AS
XEventData(xdr)
ORDER BY [Date] DESC;
```

Figure 3: SQL Deadlock Information

4 SQL STORAGE PERFORMANCE

SQL storage performance is critical to BigFix server health. The general storage latency recommendation is < 1ms for the database containers and logs. This metric is easily achievable with modern storage systems, though typically priced at a premium for cloud-based solutions. The following query will derive the average read and write latencies across the recognized databases.

```
select [database_id], [file_id],
       [io_stall_read_ms] / (1.0 + [num_of_reads]) as [Avg Read Latency],
       [io_stall_write_ms] / (1.0 + [num_of_writes]) as [Avg Write Latency],
       [size_on_disk_bytes] / (1024 * 1024) as [Size In MB]
from sys.dm_io_virtual_file_stats(NULL, NULL);
```

Figure 4: SQL Storage Performance

To derive the list of database identifiers and their associated file identifiers, the following query may be used.

```
Select d.name as [DatabaseName],
       d.database_id as [DatabaseID],
       mf.file_id as [FileID],
       mf.name as [LogicalFileName],
       mf.physical_name as [PhysicalPath],
       mf.type_desc as [FileType]
from sys.databases d inner join sys.master_files mf on d.database_id =
mf.database_id order by d.database_id, mf.file_id;
```

Figure 5: SQL Storage Identifiers

5 SQL INDEX FRAGMENTATION

The SQL fragmentation query provides a data object breakdown of fragmentation counts. It can be used to determine the health of the database and the associated index management scripts provided by BigFix. Please see the BigFix Maintenance Guide in the "Further Reading" section for comprehensive information on the BigFix index management scripts.

```
select s.name,o.name,i.name,ips.avg_fragmentation_in_percent,ips.page_count
from sys.objects o left outer join sys.schemas s on
o.schema_id= s.schema_id  left outer join sys.indexes i on
o.object_id=i.object_id left outer join sys.dm_db_index_physical_stats (db_id(),
NULL, NULL, NULL, NULL) AS IPS
on i.object_id=IPS.object_id and i.index_id=ips.index_id
where o.type='U' and i.index_id > 0 order by avg_fragmentation_in_percent desc
```

Figure 6: SQL Storage Fragmentation

6 SQL STATISTICS

MS SQL uses statistics for query optimization. For example, if a table has low row cardinality, it may prefer a table scan versus an index lookup. The following query will provide the current set of SQL statistics in descending order by the last update timestamp.

```
select sp.stats_id, name, filter_definition, last_updated, rows, rows_sampled,
steps, unfiltered_rows, modification_counter from sys.stats as stat cross apply
sys.dm_db_stats_properties(stat.object_id, stat.stats_id) as sp order by
last_updated desc
```

Figure 7: SQL Statistics Query

In the event the statistics appear old or incomplete, MS SQL offers a comprehensive stored procedure to force statistics updates.

```
exec sp_updatestats;
```

Figure 8: SQL Statistics Update

7 MISCELLANEOUS SQL

The following table provides a miscellaneous set of SQL for understanding a BigFix deployment.

Purpose	SQL
SQL configuration including lock allocation, server memory allocation (min, max), CTFP, MAXDOP, etc.	exec sp_configure;
SQL running processes with blocking information (I.e., "Blk By").	exec sp_who2;
Operating system and process level memory view.	• select * from sys.dm_os_sys_memory; • select * from sys.dm_os_process_memory; • select * from sys.dm_os_sys_info; • select sqlserver_start_time, (committed_kb/1024) AS Total_Server_Memory_MB, (committed_target_kb/1024) AS Target_Server_Memory_MB from sys.dm_os_sys_info;
Query and lock management event tracking.	1. Create a new session in SSMS: Management → Extended Events → Sessions. 2. Use the Session Creation Wizard and TSQL_LOCKS template. 3. Take standard events and log to file.
Active locks (waiting and blocking).	SELECT t1.resource_type, DB_NAME(resource_database_id) AS dat_name, T1.resource_associated_entity_id, t1.request_mode, t1.request_session_id, T2.wait_duration_ms, (SELECT TEXT FROM sys.dm_exec_requests r CROSS apply sys.dm_exec_sql_text (r.sql_handle) WHERE r.session_id = t1.request_session_id) AS wait_sql, T2.blocking_session_id, (SELECT TEXT FROM sys.sysprocesses p CROSS apply sys.dm_exec_sql_text (p.sql_handle) WHERE p.spid = t2.blocking_session_id) AS blocking_sql FROM sys.dm_tran_locks t1, sys.dm_os_waiting_tasks t2 WHERE t1.lock_owner_address = t2.resource_address;
Table row counts.	SELECT s.name AS SchemaName, t.name AS TableName, SUM(p.rows) AS TableRowCount

	FROM sys.schemas AS s JOIN sys.tables AS t ON t.schema_id = s.schema_id JOIN sys.partitions AS p ON p.object_id = t.object_id WHERE t.is_ms_shipped = 0 AND p.index_id IN (0, 1) GROUP BY s.name, t.name ORDER BY SchemaName, TableName;

Figure 9: Miscellaneous SQL

8 FURTHER READING

In the event further reading is desired, the following technical resources are available.

BigFix Platform Documentation: URL
BigFix Capacity Planning Guide: URL
BigFix Maintenance Guide: URL
BigFix Performance Toolkit: URL

MS SQL Deadlocks Guide: URL
MS SQL system_health Session: URL
MS SQL Server Management Studio Event Data: URL