Diagnosing and resolving Sporadic Applicaiton Performance Degredation
A Lab128 Case Study

Fred Habash, Oracle OCP

10/21/10

Issue
An application goes offline and users are no longer able to login. All application servers crash requiring a reboot.

Diagnostic data and analysis

Initial diagnostic data
- Initial examination of application environment revealed the following ...
  - Application servers:
    1. A sharp increase in network traffic that accompanied this event.
  - Database server:
    2. A recurrent CPU spike up to 85% usage repeating every 30 minutes.
    1. A similar increase in network traffic from/to application servers.
  - A reboot of application servers resolved the issue and application became available. However, the cpu usage pattern continued after the reboot.

Advanced diagnostic data
- Based on out initial findings, we started our analysis trying to understand the nature of the network traffic that accompanied this event. Even though, the CPU usage patterns was interesting, we did not persue it initially because it persisted after the reboot that resolved the issue.

- To analyze this network traffic, we needed to provide the following:
  1. Quantify it.
  2. Determine if it is mostly inbound vs. out-bound.
  3. Identify the source at session and if possible SQL level.
  4. Finally, use this data to understand and validate application behavior and how it may have contributed to the application server outage.

Quantify it
- This data is readily available in the Lab128 DB home screen. It displays both in/out-bound SQLNet traffic. Just a quick look at the graphs showed us immediately that it was mostly an out-bound traffic from the database. The database was sending out an average of 10MB/sec.

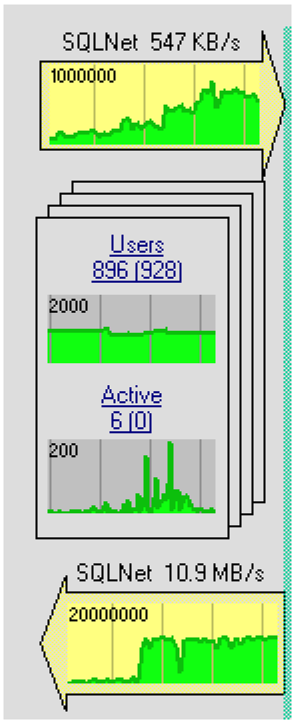


Figure (1)

- Using Lab128, we were also able to accurately identify the time frame during which this event lasted. It lined up perfectly with the application outage time window. Both events started 13:17 and lasted for about 40 minutes.

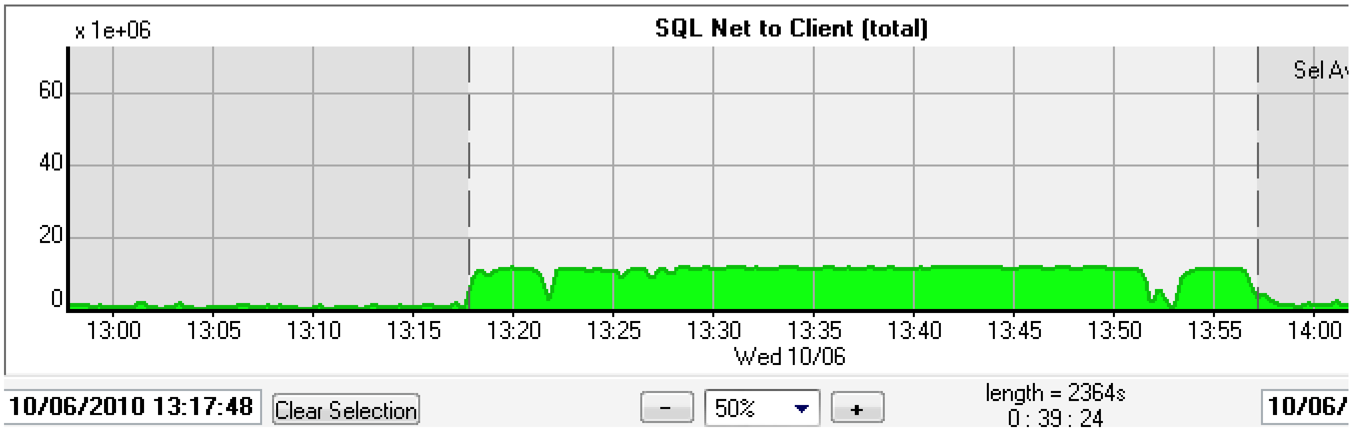


Figure (2)

Identify source (session or SQL)

- This is where the process became a little tricky. There is no direct way within Oracle to associate this statistic (sqlnet bytes to/from client) with a specific session or SQL, and certainly not historically. The closest one can get is to use v$sesstat. The limitation, though, it can not be used historically and given how sessions are reused by the applications, it becomes impractical to drill down at a specific SQL.

- To establish this association, we had these options:

1. Use ASH data wait events (specially sqlnet related events) to extrapolate what sessions could've potentially caused sqlnet bytes traffic. The problem is that these event are considered 'idle'. As such, a session that was sending bytes may have been considered idle if caught waiting on this event.
2. Identify SQL during this time frame and sort it by some dimensions that can potentially be related

to sqlnet bytes traffic. We could use stats like physical or logical IO. The assumption here is a SQL that was on the top IO list may have 'more likely' contributed to sqlnet bytes. This information will need to be eventually verified by application developers for accuracy.

- Not being fully satisfied with above tow options, we reached to the application developers for some clues that can help narrow 'suspect' list down. We were told that the most likely criteria from an application logic standby point is to focus on sql that handles (mostly selects) LOB objects.

- Using this lead, we used LAB28 to drill down as follows:

1. **Identify the time frame and duration.**
   This was done displaying the 'sqlnet to client (total)' graph (figure 2). We established begin/end times as well as duration of the event.

2. **Identify SQL executed during this time frame.**
   - Simply by dragging the mouse (lightly shaded area) to cover the entire event.
   - Then 'Show in SQL explorer' in figure 3.

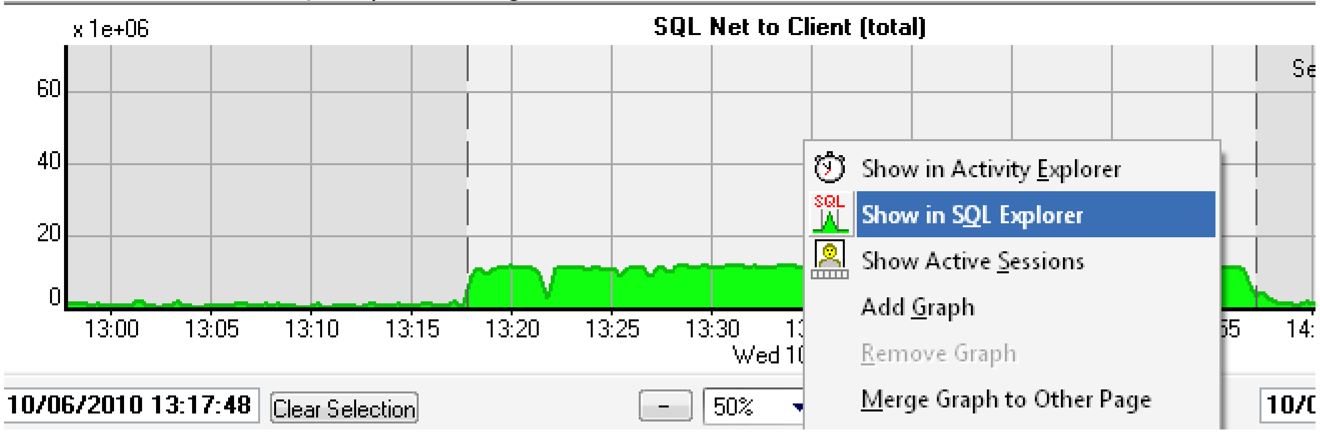


Figure (3)

3. **Isolate SQL that handled LOB data.**
   - Based on our application knowledge, we isolated 9 SQL statements (7 selects and 2 updates) that handled LOB data on candidate tables.

4. **Establish an association between a given SQL_ID executions and 'sqlnet to client' statistic.**
   - We then charted execution count for each for these statements against the 'sqlnet to client' chart looking for associations (time and magnitude).
   - It was immediately visible to us how SQL_ID 'ca9zf5v7r2jyg' lined up almost perfectly with the 'sql net to client (total)' event. This made it the number 1 suspect. At this point, we exonerated the remainder of the sql_id list.
   - We retrieved the full text for this sql_id and was able to determine that indeed was fetching a large size LOB column data.

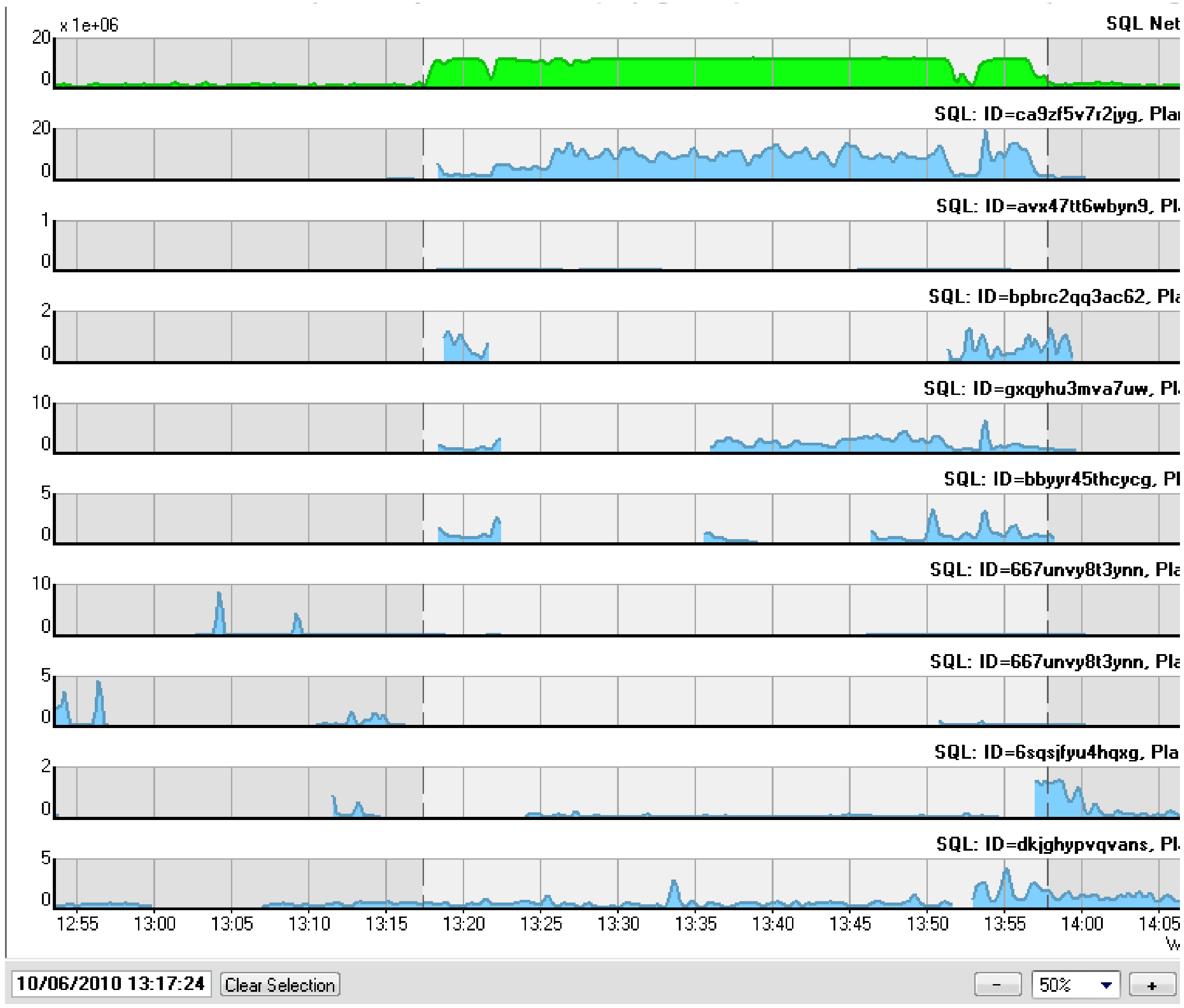


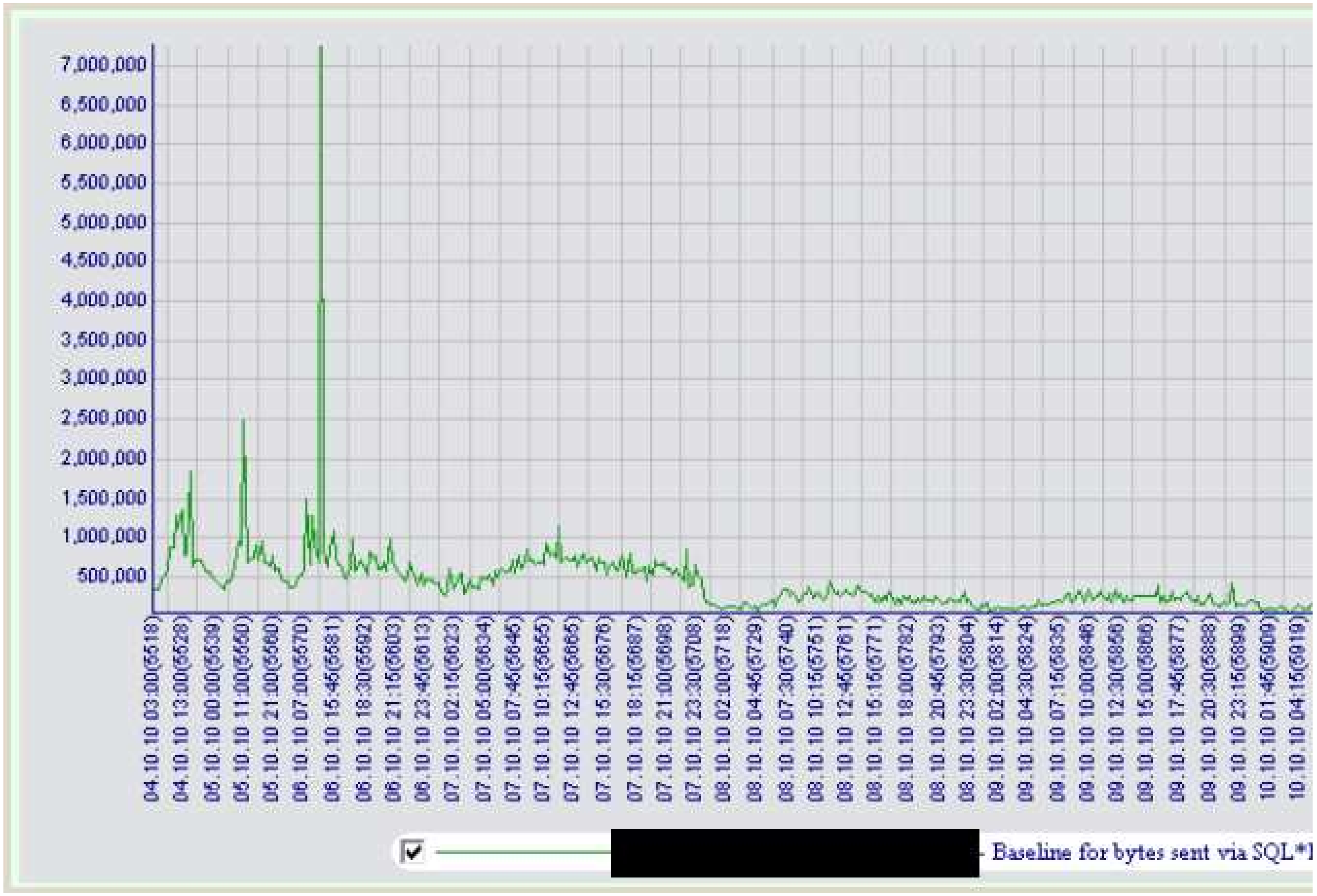
Figure (4)

5. **Verify this association by application developers.**
Upon sharing the above sql_id with the application developers, we received the following reply (an excerpt from the actual email) ...

" ... Bingo! That's the query that my patch addresses. ..."

Resolution

- Eventually, an application patch was made and issue resolved. The issue resulted mainly from an application event that triggered all application servers to request a cache update simultaneously. This flooded the network bandwidth and chocked the application servers causing them to eventually crash.
- Graph below shows reduction of sqlnet traffic after applying patch on 10/7


7,000,000
6,500,000
6,000,000
5,500,000
5,000,000
4,500,000
4,000,000
3,500,000
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
04.10.10 03:00(5518)
04.10.10 13:00(5528)
05.10.10 00:00(5539)
05.10.10 11:00(5550)
05.10.10 21:00(5560)
06.10.10 07:00(5570)
06.10.10 15:45(5581)
06.10.10 18:30(5592)
06.10.10 21:15(5603)
06.10.10 23:45(5613)
07.10.10 02:15(5623)
07.10.10 05:00(5634)
07.10.10 07:45(5645)
07.10.10 10:15(5655)
07.10.10 12:45(5665)
07.10.10 15:30(5676)
07.10.10 18:15(5687)
07.10.10 21:00(5698)
07.10.10 23:30(5708)
08.10.10 02:00(5718)
08.10.10 04:45(5729)
08.10.10 07:30(5740)
08.10.10 10:15(5751)
08.10.10 12:45(5761)
08.10.10 15:15(5771)
08.10.10 18:00(5782)
08.10.10 20:45(5793)
08.10.10 23:30(5804)
09.10.10 02:00(5814)
09.10.10 04:30(5824)
09.10.10 07:15(5835)
09.10.10 10:00(5846)
09.10.10 12:30(5856)
09.10.10 15:00(5866)
09.10.10 17:45(5877)
09.10.10 20:30(5888)
09.10.10 23:15(5899)
10.10.10 01:45(5909)
10.10.10 04:15(5919)
✓ [redacted] - Baseline for bytes sent via SQL*