


# 从一个故障案例谈数据库运维中的数字化分析之路



 白鳢的洞穴

 2024-11-18

 566

想实现数字化诊断分析或者大家喜欢说的智能化诊断其实并不容易，早期的AIOPS从业者总是和客户说他们可以不需要数据库专家就能通过算法实现智能化运维，大概十年过去了，这个梦似乎快破灭了，专家知识依然是智能化运维中的关键。随着大语言模型的兴起，最近我总是听人说生成式AI可以解决以前AIOPS没有实现的一切，真的如此吗？

前几天我说过一个比较复杂的案例，如果用AIOPS来分析难度很大。今天我们通过复盘这个案例来看一看，如果一个故障能够实现全数字化的分析，自动化的诊断需要如何来完成？

DBA朋友大概对于复杂场景故障分析都有些心得，实现方法不外乎大胆推测，细心求证。大胆推测要基于丰富的知识和实践经验；而细心求证则要补全这些推断中间的各个环节，除了需要扎实的基本功之外，还需要利用可采集到的各种数据，包括指标，日志，TRACE等。通过这些已知的数据或者证据来进行推理和分析。

在专家的帮助下，这些分析工作可以根据专家所掌握的技巧与方法进行，但是如果我们要通过完全自动化的方式来进行分析，该怎么做呢？

实际上专家如果想要完成故障分析，也必须依赖能够支撑完成分析所需要的所有数据或者证据。如果哪一个环节上的数据或者证据缺失，则需要依靠专家的推理能力来补全证据链条，或者根据推断再去补全证据数据。

而如果要实现自动化的诊断分析，则对数据的要求会更高一些。如果中间缺少明显的证据，那么推理可能无法顺利进行，因为算法无法像人一样利用历史经验和在自己脑子中的知识去进行比较跳跃的推理。

首先我们来回顾那个案例，用户在11月1日发现业务系统出现了数分钟的卡顿，导致了生产系统告警。通过AWR报告发现等待事件是library cache相关的。

## Top 10 Foreground Events by Total Wait Time



Event	Waits	Total Wait Time (sec)	Wait Avg(ms)	% DB time	Wait Class
library cache: mutex X	168,838	1172.7K	6946	88.5	Concurrency
latch: cache buffers chains	10,196	85.4K	8376	6.4	Concurrency
DB CPU		74K		5.6	
log file sync	2,939,522	8850.3	3	.7	Commit
cursor: pin S wait on X	934	8764.6	9384	.7	Concurrency
latch: row cache objects	18,020	592.2	33	.0	Concurrency
buffer busy waits	42,674	474.6	11	.0	Concurrency
SQL*Net message from dblink	332,924	240.4	1	.0	Network
enq: TX - row lock contention	1,213	184.9	152	.0	Application
latch: shared pool	3,484	128.5	37	.0	Concurrency

对于这个等待事件大家可能都不陌生。MOS上有一篇文章Troubleshooting 'library cache: mutex X' Waits. (Doc ID 1357946.1) 十分详尽地举出了一些引发此类等待的主要场景，大家根据这个文档一般就能够找到问题的根因。

'library cache : mutex X' 的等待类似于早期版本中的库缓存等待。'library cache : mutex X' 可能是由许多问题引起的（包括应用程序问题、SQL缺乏共享导致高版本数等），但本质上是某些东西持有互斥锁“太久”以至于其他会话必须等待资源。如果保护库缓存结构的 latches/mutexs 存在争用，这意味着解析系统存在压力。解析 SQL 需要更长的时间，因为它无法获取所需的资源。这会延迟其他操作，并且通常会减慢系统速度。引发此类等待的主要原因包括：

- 1) 频繁的死解析 - 如果死解析的频率非常高，则此引脚上可能会发生争用。
- 2) 高版本计数 - 当版本计数变得过多时，需要检查一长串版本，这可能会导致对此事件的争用
- 3) Invalidations （失效） - 失效是衡量缓存的游标因不再有效而从缓存中删除的次数的指标。游标失效，因为某些内容已更改，因此内存中的游标副本不再有效。例如，重新收集有关对象的统计信息或修改表定义足以使基于该对象的查询的游标失效。当游标失效时，任何想要使用该游标的会话都需要等待加载有效版本。如果存在过多或不必要的失效，则可以看到对 'library cache : mutex X' 的大量等待。
- 4) Reloads - Reload 是以前存在于缓存中的游标被搜索、发现不存在（因为它已经过期等）然后必须重新编译并重新加载到库缓存中的次数。很高的RRELOAD是很危险的，因为它们表明您正在执行的工作，如果缓存设置得当，则不必首先删除光标。如果正在重新加载游标，则session无法抓取它以进行工作，这可能导致等待 'library cache : mutex X'。



- 5) 已知 Bug

## Report Summary

### Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	734.3	0.4	0.01	0.01
DB CPU(s):	41.1	0.0	0.00	0.00
Redo size (bytes):	5,443,440.1	3,257.0		
Logical read (blocks):	5,606,305.2	3,354.5		
Block changes:	26,897.7	16.1		
Physical read (blocks):	263.5	0.2		
Physical write (blocks):	1,682.2	1.0		
Read IO requests:	115.0	0.1		
Write IO requests:	798.4	0.5		
Read IO (MB):	2.1	0.0		
Write IO (MB):	13.1	0.0		
User calls:	85,753.7	51.3		
Parses (SQL):	29,126.8	17.4		
Hard parses (SQL):	5.9	0.0		
SQL Work Area (MB):	117.5	0.1		
Logons:	6.8	0.0		
Executes (SQL):	52,709.7	31.5		
Rollbacks:	10.6	0.0		
Transactions:	1,671.3			

公众号 · baishan755

前面两个问题很容易从AWR报告中被排除掉，系统的解析数量很高，不过硬解析不多。第五个问题就比较复杂了，因为已知BUG相当多。第三第四个原因在AWR报告中也是可以看到一些端倪的。

## Library Cache Activity

- "Pct Misses" should be very low

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Invali- dations
ACCOUNT_STATUS	35,652	0.03	0		0	0
APP CONTEXT	1	0.00	224	0.45	1	0
BODY	17,131	0.16	13,772,052	0.00	42	0
CLUSTER	136	2.21	136	2.21	0	0
DBLINK	654,488	0.00	0		0	0
EDITION	12,246	0.02	24,164	0.02	0	0
HINTSET OBJECT	172	15.70	172	31.40	0	0
INDEX	36,529,795	-0.00	111,495	0.22	230	0
OBJECT ID	30	100.00	0		0	0
QUEUE	48	0.00	5,571	0.05	2	0
RULESET	0		4	0.00	0	0
SCHEMA	12,982	0.02	0		0	0
SQL AREA	1,093,798	13.17	114,634,713	-0.08	8,763	4,738
SQL AREA BUILD	10,360	27.53	0		0	0
SQL AREA STATS	7,648	36.87	7,648	36.89	1	0
SUBSCRIPTION	5	0.00	5	0.00	0	0



TABLE/PROCEDURE	96,228,726	4463.29	100,542,822	-0.01	1,857	0
TRIGGER	5,219	0.02	662,904	0.00	12	0

公众号 · baishan755

Library Cache Activity章节中我们看到了SQL AREA存在大量的Reload和Invalidations。因此很明确就是大量的CURSOR失效导致了此问题的发生。

### SQL ordered by Parse Calls

- Total Parse Calls: 52,540,305
- Captured SQL account for 44.4% of Total

Parse Calls	Executions	% Total Parses	SQL Id	SQL Module	SQL Text
2,031,532	0	3.87	<a href="#">c2jjk8s3umdbk</a>	VEY_DpbgSFC.exe	SELECT PROPERTY_01 CATEGORY_K...
1,423,918	1,423,921	2.71	<a href="#">fmgux4h671y27</a>	JDBC Thin Client	SELECT /*+ index(R_WIP IDX_R_...
1,301,204	1,287,162	2.48	<a href="#">8ys4g9a377qr2</a>	VEY_DpbgSFC.exe	SELECT nvl(a.Check_out_time, ...
1,243,488	0	2.37	<a href="#">ffndajp7dp3wk</a>	VEY_DpbgSFC.exe	SELECT CATEGORY_KEY FROM DMPD...
1,177,749	1,177,737	2.24	<a href="#">dpztdwbcc34b0</a>	JDBC Thin Client	SELECT /*+ index(R_WIP IDX_R_...
615,498	613,561	1.17	<a href="#">7yva69p1ttsvt</a>	VEY_DpbgSFC.exe	SELECT ID, Is_Closed , no FRO...
615,319	613,560	1.17	<a href="#">4rg1pg3qs8zh8</a>	VEY_DpbgSFC.exe	SELECT ID, CATEGORY_KEY FROM ...
604,614	823,525	1.15	<a href="#">4jghn3r77ftz8</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
604,479	822,776	1.15	<a href="#">athjgs18ku3xh</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
604,412	822,865	1.15	<a href="#">33n4f7qx0fajk</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
604,192	822,450	1.15	<a href="#">bz8b5c87f93ds</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
604,039	822,310	1.15	<a href="#">8nkftgpu5v1va</a>	JDBC Thin Client	SELECT NVL(MIN(KEY_VALUE), "...
603,993	822,707	1.15	<a href="#">fpqbgp90mxb</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
603,868	822,293	1.15	<a href="#">31fzrwk8sscmx</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
603,729	822,763	1.15	<a href="#">akts11f5kjg3z</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
603,605	821,713	1.15	<a href="#">17q73vgqh0czw</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
603,427	822,500	1.15	<a href="#">g05jqx3xr8y5w</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
603,335	821,893	1.15	<a href="#">6cwnmf9jagn</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
603,322	822,425	1.15	<a href="#">73kw8xrk2t6fc</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
602,958	821,447	1.15	<a href="#">a9y04kr6j2dwn</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
602,841	821,908	1.15	<a href="#">2dy8mjgmh0frp</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
602,648	821,057	1.15	<a href="#">c16hkbdm5xs4n</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
602,561	820,133	1.15	<a href="#">4pqq9ktm5quzw</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
602,479	821,684	1.15	<a href="#">61jqjmn49n7x</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
602,464	821,062	1.15	<a href="#">fhk5s1n1kr44s</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
602,263	820,162	1.15	<a href="#">fn4ap1mf5pwpz</a>	VEY_DpbgSFC.exe	SELECT NVL(MIN(KEY_VALUE), "...
576,419	571,627	1.10	<a href="#">062yxwb3fwg2h</a>	VEY_DpbgSFC.exe	SELECT /*+ index(x.r_wip_part...
536,907	528,002	1.02	<a href="#">dvxgt2sj0xxw9</a>	VEY_DpbgSFC.exe	SELECT CATEGORY_KEY FROM DMPD...
532,290	532,274	1.01	<a href="#">b04rhcvkrjyvc</a>	JDBC Thin Client	BEGIN dmpdb2.BOB_QUERY_SFC(1,...

第二个问题我们一般可以通过检查解析较高和执行较多的SQL来进行分析定位。在这里我们遇到了一个十分容易引起误导的问题，就是排在前面的SQL执行数量为0，但是解析数量很高，这大概率是应用程序存在BUG导致的，比如有一条SQL写错了，解析失败，因此每次解析后都没有执行。



通过比对以前的AWR报告，客户确认这个问题一直存在，不过以前为什么不出问题呢？这个问题也许很容易用“压垮骆驼的最后一根稻草”来搪塞，不过事实真的如此吗？

根据我对Oracle共享池的理解，很可能是某个DDL操作或者其他什么特殊的操作引发了大量的RELOAD产生。如果负载并无太大变化，那么SGA RESIZE等也可能引发类似问题。

## Memory Dynamic Components

- Min/Max sizes since instance startup
- Oper Types/Modes: INITializing, GROw, SHRink, STAtic/IMMediate, DEFerred
- ordered by Component

Component	Begin Snap Size (Mb)	Current Size (Mb)	Min Size (Mb)	Max Size (Mb)	Oper Count	Last Op Typ/Mod
ASM Buffer Cache	0.00	0.00	0.00	0.00	0	STA/
DEFAULT 16K buffer cache	0.00	0.00	0.00	0.00	0	STA/
DEFAULT 2K buffer cache	0.00	0.00	0.00	0.00	0	STA/
DEFAULT 32K buffer cache	0.00	0.00	0.00	0.00	0	STA/
DEFAULT 4K buffer cache	0.00	0.00	0.00	0.00	0	STA/
DEFAULT 8K buffer cache	0.00	0.00	0.00	0.00	0	STA/
DEFAULT buffer cache	320,320.00	320,064.00	320,000.00	322,816.00	4	SHR/IMM
KEEP buffer cache	0.00	0.00	0.00	0.00	0	STA/
PGA Target	61,440.00	61,440.00	61,440.00	61,440.00	0	STA/
RECYCLE buffer cache	0.00	0.00	0.00	0.00	0	STA/
SGA Target	334,848.00	334,848.00	334,848.00	334,848.00	0	STA/
Shared IO Pool	0.00	0.00	0.00	0.00	0	STA/
java pool	448.00	448.00	448.00	448.00	0	STA/
large pool	512.00	512.00	512.00	3,328.00	0	SHR/DEF
shared pool	11,584.00	11,840.00	9,216.00	11,840.00	4	GRO/IMM
streams pool	128.00	128.00	0.00	128.00	0	GRO/IMM

第二个可能干扰本次故障分析的问题产生了，AWR报告中真的看到了存在SGA RESIZE现象。不过幸运的是实例并未重启过，因此我们可以观察SGA RESIZE事件产生的时间，与ASH中的数据进行比对，找出先后顺序，从而排除掉这个可能性。经过分析，我们发现SGA RESIZE滞后于Library Cache发生问题。因此SGA RESIZE是果不是因。

零点刚过就发生问题，不大可能是与人为操作有关，那么就需要去检查定时任务了。同时需要检查的是10月1号和9月1号是否有类似问题发生。数据库的数据很难找到了，不过业务系统日志还是可供分析的。通过分析，发现每个月1号凌晨都会有类似情况存在，不过时间并不固定，但是有一个特点，都是0点过后的几十秒到一两分钟内。

通过ASH分析发现相关的等待与某张分区表相关，这张分区表是月分区的。而故障发生的时段与这个分区写入第一条记录的时间完全吻合。那么问题就很清晰了，分区表的段延时分配是引发本故障的主要原因。因为段延时分配导致了相关的Invalidations，从而引发了业务卡顿。如果要避免这些问题，那么提前对分区表做空间分配就可以了。



通过上面的分析，大家大致了解了本故障的發生的原因和規避方法了。但是此類問題如何能夠通過自動化分析的手段來提前發現或者自動化診斷呢？

运维对象: orcl1212    指标ID:     指标名称: mutex    指标分类: --请选择--

指标项	对象名称	指标名称	值	
1	2184123	orcl1212	library cache: mutex X 总等待时间	21057.0
2	2184223	orcl1212	library cache: mutex X 总等待次数	1262.0
3	2184323	orcl1212	library cache mutex X 平均等待时间	0.0

首先我們整個分析和預警所需的数据都需要指标化。library cache : mute X的平均等待次数和平均等待事件要作为指标存在在系统中。一旦这些指标出现异常，则立即发起告警。这里首先需要解决一个问题，那就是什么叫“异常”？简单地通过阈值设置超过某个数就是异常是不合适的，因为不同的系统，这个值很难设定好。最好通过AI算法来判别异常。

其次是如果发现了异常，要进行自动分析，如何去发现可能的路径？Oracle的官方文档虽然提供了一些分析路径，但是并不完整，比如说分区表的延时段创建的问题，在此文档中就没有提及，因此这些分析路径还是要依靠专家经验和用户自己的积累。

第三个问题是如何在众多路径中找到 正确的一条，也就是如何排除不可能的路径？这一点要自动化也十分困难，本案例中的执行数为0的SQL问题是很难被自动排除的，依赖于用户对系统的历史的了解，才很快排除掉了。如果想要自动完成，则需要对历史数据进行采集和自动特征提取才行，要实现这一点并不容易。

通过ASH数据我们可以发现产生问题的表是哪一张，也可能发现不了，因为ASH数据里依然有大量的干扰因素。如果我们能够找到哪张表引发了故障，那么我们需要去分析表的元数据。如果找准了某张表，而且我们拥有这个运维经验，那么自动化定位问题也不算太难了。

大家可以看到，要想实现自动化分析，运维经验与数据是关键，而要采集到那么丰富的数据，远不是传统的监控系统能够完成的，数据的采集是目标导向的，是基于运维知识的，这样才能够在对系统影响最小的情况下完成所需数据的采集。


在推理的全过程中，完全依靠传统的方法也十分困难，大模型应该是补全推理过程中缺失能力的关键，不过通识大模型不够用，必须有高质量的专业大模型才可以。专业大模型如何训练？其中所需的十分专业的知识，恐怕也只有专家 才能提供。问题又回来了，干技术的活，离开了专家，真的不行。

 [大数据](#) [分区表](#) [游标](#)

文章转载自白鳍的洞穴，如果涉嫌侵权，请发送邮件至：contact@modb.pro进行举报，并提供相关证据，一经查实，墨天轮将立刻删除相关内容。


## 评论

分享你的看法，一起交流吧～

- 


星星之火 5月前

打败拖延最好的方法，就是遇事提前规划

评论 0
- 

筱悦星辰 7月前

打败拖延最好的方法，就是遇事提前规划。

评论 0
- 

舒悦 8月前

从一个故障案例谈数据库运维中的数字化分析之路

评论 0

## 相关阅读

优炫数据库在山东省寿光市人民检察院成功应用！

中国信通院2025上半年“可信数据库”新增标准解读

希望和国产数据库厂商共建公共知识库

客户说 | 古茗选用阿里云PolarDB，以云端之力解锁茶饮数字化新高度

GBASE南大通用亮相2025 CHITEC，当数据库“豫”见医疗信息化

“融合进化 智领未来” 电科金仓2025产品发布会成功举办！

携手Zilliz构建向量数据库，赋能企业AI应用创新！

【新模型速递】PAI-Model Gallery云上一键部署Qwen3-Coder模型

【重磅发布】天玑数据PDCI数据库专有云平台：打造“一云多芯、一池多库”的信创新标杆



中国移动磐维数据库亮相TDBC 2025 引领可信数据基础设施新格局