DTCC

2016中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2016

数 据 定 义 未 来

SequeMedia 盛拓传媒 IT168.com ChinaUnix ITPUB

基于云的系统性能诊断和优化

鲍倚天@恩核

目录

1 性能管理及优化服务现存问题

2 基于云的系统性能管理和优化服务技术架构

3 基于云的性能优化和诊断方法

4 实施案例

系统性能管理及优化服务现存问题

IT大环境日趋恶化、人力成本与日俱增、出差费用居高不下、工程师培养难度加大等各种原因，致使线下性能管理和优化服务变得越来越难，客户和IT厂商都处于两难境地。

1

- 人力成本与日俱增，自2013年~2016年3年期间，性能优化工程师的成本至少增长2倍以上。但客户提供的项目费用不但没有增加，反倒降低了将近1/5左右。(经统计和参考同行业相关数据：中级性能优化工程师人均成本为6.3万)

2

- IT环境变化，2014年后半年至现在，互联网+，OTO，移动应用等炙热的市场需求，大量消耗了IT工程师，致使无人安心从事传统IT技术，人员稳定性无法保障。

3

- 性能管理和优化工作通常需要通过出差的形式完成，现在很多工程师不愿意长期出差，致使项目实施和服务难度加大，再加上出差费用，项目成本加大。

4

- 工程师培养周期长，根据经验一个可以独立实施项目的工程师培养周期为1年以上，目前项目型的现状，致使人员复用性非常低。

改善方案

服务云化

服务工具化

服务流程化

提升人员复用性

系统性能管理及优化服务现存问题

针对目前性能管理及优化服务现存的众多问题，必须采取相应且有效的措施予以解决。目前采用的方案为充分利用云服务、发挥工具的作用、采用基于中心化的服务模式等，经过实践具有不错效果。

改善目标

1	人力成本与日俱增，利润无法保障	• 研发功能齐全，自动化程度高的软件工具，以减少人力投入
2	工程师不稳定，人员流动快	• 以较少的人员，较高的薪酬，持续性的技术培训来保障人员的稳定性
3	项目出差频率高，周期长	• 基于云技术和工具化的方式，实现性能管理和优化目标，减少出差概率
4	工程师培养周期长，人员复用性差，效果过渡依赖人的能力	• 通过软件工具和详细化的流程化工作，减少对人员能力的依赖

改善方案

1 云技术	+	2 专业工具	+	3 中心化服务
• 采用云服务 • 使用远程连接技术 • 性能信息远程传输		• 性能管理工具 • 性能监控工具 • 性能优化工具		• 远程性能监控和服务支持 • 周期性性能信息搜集和诊断 • 基于工具实现优化服务

目录

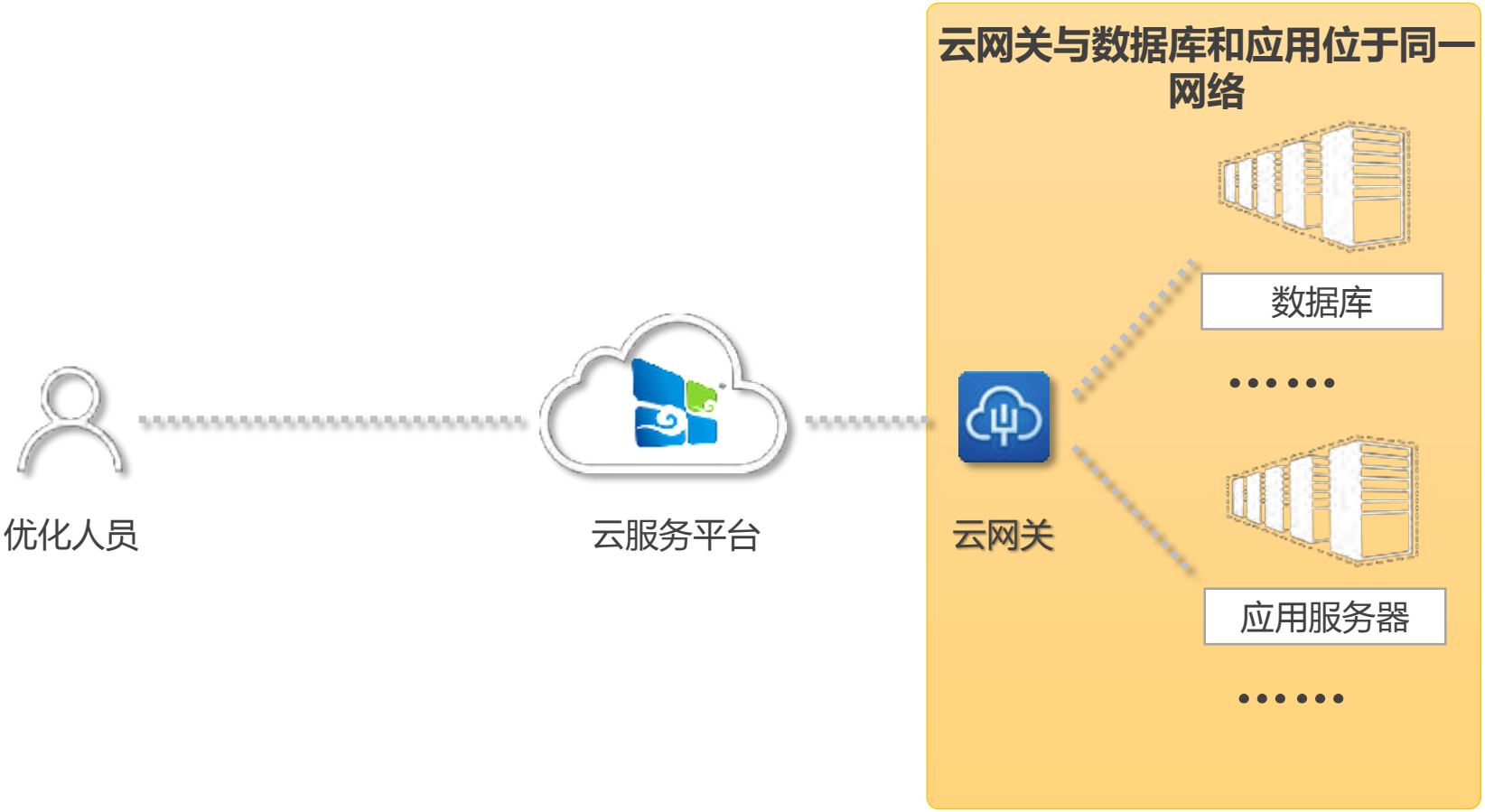
1 性能管理及优化服务现存问题

2 基于云的系统性能管理和优化服务技术架构

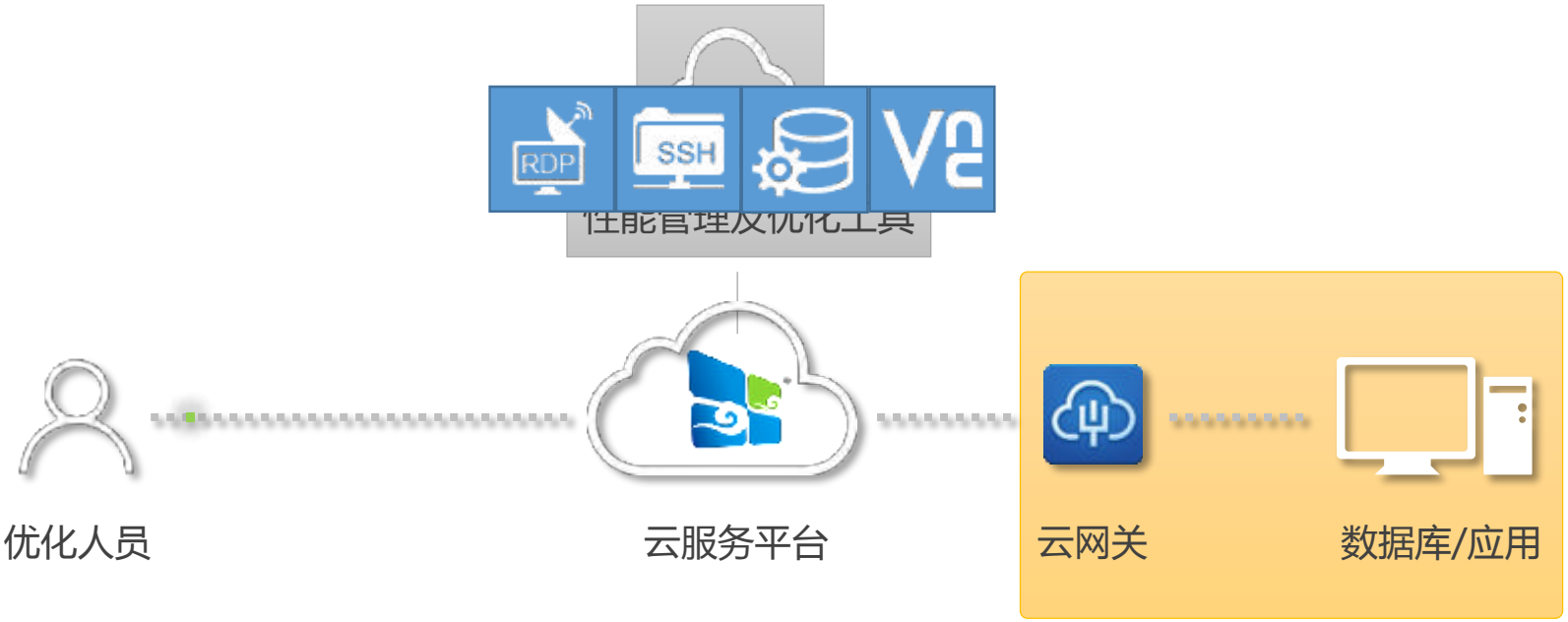
3 基于云的性能优化和诊断方法

4 实施案例

基于云的系统性能管理和服务技术架构

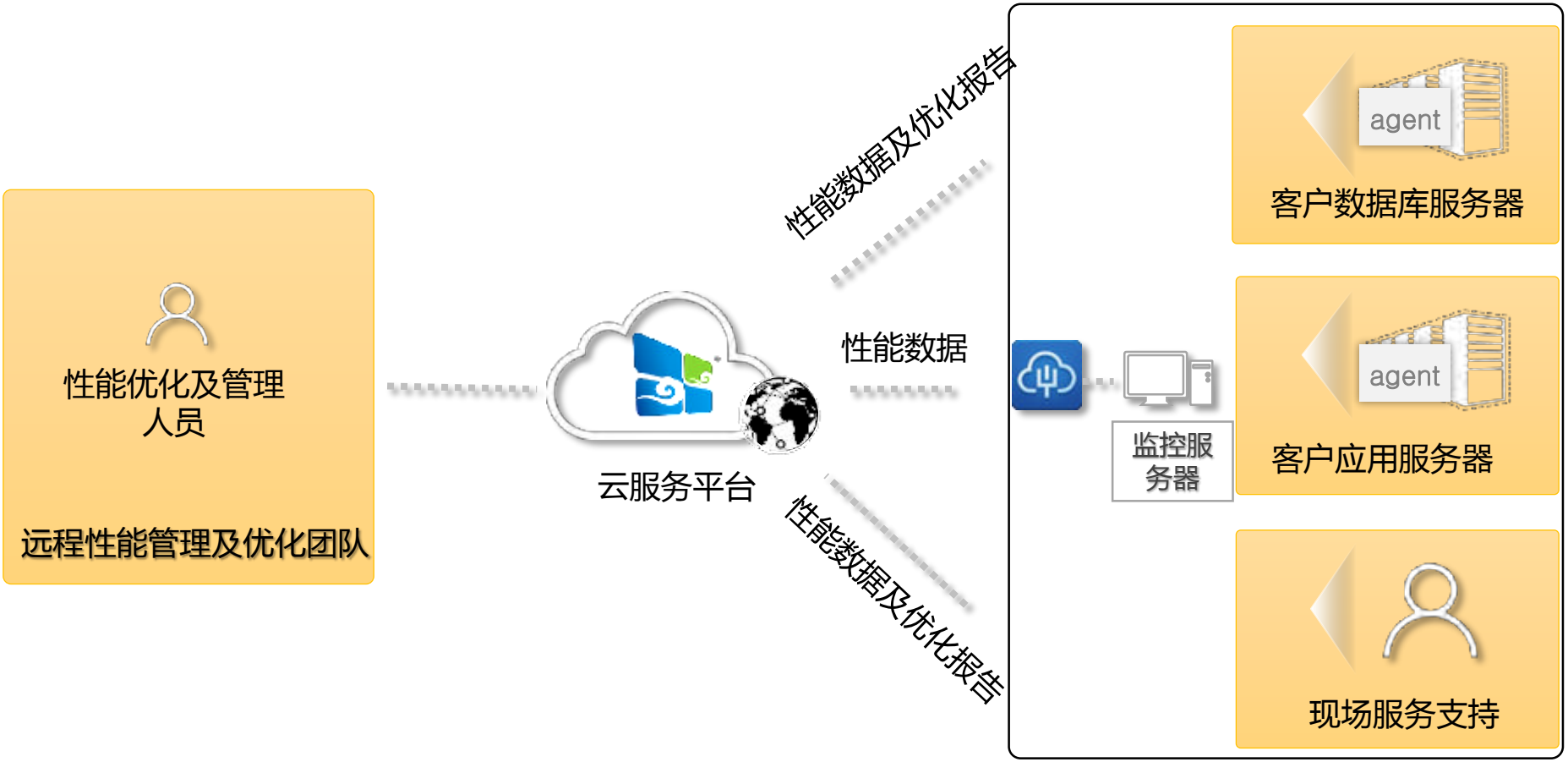


基于云的系统性能管理和服务技术架构



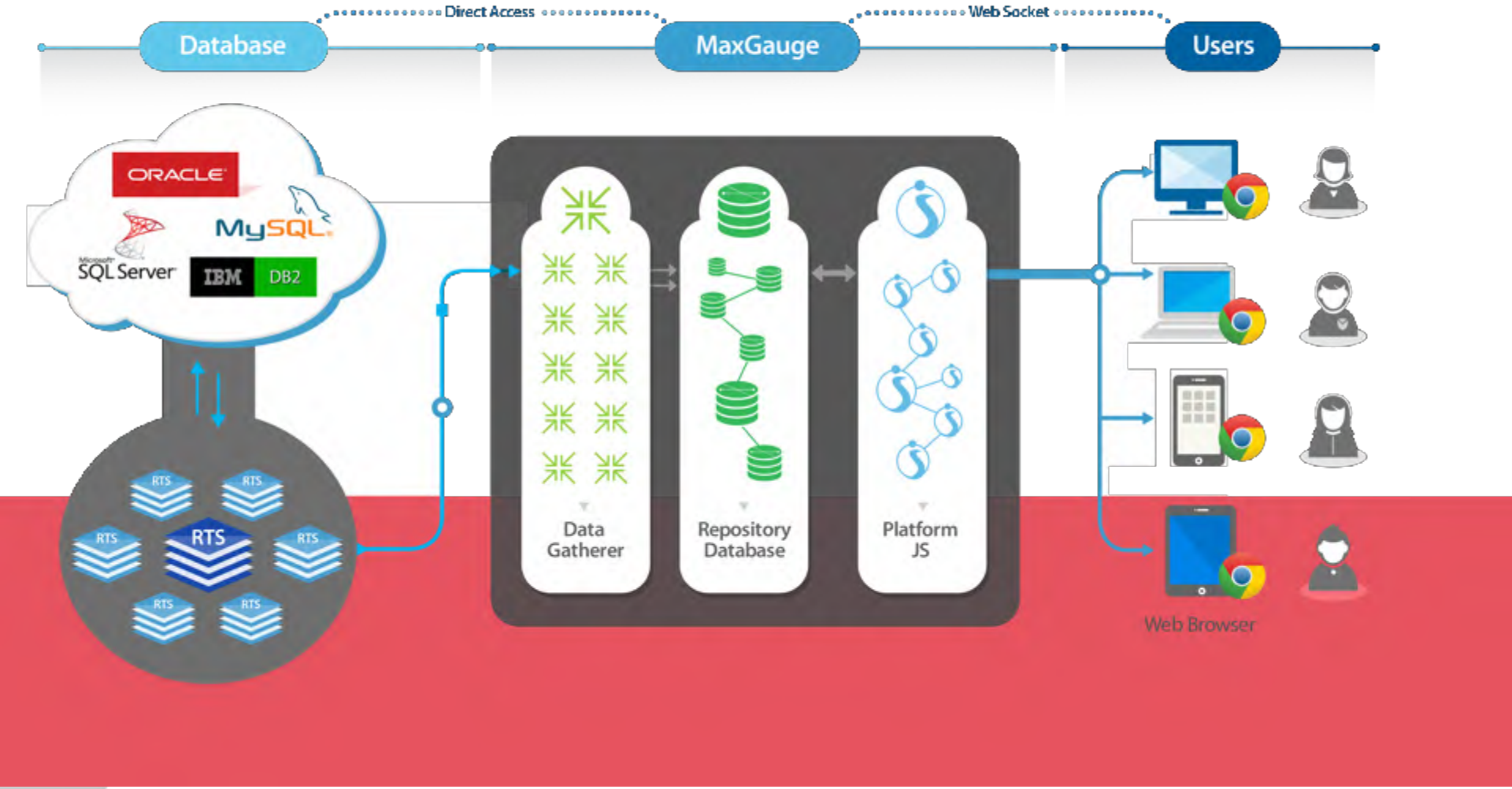
基于应用虚拟化技术，可将性能管理和性能优化工具软件发布到云中，以便优化专家在性能优化和管理过程中使用。

基于云的系统性能管理和服务技术架构



基于云的系统性能管理和服务技术架构

将性能管理或性能监控软件的AGENT部署在对象服务器上，AGENT搜集到的监控信息，可以存储在云服务平台中的知识库中，工程师通过访问知识库实现性能管理和优化。



基于云的系统性能管理和服务技术架构

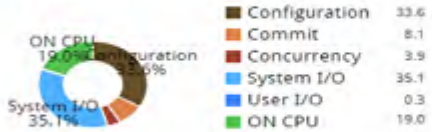
通过工具等连接数据库搜集各种性能指标

通过直接访问SGA方式搜集各种性能指标、SQL语句、其执行计划，及日志信息。

基于云的系统性能管理和服务技术架构

1 Minute Analysis

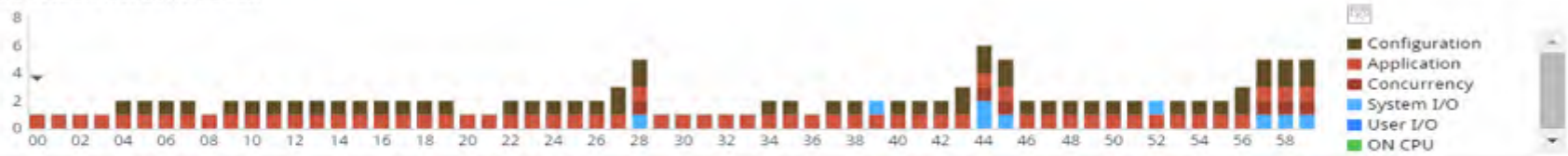
Wait Class (%)



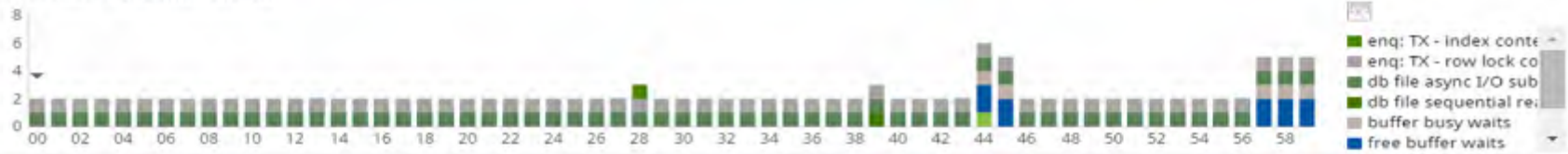
Event Name	Time (%)	Total Time (Sec)	Value/Sec	Total Waits	Wait Time/Waits (ms)	Wait Class
free buffer waits	33.6%	58	0.97	196	296.0	Configuration
db file async I/O submit	30.7%	53	0.88	11	4,823.6	System I/O
CPU Time	19.0%	32	0.55			
log file sync	8.1%	13	0.23	301	46.4	Commit
log file parallel write	3.4%	5	0.10	248	23.4	System I/O

18:42:00

Active Session Trend (Wait Class)



Active Session Trend (Wait Events)



SID	Program	Module	User Name	SPID	CPU (%)	Elapsed Time (Sec)	Event Name	Wait Time	Logical Reads/Sec (blocks)	Physica (b
384	oracle@localhost.lo...			14702	0	1,008,569	db file async I/O submit	WAITING (0)	0	
585	LitePlus.exe	summary_3	MAXGAUGE	31684	7	273,849	db file sequential read	WAITED KNOW...	1,240,017	
388	oracle@localhost.lo...	PQ_TEST	MAXGAUGE	8357	0	191,583	PX Deq Credit: send blkd	WAITING (0)	0	
772	oracle@localhost.lo...	PQ_TEST	MAXGAUGE	8359	0	191,583	PX Deq Credit: send blkd	WAITING (0)	0	

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 SUM

可以秒单位实时搜集和分析性能状况

00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 SUM

基于云的系统性能管理和服务技术架构

◘基于云的系统性能管理和服务技术架构的优点

✓云服务技术成熟

✓工具化和自动化程度高

✓提高人员复用率

✓大幅度降低项目投入和客户自身的成本

◘基于云的系统性能管理和服务技术架构的局限

✓在对于安全性较高的环境中不适用

目录

1 性能管理及优化服务现存问题

2 基于云的系统性能管理和优化服务技术架构

3 基于云的性能优化和诊断方法

4 实施案例

基于云的索引优化设计及应用步骤

1) 全面搜集表的读取类型（基于云平台上的性能数据搜集SQL语句并解析）

2) 选定索引列对象和统计分析其数据分布

3) 统计列使用频次及运算符

4) 组合索引列及列顺序的选择

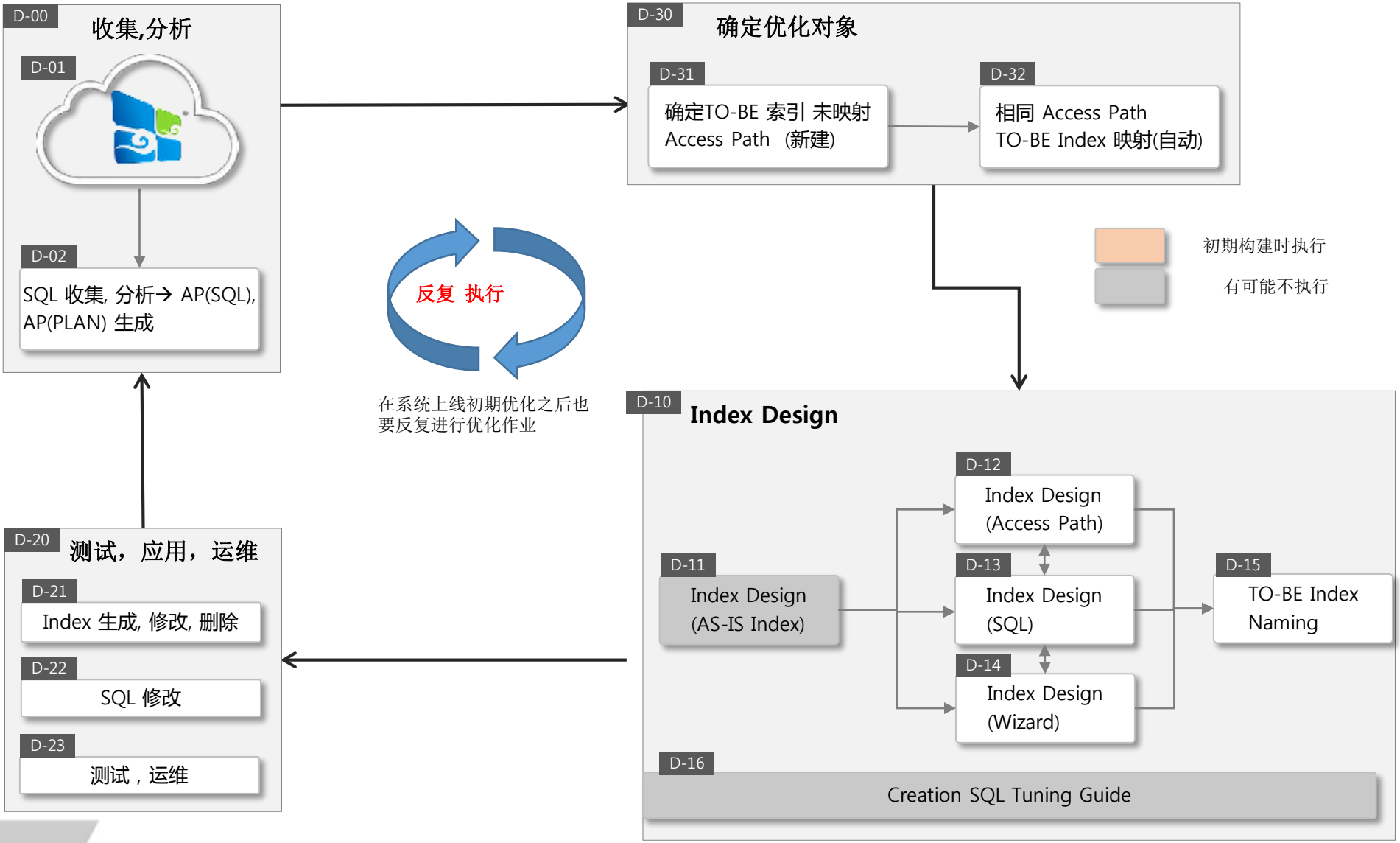
5) 模拟和测试

6) 调整部分SQL语句

7) 索引部署及监控

基于云的索引优化设计

AS-is to-be 索引比较



基于云的索引优化设计

TABLE NAME	SALES	销售信息管理表	TOTAL ROWS	201,500	平均月增加量	5,000

★No	Access Pattern	Count	Range	Old	New		(1)	(2)	(3)	(4)	(5)	(6)
★1	SALENO(=), ITEM(=)	9	1	1	1	现行索引	SALENO	SALEDATE	STATUS	SALEDEPT	CUSTNO	ITEM
★2	SALEDATE(=), SALEDEPT(=)	3	180	2	2		ITEM	SALEDEPT			SALEDATE	
★3	SALEDATE(like), SALEDEPT(=)	5	3000	4	2			SALETYPE				
★4	CUSTNO(=), SALEDATE(between)	3	300	5	3							
★5	SALEDATE(like), STATUS(=60), CUSTNO(like)	2	1200	3	4							
★6	STATUS(in), [AGENTNO(like)]	4	400	3	4							
★7	ITEM(=), SALEDATE(like), SALEDEPT(like)	4	1400	2	5	新建索引	(1)	(2)	(3)	(4)	(5)	(6)
★8	SALEDATE(like), STATUS(=), group by CUSTNO	3	1000	3	4		SALENO	SALEDEPT	CUSTNO			
★9	SALEDEPT(=), SALEDATE(like), SALETYPE(=),	4	3000	4	2		ITEM	SALEDATE				
	order by SALEDATE, SALETYPE							(CLUSTER)				
★10	SALEDATE(between), CUSTNO(=),	3	300	5	3							
	(STATUS(=) or SALETYPE(=))											
★11	AGENTNO(=), SALEDATE(between), ITEM(like)	4	800		6							

主要列的分布度

列名	种类	平均	最大	备注事项
Saleno	20,000	11	100	
Saledate	1,500	130	800	月平均(5,000行)，月末集中
Saledept, saledate	11,000	20	180	1个月平均(500)，使用期间为1年
Status	25	8,000	56000	60,90时为90%，其它：平均为300
Custno	3200	63	300	
agentno	250	1,000	4,500	

注意事项

第6个读取类型中的STATUS为60,90时能够向部分范围处理方式引导。

第2,9个读取类型中查询范围限制在12个月以内。

Sample SQL

```
SELECT COUNT( * )
FROM T_SHOP_INOUT X,
        T_SHOP_INOUT_DETAIL Y,
        T_ITEM A
WHERE X.INOUT_NO = 'N1267578'          --- (1)
  AND X.INOUT_NO = Y.INOUT_NO          --- (2)
  AND Y.ITEM_ID = A.ITEM_ID            --- (3)
ORDER BY Y.PACKING_UNIT_NO, Y.SERIAL_NO
```

自动分析

Access Path (SQL)

T_SHOP_INOUT

: **INOUT_NO (=J), INOUT_NO(=)** ← **(1),(2)**

INOUT_NO列用于常数运算以及用于条件连接运算，此为组合 Access Path.

T_SHOP_INOUT_DETAIL

: **INOUT_NO (=J), ITEM_ID(=J)** ← **(2),(3)**

INOUT_NO, ITEM_ID列应用于条件连接的Access Path.

T_ITEM

: **ITEM_ID (=J)** ← **(3)**

Index Design

TO-BE Index Design

T_SHOP_INOUT : INOUT_NO 单独作为常量条件应用的情况

➔ **INOUT_NO**

T_SHOP_INOUT_DETAIL : INOUT_NO, ITEM_ID是作为连接条件加以应用的情况. 此种情况应尽量从 T_SHOP_INOUT 表中，以常量获得 INOUT_NO 条件.

➔ **INOUT_NO**

T_ITEM : 从T_SHOP_INOUT_DETAIL表中，获得 ITEM_ID的取值.

➔ **ITEM_ID**

基于云的索引优化设计

CUST_NO	366,723	364	1	平均 40行

CUST_NO, ARR_DT(B)
CUST_NO, PRICE_STD
PRICE_STD, BNK_CD
BNK_CD, SND_CD, ARR_DT(L)
PRICE_STD, ARR_DT(B)
CRD_NO, ACT_NO, PRICE_STD
CRD_NO, ARR_DT(L)
PRICE_STD, BNK_CD(IN), ARR_DT(L)
PRICE_STD[IN], BNK_CD(IN)
BNK_CD, ARR_DT(B)
SND_CD, BNK_CD(IN), ARR_DT(B)

CUST_NO

CUST_NO, ARR_DT(B)
CUST_NO, PRICE_STD

CUST_NO + PRICE_STD

PRICE_STD+ BNK_CD+ ARR_DT

~~CUST_NO, PRICE_STD~~
PRICE_STD, BNK_CD
PRICE_STD, ARR_DT(B)
~~CRD_NO, ACT_NO, PRICE_STD~~
PRICE_STD, BNK_CD(IN), ARR_DT(L)
PRICE_STD[IN], BNK_CD(IN)

PRICE_STD, BNK_CD使用频率高

~~PRICE_STD, BNK_CD~~
BNK_CD, SND_CD, ARR_DT(L)
~~PRICE_STD, BNK_CD(IN), ARR_DT(L)~~
~~PRICE_STD(IN), BNK_CD(IN)~~
BNK_CD, ARR_DT(B)
SND_CD, BNK_CD(IN), ARR_DT(B)

BNK_CD+ ARR_DT+ SND_CD

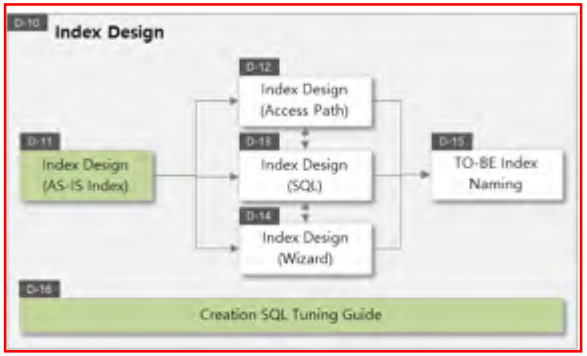
CRD_NO

CRD_NO, ACT_NO, PRICE_STD
CRD_NO, ARR_DT(L)

CRD_NO

CRD_NO	402,849	320	1	平均 36行

基于云的索引优化设计



D-11 Index Design(AS-IS Index) 1

比较现有 AsIs-Index和收集的 AP(SQL)之间的契合度，对于契合度比较高的索引尽量复用。对于契合度比较低的索引，需要设计新索引。

D-12 Index Design(Access Path) 2

对收集的SQL中提取到的 AP(SQL)正确地加以分组（Grouping）并进行 ToBe-Index设计的过程.

对索引设计技术需能够娴熟加以运用.

相比于在查验 SQL执行情况的同时进行索引设计，应倾向于仅基于AP(SQL)进行索引设计.

D-14 Index Design(Wizard) 2

将收集的 AP(SQL)按照定制的规则自动经过多次处理完成分组研判，并根据结果构建ToBe-Index的过程.

对索引设计知识没有很深积淀也能通过工具进行设计.

并非全部处理过程都采用自动方式，应按阶段适用自动处理，而部分过程应采用手动方式。

D-13 Index Design(SQL) 3

按表分别对收集的 SQL一一加以确认，并进行ToBe-Index设计的过程.

不是一开始就采用此方法，而是在上面的 Index Design(Access Path) 或者 Index Design(Wizard) 方式构建索引之后，再针对性地对SQL进行细致的分析之后，根据情况按需进行索引设计

D-15 ToBe-Index Naming 4

对上面这些构建阶段设计出的ToBe-Index加以命名，并赋予各种选项.

最终生成新增，修改这些索引的DDL,并将新的索引在工具上模拟到对应的AP(SQL)上

D-16 Creation SQL Tuning Guide

在设计ToBe-Index的过程中对各SQL标注改进事项，并据此编写Tuning Guide.

1. 尽快研判AsIs Index 中能够加以复用的部分.(Optional)
2. 手动对AP进行分组研判，并进行索引设计(Index Design(Access Path))或通过向导进行索引设计.
3. 对ToBe Index设计的各个研判项目，依次查看SQL并加以验证.（Optional）

基于云的索引优化设计

TAB1
Total rows : 340,988

AsIs Index
TAB1_idx1 : col3

TAB1
Total rows : 340,988

AsIs Index
TAB1_idx1 : col1
TAB1-idx2 : col3, col2

反复施行

优化状态

- 周期性地反复收集和分析
- 维护好日常的优化状态
- 定期进行优化前，后性能比较
- 恶性查询的检索条件上报

D-00 **搜集，解析**

D-10 **Index Design**

D-20 **测试及应用**

SQL#1 (2014-4-1)
SELECT COL1, COL2
FROM TAB1
WHERE COL1 = '23'

SQL#2 (2014-4-1)
SELECT COL1, COL2
FROM TAB1
WHERE COL2 = '23'
AND COL3 = 'dkhd'

SQL#3 (2014-4-16)
SELECT COL1, COL2
FROM TAB1
WHERE COL5 = '2014'
AND COL3 = 'dkhd'

AP#1
COL1(=)

AP#2
COL2(=), COL3(=)

AP#3
COL3(=), COL5(=)

ToBe Index
TAB1_idx1 : COL1

ToBe Index
TAB1_idx2 : COL3, COL2

ToBe Index
TAB1_idx3 : COL5, COL3

Index 创建脚本自动生成 (自动)

DBA 执行

索引构建前后性能比较，评测

优化对象，新增构建项目的确认和处置

对未映射 AP重点加以管理可以将优化状态进行常态化运维

始终确认 AsIs Index – ToBe Index是否一致，并维护其一致性

对于不使用的 SQL,AP需要周期性地进行检索以及清除

基于云的索引优化设计

Index Design Using Access

D-12

功能概要

- 按表，基于收集、分析的 AP(SQL)设计 ToBe-Index的功能.
- 此过程不是对个别SQL进行详细分析并进行索引设计，而是主要基于 AP(SQL)将主要列分组之后进行索引设计.
- 在设计过程中为了应用从DB收集的统计信息 (Cardinality 等)，可以直接对主要列进行分布度调查.

1. 按表分别进行收集、分析的 Access Path
2. 收集，分析的 AP 中使用次数最多并且以 '='运算应用的条件列需要提取出来，此步骤可以针对性进行检索
3. 单击检索按钮就会显示如图所示的窗口，显示使用的运算符以及列及其适应频次
4. 从列出的检索条件中选择排在前列的列字段并将其选定为 ToBe Index对象列.
5. 对选定为Tobe Index对象列的条件，检索与其关联的 AP.此时检索可以指定 'AND' 或 'OR'运算进行检索.

基于云的索引优化设计

Index Design Using Access

D-12

功能概要

- 按表，基于收集、分析的 AP(SQL)进行 ToBe-Index设计.
- 此过程不是对个别SQL进行详细分析并进行索引设计，而是主要基于 AP(SQL)将主要列分组之后进行索引设计.
- 在设计过程中为了应用从DB收集的统计信息 (Cardinality 等)，可以直接对主要列进行分布度调查.

6 对于选定的ToBe Index候补列将生成索引.

7 生成索引，索引将自动生成，且索引名将按照'Option'中指定的规则自动命名

8 将映射与 生成的ToBe Index一致的AP.

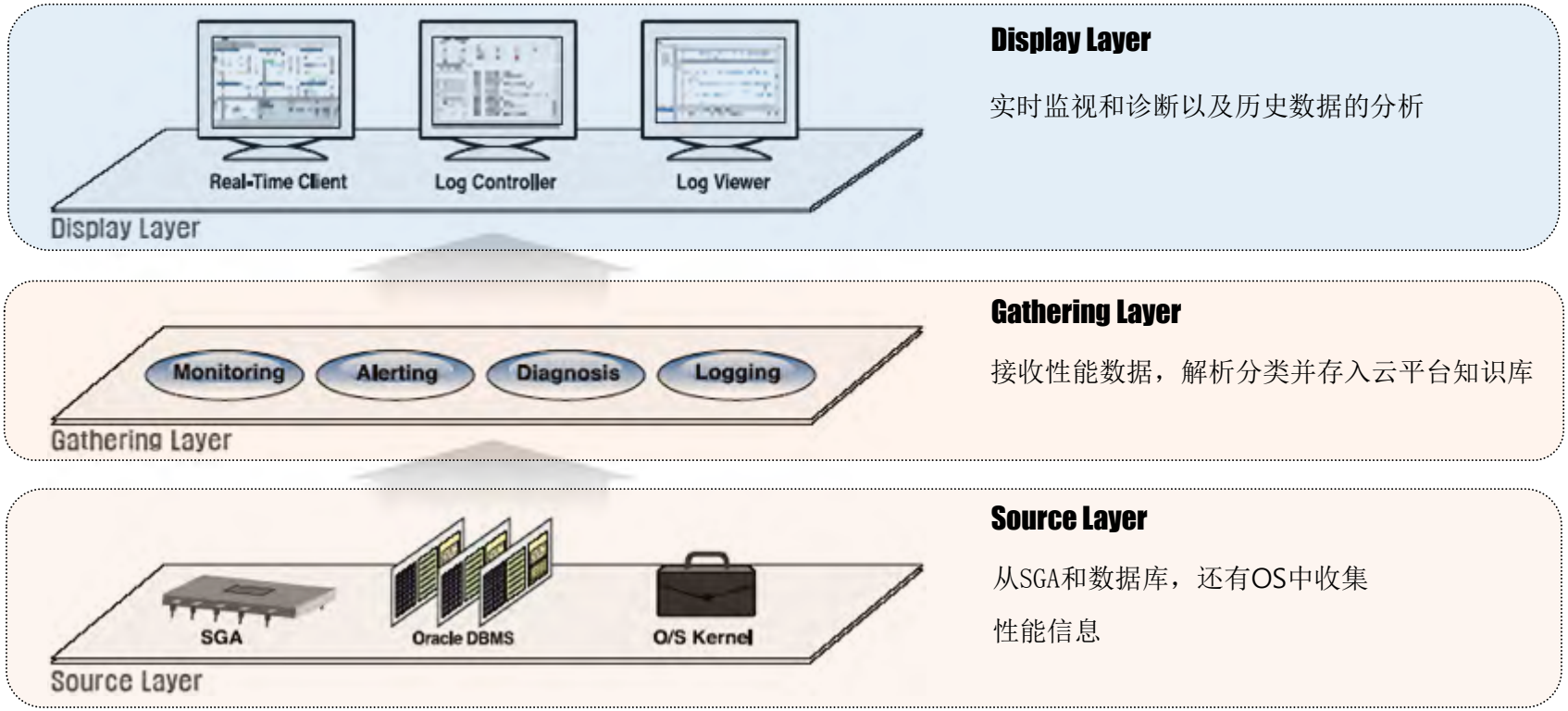
9 执行映射之后就会产生如图所示的AP-ToBe Index 间的映射关系.

10 通过 1~9步的一系列操作，对那些与 Tobe Index尚未映射的余下的AP进行同样的分析处理，并进行 ToBe Index设计以及完成映射操作

基于云的性能监控和诊断

性能监控体系结构

直接读取ORACLE的SGA信息，即便在数据库挂起的时候，也可以采集各种性能数据。



Display Layer

实时监视和诊断以及历史数据的分析

Gathering Layer

接收性能数据，解析分类并存入云平台知识库

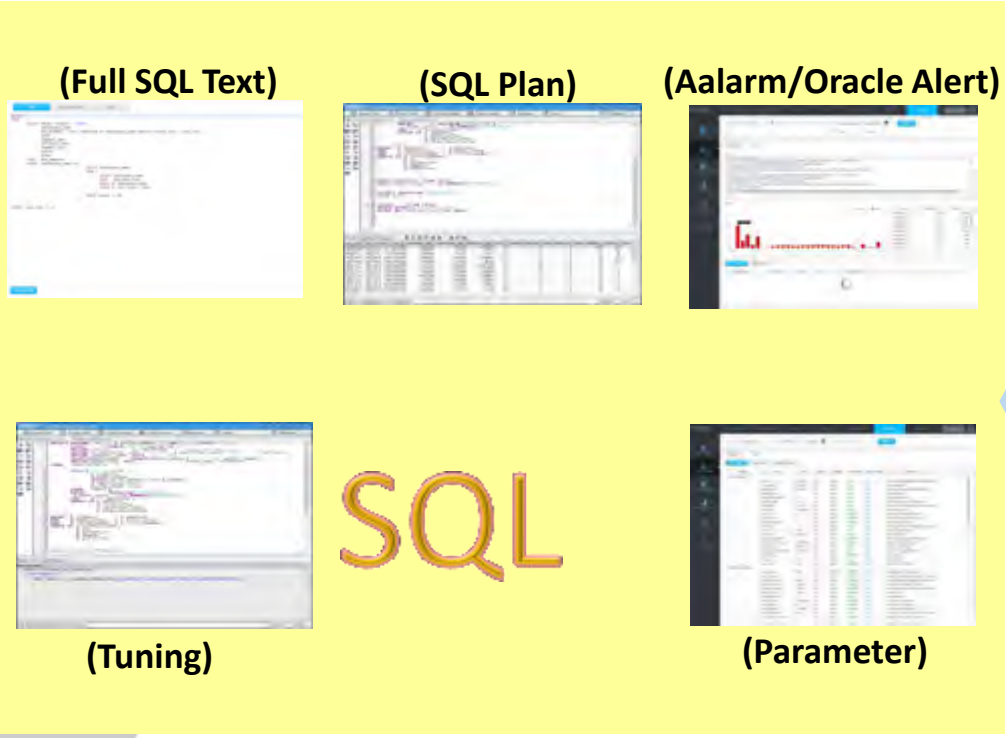
Source Layer

从SGA和数据库，还有OS中收集性能信息

基于云的性能监控和诊断

Top Down诊断方式

(Performance Trend) (Bottle-neck)



(Full SQL Text) (SQL Plan) (Aalarm/Oracle Alert)

(Tuning) (Parameter)

(Top SQL) (Top Event)

(Session Detail) (Sql Detail)

(stat/wait) (Session List)

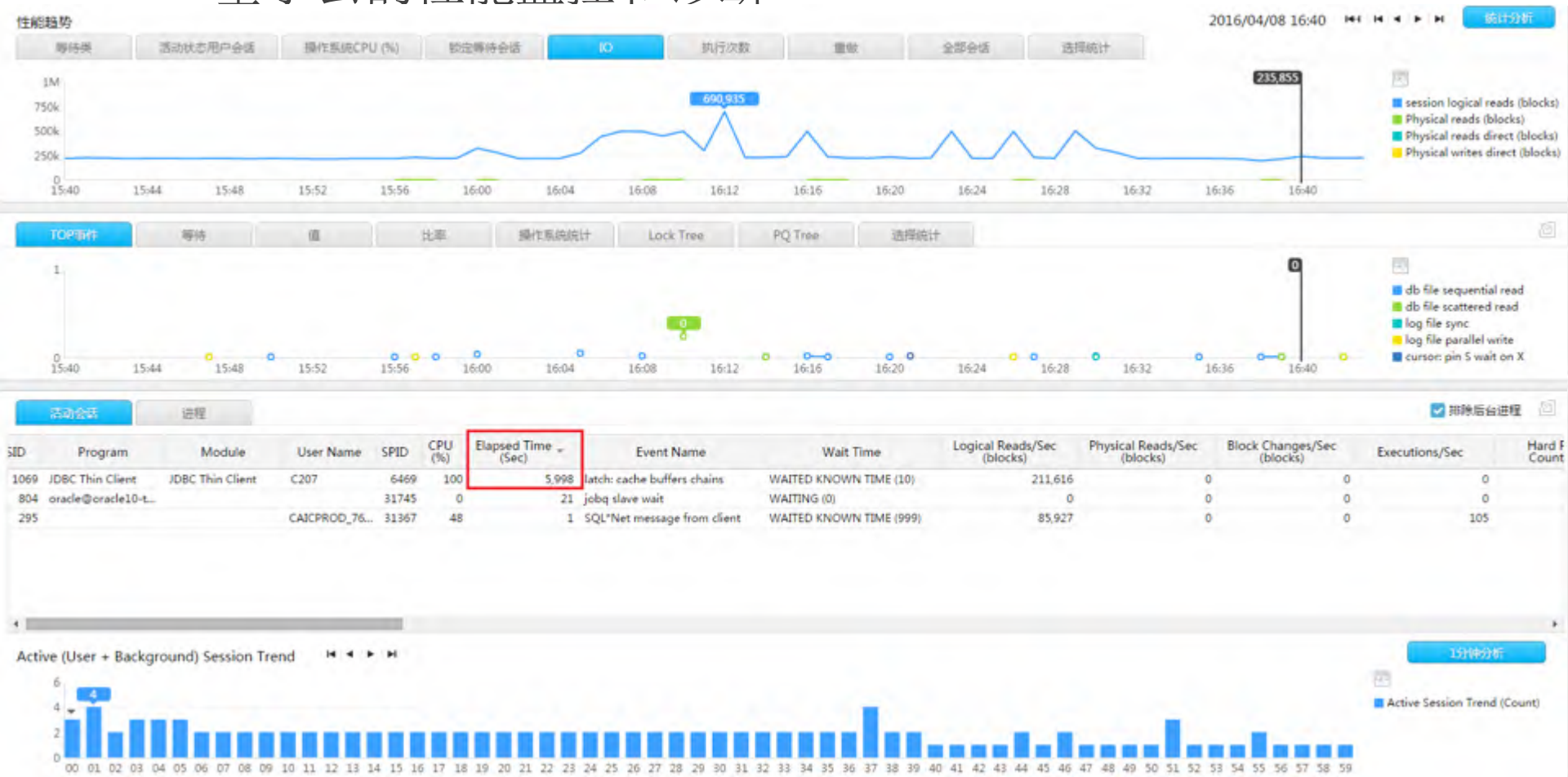
基于云的性能监控和诊断

session logical reads (blocks)

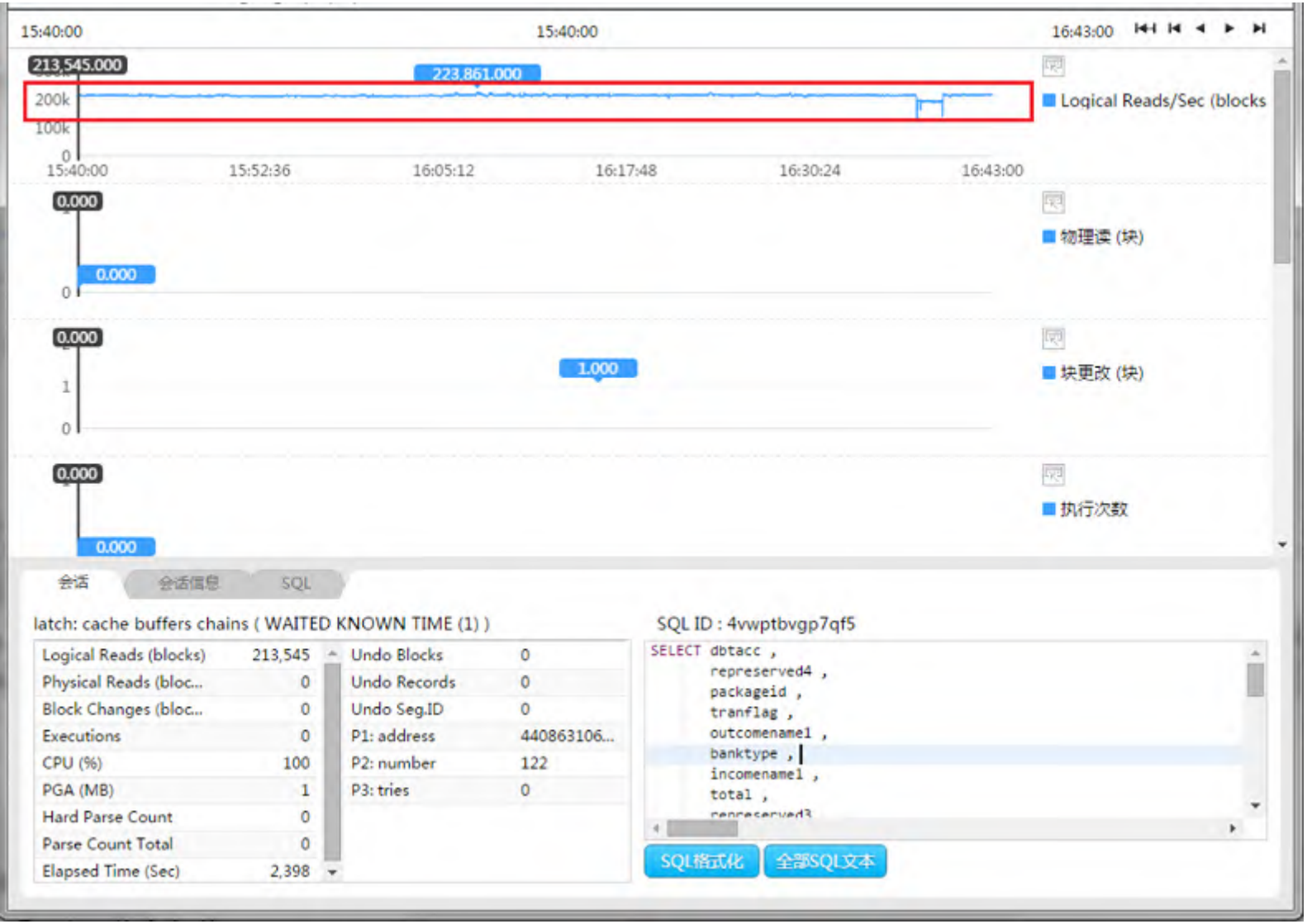
[2016/04/09] 24小时趋势

现象描述：从04-06到04-28号一直发生周期性的性能波动，逻辑读一天累计的值能达到200GB左右，通过观察一天的性能趋势，发现在一天内，也同样存在逻辑读的性能波动情况

基于云的性能监控和诊断



通过截取从15：40-16：40这一个小时的性能数据观察发现，会话1069一直处于活动状态，在一小时的时间内的每秒产生的逻辑读是20几万,等待事件为lacth:cache buffer chains



通过进一步的下钻，查看会话1069一小时的趋势可以看出，该会话中的逻辑读一直在持续

基于云的性能监控和诊断

SQL详细信息 [ORACLE10DB] SQL ID: 4vwptbvgp7qf5

一天趋势 [04/09 00:00~23:59] 2016/04/09 16:00:00

执行时间趋势 | IO 趋势 | 等待时间趋势

769,077,027

Plan Hash Value	Executions	Elapsed Time/exec (Sec)
1233049816	1	13,707.550

一小时趋势 [16:00~16:59]

执行时间趋势 | IO 趋势 | 等待时间趋势

128,207,523

SQL | 执行计划 | 变量

```
        buscod ,
        dbtbankname ,
        crdttyp
FROM    ebank_dfdk_log_h
WHERE   pk_id IN (
                SELECT DISTINCT pk_h_parent
                FROM   ebank_dfdk_log_r
                WHERE (( yurref IN ( '1070A
                AND    serial = 0
        )
```

SQL格式化 | 全部SQL文本

1天 | 1小时

Event Name	Time (%)	Time (Sec)	Time/exec (Sec)	Wa
CPU Time	100.0	3,600.4	3,600.4	

Time	Schema	Program	Module	SQL Plan Hash	Elapsed Time/exec (Sec)	CPU Time/exec (Sec)
16:00:00	C207	JDBC Thin Client	JDBC Thin Client	1233049816	600.100	600.100
16:10:00	C207	JDBC Thin Client	JDBC Thin Client	1233049816	600.050	600.050
16:20:00	C207	JDBC Thin Client	JDBC Thin Client	1233049816	600.100	600.100
16:30:00	C207	JDBC Thin Client	JDBC Thin Client	1233049816	600.050	600.050
16:40:00	C207	JDBC Thin Client	JDBC Thin Client	1233049816	600.050	600.050
						3,600.400

最终关联到SQL有性能问题的SQL语句

SQL和执行计划

SQL | 执行计划 | 变量

```
SELECT dbta ,
       represerve ,
       package ,
       tranfg ,
       outcome1 ,
       banktype ,
       incomename1 ,
       total ,
       represerved3 ,
       epttim ,
       reqreserved1 ,
       parentid ,
       eptdat ,
       bankarea ,
       dbtaccbkn ,
       serial ,
       banktypecode ,
       memo ,
       billcode ,
       c_ccynbr ,
       reqreserved2 ,
       pk_id ,
       summny ,
       bankuniqueno ,
       buscod ,
       dbtbank ,
       crd
FROM   ebank_log_h
WHERE  pk_id IN (
                 SELECT DISTINCT pk_parent
                 FROM   ebank_log_r
                 WHERE (( yurref IN ( '1070A21M4' , '1070A21M5' ) ) )
                 AND    serial = 0
       )
```

```
id   Plan Hash Value : 1233049816
0      SELECT STATEMENT
1      FILTER
2        TABLE ACCESS (FULL) C207.EBANK_LOG_H' (TABLE)
3        FILTER
4          TABLE ACCESS (FULL) C207.EBANK_LOG_R' (TABLE)
     ---------------------------------------------------------------------------
     Predicate Information (identified by operation id)
     ---------------------------------------------------------------------------
     1 - filter( IS NOT NULL)
     3 - filter(:B1=0)
     4 - filter(("PK_PARENT"=:B1 AND INTERNAL_FUNCTION("YURREF")))
```

SQL格式化

基于云的性能监控和诊断

```
SELECT dbta ,
    represerve ,
    package ,
    tranfg ,
    outcome1 ,
    type ,
    ……………….
    dbtbank ,
    crd
FROM  ebank_log_h — 572255行
WHERE  pk_id IN (
        SELECT DISTINCT pk_parent
        FROM   ebank_log_r   —157529行
        WHERE (( yurref IN ( '1070A21M4' , '1070A21M5' ) ) )
        AND    serial = 0
    )
```

```
-------------------------------------------------------------------------
| Id  | Operation            | Name         | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              |     1 |   211 |   585M  (1)|
|*  1 |  FILTER              |              |       |       |            |
|   2 |   TABLE ACCESS FULL  | EBANK_LOG_H  |  156K |   31M|  1112   (1)|
|*  3 |   FILTER             |              |       |       |            |
|*  4 |    TABLE ACCESS FULL| EBANK_LOG_R  |     1 |    42 |  3806   (1)|
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter( EXISTS (SELECT /*+ */ 0 FROM "NC207"."EBANK_LOG_R"
              "EBANK_LOG_R" WHERE :B1=0 AND "PK_PARENT"=:B2 AND
              ("YURREF"='1070A2100000000854M4' OR
"YURREF"='1070A2100000000854M5')))
   3 - filter(:B1=0)
   4 - filter("PK_PARENT"=:B1 AND ("YURREF"='1070A21M4' OR
              "YURREF"='1070A21M5'))
```

条件serial=0在表EBANK_LOG_R的过滤条件中并不存在

(TABLE) C207.EBANK_LOG_H

Columns | Indexes | Data | Scripts | Partitions | Constraints | Privileges | Triggers | Referential | Dependency(Used by) | Comment | General

Pos	Column	PK	Ref	Type Owner	Type	Null?	Default	Comments	Histogram	Num Distin
19	PARENTID				VARCHAR2 (20 Byte)				None	757
20	PK_ID	1			CHAR (20 Byte)	N			Height Balanced	1565
21	REPRESERVED3				VARCHAR2 (100 Byte)				None	
22	REPRESERVED4				VARCHAR2 (100 Byte)				None	
23	REQRESERVED1				VARCHAR2 (100 Byte)				None	
24	REQRESERVED2				VARCHAR2 (100 Byte)				None	
25	SERIAL				NUMBER (38)				Frequency	
26	SUMMNY				NUMBER (30,8)	N			Height Balanced	57
27	TOTAL				NUMBER (38)	N			None	
28	TRANFLAG				NUMBER (38)				None	
29	TS				CHAR (19 Byte)		to_char(sysdate,'yyyy-mm-dd hh24:mi:ss')		Height Balanced	755

反复扫描的行数：
572255*157529= 901,4675,7895

基于云的性能监控和诊断

6000多秒

```
SELECT dbta ,
    represerve ,
    package ,
    tranfg ,
    outcome1 ,
    type ,
    ……………….
    dbtbank ,
    crd
FROM  ebank_log_h
WHERE  pk_id IN (
        SELECT DISTINCT pk_parent
        FROM   ebank_log_r
        WHERE (( yurref IN ( '1070A21M4' ,
'1070A21M5' ) ) )
        AND    serial = 0
    )
```

SQL改写

2秒

```
SELECT dbta ,
    represerve ,
    package ,
    tranfg ,
    outcome1 ,
    type ,
    ……………….
    dbtbank ,
    crd
FROM  ebank_log_h
WHERE  pk_id IN (
        SELECT DISTINCT pk_parent
        FROM   ebank_log_r
        WHERE (( yurref IN ( '1070A21M4' ,
'1070A21M5' ) ) )
)
AND    serial = 0
```

```
---------------------------------------------------------------------------
| Id  | Operation            | Name         | Rows  | Bytes | Cost (%CPU)|
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              |     1 |   211 |   585M  (1)|
|*  1 |  FILTER              |              |       |       |            |
|   2 |   TABLE ACCESS FULL  | EBANK_LOG_H  |  156K|    31M|  1112   (1)|
|*  3 |   FILTER             |              |       |       |            |
|*  4 |    TABLE ACCESS FULL| EBANK_LOG_R  |     1 |    42 |  3806   (1)|
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter( EXISTS (SELECT /*+ */ 0 FROM "NC207"."EBANK_LOG_R"
              "EBANK_LOG_R" WHERE :B1=0 AND "PK_PARENT"=:B2 AND
              ("YURREF"='1070A2100000000854M4' OR
"YURREF"='1070A2100000000854M5')))
   3 - filter(:B1=0)
   4 - filter("PK_PARENT"=:B1 AND ("YURREF"='1070A21M4' OR
              "YURREF"='1070A21M5'))
```

```
---------------------------------------------------------------------------
| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%CPU)|
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     6 |  1518 |  3815   (1)|
|   1 |  NESTED LOOPS                |              |     6 |  1518 |  3815   (1)|
|   2 |   SORT UNIQUE                |              |     6 |   252 |  3808   (1)|
|*  3 |    TABLE ACCESS FULL         | EBANK_LOG_R  |     6 |   252 |  3808   (1)|
|*  4 |   TABLE ACCESS BY INDEX ROWID| EBANK_LOG_H  |     1 |   211 |     2   (0)|
|*  5 |    INDEX UNIQUE SCAN         | PK_E_LOG_H   |     1 |       |     1   (0)|
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("YURREF"='1070A21M4' OR "YURREF"='1070A21M5')
   4 - filter("SERIAL"=0)
   5 - access("PK_ID"="PK_PARENT")
```

基于云的性能优化和诊断的价值

监控自动化	▪自动搜集性能数据 ▪周期记录重要历史信息 ▪ TOWN-DOWN方式快速定位问题，效率提升50%以上
优化半自动化	▪可以由工具代替50%人力劳动 ▪通过工具辅助进行优化工作
服务中心化	▪降低成本 ▪提升人员复用率

目录

1 性能管理及优化服务现存问题

2 基于云的系统性能管理和优化服务技术架构

3 基于云的性能优化和诊断方法

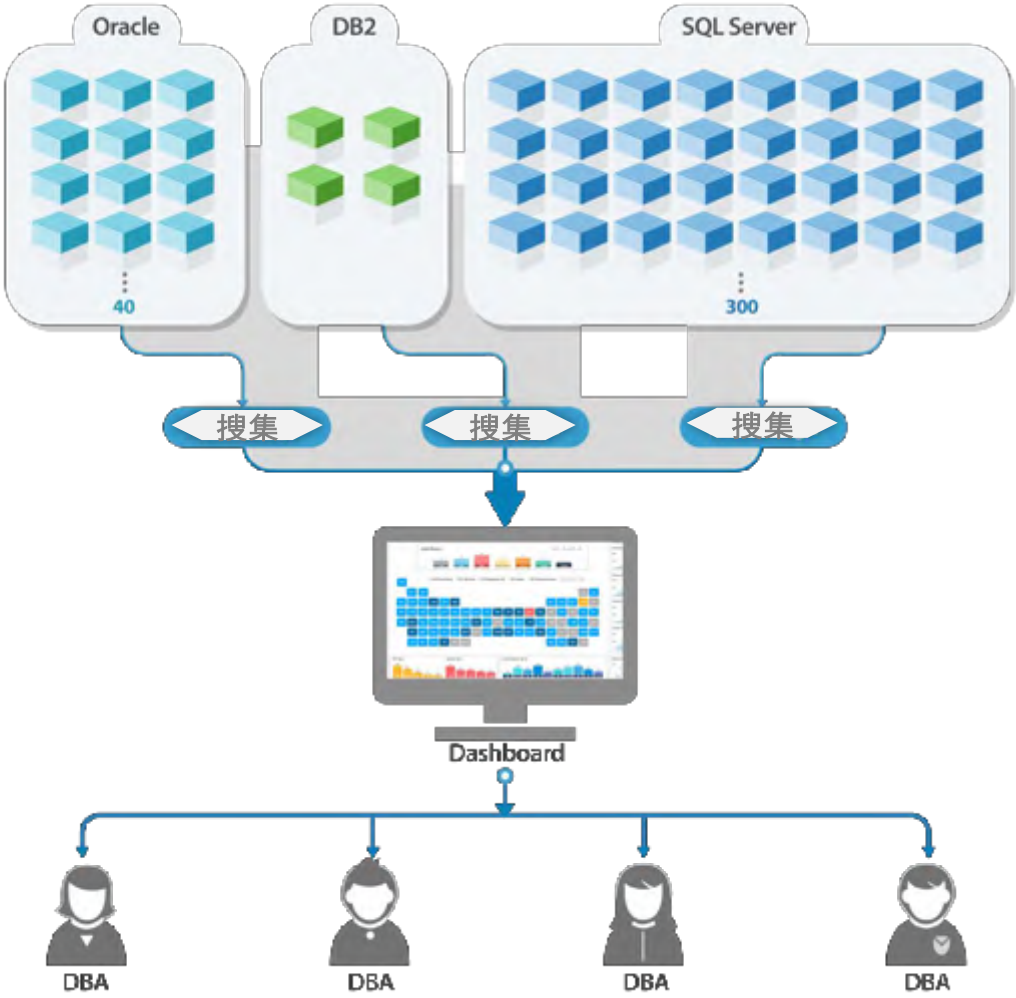
4 实施案例

某金融行业客户案例

场景

实时搜集对象数据库（ORACLE，DB2，SQL SERVER）。

- 将性能信息发送至服务器短，并存储在服务器的知识库中。
- DBA，性能优化人员，实时查看系统性能问题，并及时了解系统瓶颈。
- 长时间跟踪高消耗SQL语句和SESSION，并形成分析报告，以便准确定位性能瓶颈。
- 长期累积SQL语句和执行计划，并将其解析为读取路径，以便设计战略索引。



某金融行业客户案例

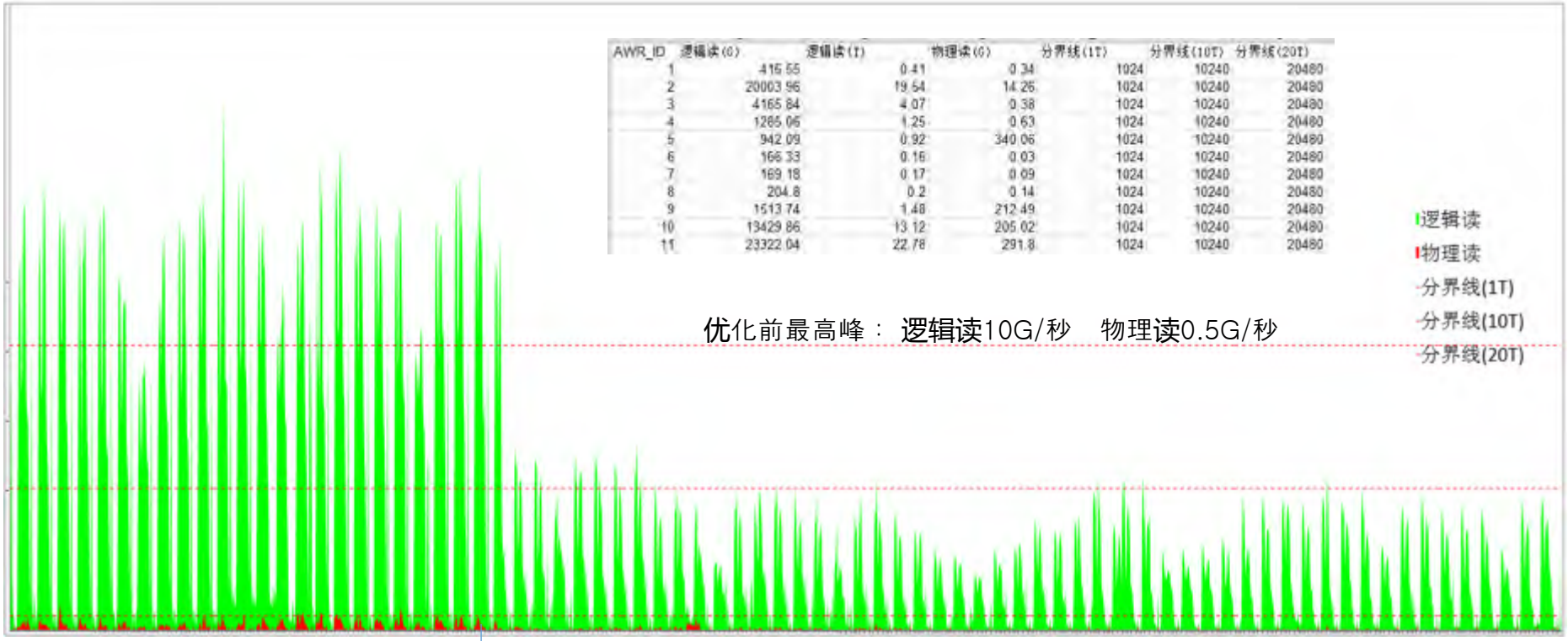
为提升优化效果，结合核心表索引设计，完成了125个TOPSQL优化。

优化效果如下：

- ✓ **SQL 逻辑IO： 由 625,386,664 Blocks 下降到 2,687,922 Blocks,改善率231.8倍。**
- ✓ **SQL 耗时： 由9774.49 S 下降到 228.68 S,改善率 42.74 倍。**
- ✓ **TOPSQL 耗时大幅改善，核心业务性能提升。**

某金融行业客户案例

AWR_ID	逻辑读(G)	逻辑读(T)	物理读(G)	分界线(1T)	分界线(10T)	分界线(20T)
1	416.55	0.41	0.34	1024	10240	20480
2	20003.96	19.54	14.25	1024	10240	20480
3	4165.84	4.07	0.38	1024	10240	20480
4	1285.06	1.25	0.63	1024	10240	20480
5	942.09	0.92	340.06	1024	10240	20480
6	166.33	0.16	0.03	1024	10240	20480
7	169.18	0.17	0.09	1024	10240	20480
8	204.8	0.2	0.14	1024	10240	20480
9	1513.74	1.48	212.49	1024	10240	20480
10	13429.86	13.12	205.02	1024	10240	20480
11	23322.04	22.78	291.8	1024	10240	20480



对比效果说明：

优化前08月06日-09月07日(共计33天)平均逻辑读(每小时) = 9262128G/837小时 = 11065G/小时 = 10.8T/小时

平均物理读(每小时) = 193957G/837小时 = 232G/小时 = 78M/秒

优化后09月07日-10月27日(共计50天)平均逻辑读(每小时) = 3351895G/1140小时 = 2940G/小时 = 2.8T/小时

平均物理读(每小时) = 40622G/1140小时 = 35G/小时 = 10M/秒

磁盘IO资源需求量逻辑读 = ： 2940 / 11065 = **1/4**

物理读 = ： 35 / 232 = **1/6**

总结

人员成本显著降低，一名工程师可以支持5~10个企业

基于云和工具化的方式实现性能信息的搜集和优化，显著降低了工程师的投入，每个工程师可以支持更多的客户服务。

企业客户无须支付昂贵的费用设立性能优化项目，通过该方式给企业提供的远程优化服务服务的持续性和质量，与现场服务没有太大区别。

企业负担降低，服务持续性好，质量有保证，效果与现场服务相近

减少工程师出差频率，工程师获得更多归属感，降低项目实施成本，有利于提升人员稳定性。

工程师出差频次降低后，不仅可以提升工程师的归属感，而且还可以为企业获取更多利润，也有利于人员的持续性培训和管理。

该服务提供方式目前仅适用于对安全要求不是特别高，且可以通过特定网管能够允许外部访问的企业，否则难以应用。只要服务厂商可以严格遵守客户企业的安全要求，其实该方式不会对企业的数据安全带来隐患，因为搜集时，仅搜集性能有关的数据，不涉及具体业务数据。

该方式目前仅适用于安全要求不是特别高的一些企业客户。

THANKS

SequeMedia 盛拓传媒 IT168.com ChinaUnix ITPUB