TiD 2017
质量竞争力大会

性能测试过程实践

曹承臻
搜狗

目录：

- 为什么要做性能测试
- 性能测试分类
- 性能测试具体流程
- 性能调优介绍

为什么要做性能测试？

- 能力验证
- 规划能力
- 性能调优
- 缺陷发现

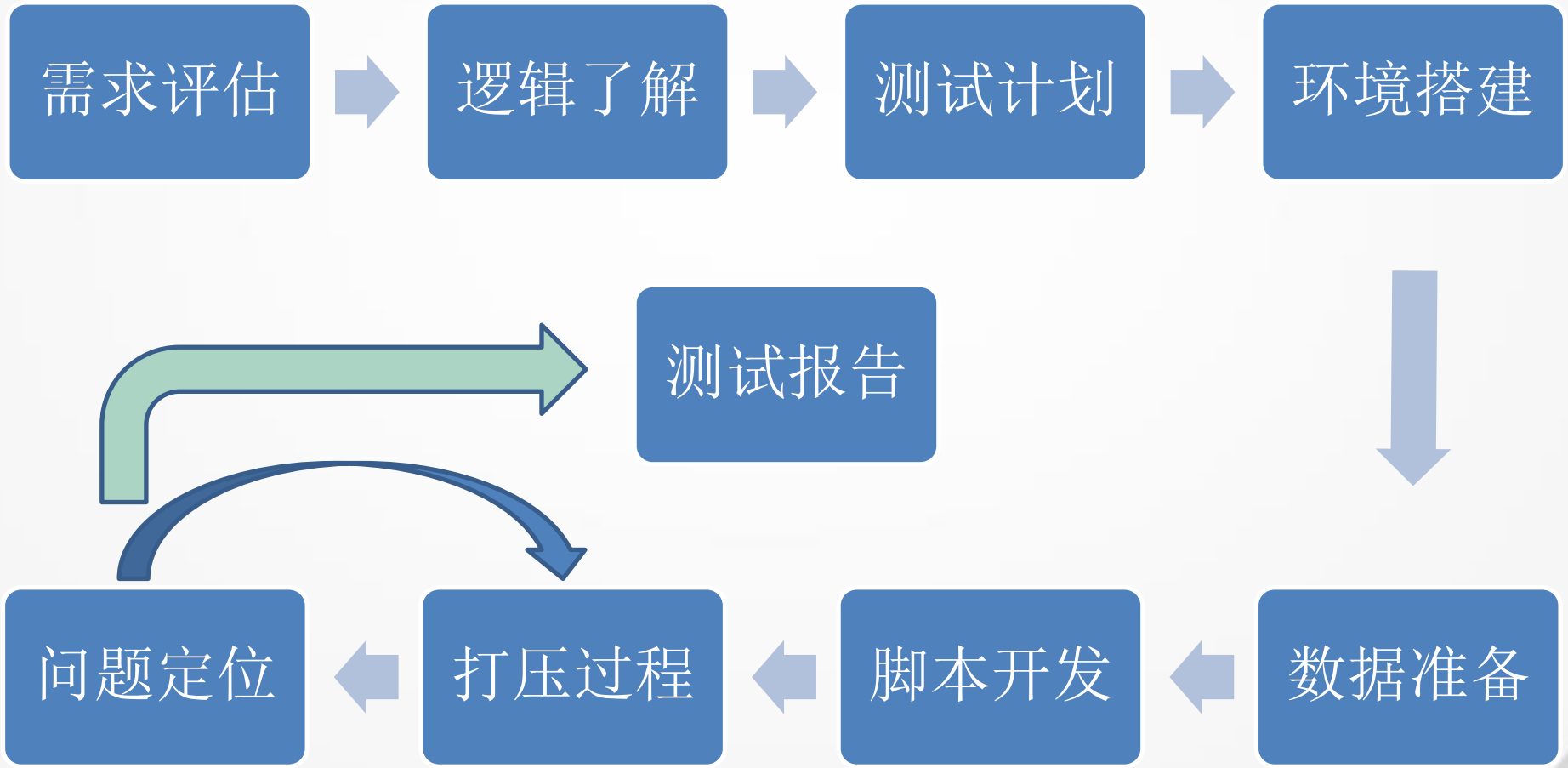
性能测试分类

试

试

试

性能测试具体流程



需求评估

- 目的：评估是否需要做性能测试

>100

<50

需求评估

- 目的：评估是否需要做性能测试
  - 需要做性能测试
    - 新产品，预估单台机器QPS峰值超过100；
    - 已经上线的产品，之前未做过性能测试，接入新业务后，预估单台机器QPS超过100
  - 不需要做性能测试
    - 预估单台机器QPS峰值低于50。
    - 有相同实现逻辑的产品，且已经做过性能测试。
    - 已上线的产品，之前做过性能测试，接入新业务后，且预估本次QPS峰值低于之前打压结果。

TiD2017

需求评估

- QPS评估方法
  - 产品已经做过灰度或者已经上过线

品未上过线，产

天类似产品

0%请求发生

4内

简单计算的结果

- ((80%*5 (24小时 60秒*40%))/1 = 115.7个请求/秒

逻辑了解

- 产品、测试、开发沟通会

沟通

。（参数、可

隔离

试目标QPS及推断方法

和参数间比例关系。

调优最终篇

2017/3/29 (周三) 22:06

(桌面事业部_测

RE: 【压力测试】ABC-8067_网吧版

收件人

抄送 Ma

转发该邮 时间为 2017/6/22 14:30,

Hi 曹哥,

这个是之前沟通的网吧版导航压力测试详细信息,

Jira地址: http://jira.sogou-inc.com/browse/

一、 **测试目的**

为网吧版网址导航123.sogou.com/wb/ 评估QPS

二、 **接口信息**

1. 接口地
2. http: /wb/
3. http: wb/getMod.api?tab=haha
4. http: /wb/getMod.api?tab=baike
5. http: 8/wb/getMod.api?tab=youxi
6. http: 9/wb/getMod.api?tab=zhub
7. http: /wb/getVideos.api?tab=da

8. 参数说明（每个参数的含义及取值范围）

   /wb/ 对应的首页请求

   getMod.api?tab: 对应网吧版导航四个tab

   getVideos.api?tab: 对应影视大全数据数据

9. 返回的结果（写明服务端返回正常的标示字段

   因数据较多,可以单独点击URL查看接口数据,方


```
[
  {
    "tab":{
      "id":"650",
      "title":"健康养生",
      "link":"http://toutiao.sogou.com/jiankang/?fr=sgbk
      "image":"",
      "color":"",
      "subtitle":null,
      "code":"bt_jiankang"
    },
    "title":"嘴里出现这种味,可能是某种疾病征兆",
    "link":"http://www.ttys5.com/xinwen/xinwenhangye/2017-03-20/136221.html?hmsr=sougoudaohang&hmpl=wenzilian419&hmcu
    "image":"",
    "color":"",
    "subtitle":"嘴里出现这种味,可能是某种疾病征兆",
    "code":""
  },
  {
    "id":"669",
    "title":"到底什么鬼！这些东西竟让你越来越笨",
    "link":"http://www.ttys5.com/xinwen/xinwenhangye/2017-03-20/136223.html?hmsr=sougoudaohang&hmpl=wenzilian420&hmcu
    "image":"",
    "color":"",
    "subtitle":"到底什么鬼！这些东西竟让你越来越笨",
    "code":""
  }
 ]
 }
]
```


三、 **测试数据**

无入参数据

四、 **测试机器信息**

Host :10 .238

用户名：root

密码：单独发送

服务端log位置：/search/nginx/log/access.log

硬件信息:

4真核, 4g内存 与线上保持一致

软件信息: php7环境 nginx:1.9 redis:redis-stable DB:mysql .64.173)

五、 **线上搜索接口信息**

QPS估值:

1. 取123.sogou.com(UV为:1800w) 线上高峰时期19:00-21:00 PV为660w 平均QPS为:611
2. 预估网吧版导航UV为200w为线上的1/9,预估导航压测QPS为 611/9≈70

TiD2017

测试计划

一、 测试目的

找到热词单台服务器可以持续承受的QPS峰值。

二、 测试任务

（1）测试阶段分布：

测试准备时间：

服务器提测时间：2016年8月30日

测试开始时间：2016年8月30日

测试完成时间：2016年9月1日

（2）测试任务：

产品需求	测试需求		测试范围	开
热词数据服务器性能测试	服务器	打压机部署	已有可用打压机	张明
		服务器逻辑了解	接口参数、服务器逻辑	张明
		打压脚本编写	用户场景模拟	张明
		测试场景部署	用户场景模拟	张明
		结果分析调优	瓶颈定位、调优，结果产出	张明

（3）测试条件：

1. 测试服务器与线上服务器隔离，不能直接或间接打压线上环境。
2. 测试服务器可以正常提供服务并保持性能稳定。
3. 测试数据要尽量模拟真实的用户操作。

三、 详细计划安排：

	曹承臻
8月30日	服务器逻辑了解&脚本编写
8月31日	场景打压&结果分析&调优
9月1日	结果分析&结果产出

四、 测试环境说明

- **硬件/系统配置**

  CPU：Core 32核（2.4GHz）

  内存：128G

  操作系统：Linux

- **程序配置**

  无

五、 测试分组

a) **测试分组一**

测试目的：

并发较多热词服务器请求时，查看热词服务器各项性能。

持续时间：

1h

测试数据

K值随机生成

日期选择8月1号到8月30号随机选择

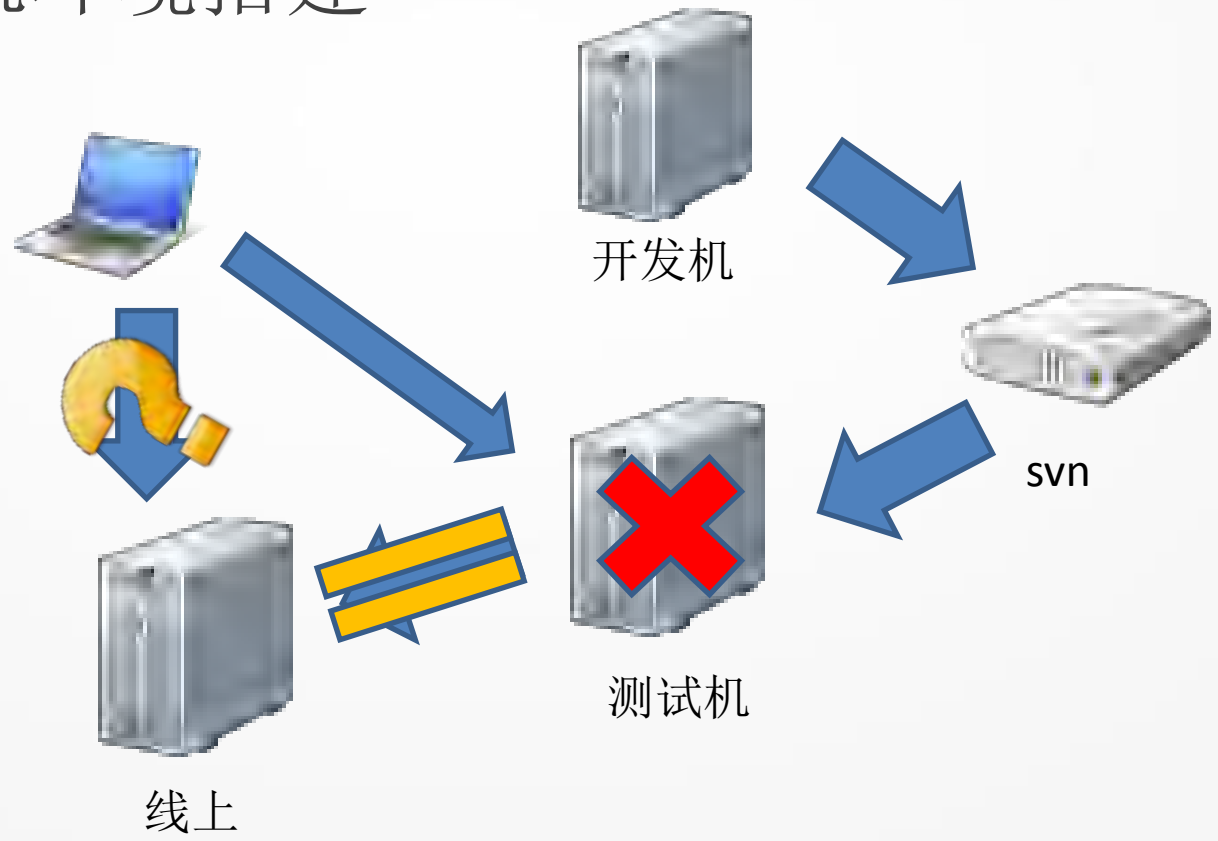
测试方法：

1. 模拟用户操作过程，组成打压请求脚本。

leader、PM要求的测试部件组。

环境搭建

- 打压机环境搭建

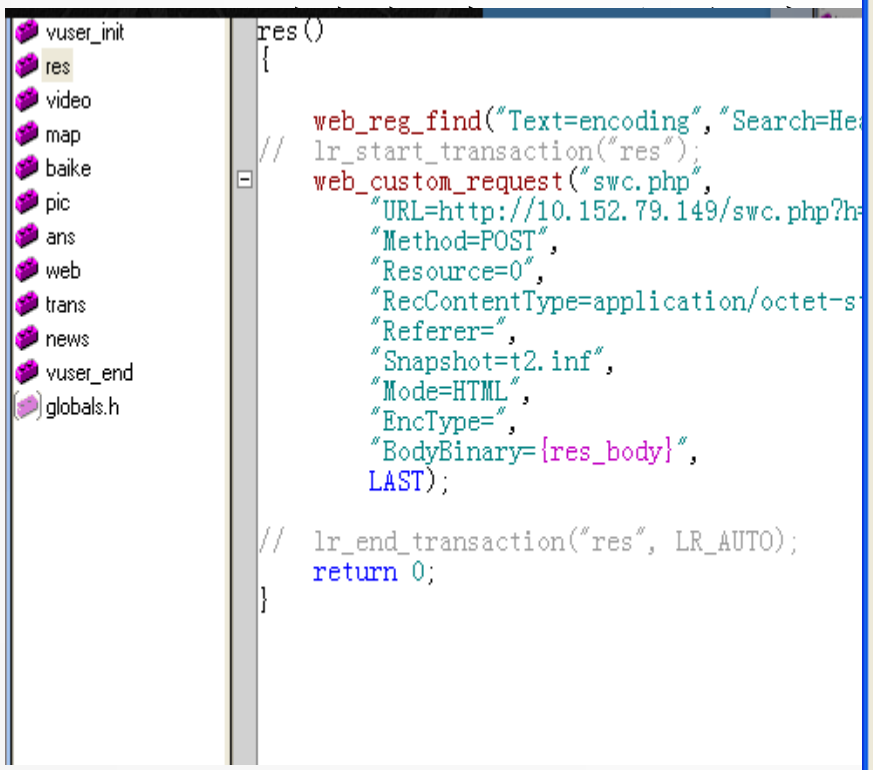


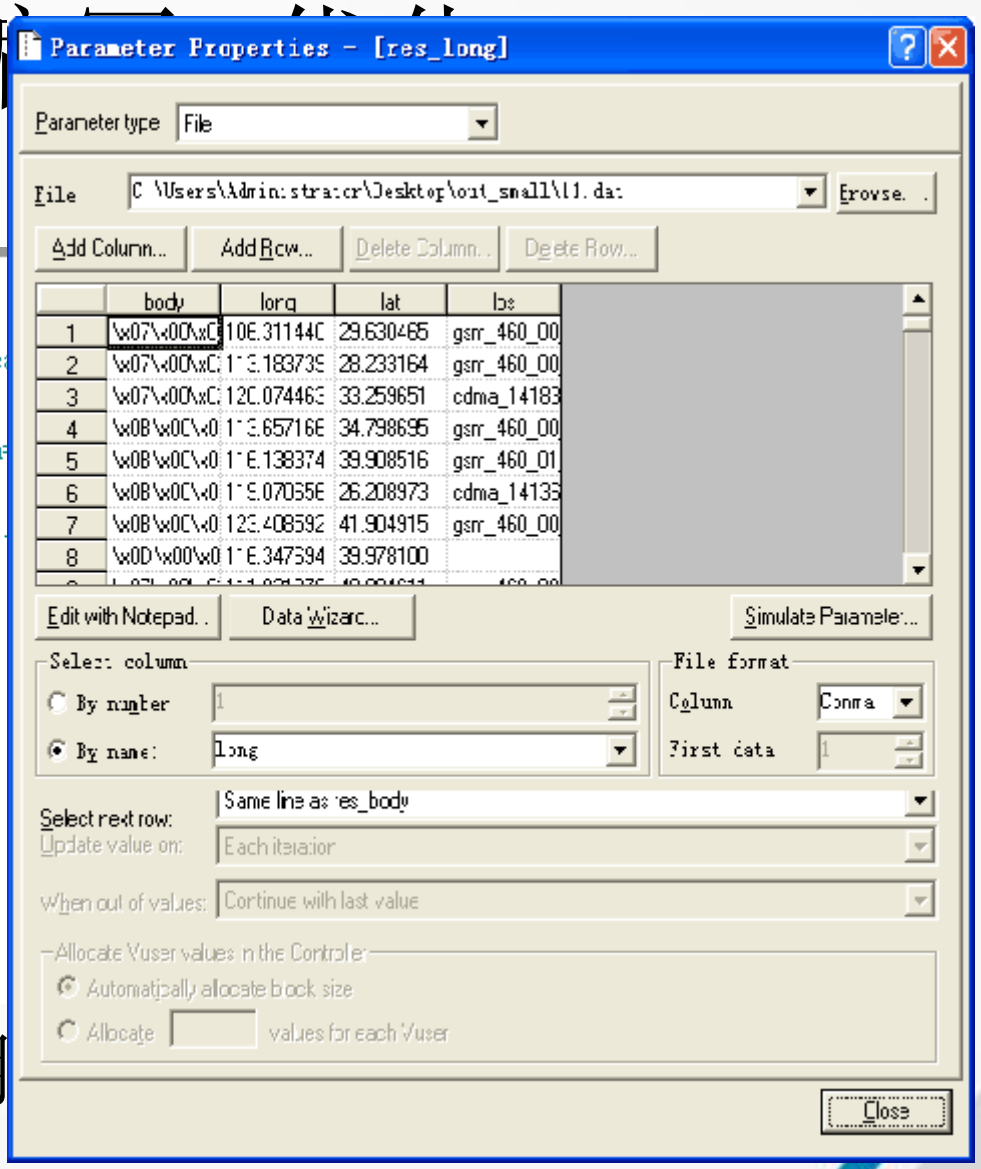
数据准备

- 数据有效性
- 测试机的查看权限
  - Log位置
  - Log信息
- 打压比例

TiD2017

脚本级





➢注：1、尽量模拟用

打压过程

- 服务器性能指标监控
  - Vmstat、top、iostat、nload、free等。
- 客户端性能监控
  - HPS、TPS、响应时间
- 打压机性能监控
- 注：每天的进度汇报

性能调优介绍

- 影响Linux性能的因素
- 系统性能评估标准
- 系统性能分析工具
- 性能评估与优化过程

一　影响Linux服务器性能的因素

- 操作系统级
  - CPU
  - 内存
  - 磁盘I/O
  - 网络I/O带宽

这些子系统之间关系是相互彼此依赖的,任何一个高负载都会导致其他子系统出现问题.比如:

- 大量的页调入请求导致内存队列的拥塞
- 网卡的大吞吐量可能导致更多的 CPU 开销
- 大量的 CPU 开销又会尝试更多的内存使用请求
- 大量来自内存的磁盘写请求可能导致更多的 CPU 以及 IO 问题

所以要对一个系统进行优化,查找瓶颈来自哪个方面是关键,虽然看似是某一个子系统出现问题,其实有可能是别的子系统导致的.

服务器性能排查过程:

全面排查→缩小范围→猜测原因→数据支持→定位问题

二　系统性能评估标准

影响性能因素	评判标准		
	好	坏	糟糕
CPU	user% + sys%< 70%	user% + sys%= 85%	user% + sys% >=90%
内存	Swap In（si）=0 Swap Out（so）=0	Swap In（si）=10左右 Swap Out（so）=10左右	More Swap In & Swap Out
磁盘	iowait % < 20%	iowait % =35%	iowait % >= 50%

其中：

%user：表示**CPU**处在用户模式下的时间百分比。
%sys：表示**CPU**处在系统模式下的时间百分比。
%iowait：表示**CPU**等待输入输出完成时间的百分比。
swap in：即**si**，表示虚拟内存的页导入，即从**SWAP DISK**交换到**RAM**
swap out：即**so**，表示虚拟内存的页导出，即从**RAM**交换到**SWAP DISK**。

三　系统性能分析工具

常用系统命令

Vmstat、sar、iostat、netstat、free、ps、top等

第三方工具

nmon_x86_64_rhel54.rhel54

四　Linux性能评估与优化

1：　确定应用类型

基于需要理解该从什么地方来入手优化瓶颈,首先重要的一点,就是理解并分析当前系统的特点,多数系统所跑的应用类型,主要为 2 种:

- IO Bound(IO 范畴): 在这个范畴中的应用,一般都是高负荷的内存使用以及较高的磁盘读写，例如频繁查询数据库操作的应用.
- CPU Bound(CPU 范畴): 在这个范畴中的应用,一般都是高负荷的 CPU 占用. CPU 范畴的应用,就是一个批量处理 CPU 请求以及数学计算的过程.例如计算当天的抢票数据后，不断向其他服务器发请求并处理返回值。

结论：一般来说， IO Bound的应用，系统IO容易成为主要瓶颈，CPU Bound的应用，CPU容易成为主要瓶颈。

2：cpu性能评估

（1）利用vmstat命令监控系统CPU

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	wa	id
1	0	138592	17932	126272	214244	0	0	1	18	109	19	2	1	1	96
0	0	138592	17932	126272	214244	0	0	0	0	105	46	0	1	0	99
0	0	138592	17932	126272	214244	0	0	0	0	198	62	40	14	0	45
0	0	138592	17932	126272	214244	0	0	0	0	117	49	0	0	0	100
0	0	138592	17924	126272	214244	0	0	0	176	220	938	3	4	13	80
0	0	138592	17924	126272	214244	0	0	0	0	358	1522	8	17	0	75
1	0	138592	17924	126272	214244	0	0	0	0	368	1447	4	24	0	72
0	0	138592	17924	126272	214244	0	0	0	0	352	1277	9	12	0	79

- 每列的含义如下
- procs
  - r //等待CPU运行时间的进程数，如果较多，说明CPU很忙
  - b //处于不可中断的进程数，通常意味着这些进程在等待I/O(磁盘，网络，用户输入等)，此值长期不为0的话需要分析一下原因。
- memory
  - swpd //总共被使用的虚拟内存，如果此值较高或者再不断增加，说明内存不够
  - free //空闲的物理内存
  - buff 用作缓冲的内存大小，用来存储，目录里面有什么内容，权限等的缓存
  - cache 用作缓存的内存大小，直接用来记忆我们打开的文件,给文件做缓冲
- swap
  - si //每秒从交换区写到内存的大小，由磁盘调入内存。
  - so //每秒写入交换区的内存大小，由内存调入磁盘
  - bi //每秒从block设备收到的块数量(读取)
  - bo //每秒发送给block设备的块数量(写入)
- system
  - in //每秒的中断数，包括时钟
  - cs //每秒的上下文切换数,多发生在操作系统结束一个进程，然后启动另一个进程的时候
- cpu
  - us：用于运行非核心(用户代码)花费的百分比
  - sy：用于运行kernel代码花费的百分比
  - id：空闲的cpu百分比
  - wa：用于等待IO的百分比

（3）案例分析：

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	wa	id
2	1	207740	98476	81344	180972	0	0	2496	0	900	2883	4	12	57	84
0	1	207740	96448	83304	180984	0	0	1968	328	810	2559	8	9	90	83
0	1	207740	94404	85348	180984	0	0	2044	0	829	2879	9	6	88	85
0	1	207740	92576	87176	180984	0	0	1828	0	689	2088	3	9	78	88
2	0	207740	91300	88452	180984	0	0	1276	0	565	2182	7	6	83	87
3	1	207740	90124	89628	180984	0	0	1176	0	551	2219	2	7	91	91
4	2	207740	89240	90512	180984	0	0	880	520	443	907	22	10	77	67
5	3	207740	88056	91680	180984	0	0	1168	0	628	1248	12	11	87	77
4	2	207740	86852	92880	180984	0	0	1200	0	654	1505	6	7	77	87
6	1	207740	85736	93996	180984	0	0	1116	0	526	1512	5	10	75	85
0	1	207740	84844	94888	180984	0	0	892	0	438	1556	6	4	80	90

根据观察值,我们可以得到以下结论:
1,上下文切换数目高于中断数目,说明 kernel 中相当数量的时间都开销在上下文切换线程.
2,大量的上下文切换将导致 CPU 利用率分类不均衡.很明显实际上等待 io 请求的百分比(wa)非常高,以及user time 百分比非常低(us).
3,因为 CPU 都阻塞在 IO 请求上,所以运行队列里也有相当数目的可运行状态线程在等待执行.

问题？

你是否遇到过系统CPU整体利用率不高，而应用缓慢的现象？

在一个多CPU的系统中，如果程序使用了单线程，会出现这么一个现象，CPU的整体使用率不高，但是系统应用却响应缓慢，这可能是由于程序使用单线程的原因，单线程只使用一个CPU，导致这个CPU占用率为100%，无法处理其它请求，而其它的CPU却闲置，这就导致了整体CPU使用率不高，而应用缓慢现象的发生。

3：内存性能评估

（1）利用vmstat命令监控内存

```
[root@node1 ~]# vmstat 2 3
procs -----------memory----------  ---swap--  -----io---- --system--  -----cpu------
 r  b   swpd   free      buff  cache   si   so    bi    bo     in     cs    us sy  id  wa st
 0  0      0   162240   8304  67032    0    0    13    21   1007    23     0  1  98   0  0
 0  0      0   162240   8304  67032    0    0     1     0   1010    20     0  1  100 0  0
 0  0      0   162240   8304  67032    0    0     1     1   1009    18     0  1  99   0  0
```

●memory

swpd列表示切换到内存交换区的内存数量（以k为单位）。如果swpd的值不为0，或者比较大，只要si、so的值长期为0，这种情况下一般不用担心，不会影响系统性能。

free列表示当前空闲的物理内存数量（以k为单位）

buff列表示buffers cache的内存数量，一般对块设备的读写才需要缓冲。

cache列表示page cached的内存数量，一般作为文件系统cached，频繁访问的文件都会被cached，如果cache值较大，说明cached的文件数较多，如果此时IO中bi比较小，说明文件系统效率比较好。

●swap

si列表示由磁盘调入内存，也就是内存进入内存交换区的数量。

so列表示由内存调入磁盘，也就是内存交换区进入内存的数量。

一般情况下，si、so的值都为0，如果si、so的值长期不为0，则表示系统内存不足。需要增加系统内存。

3：内存性能评估

（1） 利用 free命令监控内存占用

```
[@gd_57_162 python]# free -m
             total       used       free     shared    buffers     cached
Mem:          5850       2315       3535          0        214       1719
-/+ buffers/cache:        381       5468
Swap:            0          0          0
```

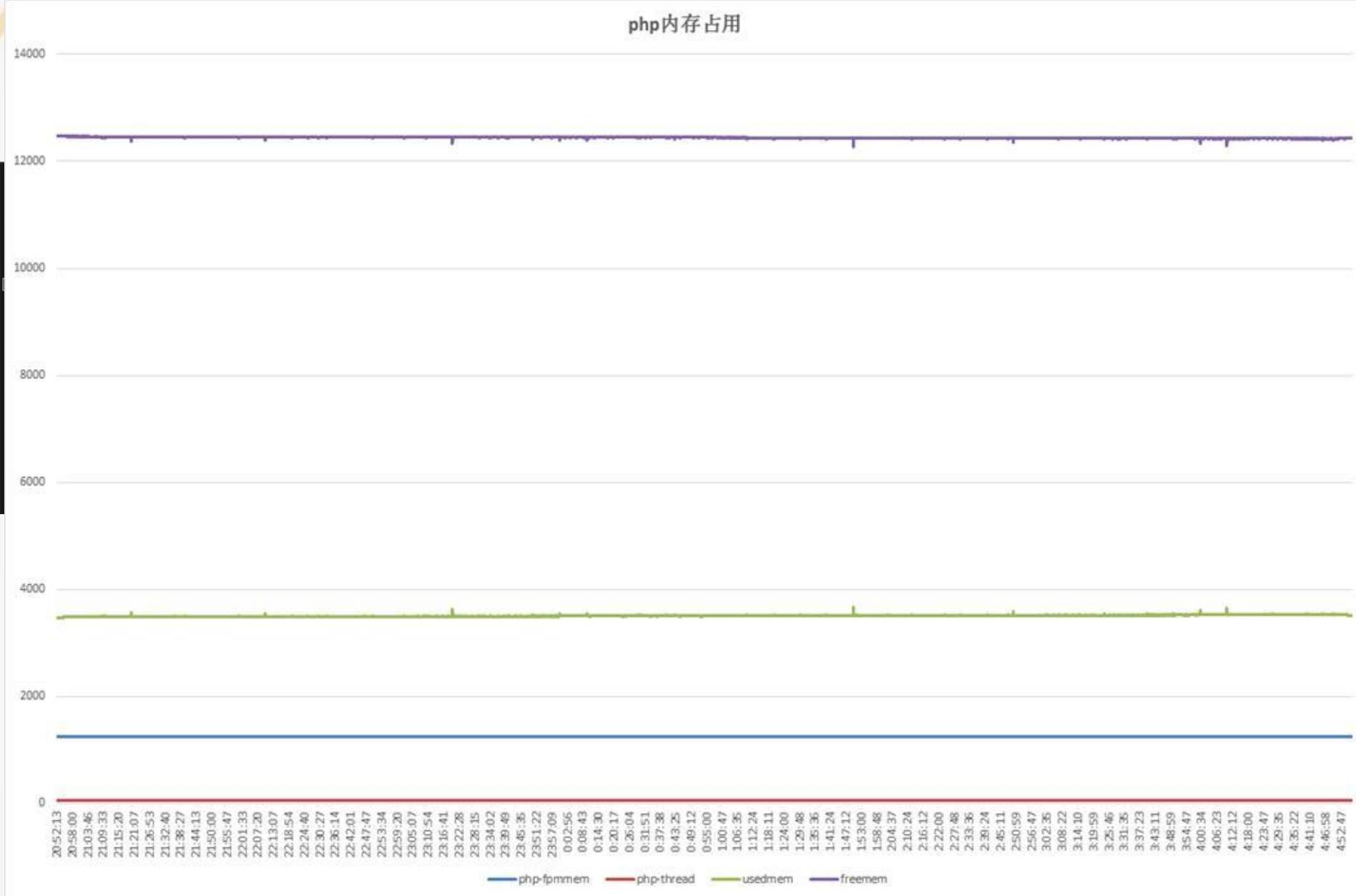
Cache大不是内存泄露：

To free pagecache: echo 1 > /proc/sys/vm/drop_caches

To free dentries and inodes: echo 2 > /proc/sys/vm/drop_caches

To free pagecache, dentries and inodes: echo 3 > /proc/sys/vm/drop_caches

TiD2017性能评估



（2） 案例分析：内存不足

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa
17	0	1250	3248	45820	1488472	30	132	992	0	2437	7657	23	77	0	23
11	0	1376	3256	45820	1488888	57	245	416	0	2391	7173	10	90	0	0
12	0	1582	1688	45828	1490228	63	131	1348	76	2432	7315	10	90	0	10
12	2	3981	1848	45468	1489824	185	56	2300	68	2478	9149	15	25	60	73
14	2	10385	2400	44484	1489732	0	87	1112	20	2515	11620	0	15	85	88
14	2	12671	2280	43644	1488816	76	51	1812	204	2546	11407	20	45	35	35

根据观察值,我们可以得到以下结论:

1. 大量的读请求回内存(bi),导致了空闲内存在不断的减少(free).这就使得系统写入 swap device 的块数目(so)和 swap 空间(swap)在不断增加.
2. si、so的值长期不为0，则表示系统内存不足,同时看到 CPU Wati I/O time(wa)百分比很大.这表明 I/O 请求已经导致 CPU 开始效率低下.

（3） 内存不足定位

#valgrind --tool=memcheck --leak-check=yes --show-reachable=yes ./test

测试代码

```cpp
#include <stdlib.h>
int* func(void)
{
    int* x = malloc(10 * sizeof(i
    x[10] = 0;  //问题1: 数组下标越
}
 int main(void)
{
    int* x=NULL;
    x=func();
    //free(x);
    x=NULL;
    return 0;   //问题2: 内存没有释
 }
```

```
==12957== Memcheck, a memory error detector
==12957== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==12957== Using Valgrind-3.6.0 and LibVEX; rerun with -h for copyright info
==12957== Command: ./test
==12957==
==12957== Invalid write of size 4
==12957==    at 0x80483DF: func (test.c:5)
==12957==    by 0x80483FC: main (test.c:10)
==12957==  Address 0x41d3050 is 0 bytes after a block of size 40 alloc'd
==12957==    at 0x402682F: malloc (vg_replace_malloc.c:236)
==12957==    by 0x80483D5: func (test.c:4)
==12957==    by 0x80483FC: main (test.c:10)
==12957==
==12957==
==12957== HEAP SUMMARY:
==12957==     in use at exit: 40 bytes in 1 blocks
==12957==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==12957==
==12957== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==12957==    at 0x402682F: malloc (vg_replace_malloc.c:236)
==12957==    by 0x80483D5: func (test.c:4)
==12957==    by 0x80483FC: main (test.c:10)
==12957==
==12957== LEAK SUMMARY:
==12957==    definitely lost: 40 bytes in 1 blocks
==12957==    indirectly lost: 0 bytes in 0 blocks
==12957==      possibly lost: 0 bytes in 0 blocks
==12957==    still reachable: 0 bytes in 0 blocks
==12957==         suppressed: 0 bytes in 0 blocks
==12957==
==12957== For counts of detected and suppressed errors, rerun with: -v
==12957== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 13 from 8)
```

Memwatch监测工具

4：磁盘I/O性能评估

（1）磁盘转速和每秒读写次数的关系

结论：10K RPM 磁盘有能力提供 120~150 次读写。

以一天
1, 100( )S)
2, 转换 nds (Rotation)
3, 单位
4, 旋转
5, 加上 MS
6, 加上
7, 100(
每次应 上都要花费下列时间里，
磁盘将
有效率 的 I/O 请求数

（2）利用iostat评估磁盘性能

Device:	rrqm/s	wrqm/s	r/s	w/s	rsec/s	wsec/s	avgrq-sz	avgqu-sz	await	svctm	%util
vda	0	29	0	5	0	272	54.4	0.02	3.8	1.4	0.7
vda1	0	0	0	0	0	0	0	0	0	0	0
vda2	0	29	0	5	0	272	54.4	0.02	3.8	1.4	0.7
vdb	0	0	0	0	0	0	0	0	0	0	0
dm-0	0	0	0	0	0	0	0	0	0	0	0
dm-1	0	0	0	34	0	272	8	0.09	2.59	0.21	0.7
dm-2	0	0	0	0	0	0	0	0	0	0	0
dm-3	0	0	0	0	0	0	0	0	0	0	0
dm-4	0	0	0	0	0	0	0	0	0	0	0

参数说明：
r/s+w/s:读写次数
Await：I/O请求平均执行时间.包括发送请求和执行的时间.单位是毫秒.
Svctm：发送到设备的I/O请求的平均执行时间.单位是毫秒.
%util：在I/O请求发送到设备期间,占用CPU时间的百分比.用于显示设备的带宽利用率.当这个值接近100%时,表示设备带宽已经占满

（3）案例分析：磁盘I/O负载过高

Device:	rrqm/s	wrqm/s	r/s	w/s	rsec/s	wsec/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	svctm	%util
/dev/sda	0	1766.67	4866.67	1700	38933.33	31200	19466.67	15600	10.68	6526.67	100.56	5.08	3333.33
/dev/sda1	0	933.33	0	0	0	7733.33	0	3866.67	0	20	2145.07	7.37	200
/dev/sda2	0	0	4833.33	0	38666.67	533.33	19333.33	266.67	8.11	373.33	8.07	6.9	87
/dev/sda3	0	833.33	33.33	1700	266.67	22933.33	133.33	11466.67	13.38	6133.33	358.46	11.35	1966.67

结论：

1、r/s+w/s的值远远超过磁盘可以承受的最大读写次数。

2、%util：这个值大于100%时,表示设备带宽已经占满 。

综上所述：磁盘I/O成为系统瓶颈。

5：网络性能评估

使用工具**nmon**

```
┌nmon-14i──────[H for help]──Hostname=sjs_68_232──Refresh= 2secs ──20:36.04─────
│ Network I/O ──────────────────────────────────────────────────────────────────
│I/F Name Recv=KB/s Trans=KB/s packin packout insize outsize Peak->Recv Trans
│      lo      0.0      0.0      0.0      0.0      0.0      0.0        0.0     0.0
│    eth0      1.3      0.1     23.0      0.5     60.0    234.0        5.2     4.4
│    eth1     36.4      0.0    120.8      0.0    308.3      0.0      342.7     8.6
│    sit0      0.0      0.0      0.0      0.0      0.0      0.0        0.0     0.0
```

参数说明：

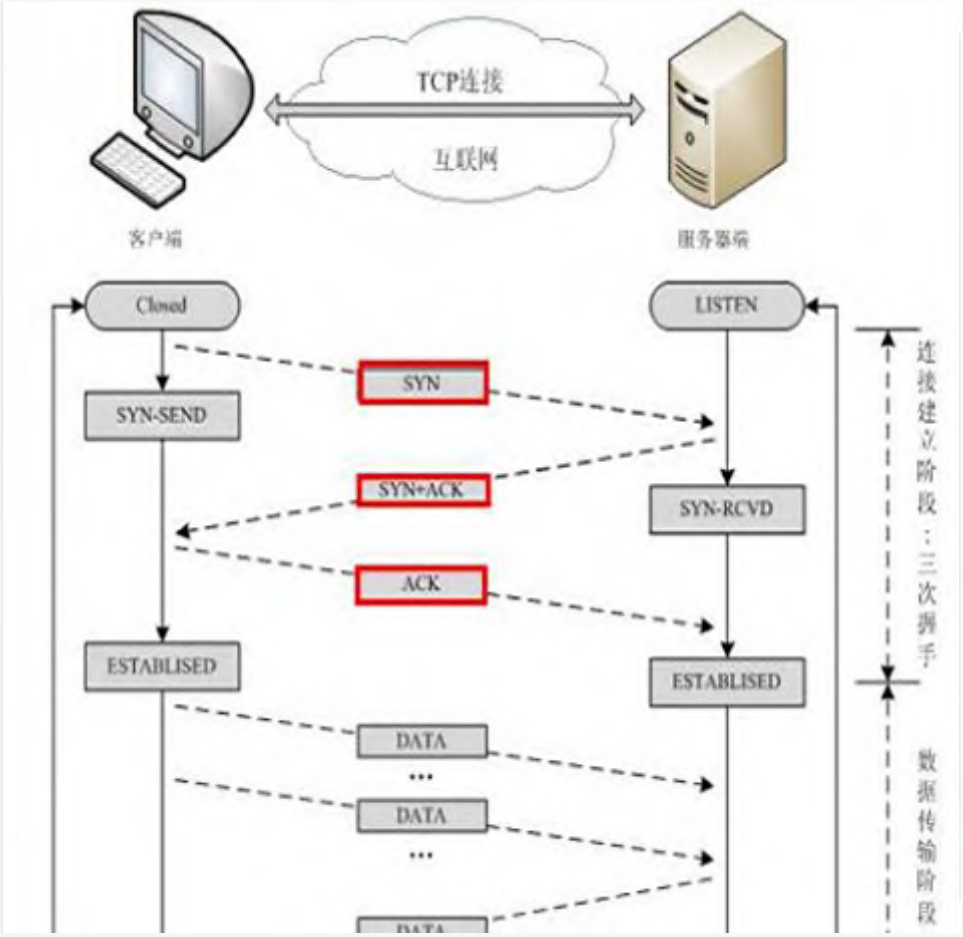
1、recv：每秒接收千字节大小。

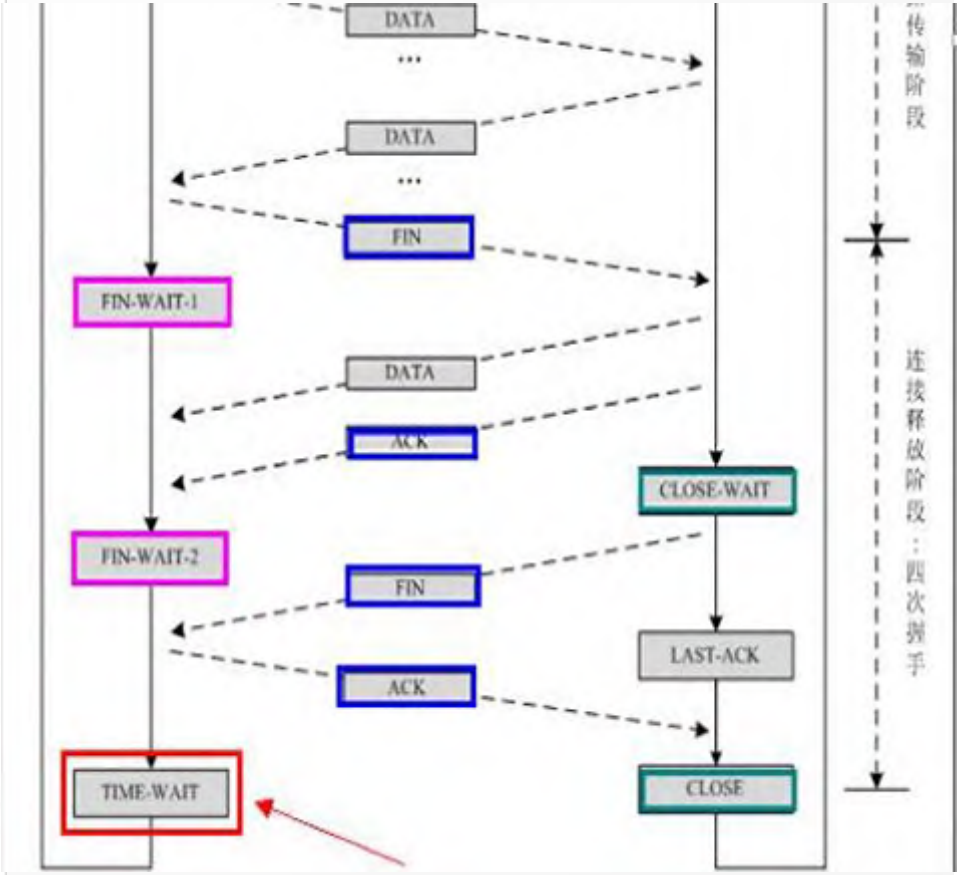
2、trans：每秒发出的千字节大小。

3、packin：当前正在接收的包个数。

4、packout：当前正在发出的包个数

TCP三次握手&四次挥手介绍





Time-Wait

1、服务端time-wait个数如何产生的？
2、time-wait的类型
3、查看time-wait个数的命令
netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}‘

```
[@bjzw_111_238 nginx_logs]# netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print
TIME_WAIT 43
CLOSE_WAIT 5
ESTABLISHED 377
```

查看系统可用端口号个数
cat /proc/sys/net/ipv4/ip_local_port_range

```
[@bjzw_111_238 nginx_logs]# cat /proc/sys/net/ipv4/ip_local_port_range
32768   61000
```

如何设置

- vi /etc/sysctl.conf

- net.ipv4.tcp_tw_reuse = 1
- //表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；
- net.ipv4.tcp_tw_recycle = 1
- //表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭
- net.ipv4.tcp_fin_timeout = 30
- //修改系统默认的 TIMEOUT 时间
- /sbin/sysctl –p

TiD2017

实例分享

r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa
1	0	249844	19144	18532	1221212	0	0	7	3	22	17	25	8	17	18
0	1	249844	17828	18528	1222696	0	0	40448	8	1384	1138	13	7	65	14
0	1	249844	18004	18528	1222756	0	0	13568	4	623	534	3	4	56	37
2	0	249844	17840	18528	1223200	0	0	35200	0	1285	1017	17	7	56	20
1	0	249844	22488	18528	1218608	0	0	38656	0	1294	1034	17	7	58	18
0	1	249844	21228	18544	1219908	0	0	13696	484	609	559	5	3	54	38
0	1	249844	17752	18544	1223376	0	0	36224	4	1469	1035	10	6	67	17
1	1	249844	17856	18544	1208520	0	0	28724	0	950	941	33	12	49	7
1	0	249844	17748	18544	1222468	0	0	40968	8	1266	1164	17	9	59	16
1	0	249844	17912	18544	1222572	0	0	41344	12	1237	1080	13	8	65	13

TiD2017

实例分享

```
#   iostat  -x  1

avg-cpu:    %user   %nice   %sys    %idle
30.00   0.00    9.33    60.67
Device: rrqm/s  wrqm/s  r/s w/s rsec/s  wsec/s  rkB/s   wkB/s   avgrq-sz    avgqu-sz    await   svctm   %util
/dev/sda    7929.01 30.34   1180.91 14.23   7929.01 357.84  3964.50 178.92  6.93    0.39    0.03    0.06    6.69
/dev/sda1   2.67    5.46    0.40    1.76    24.62   57.77   12.31   28.88   38.11   0.06    2.78    1.77    0.38
/dev/sda2   0.00    0.30    0.07    0.02    0.57    2.57    0.29    1.28    32.86   0.00    3.81    2.64    0.03
/dev/sda3   7929.01 24.58   1180.44 12.45   7929.01 297.50  3964.50 148.75  6.90    0.32    0.03    0.06    6.6
avg-cpu:    %user   %nice   %sys    %idle
9.50    0.00    10.68   79.82
Device: rrqm/s  wrqm/s  r/s w/s rsec/s  wsec/s  rkB/s   wkB/s   avgrq-sz    avgqu-sz    await   svctm   %util
/dev/sda    0.00    0.00    1195.24 0.00    0.00    0.00    0.00    0.00    0.00    43.69   3.60    0.99    117.86
/dev/sda1   0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
/dev/sda2   0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
/dev/sda3   0.00    0.00    1195.24 0.00    0.00    0.00    0.00    0.00    0.00    43.69   3.60    0.99    117.86
avg-cpu:    %user   %nice   %sys    %idle
9.23    0.00    10.55   79.22
Device: rrqm/s  wrqm/s  r/s w/s rsec/s  wsec/s  rkB/s   wkB/s   avgrq-sz    avgqu-sz    await   svctm   %util
/dev/sda    0.00    0.00    1200.37 0.00    0.00    0.00    0.00    0.00    0.00    41.65   2.12    0.99    112.51
/dev/sda1   0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
/dev/sda2   0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
/dev/sda3   0.00    0.00    1200.37 0.00    0.00    0.00    0.00    0.00    0.00    41.65   2.12    0.99    112.51
```

TiD2017

实例分享

```
#top -d 1
11:46:11    up  3   days,   19:13,  1   user,   load    average:    1.72,   1.87,   1.80
176 processes:  174 sleeping,   2   running,    0   zombie, 0   stopped
CPU states: cpu user    nice    system  irq softirq iowait  idle
total   12.8%   0.0%    4.6%    0.2%    0.2%    18.7%   63.2%
cpu00   23.3%   0.0%    7.7%    0.0%    0.0%    36.8%   32.0%
cpu01   28.4%   0.0%    10.7%   0.0%    0.0%    38.2%   22.5%
cpu02   0.0%    0.0%    0.0%    0.9%    0.9%    0.0%    98.0%
cpu03   0.0%    0.0%    0.0%    0.0%    0.0%    0.0%    100.0%
Mem:    2055244k    av, 2032692k    used,   22552k  free,   0k  shrd,   18256k  buff
1216212k    actv,   513216k in_d,   25520k  in_c
Swap:   4192956k    av, 249844k used,   3943112k    free    1218304k    cached
PID USER    PR  NI  VIRT    RES SHR S   %CPU    %MEM    TIME+   COMMAND
14939   mysql   25  0   379M    224M    1117    R   38.2    25.7%   15:17.78    mysqld
4023    root    15  0   2120    972 784 R   2.0 0.3 0:00.06 top
1   root    15  0   2008    688 592 S   0.0 0.2 0:01.30 init
2   root    34  19  0   0   0   S   0.0 0.0 0:22.59 ksoftirqd/0
3   root    RT  0   0   0   0   S   0.0 0.0 0:00.00 watchdog/0
4   root    10  -5  0   0   0   S   0.0 0.0 0:00.05 events/0
```

TiD2017

实例分享

```
#top -d 1
11:46:11    up  3   days,   19:13,  1   user,   load    average:    1.72,   1.87,   1.80
176 processes:  174 sleeping,   2   running,    0   zombie, 0   stopped
CPU states: cpu user    nice    system  irq softirq iowait  idle
total   12.8%   0.0%    4.6%    0.2%    0.2%    18.7%   63.2%
cpu00   23.3%   0.0%    7.7%    0.0%    0.0%    36.8%   32.0%
cpu01   28.4%   0.0%    10.7%   0.0%    0.0%    38.2%   22.5%
cpu02   0.0%    0.0%    0.0%    0.9%    0.9%    0.0%    98.0%
cpu03   0.0%    0.0%    0.0%    0.0%    0.0%    0.0%    100.0%
Mem:    2055244k    av, 2032692k    used,   22552k  free,   0k  shrd,   18256k  buff
1216212k    actv,   513216k in_d,   25520k  in_c
Swap:   4192956k    av, 249844k used,   3943112k    free    1218304k    cached
PID USER    PR  NI  VIRT    RES SHR S   %CPU    %MEM    TIME+   COMMAND
14939   mysql   25  0   379M    224M    1117    R   38.2    25.7%   15:17.78    mysqld
4023    root    15  0   2120    972 784 R   2.0 0.3 0:00.06 top
1   root    15  0   2008    688 592 S   0.0 0.2 0:01.30 init
2   root    34  19  0   0   0   S   0.0 0.0 0:22.59 ksoftirqd/0
3   root    RT  0   0   0   0   S   0.0 0.0 0:00.00 watchdog/0
4   root    10  -5  0   0   0   S   0.0 0.0 0:00.05 events/0
```

实例分享

```
# strace -p 14939
Process 14939 attached - interrupt to quit
read(29, "\3\1\237\1\366\337\1\222%\4\2\0\0\0\0\0012P/d", 20) = 20
read(29, "ata1/strongmail/log/strongmail-d"..., 399) = 399
_llseek(29, 2877621036, [2877621036], SEEK_SET) = 0
read(29, "\1\1\241\366\337\1\223%\4\2\0\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 400) = 400
_llseek(29, 2877621456, [2877621456], SEEK_SET) = 0
read(29, "\1\1\235\366\337\1\224%\4\2\0\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 396) = 396
_llseek(29, 2877621872, [2877621872], SEEK_SET) = 0
read(29, "\1\1\245\366\337\1\225%\4\2\0\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 404) = 404
_llseek(29, 2877622296, [2877622296], SEEK_SET) = 0
read(29, "\3\1\236\2\366\337\1\226%\4\2\0\0\0\0\0012P/d", 20) = 20
```

TiD2017

实例分享

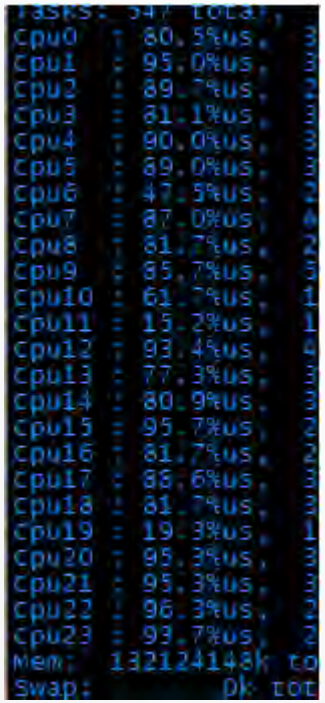
```
# ./mysqladmin -pstrongmail processlist
+----+------+-----------+-----------+---------+------+---------+-----------------
| Id | User | Host | db | Command | Time | State | Info
+----+------+-----------+-----------+---------+------+---------+-----------------
| 1 | root | localhost | strongmail | Sleep | 10 | |
| 2 | root | localhost | strongmail | Sleep | 8 | |
| 3 | root | localhost | root | Query | 94 | Updating | update `failures` set
`update_datasource`='Y' where database_id='32' and update_datasource='N' and |
| 14 | root | localhost | | Query | 0 | | show processlist
```

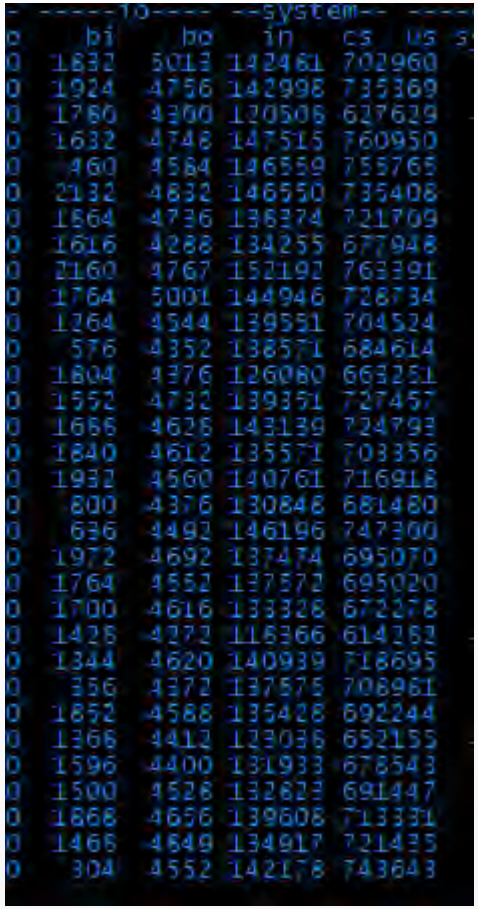
- 结论: 从以上总结出,MySQL 里这些 update 查询问题,都是在尝试对所有 table 进行索引.这些产生的读请求正 是导致系统性能下降的原因.

TiD2017

实例分享

问题：发现A请求压力80tps后，cpu占用就非常高了（24核的机器，每个cpu占用率全面飙到80%以上），且设置的检查点没有任何报错





实例分享

Device:	rrqm/s	wrqm/s	r/s	w/s	rsec/s	wsec/s	avgrq-sz	avgqu-sz	await	svctm	%util
sda	0.00	1.00	0.00	3.00	0.00	24.00	8.00	0.00	0.33	0.33	0.10
dm-0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dm-1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dm-2	0.00	0.00	0.00	3.00	0.00	24.00	8.00	0.00	0.33	0.33	0.10
dm-3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
dm-4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
sdb	0.00	88.00	202.00	[illegible]			43.36	1.50	1.98	1.00	75.70

bo	in	cs	us	sy	[illegible]
0	26411	420645	61	20	
0	26695	417673	60	20	
28	26540	440458	61	19	
0	26453	432661	61	19	
8	27997	404315	61	22	
0	26044	418908	60	20	
144	25986	421930	58	22	
32	25996	438852	58	22	
0	25743	418022	61	19	
40	27868	417311	60	23	
0	26461	411375	63	18	
0	26306	418295	60	20	
12	25817	431984	61	20	
0	26382	418646	59	21	
99	27862	404357	62	21	
59	26362	424842	62	18	
0	26266	430365	60	20	
12	25909	419956	57	23	
0	26234	421758	62	18	
0	27847	404736	60	24	
9	26701	410039	62	19	
0	26439	423951	62	16	
12	26283	416130	62	18	
0	25978	413793	56	24	
0	28240	410933	64	20	
0	26226	407984	60	20	
0	26207	417897	58	22	
12	26039	427543	58	22	
0	25587	415449	60	20	
120	28179	426880	61	23	

TiD2017

实例分享

```
#! /usr/bin/env stap

global csw_count
global idle_count
probe scheduler.cpu_off {
csw_count[task_prev, task_next]++
idle_count+=idle
}
function fmt_task(task_prev, task_next)
{
return sprintf("%s(%d)->%s(%d)",
task_execname(task_prev),
task_pid(task_prev),
task_execname(task_next),
task_pid(task_next))
}
function print_cswtop () {
printf ("%45s %10s\n", "Context switch", "COUNT")
foreach ([task_prev, task_next] in csw_count- limit 20) {
printf("%45s %10d\n", fmt_task(task_prev, task_next), csw_count[task_prev, task_next])
}
printf("%45s %10d\n", "idle", idle_count)
delete csw_count
delete idle_count
}
probe timer.s($1) {
print_cswtop ()
printf("----------------------------------------------------------\n")
}
```

```
                                  Context switch      COUNT
discover_agent_(12250)->swapper(0)      70395
swapper(0)->discover_agent_(12250)      70370
discover_agent_(12250)->swapper(0)      64330
swapper(0)->discover_agent_(12250)      64303
discover_agent_(12250)->swapper(0)      61493
swapper(0)->discover_agent_(12250)      61466
discover_agent_(12250)->swapper(0)      33717
swapper(0)->discover_agent_(12250)      33706
discover_agent_(12250)->swapper(0)      32897
swapper(0)->discover_agent_(12250)      32881
discover_agent_(12250)->swapper(0)      19274
swapper(0)->discover_agent_(12250)      19265
discover_agent_(12250)->swapper(0)      17311
swapper(0)->discover_agent_(12250)      17306
discover_agent_(12250)->swapper(0)      13720
swapper(0)->discover_agent_(12250)      13714
discover_agent_(12250)->swapper(0)       6985
swapper(0)->discover_agent_(12250)       6984
discover_agent_(12250)->swapper(0)       5184
swapper(0)->discover_agent_(12250)       5179
                                  idle  356852
```

TiD2017

实例分享

Context Switching(內文切換)

1. Def：
   當CPU從一個process切到另一個process執行之前
   Context switching
   而 Context switching是一**系統負擔**，其時間長短
2. 如何降低context switch之負擔
   - 法一：多加利用registers
     - Def：如果registers數量夠多，則每一
       所以在context switching時，OS只要
     - 優點：負擔最小(速度最快)
       (∵避免Memory store/Load之時間)
     - 缺點：不適用於register數量少之情況
   - 法二：利用Thread(light-weighted process)來
     得以降低context switching負擔
     ∵同一個process內的Threads彼此共享code
     ∴私有資訊不多，context switching時不需
   - 法三：System process擁有自己的register se
     當user process與system process之間的conte

ess的執行狀態(eg. pc, CPU register store in PCB)

ware因素

己的registers set

isters set即可完成

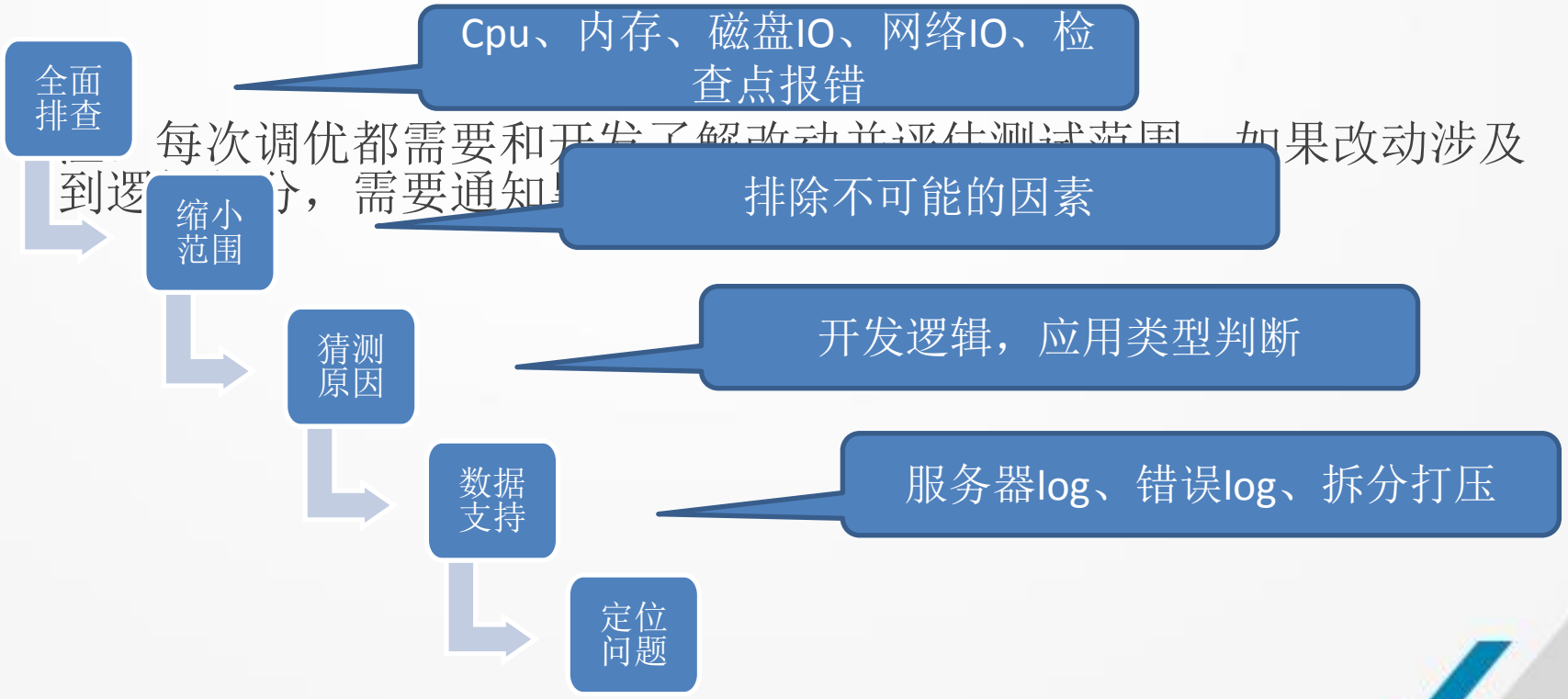
file, OS resources

tore/Load量少

變Register Set指標即可

问题定位&优化

- 知己知彼
  - 服务端逻辑、使用的技术、其他外围知识掌握。



每次调优都需要和开发了解改动并评估测试范围，如果改动涉及到逻辑拆分，需要通知

报告产出

- 时机：性能测试达到预期结果之后。
- 发送人：同测试方案
- 工具：Lranalysis，nmon。
- 包含信息：
  - 整体结论
  - 具体性能图标及对应子结论
  - 测试过程中发现的问题及解决方案
  - 风险备忘
  - 实际打压分组
  - 上线域名信息

Q&A