

6 代监控数据中心

文档版本:1.0

撰写时间:2024-4-22

最后修改日志:

目录

数据中心软件系统概述.....	4
数据中心设备概述.....	4
部署数据中心.....	5
数据中心操作系统需求.....	7
回到主题.....	8
现在数据中心已经可以运行,但没有策略支持.....	8
解释一下什么是策略.....	9
在真实的运营环境下 CPM 是一个独立服务.....	9
系统策略.....	9
运营级部署第一步,首次运行.....	10
数据引擎工作参数.....	12
首次运行的 RealTimeVideo2 配置范例.....	22
运营级部署第二步,监视数据中心的方法.....	23
理解子库工作机制.....	23
理解 CPU 使用率.....	23
使用自带工具分析单进程内核时间占用率.....	24
在数据中心运行中,监视句柄和线程变化.....	24
监视磁盘.....	25
建议专机专用.....	25
运营级部署第三步,部署群集 IP 转发.....	25
运营级部署第四步,部署 AP 服务器(GPU 群集).....	25
运营级部署第五步,部署外围服务器:直播服务器,web 服务器,CS 服务器,小程序服务器,远程桌面.....	25
运营级部署第六步,拓扑网络交换机优化.....	26
运营级部署第七步,给所有服务器接入进程保护系统.....	26
帮助数据中心二次开发.....	26
并入的模块都会有 Test 方法.....	27
砍而不删.....	27
蝴蝶效应下的崩溃记忆.....	27
关闭数据中心的方法.....	28
数据中心硬盘坏了怎么办.....	28
Z-AI 自带远程桌面.....	28
6 代监控的数据中心服务是不可复制的独特程序.....	28

数据中心软件系统概述

6代监控数据中心的主要工作是大数据存储,线程调度数据结构,高流量通讯.

在大数据存储环节主要运行 3 套数据引擎,这些数据引擎是使用 ZDB2 体系量身设计的专用引擎.

- AI 识别数据引擎:这是由 AP 服务器群集传来的 AI 识别结果,具体规模不好说,当 AI 数据为 0,服务器开机 5 分钟以后用 `zdb2_info` 命令查出来会达到 200 万条.
- H264 片段数据引擎:由 AP 服务器对监控视频重建编码后生成的 h264 裸码流(下个版本可能会使用 h265),不含音轨数据,500 路 2k 监控头的单日数据规模在 50-80TB 间,该数据引擎充当 NVR 群集的替代功能.
- Forever 数据引擎:项目中期由甲方提出的需求,用于解决摘录问题,当发生条件识别,把视频+AI 一并摘录下来并永久保存.该设计后面被判定为失误,数据中心保留了 Forever 数据引擎,但处于空置不用状态.

高流量通讯是指在多张万兆级电口+光口的物理网卡高负荷工作,该环节主要使用 ZNet 体系中的 C4 框架.这里细节非常多,本文后续会详细介绍.

线程调度是将高流量与大数据引擎相结合,例如 AI 识别数据引擎有 20 个子库,运营期间就会有 20 个 cpu 占用率 100%的线程,线程调度会在高流量进来时用线程执行存储流程.另外,在大数据查询,下载,也会使用线程.整个数据中心实现的技术面是以堆算法+数据结构完成,这些算法+数据结构也都工作于线程.

数据中心设备概述

数据中心对 CPU,内存体量,阵列规模,有所要求.

- CPU:在数据中心服务器启动后,可以有 100 个 cpu 占用率达到 100%的线程,也可以只有 10 线程,这取决于物理阵列规模,数据中心对 cpu 的要求是核心数量或超线程数量,例如 20 盘的 hdd 阵列,这时数据引擎至少会开 20-40 个子库,等同于数据引擎使用了 40 条线程.
- 内存体量:内存规模决定了 HDD 阵列系统存储效率,最低要求:内存体量=阵列存储体积*(0.5%..1%),例如 100TB 的阵列存储空间,内存最少需要 500G 规格.在实际运营中 500G 内存大约可以支持候 1-5 分钟内存 IO 读写分离暂存,这种机制会截获写入 IO 数据,存储与内存中,当调用 flush,内存会往阵列 IO 里面物理写入.物理写入时,如果内存体量可以容纳本次 flush 规模,IO 会瞬间返回,否则写入 IO 会发生等待,而 IO 等待会传导给对数据的查询.在正规使用情况下,阵列服务器的内存槽应该会全部插满.
- 阵列规模:SSD 盘 IO 高容量小,HDD 盘 IO 低容量大,最小规则建议 20TB 向上,组盘期间如果走 HDD 路线需计算好带宽开销.

建议自己 diy 数据中心服务器,内存槽子不够插就用双路主板,主板只要支持 OCuLink,SlimSAS 这类接口,可以一根线接 4-8 盘.只要服务器机箱有物理盘架,主板 cpu 内存这些会非常好配.

6代监控的 HDD 避免选择监控红盘,寻址比黑盘慢 10 倍,这种盘只能写,查询时寻道时间非常久,建议往企业级 hdd 方向靠.在存储做优化环节,红盘 HDD 会让人吐血,内存如果能跟上,黑盘 HDD 只会让人折腾几天,SSD 盘几乎无需优化.

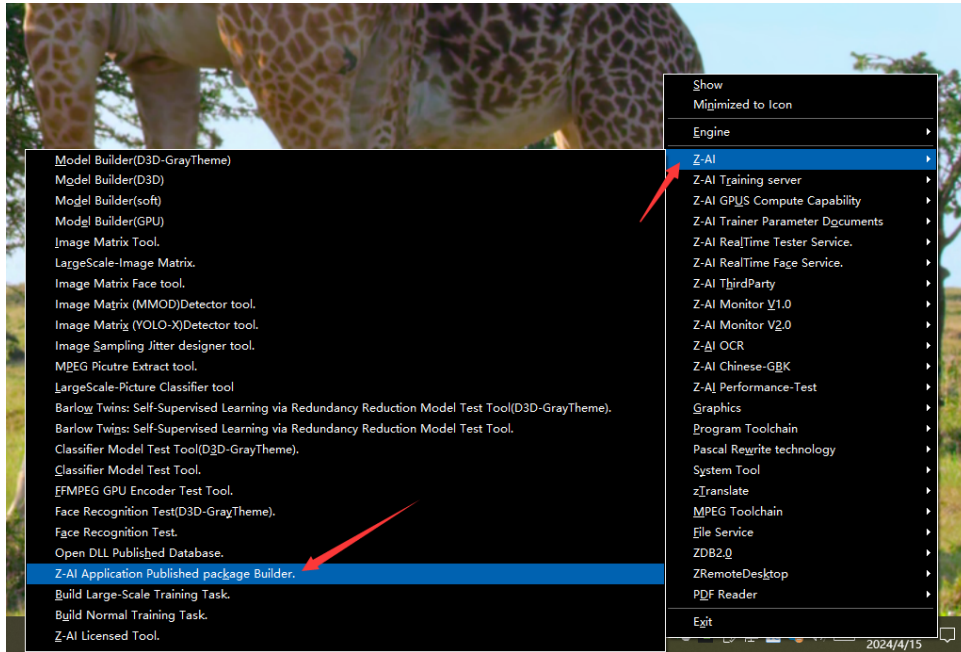
上面只是建议,因为阵列服务器会花钱的,看在钱的面子上,大家会自己去研究琢磨.

部署数据中心

数据中心分为 GPU 版和 AMD64 版,两个版本都要求服务器在 CPU,内存,阵列,网络,这些环节达到 hpc 级。

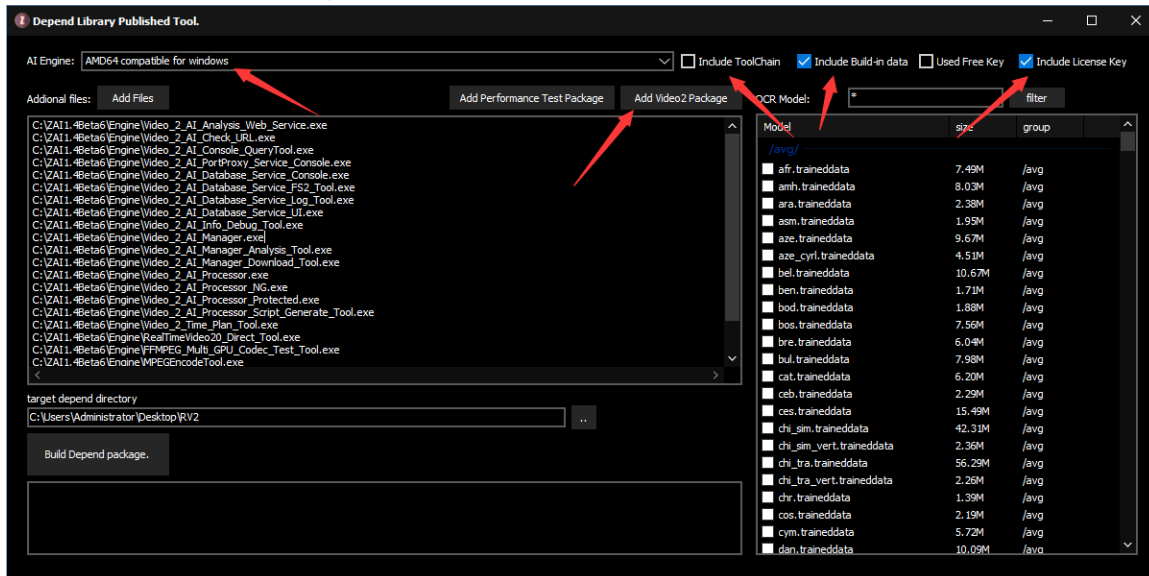
部署数据中心前需要在本机有一个 Z-AI 的副本,如果没有 Z-AI 可以通过 <https://zascal.net> 下载.安装 Z-AI 会需要授权码,申请试用,以及老的授权用户无法安装,可以联系 qq600585 帮忙解决.在 2024-3 到 5 月间,Z-AI 的美国服务器一直处于故障状态。

Z-AI 安装完成后,在主工具链启动应用程序发行系统

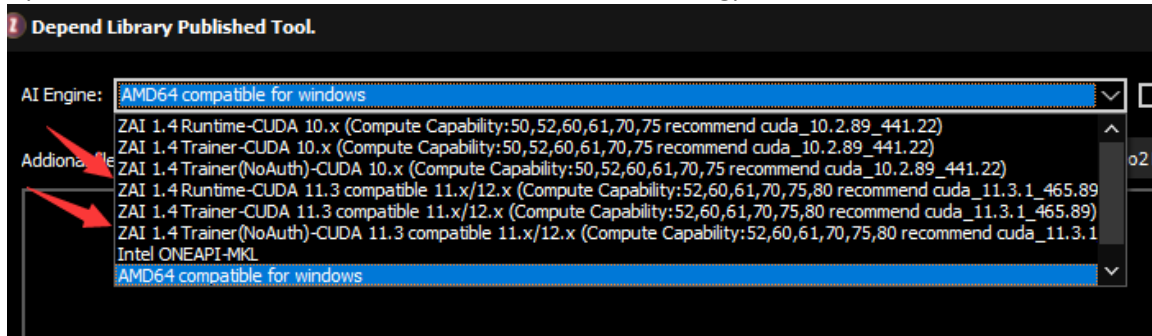


Add Video2 Package

操作环节按下列箭头进行,该按钮会生成 6 代监控的所有依赖文件。

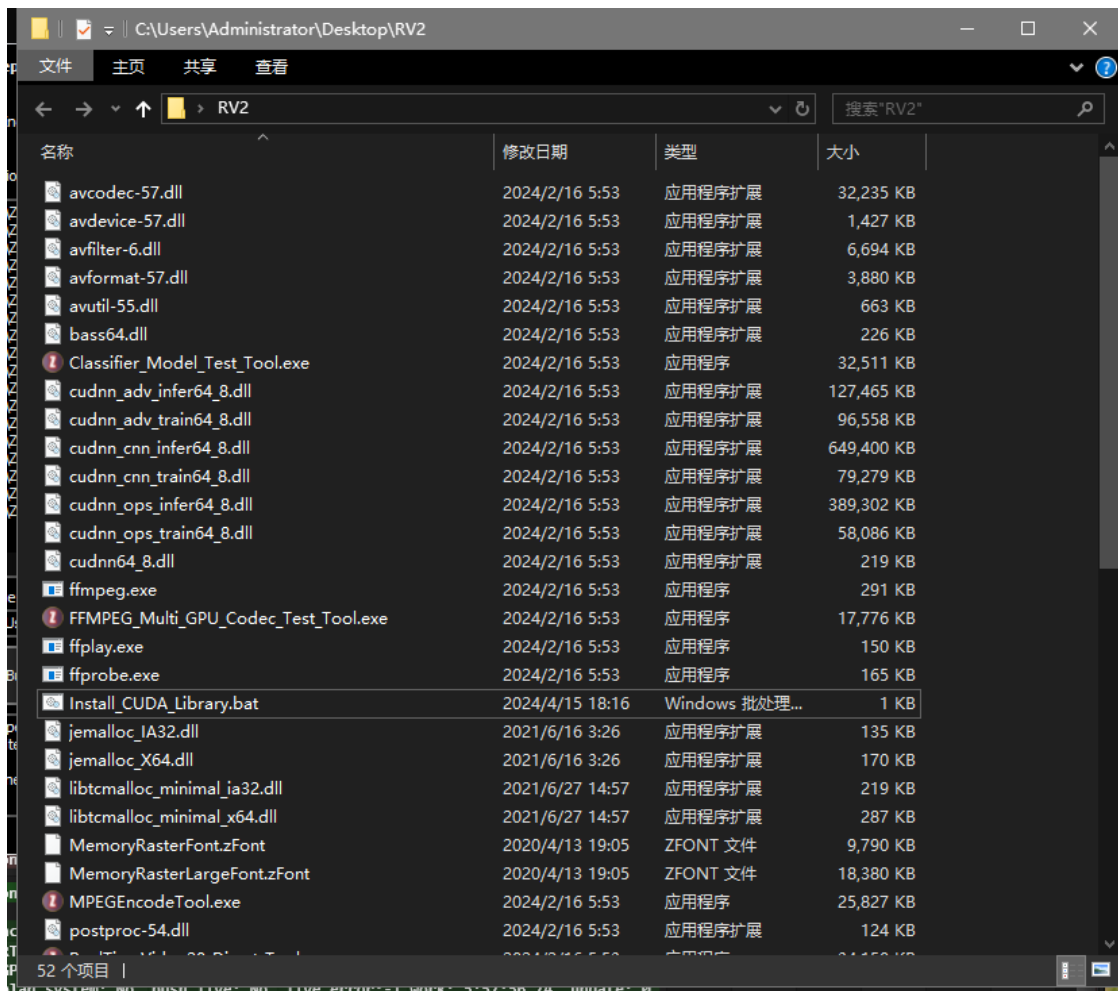


如果数据中心有 GPU 设备(必须是 nv 系的 gpu),建议在这里选择 RunTime 或则 NoAuth 的计算引擎来部署. Gpu 设备必须是数据中心规格,切勿使用 3090,4090,家用 gpu 有编码器限制!



设置完成后点该按钮执行部署

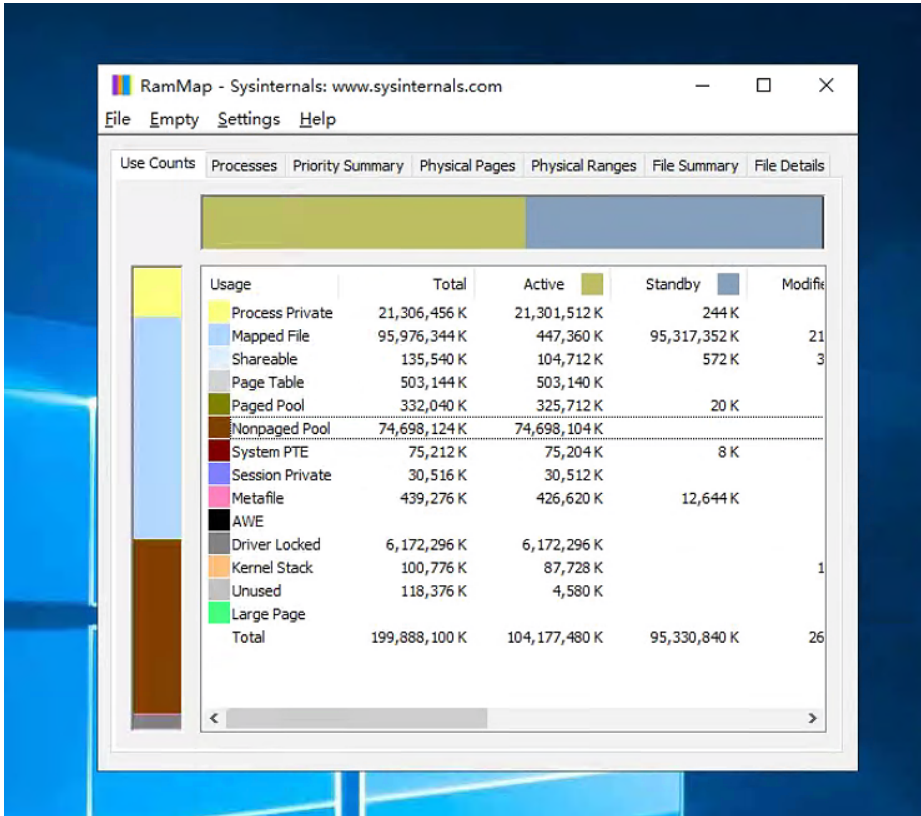
完成后在目标目录会有许多文件,这时候,先不管.



数据中心操作系统需求

建议数据中心安装 Windows Server 2022 Desktop Experience,因为数据流量非常大,曾经我们在交付项目中使用的 windows server 2019 出现过下图的操作系统 bug,感兴趣可以搜索一下非页面内存发生泄漏的严重后果.

当数据中心发生泄漏期间,网络流量大约在 3000Mbps/s,当物理断网后泄漏停止,初步判定与网络相关,后来尝试过更新系统驱动,更新 bios,因为服务器主板自带几个网口,同时也有别的网卡,还尝试过,走正版渠道,结果,均无法解决非页泄漏问题.最后拿 ipmi 重装系统使用 2022,接下来,重新编译数据中心,检查所有跟 socket 相关的 received buffer 这类东西,具体改动现在已经无从考证.最后谜之泄漏被谜之解决.真实经历,仅供参考.



非页面泄漏的症状:系统启动后,刚开始非页会低于 100M,这时候断网,非页开始慢慢下降到 10M,然后开流量,大约 3 分钟内,非页飙升到 70G,这时候再断网,等 30 分钟,非页一直保持在 70G.

在操作系统环节,建议沿走我们的老路,选择 IDC 用的品牌主板+IDC 用的品牌网卡跑 server 2022,避免山寨硬件,高流量服务器网络和 pc 网络是不同的.

如果大家在数据中心环节遇到类似情况,恭喜中奖!麻烦详细记录一下问题发生的环境,不要像我们这样匆忙间只有几张截图.!

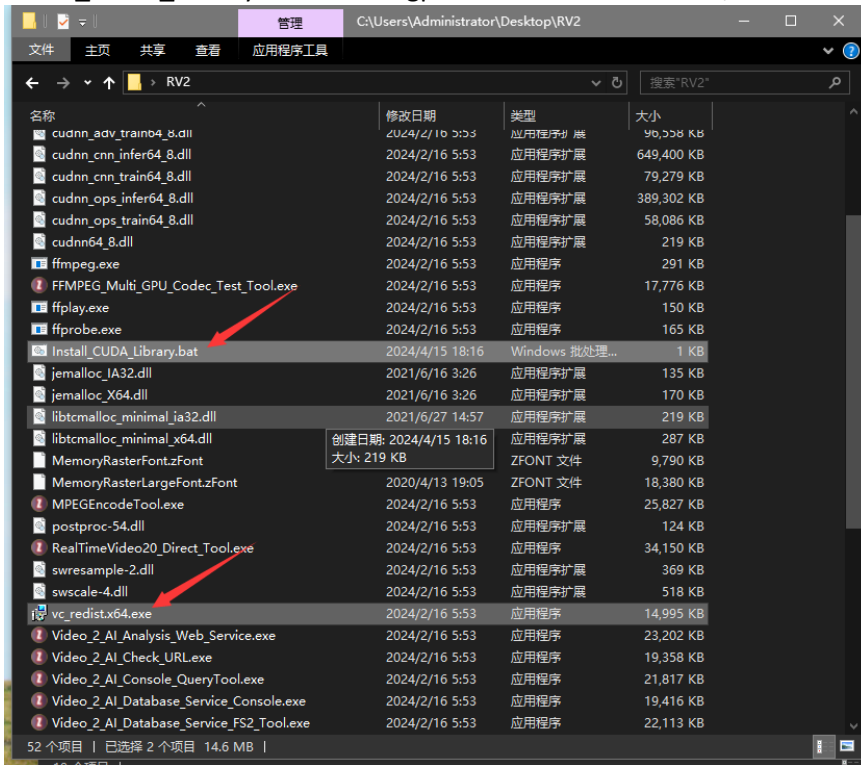
回到主题

在服务器系统环节,如果选择 AMD64 这类非 GPU 的数据中心系统,可以选择 Hyper-V 来拉数据中心(我们还没有尝试过 HV 下的系统跑高流量),亦可直接让数据中心工作与裸系统中(这是我们切身使用的方式).

当使用 Z-AI 的发行工具完成部署以后,只需要把这些文件 copy 到目标操作系统中即可.

在依赖库环节,需要安装下图箭头所指的 `vc_redist.x64.exe`,这是 vs2017 的依赖库.

`Install_CUDA_Library.bat` 会部署 gpu 所使用的 cudnn 环境,需要先安装 cuda,然后再运行 `Install_CUDA_Library.bat`



经验技巧:在 windows server(2016 或则以后的版本)家族,文件的 copy,可以被多网卡同步加速,假如有 4 网卡,那么物理 copy 速度会是单网卡的 4 倍,多网卡 copy 加速对大文件来说是非常棒的机制,必须是大文件,因此使用 zip,rar 打个包来传会非常快.

现在数据中心已经可以运行,但没有策略支持

下面两个程序都是数据中心服务器,直接启动即可.不要求启动顺序.

- Video_2_AI_PortProxy_Service_Console.exe:负责运行 CPM 服务
- Video_2_AI_Database_Service_Console.exe:负责运行数据中心主体服务

解释一下什么是策略

策略需要区分系统和部署,就部署而言,在 ZNet 的 C4 体系中,服务器是互相依赖的,并且,所有服务器的启动模式都是脚本化的.在数据中心,总共有 5 种服务.

- **RealTimeVideo2**:数据中心核心服务器,所有的高流量通讯,高度复杂的线程,跑磁盘阵列的数据引擎都在这里,该服务涉及面非常大,档后续会有大量章节围绕该服务进行陈述.
- **RealTimeVideo2_FS2**:数据中心专用的文件系统 FS2.0,主要提供给 AP 服务器调试 AI 数据使用,目前是空置状态
- **RealTimeVideo2_Log**:服务器工作日志信息,该服务目前基本处于空置中,因为 AI 数据量非常多,经常单日 log 文件达到 500G 规模,根本没法使用,直接做空置处理(空置化处理是因为砍掉它会修改运营服务器,容易滋生隐藏 bug),Log 的空置不是完全空置,是基本上空置.
- **DP**:C4 的入网服务器,这里不做解释,细节去看 ZNet 项目, <https://github.com/PassByYou888/ZNet>
- **CPM**:群集端口转发服务,这也就是 6 代监控支持的 IP 转发,AI 盒子,拓展的 app 服务器,例如 web,都使用 CPM 来转发

如果以策略方式来启动数据中心,在 windows shell 中命令行书写如下,作用是通过 `Video_2_AI_Database_Service_Console.exe` 程序同时启动 5 个服务

```
Video_2_AI_Database_Service_Console.exe Service("0.0.0.0", "127.0.0.1", "9290", "RealTimeVideo2|RealTimeVideo2_FS2|RealTimeVideo2_Log|DP|CPM")
```

另一种写法是只启动 3 个关键服务,这种写法数据中心也是可以正常运行的

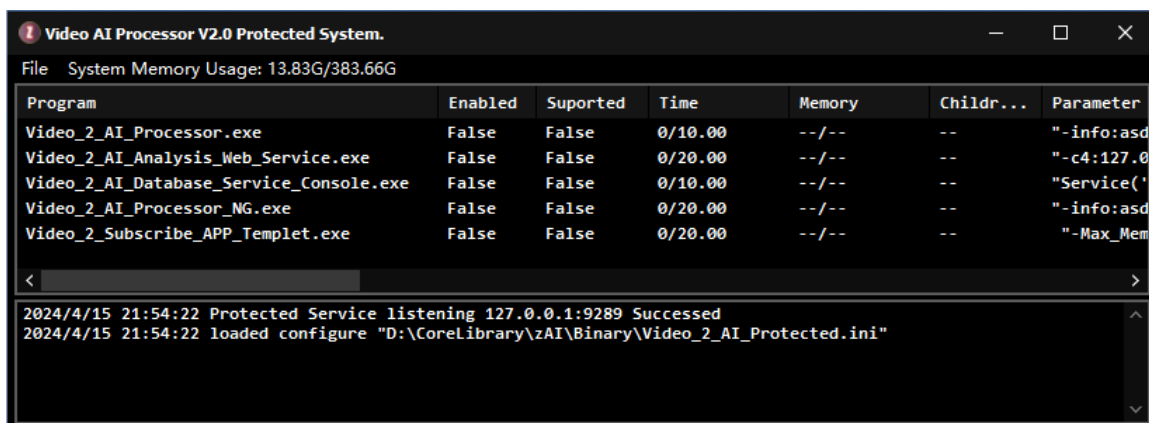
```
Video_2_AI_Database_Service_Console.exe Service("0.0.0.0", "127.0.0.1", "9290", "RealTimeVideo2|DP|CPM")
```

在真实的运营环境下 CPM 是一个独立服务

这是因为 RealTimeVideo2 服务占据了 90%的网络流量+CPU 资源,CPM 作为 IP 转发的中心服务会或多或少发生延迟,在 6 代监控的应用端,有大量的实时监控都使用 CPM 播放 RTSP/RTMP/HLS/WebRTC,哪怕是 0.1s 延迟也会发生画面顿挫问题.在独立服务或则独立 IP 下跑 CPM 服务可以很好解决延迟问题.

系统策略

系统策略就是如何启动,如何维护数据中心在运行中的稳定性,使用 `Video_2_AI_Processor_Protected.exe` 来干,这是一个进程+操作系统的守护程序.因为可以支持内存 NUMA+CPU 亲和性,对于数据中心来说这是最好系统策略.



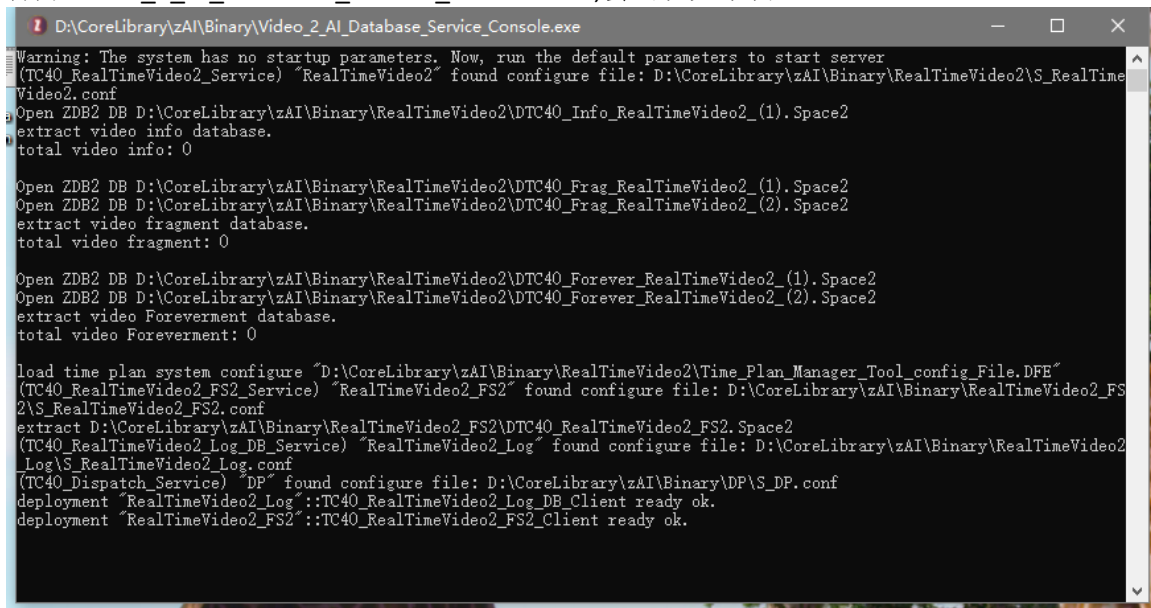
使用守护程序运行数据中心服务只需要将正常的 windows shell 程序按下列参数填入即可.



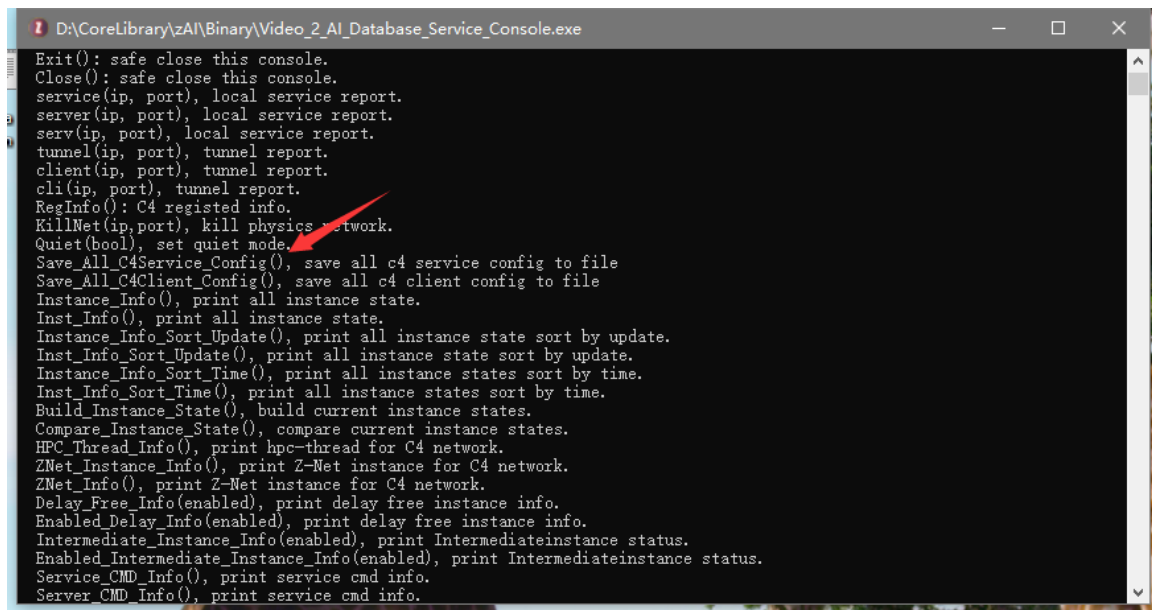
运营级部署第一步,首次运行

任何 C4 的服务都必须经过首次运行来得到内置参数,

打开 Video_2_AI_Database_Service_Console.exe,会出现如下窗口

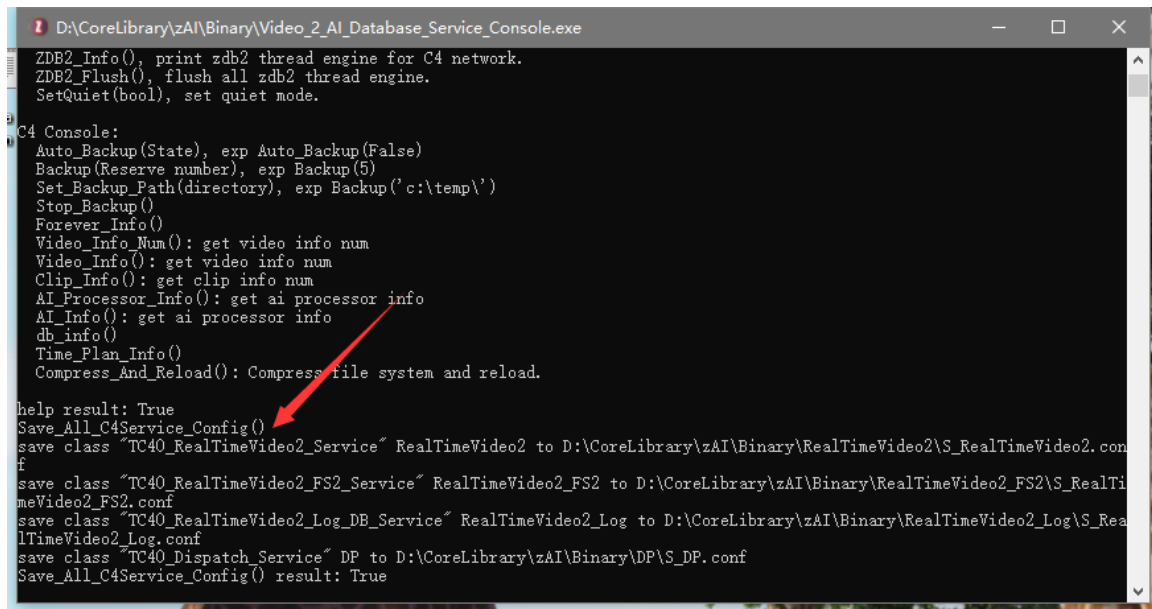


这时候,在控制台敲入 **help** 命令,然后往上翻,会看到箭头的 **Save_All_C4Service_Config()** 命令



```
D:\CoreLibrary\zAI\Binary\Video_2_AI_Database_Service_Console.exe
Exit(): safe close this console.
Close(): safe close this console.
service(ip, port), local service report.
server(ip, port), local service report.
serv(ip, port), local service report.
tunnel(ip, port), tunnel report.
client(ip, port), tunnel report.
cli(ip, port), tunnel report.
RegInfo(): C4 registered info.
KillNet(ip,port), kill physics network.
Quiet(bool), set quiet mode.
Save_All_C4Service_Config(), save all c4 service config to file
Save_All_C4Client_Config(), save all c4 client config to file
Instance_Info(), print all instance state.
Inst_Info(), print all instance state.
Instance_Info_Sort_Update(), print all instance state sort by update.
Inst_Info_Sort_Update(), print all instance state sort by update.
Instance_Info_Sort_Time(), print all instance states sort by time.
Inst_Info_Sort_Time(), print all instance states sort by time.
Build_Instance_State(), build current instance states.
Compare_Instance_State(), compare current instance states.
HPC_Thread_Info(), print hpc-thread for C4 network.
ZNet_Instance_Info(), print Z-Net instance for C4 network.
ZNet_Info(), print Z-Net instance for C4 network.
Delay_Free_Info(enabled), print delay free instance info.
Enabled_Delay_Info(enabled), print delay free instance info.
Intermediate_Instance_Info(enabled), print Intermediateinstance status.
Enabled_Intermediate_Instance_Info(enabled), print Intermediateinstance status.
Service_CMD_Info(), print service cmd info.
Server_CMD_Info(), print service cmd info.
```

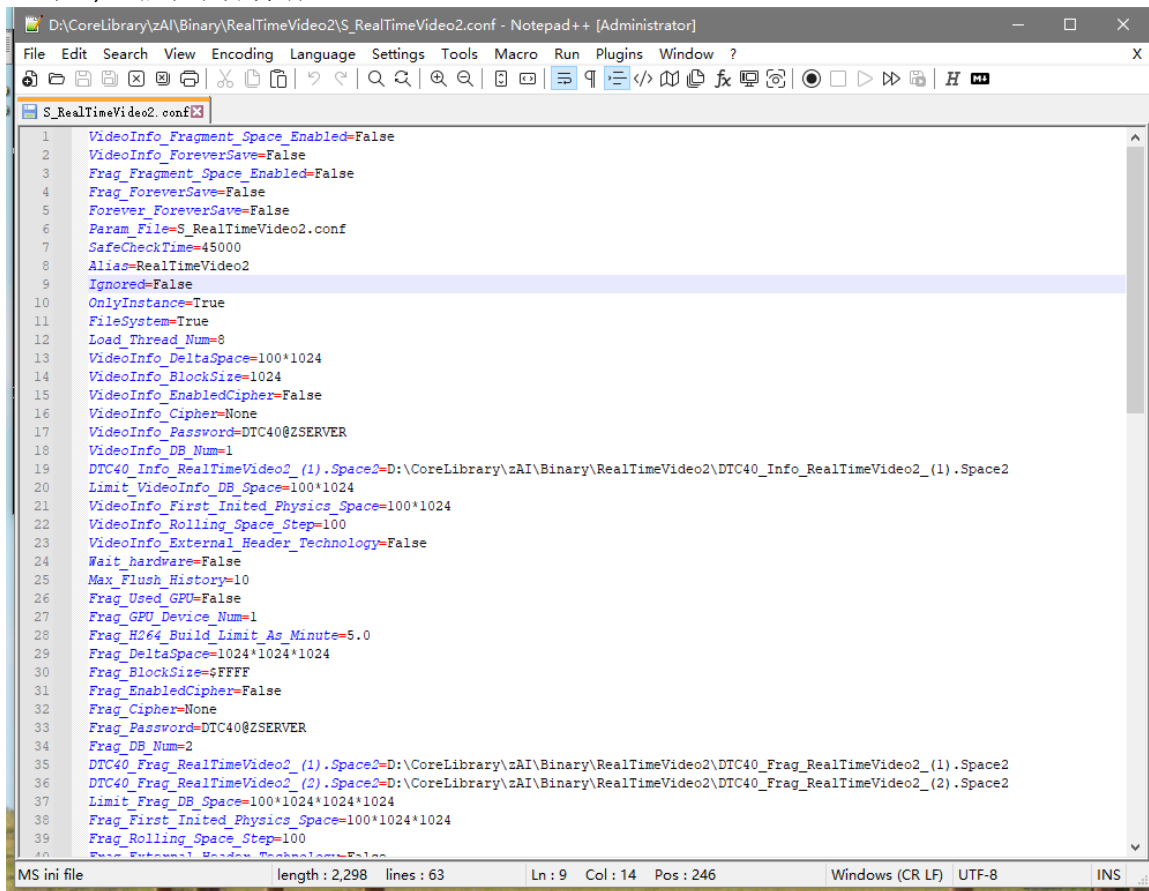
然后敲入 **Save_All_C4Service_Config()** 命令,如下箭头所指,这时候,所有已经启动的服务参数都会生成在对应的文件中.



```
D:\CoreLibrary\zAI\Binary\Video_2_AI_Database_Service_Console.exe
ZDB2_Info(), print zdb2 thread engine for C4 network.
ZDB2_Flush(), flush all zdb2 thread engine.
SetQuiet(bool), set quiet mode.
C4 Console:
Auto_Backup(State), exp Auto_Backup(False)
Backup(Reserve number), exp Backup(5)
Set_Backup_Path(directory), exp Backup('c:\temp\')
Stop_Backup()
Forever_Info()
Video_Info_Num(): get video info num
Video_Info(): get video info num
Clip_Info(): get clip info num
AI_Processor_Info(): get ai processor info
AI_Info(): get ai processor info
db_info()
Time_Plan_Info()
Compress_And_Reload(): Compress file system and reload.
help result: True
Save_All_C4Service_Config()
save class "TC40_RealTimeVideo2_Service" RealTimeVideo2 to D:\CoreLibrary\zAI\Binary\RealTimeVideo2\S_RealTimeVideo2.conf
save class "TC40_RealTimeVideo2_FS2_Service" RealTimeVideo2_FS2 to D:\CoreLibrary\zAI\Binary\RealTimeVideo2_FS2\S_RealTimeVideo2_FS2.conf
save class "TC40_RealTimeVideo2_Log_DB_Service" RealTimeVideo2_Log to D:\CoreLibrary\zAI\Binary\RealTimeVideo2_Log\S_RealTimeVideo2_Log.conf
save class "TC40_Dispatch_Service" DP to D:\CoreLibrary\zAI\Binary\DP\S_DP.conf
Save_All_C4Service_Config() result: True
```

然后敲 **exit** 关闭服务器,并且打开 RealTimeVideo2 服务所生成的那个.conf 配置文件

这里的主要是使用磁盘阵列的数据库引擎参数,优化数据中心的存储 IO,就是在这里面来干.这些参数非常庞大,并且晦涩,也非常不易讲清.



```
1 VideoInfo_Fragment_Space_Enabled=False
2 VideoInfo_ForeverSave=False
3 Frag_Fragment_Space_Enabled=False
4 Frag_ForeverSave=False
5 Forever_ForeverSave=False
6 Param_File=S_RealTimeVideo2.conf
7 SafeCheckTime=45000
8 Alias=RealTimeVideo2
9 Ignored=False
10 OnlyInstance=True
11 FileSystem=True
12 Load_Thread_Num=8
13 VideoInfo_DeltaSpace=100*1024
14 VideoInfo_BlockSize=1024
15 VideoInfo_EnabledCipher=False
16 VideoInfo_Cipher=None
17 VideoInfo_Password=DTC40@ZSERVER
18 VideoInfo_DB_Num=1
19 DTC40_Info_RealTimeVideo2_(1).Space2=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Info_RealTimeVideo2_(1).Space2
20 Limit_VideoInfo_DB_Space=100*1024
21 VideoInfo_First_Inited_Physics_Space=100*1024
22 VideoInfo_Rolling_Space_Step=100
23 VideoInfo_External_Header_Technology=False
24 Wait_hardware=False
25 Max_Flush_History=10
26 Frag_Used_GPU=False
27 Frag_GPU_Device_Num=1
28 Frag_H264_Build_Limit_As_Minute=5.0
29 Frag_DeltaSpace=1024*1024*1024
30 Frag_BlockSize=$FFFF
31 Frag_EnabledCipher=False
32 Frag_Cipher=None
33 Frag_Password=DTC40@ZSERVER
34 Frag_DB_Num=2
35 DTC40_Frag_RealTimeVideo2_(1).Space2=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Frag_RealTimeVideo2_(1).Space2
36 DTC40_Frag_RealTimeVideo2_(2).Space2=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Frag_RealTimeVideo2_(2).Space2
37 Limit_Frag_DB_Space=100*1024*1024*1024
38 Frag_First_Inited_Physics_Space=100*1024*1024
39 Frag_Rolling_Space_Step=100
40 Frag_External_Header_Technology=False
```

数据引擎工作参数

该环节紧接上文,这里会比较复杂,繁多,此处可以先初略过一下,待实际部署时可以使用搜索方式进行

SafeCheckTime=45000

SafeCheckTime 由 C4 框架定时触发,在数据中心服务器中,每隔 45 秒会做一次 flush 写 IO 操作

如果打开了读写分离机制,例如 Frag_Fragment_Space_Enabled=True,并且希望在内存保存 2 分钟内的所有写 IO 数据,那么可以写成 SafeCheckTime=2*60*1000

当 SafeCheckTime 触发时会使用线程引导物理 flush 机制启动,因此不会阻塞网络通讯,但会影响数据引擎的读操作,在 flush 运行期间,读操作会等待 flush 结束,而读操作会传递给高级的视频查询,视频下载,用户端 app 将会出现短暂卡顿感.

Flush 如何结束

- 当参数 Wait_hardware=True 时,这时 flush 会等待硬件 IO 完成写入.
- 当参数 Wait_hardware=False 时,Flush 不会等待硬件 IO,而会直接写入缓存,当缓存不够用,会等待缓存刷新.

说明:flush 操作是并发 IO,例如有 20 个数据库,会是同时执行 flush,不会等 flush 结束再做下一个 flush.

Alias=RealTimeVideo2

C4 框架特殊数据,别名,一般用于搜索服务器群,在数据中心服务器该选项无用,忽略即可

Ignored=False

DP 服务运行同步机制时忽略该服务,这一部分去了解 C4 框架,<https://github.com/PassByYou888/ZNet>

OnlyInstance=True

独立化实例,这一部分去了解 C4 框架,<https://github.com/PassByYou888/ZNet>

FileSystem=True

这是一个安全选项,表示是否会包含底层 DT 框架中的文件系统。

DT 框架的文件服务总是单目录文件服务,一般是进程当前目录,并且不提供文件列表和目录切换。

FileSystem 无论开关都是安全的.细节去了解 C4 框架,<https://github.com/PassByYou888/ZNet>

Load_Thread_Num=8

讲解线程前,必须说明,zdb2 数据需要经过一次 load 才会工作,load 作用是建立数据结构从而达到提升数据库工作效率目的,并且,必须是逐条 load:ZDB2 的 load 模型衍生很长,load 需要区分:全条目 load,条目块 load,最后是 external-header 方式 load.

- **全条目 load:**zdb2 的数据条目是数据体,在代码中会出现很多 body 这类名词,例如单个条目体积在 10M,这时候 load 10 万条,数据量会接近 1TB,对于 hdd 来说 load 将非常耗时.全条目 load 大多用于非常小的数据 load,例如 AI 识别结果的数据结构.
- **条目块 load:**zdb2 是空间数据库,与硬盘分区类似,zdb2 会有一张分区表,这张表会指向数据块,每一个数据条目都会占据若干数据块.全条目 load 会读取若干数据块,而条目块 load 则是指定读取数据条目的其中一个块,从而减少阵列盘的 io 开销,达到加速的作用.例如 h264 的监控视频数据大都在 10-300M 不等,这种是走的条目块 Load.在大型阵列工作期间,hdd 的 load 会有寻道时间,即使减少 io 读取体量仍然非常耗时.所以条目块 load 模型最终是作为后备 load 方式使用.
- **External-header 方式 load:**这种方式是生成 load 的预置数据,在设计专用数据引擎时,都会针对目标需求设计一些结构体来提速工作,当数据库关闭再重新打开时,这些结构体需要 load 进内存才能工作.External-header 方式 load 是提前生成一个巨型数据结构体,这样来替代前面两种 load.在监控视频数据引擎中,主要使用这种方式,只有在找不到可以被 load 的预置数据时,会使用条目块 load.

ZDB2 的 Load 每次只读一个库,每个数据引擎都会有一堆子库,ZDB2 在进行大规模 Load 时会逐库读,例如条目 1,2,3 分别存放于 3 个子库,Load 时不会按 1,2,3 顺序进行,而是直接一次性读完全库再读取下一个.

从 Load 到线程是流水线模型:Load 是连续性流程,把 load 完成的数据放到内存,然后再使用 IO-Thread 模型对这些 load 完的数据进行结构化处理,这里可以理解 2 个闸口,一个负责物理 Load,另一个负责结构化计算.

Load_Thread_Num=8 表示结构计算会有 8 条线程以抢占式方法从 Load 闸口抢数据进行结构计算,也许大家会疑惑,为什么用 8 条线程而不用 100 条线程提速?这点我在 ZNet 手册已经解释过,8 在 hpc 的机器上是速度+效率最优的参数.使用 100 条线程反而更慢.

注意:如果使用 External-header 方式 Load,那么 Load_Thread_Num 会无效,External-header 方式会把最终的数据结构直接通过 load 方式生成出来,不再需要走数据遍历流程.同时 External-header 方式也是瞬间完成的.这是最高的效率,远超线程 Load 方式.

External-header 方式是内部结构被程序化处理的产物,不是通过简单开关控制,必须是数据引擎支持这种技术.AI 状态数据引擎天生不支持 External-Header 技术,因为 AI 状态数据非常小,且数量繁多,直接全部仍内存,减少硬盘 IO 开销.而视频数据引擎则是全方位支持 external-header 技术的.

当数据中心的存储规模达到或则接近 50TB 以后,会出现 video_info 数据库每次打开 load 等 10 分钟,而 video_info 的全部数据 100GB 不到.视频数据库可以高达 49TB,每次打开都是瞬间完成,发生这种现象就是 external-header 技术.

Wait_hardware=False

Flush 时是否等待所有数据全部完成物理写入。

当 Wait_hardware=True 此处必须理解 flush 的前置流程是将内存数据依次按 position 排序,坐标由小到大以物理方式写入.这一步的写入规模会由 SafeCheckTime 决定.如果 SafeCheckTime 是 5 分钟,期间使用的内存规模大约可以达到 100GB,这时候,Flush 的等待时间大约为 100GB/每秒阵列吞吐量+2 秒.

当 Wait_hardware=False,那么 Flush 操作将会是无等待的,但这种无等待并不是立即,会发生 2 秒的作用时间,这 2 秒大致可以理解成为页面内存待写入数据传递到阵列非页面内存缓存内存区所使用的时间.这是在内存足够使用的情况下发生的.如果内存紧缺,这一步会等缓存刷新.

如果 Wait_hardware=True,这是存储设备为 SSD,M2 这类高速介质时给的,在 hdd 和低速 ssd 环境不可以等.曾经我们运营项目都以 Wait_hardware=True 状态跑,有一次,甲方在 hdd 阵列 copy 很大的东西,期间耗时大约 5 分钟,在 copy 过程中刚好触发 flush,然后,flush 一直处于等待状态,因为 hdd 的读写一旦叠加性能会变成原来 1/10,但是网络仍然还接收巨量的视频数据,一直往写缓存里面无限堆,结果是,服务器触发安全条件被强制关闭,等同于崩溃,并且丢失很多数据.

Max_Flush_History=10

Flush 在执行前会生成一个历史文件,该文件会记录全部写入,并且会以物理方式关闭,然后才执行 Flush.在任何时间,如果发生断电,系统都以历史文件进行恢复:就是从最早的历史文件依次写入.

说明一下读写分离技术的恢复模型:读写分离可以每次只恢复最近写入状态,但是这样会要求每次 flush 必须等待物理 IO,因此 flush 的恢复一律是全历史恢复.例如 SafeCheckTime 的间隔是 5 分钟,每次 flush 的规模 100GB,10 个历史文件就会有 1TB 的写入历史,当发生断电重启,这 1TB 写入数据会从最早到最新的重新写入.从而实现断电重启的自动化恢复.

VideoInfo_Fragment_Space_Enabled=False

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

VideoInfo_Fragment_Space_Enabled 表示是否启用 IO 读写分离技术.

如果不启用读写分离,那么所有的写都会是实时的,一旦发生异常关机,可能造成不可逆的数据丢失.

VideoInfo_ForeverSave=False

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

VideoInfo_ForeverSave 表示持续使用数据库.

如果 VideoInfo_ForeverSave=False 在关闭数据中心服务时会直接删除数据库,同时在启动时也会先删除老数据库然后再建立一个新数据库.等同于数据库只存在于运行期间,服务器关闭即数据全部消失.

VideoInfo_DeltaSpace=100*1024*1024

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

ZDB2 是一种空间数据库,头部是一张空间分配表,body 是数据主体,因此 ZDB2 的数据文件不可以随意增容,需要使用增容机制(AppendSpace),并且每次增容都要指定空间尺寸.ZDB2 在运行过程中数据会不断的追加,增容机制也是不断在反复运行.

VideoInfo_DeltaSpace=100*1024*1024 表示每次增容 100M,增容机制会自动化触发,每次在写入时发现空间不足就会触发增容机制.

VideoInfo_BlockSize=1024

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

ZDB2 是一种空间数据库,头部是一张空间分配表,在这张空间表中每一项都是一个 block,这个 block 会用于保存数据,例如待数据的数据为 1.5k,而 blocksize=1024,就会需要 2 个 block.

VideoInfo_BlockSize=1024 表示每个 block 是 1k,这是因为 AI 数据都很小,在某些场景,一张图会检测出几十个框框,AI 数据可能会在 4-8k 之间,这时候,这条 AI 数就会用掉 4-8 个 block,这种情况是少数,大多数情况下 AI 数据就是 1k.

VideoInfo_EnabledCipher=False

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

VideoInfo_EnabledCipher 表示是否启用加密编码器

VideoInfo_Cipher=None

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

VideoInfo_Cipher 是指定加密编码器算法,可以使用的算法如下

None,DES64,DES128,DES192,Blowfish,LBC,LQC,RNG32,RNG64,LSC,XXTea512,RC6,Serpent,Mars,Rijndael,TwoFish,AES128,AES192,AES256

建议加密算法:RC6,Serpent,Mars,Rijndael,TwoFish,AES128,AES192,AES256

注意:加密算法会影响 IO 性能,如果数据量达到 1000M/s,开加密以后大约会降级为 500M/s,IO 性能会下降 50%左右.

VideoInfo_Password=DTC40@ZSERVER

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

VideoInfo_Password 表示加密算法的 Key 算子, DTC40@ZSERVER 是 C4 框架的默认 Key

VideoInfo_DB_Num=1

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

VideoInfo_DB_Num 表示数据引擎的子库数目,这项参数的正确用法并不是直接给值就能完事,

VideoInfo_DB_Num 会影响后面的 DTC40_Info_RealTimeVideo2_(1).Space2,如果给 2 就会有 2 个 Key 值

- DTC40_Info_RealTimeVideo2_(1).Space2=xxx
- DTC40_Info_RealTimeVideo2_(2).Space2=xxx

技巧: VideoInfo_DB_Num 给值以后,立即保存,并且重启数据中心服务,使用 **Save_All_C4Service_Config()** 命令 [重生成参数](#)

DTC40_Info_RealTimeVideo2_(1).Space2=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Info_RealTimeVideo2_(1).Space2

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

非常重要:指定数据库的文件路径,如果阵列服务器内存不够,或则优化需求,这里直接给硬盘的路径,例如

d:\mydb\disk1\1.space

d:\mydb\disk2\2.space

以上是极限优化的做法,如果懒一点,可以直接组 10 盘阵列(后面会介绍 server2022 组阵列),然后把整个数据中心全部仍阵列里面运行,不需要单独指定子库.因为内存并不便宜,并且插槽有限,在内存不够的情况下,优先考虑以指定每个子库来跑.

Limit_VideoInfo_DB_Space=1024*1024*1024

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

Limit_VideoInfo_DB_Space 表示单个数据引擎子库的滚动临界点

当数据库规模达到该临界点,会以 VideoInfo_Rolling_Space_Step 参数执行删头机制,当子库规模达到或接近磁盘容量时,删头意味着新增数据必定发生 hdd 寻址延迟.删除大文件头部,再从头部重新写入,意味着 hdd 的机械寻址会由外向内以盘面半径反复定位.

在真实运营时,这是一个稳定性的重要转折点条件,增容模型往滚动模型切换以后会持续走滚动模型,如果数据中心走了极限优化,当接近或则达到滚动模型当天,应该全时间观察服务器稳定性,因为寻址延迟大概率会导致 IO 性能下降,传导效应会影响服务器的整体运行.只有在滚动模型稳定以后,服务器才可以是长时间稳定运行.

运营时的滚动模型通常会在 15-45 天以后,开始启动,视阵列容量规模而定.

Limit_VideoInfo_DB_Space=1024*1024*1024,表示单库临界点为 1GB

VideoInfo_First_Inited_Physics_Space=500*1024*1024

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

VideoInfo_First_Inited_Physics_Space 表示首次运行时,对数据库的初始容量,达到该容量会启用增容机制,一直会持续容量到达到 Limit_VideoInfo_DB_Space 参数的限制,然后进入滚动模型.

VideoInfo_First_Inited_Physics_Space 切勿给大,切勿给大,切勿给大,重要的事情说三次!

ZDB2 的空间结构是空间表在前面,数据存储在后面,每一次增容并不是重建空间表,内部设计是在尾部追加一张新的空间表,并在新空间表尾部给出数据空间,这一步可以理解成 Link 新的硬盘.当每次有数据保存时,对空间表的修改,会在局部范围中,如果初始容量给大,例如 5TB,这时候,每一次写入数据,修改空间表,在 6TB 物理空间的硬盘中磁头会来回跳,IO 能力变成 20%都有可能.

正确的做法是给一个合适的初始化容量,然后依赖于 VideoInfo_DeltaSpace 参数,一次一次的走增容机制,这样每次写入数据会是靠近的.这是很优化的存储模型.

VideoInfo_Rolling_Space_Step=100000

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

VideoInfo_Rolling_Space_Step 表示当子库数据容量达到 Limit_VideoInfo_DB_Space 所指定的额度,立即移除头部的 10 万条数据.

VideoInfo_External_Header_Technology=False

凡 VideoInfo 开头=存储 AI 识别状态的数据引擎

在 VideoInfo 数据引擎中没有使用 External-Header 技术,无论是否开关数据条目都是完全 Load

Time_Plan_Manager_Tool_config_File=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\Time_Plan_Manager_Tool_config_File.DFE
监控时间计划系统的配置文件,主要作用是服务器重启以后还原.

关于 ZDB2 的备份技术

备份就是 copy,在 ZDB2 体系中 copy 技术有 2 种模型,在讲解 copy 技术前需要说一个名词:命令队列,ZDB2 本身只是一个数据结构的存储支持库,如果要在 ZDB2 基础上做出多线性和支持并行的增删查改,会用到一种线程模型,这就是命令队列,它的作用是往队列里面仍命令,线程里面的主循环反复执行这些命令.因此 ZDB2 的所有数据操作都是队列化的:在这时候,会有个卡队列的问题,如果队列在前面发生了卡顿,会传导给后面.因此 copy 技术需要用如下方式区分:

- 静态 copy 模型:数据量很小使用的方案,静态 copy 是作为一种独立的 copy 命令存在,例如数据量只有 10GB,那么就是一条命令干完.10GB 很小,后面的等待不会超过 2 秒,因为有缓存.
- 动态 copy 模型:这种模式是建立一个 checkpoint 点(起始 copy 位置和结束 copy 位置),然后额定每次 copy 的数目 500 条,待完成再 copy 后面的 500 条.这种模式不会因为数据量很大发生队列阻塞.即使在 copy 过程中,也可以立即执行各种增删查改,500 秒这种数据规模会瞬间完成(每次完成 500 条 copy 再往命令队列仍下一个 500 条 copy).动态 copy 模型也是备份系统使用的主力技术模型.

Backup_Directory=d:\backup

备份系统的目标目录

Max_Backup=2

最大备份副本

Auto_Backup=False

自动备份,不要打开,监控数据很廉价,无需备份,丢了就丢了.
备份还原非常耗时,花 2 天时间还原会等同于数据中心瘫痪 2 天.

Auto_Backup_Hour_Span=8

自动化备份的时间跨度,单位是小时

Frag_Fragment_Space_Enabled=False

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_Fragment_Space_Enabled 表示是否启用 IO 读写分离技术.

如果不启用读写分离,那么所有的写都会是实时的,一旦发生异常关机,可能造成不可逆的数据丢失.

Frag_ForeverSave=False

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_ForeverSave 表示持续使用数据库.

如果 Frag_ForeverSave =False 在关闭数据中心服务时会直接删除数据库,同时在启动时也会先删除老数据库然后再建立一个新数据库.等同于数据库只存在于运行期间,服务器关闭即数据全部消失.

Frag_Used_GPU=False

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_Used_GPU 表示在使用数据中心服务器重建视频时是否使用 gpu 进行加速

6 代监控有专用视频下载工具端.无论数据中心是否使用 GPU 都应该避免使用使用数据中心服务器编码.

Frag_GPU_Device_Num=1

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_GPU_Device_Num 表示 GPU 的设备数量,如果服务器安装了多块 GPU,数据中心在做视频编码时会自动切换设备.

6 代监控有专用视频下载工具端.无论数据中心是否使用 GPU 都应该避免使用使用数据中心服务器编码.

Frag_H264_Build_Limit_As_Minute=5.0

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

作为对页面内存溢出保护,数据中心服务器限制了多个连续视频的拼接长度,最长 5 分钟.

6 代监控有专用视频下载工具端.无论数据中心是否使用 GPU 都应该避免使用使用数据中心服务器编码.

Frag_DeltaSpace=1024*1024*1024*10

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

ZDB2 是一种空间数据库,头部是一张空间分配表,body 是数据主体,因此 ZDB2 的数据文件不可以随意增容,需要使用增容机制(AppendSpace),并且每次增容都要指定空间尺寸.ZDB2 在运行过程中数据会不断的追加,增容机制也是不断在反复运行.

Frag_DeltaSpace =100*1024*1024*10 表示每次增容 10GB,增容机制会自动化触发,每次在写入时发现空间不足就会触发增容机制.

Frag_BlockSize=\$FFFF

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

ZDB2 是一种空间数据库,头部是一张空间分配表,在这张空间表中每一项都是一个 block,这个 block 会用于保存数据,例如待数据的数据为 1.5k,而 blocksize=1024,就会需要 2 个 block.

Frag_BlockSize =\$FFFF 表示每个 block 是 64k,这是因为视频数据都很大.BlockSize 最大就是\$FFFF.

Frag_EnabledCipher=False

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_EnabledCiphe 表示是否启用加密编码器

Frag_Cipher=None

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_Cipher 是指定加密编码器算法,可以使用的算法如下

None,DES64,DES128,DES192,Blowfish,LBC,LQC,RNG32,RNG64,LSC,XXTea512,RC6,Serpent,Mars,Rijndael,TwoFish,AES128,AES192,AES256
建议加密算法:RC6,Serpent,Mars,Rijndael,TwoFish,AES128,AES192,AES256

注意:加密算法会影响 IO 性能,如果数据量达到 1000M/s,开加密以后大约会降级为 500M/s,IO 性能会下降 50%左右.

Frag_Password=DTC40@ZSERVER

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_Password 表示加密算法的 Key 算子, DTC40@ZSERVER 是 C4 框架的默认 Key

Frag_DB_Num=2

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_DB_Num 表示数据引擎的子库数目,这项参数的正确用法并不是直接给值就能完事,

Frag_DB_Num 会影响后面的 DTC40_Frag_RealTimeVideo2_(1).Space2,如果给 2 就会有 2 个 Key 值

- DTC40_Frag_RealTimeVideo2_(1).Space2=XX
- DTC40_Frag_RealTimeVideo2_(2).Space2=XX

技巧: Frag_DB_Num 给值以后,立即保存,并且重启数据中心服务,使用 **Save_All_C4Service_Config()** 命令 [重新生成参数](#)

DTC40_Frag_RealTimeVideo2_(1).Space2=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Frag_RealTimeVideo2_(1).Space2

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

非常重要:指定数据库的文件路径,如果阵列服务器内存不够,或则优化需求,这里直接给硬盘的路径,例如

d:\mydb\disk1\1.space

d:\mydb\disk2\2.space

以上是极限优化的做法,如果懒一点,可以直接组 10 盘阵列(后面会介绍 server2022 组阵列),然后把整个数据中心全部仍阵列里面运行,不需要单独指定子库.因为内存并不便宜,并且插槽有限,在内存不够的情况下,优先考虑以指定每个子库来跑.

DTC40_Frag_RealTimeVideo2_(2).Space2=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Frag_RealTimeVideo2_(2).Space2

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

非常重要:指定数据库的文件路径,如果阵列服务器内存不够,或则优化需求,这里直接给硬盘的路径,例如

d:\mydb\disk1\1.space

d:\mydb\disk2\2.space

以上是极限优化的做法,如果懒一点,可以直接组 10 盘阵列(后面会介绍 server2022 组阵列),然后把整个数据中心全部仍阵列里面运行,不需要单独指定子库.因为内存并不便宜,并且插槽有限,在内存不够的情况下,优先考虑以指定每个子库来跑.

Limit_Frag_DB_Space=100*1024*1024*1024

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Limit_Frag_DB_Space 表示单个数据引擎子库的滚动临界点

当数据库规模达到该临界点,会以 Frag_Rolling_Space_Step 参数执行删头机制,当子库规模达到或接近磁盘容量时,删头意味着新增数据必定发生 hdd 寻址延迟.删除大文件头部,再从头部重新写入,意味着 hdd 的机械寻址会由外向内以盘面半径反复定位.

在真实运营时,这是一个稳定性的重要转折点条件,增容模型往滚动模型切换以后会持续走滚动模型,如果数据中心走了极限优化,当接近或则达到滚动模型当天,应该全时间观察服务器稳定性,因为寻址延迟大概率会导致 IO 性能下降,传导效应会影响服务器的整体运行.只有在滚动模型稳定以后,服务器才可以是长时间稳定运行.

运营时的滚动模型通常会在 15-45 天以后,开始启动,视阵列容量规模而定.

Limit_Frag_DB_Space=100*1024*1024*1024,表示单库临界点为 100GB

Frag_First_Inited_Physics_Space=100*1024*1024

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_First_Inited_Physics_Space 表示首次运行时,对数据库的初始容量,达到该容量会启用增容机制,一直会持续容量到达到 Limit_Frag_DB_Space 参数的限制,然后进入滚动模型.

Frag_First_Inited_Physics_Space 切勿给大,切勿给大,切勿给大,重要的事情说三次!

ZDB2 的空间结构是空间表在前面,数据存储在后面,每一次增容并不是重建空间表,内部设计是在尾部追加一张新的空间表,并在新空间表尾部给出数据空间,这一步可以理解成 Link 新的硬盘.当每次有数据保存时,对空间表的修改,会在局部范围中,如果初始容量给大,例如 5TB,这时候,每一次写入数据,修改空间表,在 6TB 物理空间的硬盘中磁头会来回跳,IO 能力变成 20%都有可能.

正确的做法是给一个合适的初始化容量,然后依赖于 Frag_DeltaSpace 参数,一次一次的走增容机制,这样每次写入数据会是靠近的.这是很优化的存储模型.

Frag_Rolling_Space_Step=10000

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

Frag_Rolling_Space_Step 表示当子库数据容量达到 Limit_Frag_DB_Space 所指定的额度,立即移除头部的 1 万条数据.

Frag_External_Header_Technology=True

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

参见 [Load 章节](#)

Frag_Full_Data_Load=True

凡 Frag 开头=存储 H264 视频剪辑的数据引擎

参见 [Load 章节](#)

Forever 数据引擎是一个无效设计,目前处于空置状态,直接使用默认参数即可

- Forever_ForeverSave=False
- Forever_Used_GPU=False
- Forever_GPU_Device_Num=1
- Forever_Save_Limit_As_Minute=5.0
- Forever_DeltaSpace=1024*1024*1024
- Forever_BlockSize=\$FFFF
- Forever_EnabledCipher=False
- Forever_Cipher=None
- Forever_Password=DTC40@ZSERVER
- Forever_DB_Num=2
- DTC40_Forever_RealTimeVideo2_(1).Space2=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Forever_RealTimeVideo2_(1).Space2
- DTC40_Forever_RealTimeVideo2_(2).Space2=D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Forever_RealTimeVideo2_(2).Space2
- Limit_Forever_DB_Space=50*1024*1024*1024
- Forever_First_Inited_Physics_Space=1024*1024*1024
- Forever_Rolling_Space_Step=100
- Forever_Header_Technology=True
- Forever_Fragment_Space_Enabled=True

首次运行的 RealTimeVideo2 配置范例

范例服务器配置:CPU>20Core(超线程>40),内存配置 384G,阵列用 6 张 hdd 总容量 40TB,使用时先按首次运行方法,生成配置文件,然后把下列参数粘贴进去全部覆盖,保存.重启服务器.

- SafeCheckTime=45000
- Ignored=False
- Load_Thread_Num=8
- VideoInfo_DeltaSpace=200*1024*1024
- VideoInfo_BlockSize=1536
- VideoInfo_DB_Num=8
- Limit_VideoInfo_DB_Space=1*1024*1024*1024
- VideoInfo_ForeverSave=True
- VideoInfo_First_Inited_Physics_Space=200*1024*1024
- VideoInfo_Rolling_Space_Step=10000
- VideoInfo_External_Header_Technology=True
- VideoInfo_Fragment_Space_Enabled=True
- Wait_hardware=False
- Max_Flush_History=5
- Frag_Used_GPU=False
- Frag_GPU_Device_Num=1
- Frag_H264_Build_Limit_As_Minute=5.0
- Frag_DeltaSpace=10*1024*1024*1024
- Frag_BlockSize=\$FFFF
- Frag_DB_Num=10
- Limit_Frag_DB_Space=1024*1024*1024*1024
- Frag_ForeverSave=True
- Frag_First_Inited_Physics_Space=10*1024*1024*1024
- Frag_Rolling_Space_Step=1000
- Frag_External_Header_Technology=True
- Frag_Fragment_Space_Enabled=True
- Frag_Full_Data_Load=False

启动服务器后,在 AI 数据载入环节会走完全 Load 模型,与配置无关,并行化 load 数据效率大约会在 1-2W/s 左右. Load 完成后 compute video-info...环节会做 3 件事,关联实例,统计监控名,生成 hour-span 结构,后续会详细说明. Rebuild sequence 环节会做一次重新排序,子库条目会是离散的,以大库方式还原条目序列.

```
D:\CoreLibrary\zAI\Binary\Video_2_AI_Database_Service_Console.exe
'DTC40_Info_RealTimeVideo2 (7).Space2' load sequence table
'DTC40_Info_RealTimeVideo2 (7).Space2' compute sequence space
'DTC40_Info_RealTimeVideo2 (7).Space2' load external header data 0
'DTC40_Info_RealTimeVideo2 (7).Space2' open done.
Open ZDB2 DB D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Info_RealTimeVideo2 (8).Space2
'DTC40_Info_RealTimeVideo2 (8).Space2' load sequence table
'DTC40_Info_RealTimeVideo2 (8).Space2' compute sequence space
'DTC40_Info_RealTimeVideo2 (8).Space2' load external header data 0
'DTC40_Info_RealTimeVideo2 (8).Space2' open done.
extract video info database.
full load 6477/80664 error:0
full load 13434/80664 error:0
full load 23112/80664 error:0
full load 32887/80664 error:0
full load 43328/80664 error:0
full load 53839/80664 error:0
full load 63776/80664 error:0
full load 74094/80664 error:0
all load done: 80664/80664, error:0
compute video-info...
rebuild sequence
```

Frag 载入环节走的大数据路线,同样也会关联实例,生成 hour-span 结构.

```
Open ZDB2 DB D:\CoreLibrary\zAI\Binary\RealTimeVideo2\DTC40_Frag_RealTimeVideo2 (10).Space2
'DTC40_Frag_RealTimeVideo2 (10).Space2' load sequence table
'DTC40_Frag_RealTimeVideo2 (10).Space2' compute sequence space
'DTC40_Frag_RealTimeVideo2 (10).Space2' load external header data 0
'DTC40_Frag_RealTimeVideo2 (10).Space2' open done.
extract video fragment database.
all block load done: 1190/1190, error:0
source: "5人跳舞" fragment analysis:402 last time 2024/4/18 1:54:22
source: "单人高空模拟" fragment analysis:261 last time 2024/4/18 1:54:21
source: "5人跳舞1.0模拟" fragment analysis:401 last time 2024/4/18 1:54:22
source: "测试模拟" fragment analysis:126 last time 2024/4/16 18:02:34
finish compute analysis and rebuild sequence, total num:1190, classifier/clip:4/822, last sequence id:1191
total video fragment: 1190
```

运营级部署第二步,监视数据中心的方法

当数据中心的完成脚本配置,启动它,如果有数据它这时候会 Load,耐心等待.当启动完成后,敲入命令 ZDB2_Info,如下图所示,会输出数据库及其子库的状态.记住这条命令,运营中的使用频率很高.

```
D:\CoreLibrary\zAI\Binary\Video_2_AI_Database_Service_Console.exe
total static-technology copy task: 0
total dynamic-technology copy task: 0
TZDB2_Th_Engine_Marshal Owner TZDB2_Video_Data_Tool database 80664/422.71M/3.90G
1: DTC40_Info_RealTimeVideo2 (1).Space2 Data-Num:26915 IO-Queue:0 Size:140.90M/1004.00M Fragment:0/0
2: DTC40_Info_RealTimeVideo2 (2).Space2 Data-Num:26905 IO-Queue:0 Size:140.90M/1004.00M Fragment:0/0
3: DTC40_Info_RealTimeVideo2 (3).Space2 Data-Num:26944 IO-Queue:0 Size:140.90M/1004.00M Fragment:0/0
4: DTC40_Info_RealTimeVideo2 (4).Space2 Data-Num:0 IO-Queue:0 Size:0/196.09M Fragment:0/0
5: DTC40_Info_RealTimeVideo2 (5).Space2 Data-Num:0 IO-Queue:0 Size:0/196.09M Fragment:0/0
6: DTC40_Info_RealTimeVideo2 (6).Space2 Data-Num:0 IO-Queue:0 Size:0/196.09M Fragment:0/0
7: DTC40_Info_RealTimeVideo2 (7).Space2 Data-Num:0 IO-Queue:0 Size:0/196.09M Fragment:0/0
8: DTC40_Info_RealTimeVideo2 (8).Space2 Data-Num:0 IO-Queue:0 Size:0/196.09M Fragment:0/0
Total Data/Queue:80664/0
TZDB2_FFMPEG_Engine_Marshal Owner TZDB2_FFMPEG_Data_Marshal database 1190/3.71G/99.88G
1: DTC40_Frag_RealTimeVideo2 (1).Space2 Data-Num:590 IO-Queue:0 Size:1901.02M/10227.48M Fragment:0/0
2: DTC40_Frag_RealTimeVideo2 (2).Space2 Data-Num:600 IO-Queue:0 Size:1901.52M/10227.48M Fragment:0/0
3: DTC40_Frag_RealTimeVideo2 (3).Space2 Data-Num:0 IO-Queue:0 Size:0/10227.48M Fragment:0/0
4: DTC40_Frag_RealTimeVideo2 (4).Space2 Data-Num:0 IO-Queue:0 Size:0/10227.48M Fragment:0/0
5: DTC40_Frag_RealTimeVideo2 (5).Space2 Data-Num:0 IO-Queue:0 Size:0/10227.48M Fragment:0/0
6: DTC40_Frag_RealTimeVideo2 (6).Space2 Data-Num:0 IO-Queue:0 Size:0/10227.48M Fragment:0/0
7: DTC40_Frag_RealTimeVideo2 (7).Space2 Data-Num:0 IO-Queue:0 Size:0/10227.48M Fragment:0/0
8: DTC40_Frag_RealTimeVideo2 (8).Space2 Data-Num:0 IO-Queue:0 Size:0/10227.48M Fragment:0/0
9: DTC40_Frag_RealTimeVideo2 (9).Space2 Data-Num:0 IO-Queue:0 Size:0/10227.48M Fragment:0/0
10: DTC40_Frag_RealTimeVideo2 (10).Space2 Data-Num:0 IO-Queue:0 Size:0/10227.48M Fragment:0/0
Total Data/Queue:1190/0
TZDB2_Th_Engine_Marshal Owner TZDB2_Video_Forever_Data_Tool database 0/0/2.00G
1: DTC40_Forever_RealTimeVideo2 (1).Space2 Data-Num:0 IO-Queue:0 Size:0/1022.73M Fragment:0/0
2: DTC40_Forever_RealTimeVideo2 (2).Space2 Data-Num:0 IO-Queue:0 Size:0/1022.73M Fragment:0/0
Total Data/Queue:0/0
```

上图入箭头所指信息

- **total static-technology copy task**,正在运行中的静态备份任务状态,这里会输出当前备份进度
- **total dynamic-technology copy task**,正在运行中的动态备份状态,这里会输出当前备份进度
- **"TZDB2_Th_Engine_Marshal" Owner "TZDB2_Video_Data_Tool" database 80664/422.71M/3.90G**,AI 识别状态数据引擎的统计结果,当前条目,条目所占空间,物理空间(物理空间受扩容机制影响)
- **1: DTC40_Info_RealTimeVideo2 (1).Space2 Data-Num:26915 IO-Queue:0 Size:140.90M/1004.00M Fragment:0/0**, AI 识别状态数据引擎的单库状态,就是单独文件的存储状态,IO-Queue 表示正在队列中等待处理的命令,Size 表示数据占用空间和物理空间(物理空间受扩容机制影响),Fragment 表示读写分离状态,当前写分离的数据块/当前写分离的总量
- **"TZDB2_FFMPEG_Engine_Marshal" Owner "TZDB2_FFMPEG_Data_Marshal" database 1190/3.71G/99.88G**,这是 h264 视频剪辑数据引擎,信息含义参照上面

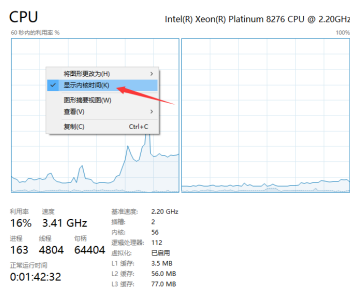
理解子库工作机制

子库使用卷动写入模型,例如数据 1,2,3,在子库写入模型中,会优选物理容量最小的子库写入.不是无脑卷动.

在选择容量最小的子库会计算当前子库物理容量+处于队列中尚未执行 IO 写入.最后的写入结果会是数据 1,2,3 分别写入不同的子库.

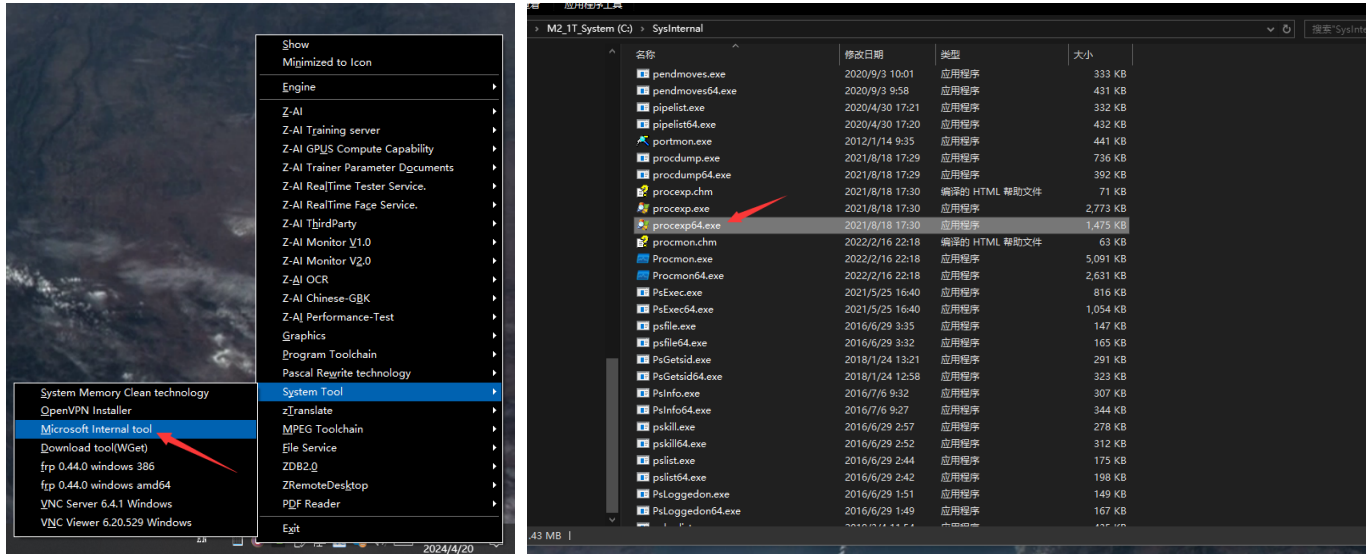
理解 CPU 使用率

请在 windows 进程工具勾出内核时间,该时间表示 windows core 层的 cpu 占用率,数据中心内部实现以堆砌结构+算法为主,内核时间消耗极少,GPU 环节主要异构不会消耗内核,不确定某些阵列卡和网卡设备是否会消耗内核,品牌设备跑高负载拉满也只会轻度消耗内核(<10%).如果发现内核时间消耗占比达到 20%或则更高数据中心服务器都会是不稳定的,这表示计算会被干扰.这个时候,可以查询一下进程,某些刷过参数的内存条也会出高内核时间问题.



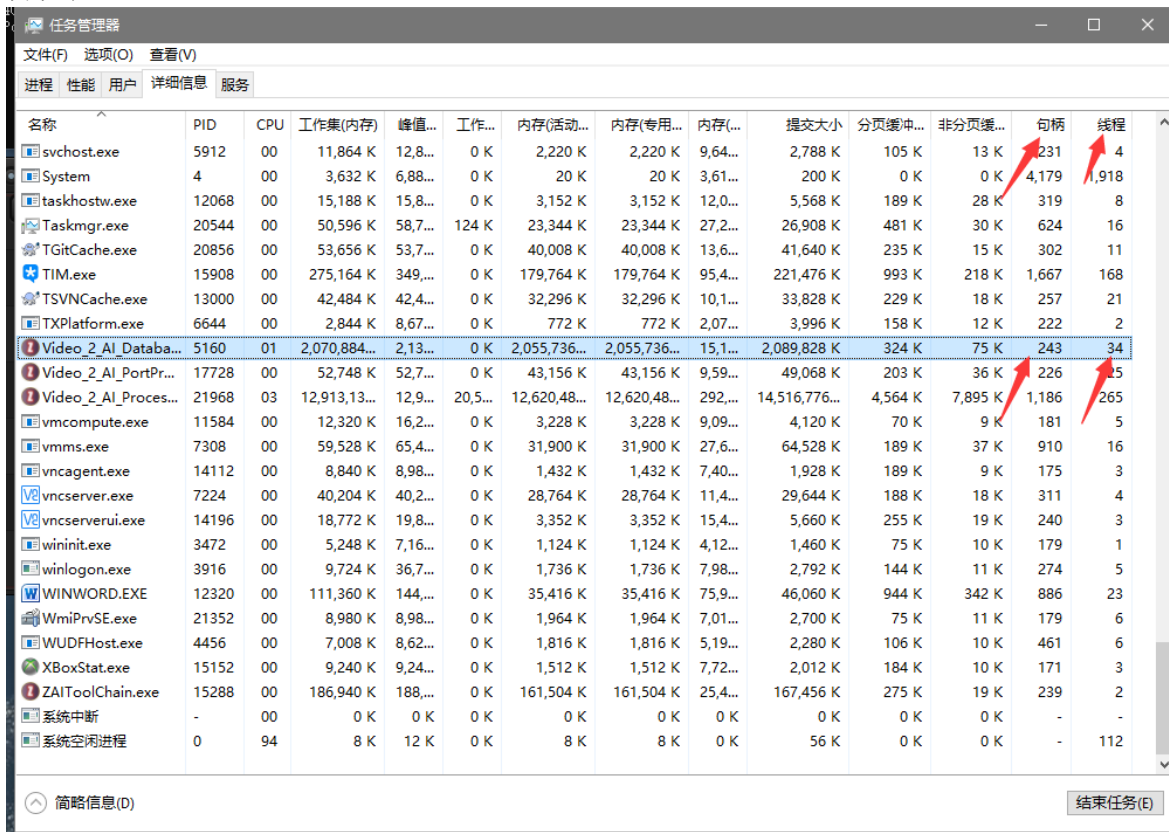
使用自带工具分析单进程内核时间占用率

当 Z-AI 被正常安装以后,会包含一套 windows 内部工具.建议使用 `procexp64.exe+ Autoruns64.exe` 来分析单进程和自动任务.



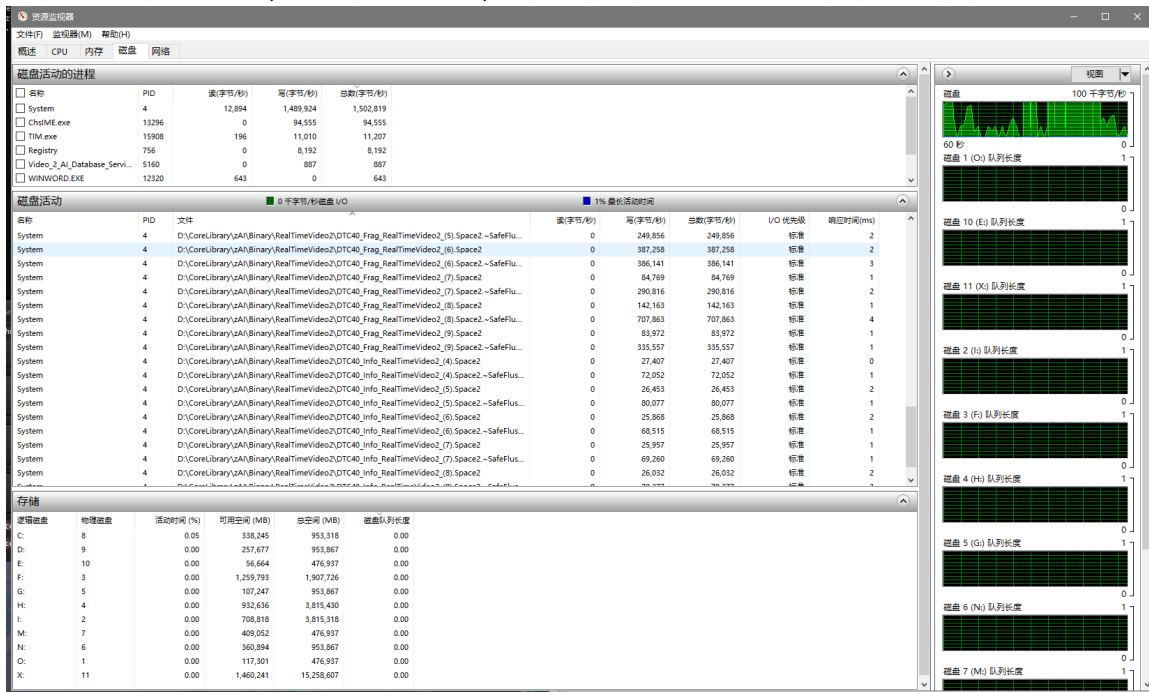
在数据中心运行中,监视句柄和线程变化

在进程工具给出句柄,线程状态,在数据启动以后该值通常会比较稳定,即使在高流量期间该值也会长时间保持稳定.如果发生跳跃,或则不断增加,这种问题如果不是驱动程序 bug(某些便宜的山寨网卡,插 usb 的 wiki 网卡),一定是擅自修改过程序,亦或是并入了没有经过 test case 的模块.数据中心在随发行版本部署后,是不会有泄漏的,已经历运营洗礼.



监视磁盘

Windows 自带的资源监视器是最好的阵列监视工具,在调教数据中心时,我都是根据该工具来分析磁盘阵列状态.在启用 IO 读写分离机制以后,每次 flush 都会出现短暂爆发期,在爆发期当中,IO 会是拉满,关键是监视爆发期持续时长.首次运营会有点生疏,后面会熟能生巧,根据硬件能力给出合理的数据中心脚本.



建议专机专用

规范部署,专机专用,跑数据中心的机器不要跑 AI Processor 这类服务.AP 不光消耗北桥+GPU,也非常消耗 CPU,而网络,阵列,内存中的数据传递,都会依靠 cpu.即使跑 20 路 1080p 也会吃掉很多 cpu 算力.

GPU 服务器在机房中是比较下贱的存在,坏了走修理,退货流程,平时都可以随时关机重启,但数据中心不同,数据都在里面,所以,还是专机专用把.

运营级部署第三步,部署群集 IP 转发

群集 IP 转发是自动化服务,只需要启动 Video_2_AI_PortProxy_Service_Console.exe,因为数据中心承载了高流量,磁盘阵列,在数据中心里面不适合集成群集 IP 转发,这会导致延迟,故专门给出小型 C4 服务.

群集 IP 转发与数据中心的通讯方式最好走内网,不给参数直启会以内环回接口通讯(ipv4(127.0.0.1),ipv6(::1)).

运营级部署第四步,部署 AP 服务器(GPU 群集)

AP 服务器环节会有专门文档做细节介绍,这里面会有许多侦错,调试,建模,编写 AP 脚本,修正脚本等等步骤.

运营级部署第五步,部署外围服务器:直播服务器,web 服务器,CS 服务器,小程序服务器,远程桌面

直播服务器环节会有推流规范,不会是直接自建直播服务器就完结,需要 AP 脚本配合,推流服务器亦可使用云服务器运营商资源.直播服务器搭建细节在 AP 文档给.

直播,web,cs,小程序,完全可以在本机直接用 Hyper-V 来跑(远比 kvm+vmphere 灵活),不必购买服务器.数据中心都是高配,划分出 1/20 的系统资源即可.

我们的数据中心有 GPU,但不是用于跑业务,而是专门跑 AI 训练.跑业务模型用专用 GPU 服务器.远程桌面可以选择 ToDesk, Windows-RD,VNC,亦可使用 Z-AI 自带的远程桌面.

运营级部署第六步,拓扑网络交换机优化

500 路 2k 监控如果工作与于无忧化的拓扑网络,会导致整个网络陷入瘫痪,即使万兆交换机,即使 4 万兆交换机,瘫痪的症状会是内网连接频繁断线,内部原因会是 ip 包丢失.优化分两方面进行.

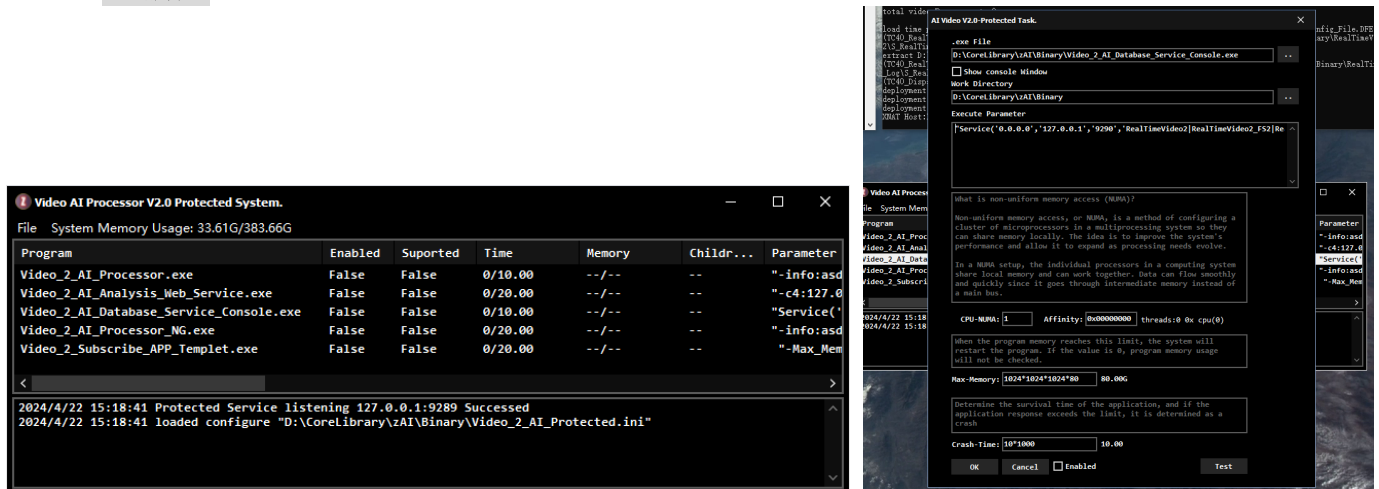
- 第一,主力交换机给出 truck 隧道,尽量保证网段点对点,现场的拓扑网络情况通常很复杂,应该根据网管给出的信息进行网段流量优化,让现场的人去优化,不要迷信那些不了解拓扑情况的网管,即使不了解拓扑网络的技术,也是可以现学现用的,chatGPT,文心一言,这些都是解决问题的利器.现场总是优于远程协助.
- 第二,所有的 AP 服务器可以与数据中心走专线,弄个交换机来解决,光口+电口均可直连.这样能降低拓扑网络的中枢主干流量.

运营级部署第七步,给所有服务器接入进程保护系统

保护系统主要作用于物理内存超额保护+无节制的 CPU 使用保护.同时保护系统也是一种半自动化的部署工具.

保护系统是针对 6 代监控体系专用的,可以支持 AP 系统,数据中心,web 统计,Subscribe 系统.

- 简单说一下 CPU 占用率:进程管理面板上的 CPU 占用率并不是真实的 CPU 开销,在服务器程序中,会大量使用线程,并行,SSE,AVX2 这类技术,凡是硬件优化类技术都是会带来瓶颈的,例如面板 cpu 使用率 80%,但点不开“开始”菜单,随便打开一个 app 等 30 秒.这在运行计算程序类的服务器很常见,甚至是算法 author 都无法准确控制算力资源(CPU 高负荷以后,内存,HDD-IO,网络,都会出现卡顿),这种工作会交给系统集成自动化,而解决办法就是“亲和性”参数,也就是给 app 限核,只有这种办法可以控制住多任务.
- 简单说一下内存超额:在高流量+无人值守项目中,内存使用率很难确定,例如在学校放学,街道上千人在行走,这个时候服务器的内存,HDD,网络,算力,等等使用率会比平时高出 20%,而半夜到凌晨,服务器又几乎是空闲状态.保护系统的作用就是当内存超额,以及系统接近崩溃,就重启,然后还原服务器.



我们平日做更新升级,都是直接 copy 文件到目标服务器群,然后直接重启系统,当系统启动完毕,保护系统会自动启动目标服务.该功能主要解决系统崩溃重启后自动还原服务器.

帮助数据中心二次开发

数据中心设计采用 C4 服务器框架,做法是往里面堆砌各种通讯命令,这些命令有很多线程分载机制,这样干是某些查询,下载,数据搜索,这类命令会有很长的流程,直接在主线程会造成卡顿.但这些并不是关键.最关键的地方是长期升级+更新+维护的可行性,以及规范化的编程.

在升级更新可行性环节:当远程请求过来,使用 HPC_Thread 方法开一条线程,处理请求,然后反馈回去.亦或是请求过来,新建一个事件实例,通过触发事件来执行反馈.这种模式有绝对可行性,并且在未来更新升级环节,可以一直沿用.一般我把这种做法叫做程序模型,也许我的叫法不太准确,以程序范式来理解即可:不需要用发掘性方式去思考机制可行性和人性化,既然定出范式就一定会可行,具体二次开发时只需要借鉴相近的功能,然后复刻这种范式,一直保持它就能做好持续的升级更新,这种方式可以长期更新推进.

数据中心的核心服务只有一个,与核心服务器对应的客户端也是只有一个,这种 1 对 1 服务器+客户端机制是 C4 的规则要求(所有 C4 通讯模型都要求服务器+客户端必须成对提供).因此,在数据中心外围体系,AP 服务器,下载工具,统计工具,WEB 接口,监控员端,Subscribe 系统,很多很多环节,它们都使用同一个客户端,这个客户端就是与核心服务所对应的.

并入的模块都会有 Test 方法

数据中心的网络服务器+客户端是用一个接近万行源码单元实现的.这个源码文件周围有许多依赖模块,尤其是数据引擎环节,这些数据引擎都是在通过了单独 test case 以后并入到服务器中的.在并入以后,因为无法单独测试,我是直接拿实际项目来跑,在运行过程中,随时[监视服务器运行状态](#).单独写完流程,通过 test case,再并入到数据中心,只要严格按流程走,正确率几乎 99%.

砍而不删

在数据中心服务器中,很多流程和方法,是不用的,甚至一些比较大型的结构也会直接空置化处理.

蝴蝶效应下的崩溃记忆

数据中心大约稳定运行了 1 年以后,处于对大型模块的深入实测,我们希望能更逼近物理设备极限,于是修改策略,把单日 AI 的数据条目从 1000 万,提升至了 5000 万.这时候原本 5 秒可以完成的遍历搜索,变成 40 秒,直接导致前端查询这类应用变得很卡.

为了解决 40 秒问题,数据中心提出了一种跨度方法,思路上是使用内存来加速范围时间搜索,将原本数千万次遍历计算量降低成 2000,因为是优化工作,需要数据中心以最简单的改动来提速,这也导致了在结构设计做的很复杂(用了很多 Pair 做结构体).在记忆中,当时已经全部通过 test case,可最后漏了一个逻辑:test case 是走构建->释放,然后,检查内存泄漏,没有泄漏,就算 passed.而对数据中心来说,构建既永久,不会释放,算法必须是在处于永久性运行中动态管理释放(漏在这一步).

大致说一下泄漏机制,例如 2024-1-1 9:00:00 到 2024-1-1 10:00:00,这 1 小时跨度有 2 万条数据,这时候,随着数据引擎运行,时间来到, 2024-2-1 9:00:00,以前这 2 万条已经被释放了,但是, 2024-1-1 9:00:00 到 2024-1-1 10:00:00 这个跨度小结构还在,这几乎是个极小的内存粒度空间.服务器必须进入滚动存储模型以后(1 个月),然后再连续运行 15 天,最开始是微小泄漏,后面变成物理内存使用量超出常规(设计阵列系统参数时会会有一个内存规格,可它严重超标).

上述的微小泄露是靠猜解决的,修复以后,很不确定的重启了服务器,当服务器再次连续运行 15 天以后,才真正的确定解决.修复库为 Z.HashHours.Templat.pas + Z.HashMinutes.Templat.pas.

```
emp10000 Revision 1982 : Z.HashHours.Templat.pas
Z.HashHours.Templat.pas Revision 1982
end;
procedure THours_Buffer_Pool(T_)..Remove_Span(Value: T_);
var
  obj2: TTime_Data_L;
begin
  FCritical.Lock;
  key
  obj2 := FTime_Data_Pool.Key_Value(Value);
  if obj2 = nil then
  begin
    if Num > 0 then
      with Repeat..do
        repeat
          queue^.data^.data.Second.Delete(Value);
        until not Next;
      end
    else
      begin
        if obj2.Num > 0 then
          with obj2.Repeat..do
            repeat
              queue^.data.Delete(Value);
            until not Next;
          end;
        end;
        queue^.data.Clear;
        Discard;
        end;
        until not Next;
      end;
  finally
    FCritical.Unlock;
  end;
end;

emp10000 Revision 1982 : Z.HashMinutes.Templat.pas
Z.HashMinutes.Templat.pas Working Copy
end;
procedure THours_Buffer_Pool(T_)..Remove_Span(Value: T_);
var
  obj2: TTime_Data_L;
begin
  FCritical.Lock;
  key
  obj2 := FTime_Data_Pool.Key_Value(Value);
  if obj2 = nil then
  begin
    if Num = 0 then
      with Repeat..do
        repeat
          queue^.data^.data.Second.Delete(Value);
        until not Next;
      end
    else
      begin
        // remove queue
        if obj2.Num > 0 then
          with obj2.Repeat..do
            repeat
              queue^.data.Delete(Value);
            until not Next;
          end;
        end;
        // remove primary key
        if obj2.Span_Time_L.Num > 0 then
          with obj2.Span_Time_L.Repeat..do
            repeat
              queue^.data.Delete(Value);
            until not Next;
          end;
        end;
        // remove link
        FTime_Data_Pool.Delete(Value);
      end;
  finally
    FCritical.Unlock;
  end;
end;
```

关闭数据中心的办法

命令行敲入 `exit`,然后等待,如果读写缓存规模上来,也许会等上 15 分钟,不会卡死,一直等。

通过 `exit` 退出,不会遗留读写缓存,所有的数据库都将会是正常关闭状态,并且在下一次启动也会更快。

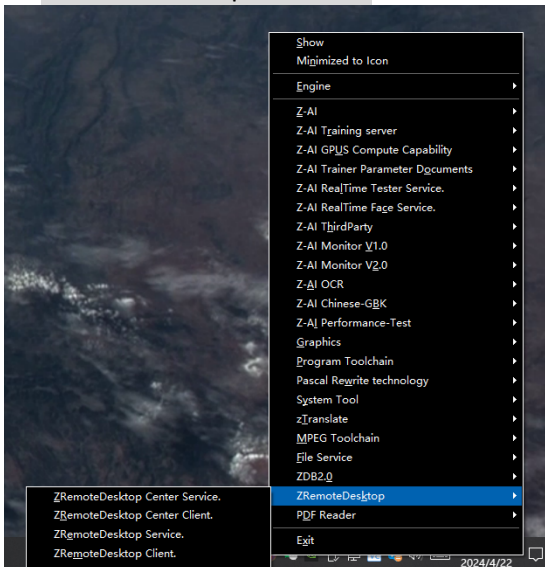
如果直接点 X 关闭会等同于断电,数据库同目录下将会遗留许多 `flush` 临时文件,并且在下一次启动时,会自动化的进入断电恢复模式,这会比正常启动更加耗时。

数据中心硬盘坏了怎么办

监控数据不像金融数据,监控数据很廉价,坏了仍掉.重新配置一下存储策略就行了。

Z-AI 自带远程桌面

使用新版本的 Z-AI 安装程序会包含 ZRemoteDesktop,不用再跑去配置网关,穿透,ToDesk,向日葵.详细使用方法请参考 ZRemoteDesktop 产品文档



6 代监控的数据中心服务是不可复制的独特程序

数据中心服务是整个监控体系的后台中枢,做这一步的开发,需要是主程序级。

如果技能点加在,MIS,erp 这些环节上,切入会比较吃力,应该以净化灵魂的方式对待。

对于全堆结构+算法的大型服务器后台,使用常年操作 `sql,UI,app` 这类程序,大概率无法胜任后续开发.同理,放眼整个 `pas` 圈,能写出数据中心后台的人也会屈指可数.从数据库到通讯框架,再到细微的各种算法,数据中心是独特程序。

数据中心是一种具有策略性质的项目:使用社区的开源机制接入一些新技术,例如 `gpt,sam`,也许只会花上 2 周,而作为 `gpt,sam` 这类前沿技术的老人,如果要用 AI 替代监控,会花上 `n` 年重走我们的老路,而更大概率也会是在监控周围酱油。

2024-4-22

全文完.