ZNet User manual

Document version :1.1

Updated :2023-10-12

Changelog:

2023-10-4 Start writing

2023-10-6 Complete 1.0 writing

2023-10-9 Add technical information on dual main thread and mutex

2023-10-11 Add Performance toolbox information

2023-10-12 Add TCompute concepts +struct concepts, using concepts to tell the story Because the code can't tell the whole story, the code can be mined from large to small by yourself.

2023-10-12 Append compiler class technology description, the specific code you go to dig

I wish you all a smooth work and a great project.

ZNet related websites

Open source host https://github.com/PassByYou888/ZNet

Open source spare - https://gitlab.com/passbyyou888/ZNet

Author personal station https://zpascal.net

ZNet author qq600585

qq group, 490269542(open source group),811381795(AI group)

Table of Contents

Noun explanation 4
ZNet's 6 command sending and receiving models 4
ZNet Dual channel model 4
ZNet thread support 5
    HPC Compute thread offloading scheme 5
10 Gigabit Ethernet support 5
Main loop three things you must know to use ZNet 6
    First thing: Iterate over and process the data receiving flow for each IO 6
    Second: Iterate through and process the data sending flow for each IO 6
    Third thing: Manage the cpu cost of each traversal 7
Optimize data transfer 7
    Analyze bottlenecks 7
    Once performance bottlenecks have been identified, the next step is to address them 7
Building a server is always about hardware programming 8
Erection of Bridges 8
ZNet bridge support 9
C4 Start script writing mode 9
    win shell command-line way: 9
    linux shell command-line mode 9
    Code way 10
C4 Startup script cheat sheet 10
    Functions :KeepAlive(connect IP, connect Port, register client), ... 10
    Function: Client (connect IP, connect Port, register client) 10
    Function: Service (listen to IP, native IP, listen to Port, register server) 10
    Function: Wait(delay in milliseconds) 11
    Function :Quiet(Bool) 11
    Function :SafeCheckTime(in milliseconds) 11
    Function: PhysicsReconnectionDelayTime (floating point Numbers, the unit seconds) 11
    Function: UpdateServiceInfoDelayTime (ms) 11
    Function: PhysicsServiceTimeout (in milliseconds) 11
    Function: PhysicsTunnelTimeout (in milliseconds) 11
    Function: KillIDCFaultTimeout (in milliseconds) 11
    Function: Root (string) 11
    Function: Password (string) 11
    UI function: Title (string) 12
    UI function: AppTitle (string) 12
    UI function: DisableUI (string) 12
    UI function: Timer (in milliseconds) 12
C4 Help command 12
    Command :help 12
    Command :exit 12
    Command :service(ip address, port) 12
    Command :tunnel(ip address, port) 12

Command :reginfo() 13
Command :KillNet(ip address, port) 13
Command :Quiet(Boolean) 13
Save_All_C4Service_Config() 13
Command: Save_All_C4Client_Config() 13
Command: HPC_Thread_Info() 13
Command :ZNet_Instance_Info() 13
Command: Service_CMD_Info() 14
Command: Client_CMD_Info() 14
Command: Service_Statistics_Info() 14
Command: Client_Statistics_Info() 14
Command: ZDB2_Info() 14
Command: ZDB2_Flush() 14
ZNet Kernel Technology - Lock multiplexing 14
ZNet Kernel Technology -Soft Synchronize 15
Kernel :Check_Soft_Thread_Synchronize 15
Kernel :Check_System_Thread_Synchronize 15
ZNet Kernel Technology - Dual main thread 15
RTL primary thread is synchronized to secondary main thread 16
The secondary main thread is synchronized to the RTL primary thread 16
The main loop after the dual main thread is opened 16
Guide to using the ZNet Performance Toolbox 17
Performance Bottleneck Analysis 17
Exclude ZNet overlapping Progress 18
Awe server main loop progress 18
ZNet kernel technology :TCompute thread model brief introduction 19
The kernel technology of ZNet: a brief introduction to its architecture 20
ZNet kernel technology: a brief introduction to the structure of the combined fist 21
Review: Designing Universal Structures TBigList<> 21
Review: Designing the scripting engine ZExpression 22
ZNet's parent porting technology: Z.Composing 22
How to overturn a project using ZNet 23
How to develop a web-like project using ZNet 23
ZNet with http and web 23
The text ends with a CS demo with minimalist C4 24

Nouns and Terminology

- Blocking queue, waiting for completion, waiting queue: Waiting is a special mechanism of ZNet, after the queue will wait for the completion of the front, the execution will be strictly in order, the waiting will all wait in the local machine, only after the remote impact, the local queue will continue, not send the whole queue to the past
- Serialization queue: do not wait for data feedback, directly send data, and the order of data reception is strict, according to the sequence of 1,2,3, then the reception will also be triggered by 1,2,3. For example, use the serialization queue to send 100, and then send a blocking queue instruction, and wait for the blocking return to indicate that the 100 sequences have been sent. Similarly, for example, uploading a file takes one hour, then sending the file first, and then winding up blocking, and waiting for blocking to return, means that the file has been sent successfully.
- Non-serialization queue: sending and receiving are processed according to the strict sequence mechanism, but after triggering the receiving, ZNet will do such program processing as decoding in some child thread or coroutine, and send the data in the sequence of 1,2,3. The received data will be put into the thread for processing later, and the receiving event will not be triggered in the strict sequence of 1,2,3. Non-serialization is often used in communication without requirements for data before and after.

6 kinds of ZNet command sending and receiving model

- SendConsole: support encryption, support compression, blocking queue model, waiting for feedback after each send, for example, first send 100 commands, finally send a SendConsole, feedback back also means that 100 commands have been sent successfully. SendConsole is very lightweight, suitable for sending and receiving small text below 64K. Examples are json,xml,ini,yaml.
- SendStream: support encryption, support compression, blocking queue model, after each send will wait for feedback, all received and sent data will be encoded and decoded in TDFE, because TDFE has data container capabilities, so SendStream is often used for application data sending and receiving,SendStream also has blocking queue capabilities. In terms of traffic, it can support larger data.
- SendDirectConsole: supports encryption, supports compression, serializes the queue model, does not wait for feedback, for sending and receiving string-based serialization of data.
- SendDirectStream: supports encryption, supports compression, serialization queue model, does not wait for feedback, uses container to package data, can send and receive larger serialization data.
- SendBigStream: does not support encryption and compression, serializes the queue model, and solves the sending and receiving of very large streams, such as files, large memory block data. SendBigStream works by sending only a part at a time and waiting for the signal to appear before continuing to send, without bursting the socket buffer. Both the bandwidth and latency of the physical network affect SendBigStream's efficiency.
- SendCompleteBuffer: Does not support encryption and compression, serializes the queue model, does not wait for feedback, high-speed transceiver core mechanism, core mechanism supported by 10 Gigabit Ethernet. The design idea of CompleteBuffer is to copy Buffer around the network, high-speed transceiver is a very important mechanism, named CompleteBuffer.

ZNet double channel model

double channel is a concept at the design level, indicating that the receiving and sending are an independent channel connection, the signal receiving and sending are designed separately, and the early dual channel is to create two connections to work. After countless exploration and upgrading, the current dual channel is based on p2pVM,p2pVM can create infinite virtual connections on the basis of a single connection, and these p2pVM connections are dual channels in the application layer.

Imagine that in the past, we piled up multiple servers and needed to define countless listens, connections and ports. Now p2p VM is used to virtualize all connections. Because it reduces the complexity of the backend technology, it provides a large heap of maintainability and normative space for the backend system. For example, all services of C4 take p2pVM dual channel.

ZNet thread support

ZNet in herently supports sending data in threads, and even for parallel programs, the data sent will be non-serialized queues.

ZNet's data reception is always in the Progress focus, which is the thread that triggers Progress. In most cases, ZNet recommends executing the main Progress loop in the main thread, such as the C4 framework.

ZNet can support running Progress in thread to achieve thread sending and receiving, but it is not recommended to do so, because ZNet has a better thread offloading scheme.

HPC Compute thread offloading scheme

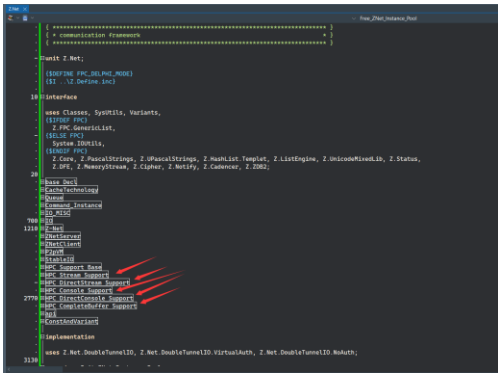
Folded code that starts with HPC in ZNet is the thread offloading solution

Process: When the data arrives from the main thread, the receive event is triggered, the offloading is started, the data is transferred, and a new thread is created to perform the processing. After this, the server can run in a state of no delay. Because the main thread logic is protected, the stability is better than opening the Critical management thread yourself.

The offloading scheme can support all command sending and receiving models except BigStream.

Offloading can not overlap: command ->HPC->HPC-> return, only: command ->HPC-> return

When the command ->HPC is triggered, the data is transferred from the main thread to the thread, and no data copy occurs



10 Gigabit Ethernet support

ZNet uses the CompleteBuffer mechanism to support 10 Gigabit Ethernet

- Internally, ZNet's SendCompleteBuffer works with threads: The threading mechanism can provide pre-processing prerequisites for large data flows, such as 10 threads for data generation, and then SendCompleteBuffer
- SendCompleteBuffer has high traffic buffering ability. When the scale of SendCompleteBuffer is called, it will be buffered to a temporary file, and a piece of CompleteBuffer will be sent out when the main loop is triggered: This mechanism provides a caching mechanism for long queue data, but it can not brute Se nd, in general,10 SendCompleteBuffers can be matched with one SendNull(blocking queue), which will keep the overall load and throughput of the network in good condition.
- Sendcompletebuffers can be overlapped with DirectStream
  For example, the command model registered by the server is DirectStream, which can be sent using SendCompleteBuffer
- SendCompleteBuffer supports sending TDFE data directly
- SendCompleteBuffer can be overlapped with the Stream blocking model

If the server registers the command model as a blocking Stream, the client sends a buffer and receives a buffer using SendCompleteBuffer in nonblocking response mode, without blocking.

- If you do not use CompleteBuffer to receive and send big data in 10 Gigabit Ethernet, a 100M data may make the sender freeze for 5 seconds and appear UI suspended, and after it is sent out, the receiver will freeze for 5 seconds and stop the response. This is because a large number of data replication, encoding, compression, and decompression take up the computing resources of the main thread Can deal with very small volume of data.
- Using a CompleteBuffer at 10 gigabit to send a 100 megabit of data, the processing delay on both sides can be less than 10ms from send to receive, so that the server model is always immediately responsive.
- Summary: Thread + disk caching +SendNull blocking +DirectStream overlap +Stream overlap = successfully solve 10Mbps Ethernet problem with CompleteBuffer

Three things you must know about the main loop to use ZNet

First thing: Iterate over and process the data receiving flow for each IO

ZNet's physical network interface mostly uses a separate thread, which is card free. When the main thread is fully occupied, such as when processing 100M compression + encoding tasks, the receiving thread is still working normally. The data received by the child thread will not be placed in the memory: the receiving program will judge whether to use a temporary file to temporarily store the data according to the data volume.

When the main loop is triggered, the main loop will work with the corresponding thread, the main loop is always single-threaded model, the program will take the received data from the child thread, including the temporary file data, and then enter the network data packet processing link.

In the process of data packet sticking,ZNet uses a large number of memory projection technology to avoid memory copy, which is to map the memory address into a TMemoryStream. In ZNet, the memory projection uses tools such as TMS64 and TMem64.

The packet processing system will contain cpu time consumption. Since the packet processing system contains large subsystems such as sequence packet and p2pVM, ZNet gives the cpu time consumption. When the value reaches a critical point, the packet processing system will interrupt the packet processing and continue processing in the next main loop. For example, when 10,000 commands are sent, the critical point of packet sticking is 100ms. Only 2000 commands have been processed when the number reaches 100ms, and the remaining 8000 commands will be processed in the next packet sticking.

In the actual operation of ZNet, there is almost no memory copy in the sticky packet process, and the processing capacity of sticky packets in a single thread can reach hundreds of thousands of processing levels per second. There is no need to open a thread or coroutine for processing. Do a thread acceleration of this kind of idea, it is not practical, this will not only increase the kernel complexity of ZNet, efficiency improvement will be very general, can only be to Newbie to solve the random high-speed sticky package, and the correct high-speed sticky package, only need to use SendCompleteBuffer to send data in.

The second thing: traverse and process the data sending process of each IO

All send commands in ZNet will eventually land on each IO

In each IO, there will be a queue of command data waiting to be sent, some of which will be stored in memory and some of which will be stored in temporary files.

ZNet will iterate over and send the strictly serialized data from the IO pending queue, which will be placed in the physical IO pending buffer, which is not under ZNet's control.

When using virtualized communication protocols like p2pVM or StableIO, the strict serialization data will be directly repackaged and placed in the physical IO buffer when traversing the IO.

All the physical buffered data sent is controlled by the signaling system of the topology network. Each ip packet containing a tcp tag requires an endpoint feedback signal (the remote receiver, not the internal topology) for the packet to be delivered. This feedback signal is the

network latency. The feedback in ZNet is the speed of delivery.

Third thing: Manage the cpu cost of each pass

ZNet will record the time cost of each training traversal IO. When it reaches a critical point, the traversal IO will be terminated and the next iteration will be resumed in the main loop. In this way, when the server reaches a certain load, the remote response will slow down, and the server itself will take the technical route of fragmented load in the single-threaded main loop.

Optimize Data Transfer

Start with a dear goal: to keep the dient from getting stuck in the UI and the server from getting stuck in the main loop. This needs to be treated as a project or product as a whole. For example, if the server is stuck in the main loop, then there will be a delay in response time. The frontend will send a request and wait 5 seconds for the response.

When the goal is dear, the primary task is to analyze the bottleneck of the lag. Most CS or web projects can directly locate the bottleneck through the command sent by the dient. For example, the GetDataList command, if there is a 5-second lag, it is directly located to the server's GetDataList response link. But sometimes,ZNet command throughput has unprecedented scale, such as hundreds of servers, and thousands of IOT network devices, these commands are dense, and ZNet is required to provide support information

Analyzing Bottlenecks

If you're not using the C4 framework, you'll need to add your own code to the server application or console.

Outputs the cpu cost of each command on the server

ZNet_Instance_Pool.Print_Service_CMD_Info;

Output server running status statistics

ZNet_Instance_Pool.Print_Service_Statistics_Info;

If you are using the C4 framework, you can enter Service_CMD_Info directly in the console

In the output command, the cpu consumption of all commands sent and received will be included. If some information contains the words ": HPC Thread", it means that this command uses HPC Thread offloading, for example, GetDataList:HPC Thread": time 123078ms

It means that the longest processing of GetDataList in HPC thread offloading ran for 2 minutes

If you are using the C4 framework, you can monitor thread offloading in real time through the HPC_Thread_Info console command

Thread offloading gives each thread information about the command being executed, and HPC_Thread_Info outputs the status of all the threads of the C4 server. Then, combined with the system's task and resource monitoring tools, keep the server running, basically can analyze the performance bottleneck.

Once the performance bottleneck is identified, the next step is to solve the bottleneck

If the server is on the main thread route, there are only two ways to solve the bottleneck. The first is to consider thread offloading and use HPC functions to open thread processing commands. The second is to consider using SendCompleteBuffer on the dient instead of SendDirectStream.

If the server itself is a multithreaded design route, it will need to start from locks, structure, flow, etc. If you optimize a multithreaded server as a whole, things are more complex: you will go from conducting analysis of the problem to locating the problem, and finally adjusting the flow. In the end, most likely an algorithm will be used to solve the problem. For example, the author of ZNet's monitoring project always waited a long time when searching for videos. Finally, the author designed an acceleration algorithm to store videos by time span.

When the server heap becomes large, a background service may reach the scale of 100,000 rows, and a lot of optimization work will also be the work of fixed bugs. These optimization work will be divided into early stage, middle stage and later stage. The closer to the early stage,

the higher the frequency of fixed bugs. In the later stage, the whole server background may be overturned and reconstructed 1-2 times, which is an experience and design problem.

Doing server is always about hardware programming

Many open source projects seem simple and easy to understand, but a real server project might be 100 times larger. Going from quantitative change to qualitative change, 100 times larger, is no longer a conventional technical solution.

Many people do server scheduling database + communication system, if it is to do the web, the server will also include the design of ui system. If the project scale is small: low frequency of communication + small amount of communication data, how to do this kind of server is no problem.

When the server scale begins to be large: high communication frequency + large amount of communication data, it will face: programming around hardware.

6 core 12 super line and 128 core 256 super line, this is not the same in the hardware location, it will affect the thread and server design model,6 core specifications of hardware can hardly open a parallel program model, even if a link to open parallel for, perhaps the whole server will be affected,6 core can only open regular threads for particularly heavy calculation links to run thread load. When the server runs on the 128 core platform, there is no problem of cpu computing resources, but reasonable arrangement of computing resources, for example,4 threads of parallel computing, sometimes faster than the use of 40 threads. Finally, the system bottleneck, in the case of not using the tripartite thread support library, a single process of win system can only use 64 hyperlines at the same time, only after mounting TBB such libraries, can use 256 hyperlines at the same time. Full thread, partial thread, single thread, this kind of server at the beginning of the design positioning has been completely different. The center of the server program is around the hardware trend of different times, the biggest reason for the hardware platform, the framework design link is the path, only clear around the hardware as the center, and then do server design and programming, this is the correct route.

Another example is 8 4T full flash sdd with 192G memory, and 8 16T hdd with 1TB memory, this kind of server in the data storage, cache system design is not the same, for example, the data scale to 3 billion, space occupation to 30T, this scale basically 192G memory will be very tight, but it is not a problem, after careful optimization can also run, because of the number According to the acceleration of search or storage speed is always to use cache, and when the hardware configuration of the storage device uses hdd and the capacity reaches 100T, memory 1TB, this design will need to optimize the cache more than 192G, after the flow comes in, it is possible to directly eat 0.99TB memory by writing cache, the program has been located at the beginning of the design with 10G to ha sh index, open the structure of various optimization algorithms, these two different hardware configurations, in the server system design level, is not the same :192G only need to consider the optimization of cache scale,1TB need to consider the optimization of cache scale + prevent crash, because hdd write big data encounter traffic > array write limit is very easy to crash, once the memory of the large array is used up, the array The IO capacity of HDD may be reduced to 5% of the original capacity, and there is no difference between the crash. Buffer control (flush) will be one of the core mechanisms directly brought to the foreground, which needs to control the key elements of big data input, array write mechanism and hardware lock as a whole.

Finally, gpu, once the server meets the gpu, the support devices from the cpu to the storage will be almost all high-end, at this time, the server in the program design link will completely break away from the classical single-threaded way, the process model will be replaced by the pipeline model, these pipelines will flow from one operating system to another operating system. At this time, the ZNet program will all take the route of Buffer+ thread, and this model just takes the data in, and the calculation body is a bunch of independent + huge computing support system.

Building Bridges

Bridge building is a communication pattern, a bit of a design pattern, which can actually improve the efficiency of server farm programming.

Bridge building only works in the command model with feedback requests

- SendStream+SendConsole, requests with queue blocking mechanism, will wait for a response
- SendCompleteBuffer_NoWait_Stream, does not queue requests, does not wait for responses

The workflow of bridging:

- A-> Make a request -> Command queue starts waiting for the model
- B-> Request received -> Request to enter delayed feedback model -> Bridge C
- C-> Receive request ->C Respond request
- B-> Receive C respond ->B respond back to A
- A-> Response from B is received, cross-server communication process is finished

Bridging is to solve the tedious data transfer process between server clusters with 1-2 lines of code,

```
procedure TDemo_Server.cmd_cb_bridge_stream(Sender: TCommandCompleteBuffer_NoWait_Bridge; InData, OutData: TDFE);
var
  bridge_: TCompleteBuffer_Stream_Event_Bridge;
begin
  // 架桥就是事件指向,剩下的让桥自动处理
  bridge_ := TCompleteBuffer_Stream_Event_Bridge.Create(Sender);
  deploy_bridge.DTNoAuth.SendTunnel.SendCompleteBuffer_NoWait_StreamM('cb_hello_world', InData, bridge_.DoStreamEvent);
end;
```

In the ZNet-C4 framework, there are dozens of kinds of servers, and the bridge technology is an efficient way to enjoy these server resources. All servers in the C4 framework support bridging and being bridged.

Write C4 server only according to the normal communication operation programming, directly consider handling the C end, infinite stack. To be completed after the opening of an application server, all the servers in the way of bridge scheduling up to use.

ZNet bridge support

ZNet Bridge support is the event prototype, reactive communication will have a feedback event, the event points to an existing automatic program flow, the feedback event is directly automatically processed when triggered.

- TOnResult_Bridge_Templet: Bridge feedback event prototype template
- TProgress_Bridge: main loop bridge, after being attached to the main loop of ZNet, each progress will trigger an event. This model in the early ZNet has not solved hpc load thread to do data search big process, the main loop bridge is often used for fragment calculation, for example, 10 million data search in the main loop stem is p each time rogress, search 100000, in order to ensure that the server is not stuck.
- TState_Param_Bridge: The bridge with Boolean state feedback
- TCustom_Event_Bridge: semi-automated reactive model bridge, a bridge that requires programming, such as accessing 10 servers, waiting for all of them to be accessed, and then responding to the requestor at once. C4 is used in large quantities.
- TStream_Event_Bridge:SendStream's automated response bridge
- TConsole_Event_Bridge:SendConsle's automated response bridge
- TCustom_CompleteBuffer_Stream_Bridge: Semi-automated CompleteBuffer response event bridge
- TCompleteBuffer_Stream_Event_Bridge: Automated response bridge for CompleteBuffer

C4 Start the way the script is written

win shell command-line way:

C4. Exe "server (127.0.0.1 '0.0.0.0', ', '8008,' DP ')" "KeepAlive (127.0.0.1 ', '8008,' DP ')"

linux shell command-line mode

./C4 \

```
"server(' 0.0.0.0 ', '127.0.0.1',8008, 'DP')" \
"KeepAlive(' 127.0.0.1 ',8008, 'DP')" \
```

Code method

```
C40AppParsingTextStyle := TTextStyle.tsC; // Use C-style text expressions for ease of writing scripts
C40_Extract_CmdLine([
'Service("0.0.0.0", "127.0.0.1", 8008, "DP")',
'Client("127.0.0.1", 8008, "DP")'
        ]);
```

C4 Start Script cheat sheet

Functions :KeepAlive(connect IP, connect Port, register client),

Parameter overloading :KeepAlive(connect IP, connect Port, register client, filter load)
Aliases, supports parameter overloading :KeepAliveClient, KeepAliveCli, KeepAliveTunnel,
KeepAliveConnect, KeepAliveConnection, KeepAliveNet, KeepAliveBuild
Client connection server, deployment type network connection, if the connection target is not successful will keep trying, after successful connection will automatically start disconnection reconnect mode. When deploying the server group,KeepAlive mode is mainly used to join the network. No matter how the C4 construction parameters change,KeepAlive will always repeatedly try the unsuccessful connection. KeepAlive mode solves the problem of the start sequence of the deployment server The order problem. KeepAlive does not search the entire communication server stack. It needs to deploy the DP service in the C4 network so that KeepAlive can join the network across servers. Simple explanation: to use KeepAlive to access C4 network, you need to mount a DP service.

Function :Auto(connect IP, connect Port, register client)
Parameter overloading :Auto(connect IP, connect Port, register client, filter load)
Aliases, support parameter overloading :AutoClient, AutoCli, AutoTunnel, AutoConnect, AutoConnection,
AutoNet, AutoBuild
Client connection server, non-deployment network mechanism, after network failure can not automatically repeatedly join the network,Auto is automatic network connection, can work in C4 network without DP service, suitable for use in the server started by someone's operation, once the network is successful, it will enter the disconnection mode.

Function: Client (connect IP, connect Port, register client)

Parameter overloading is not supported
Aliases, supports parameter overloading :Cli, Tunnel, Connect, Connection, Net, Build
Client connection server, non-deployment network mechanism, can not automatically repeatedly enter the network after network failure,Client function needs C4 target IP network DP service to enter the network.

Function: Service (listen to IP, local IP, listen to Port, register server)

Overloading: Service (native IP, listening Port, register server)
Alias, supports parameter overloading :Server, Serv, Listen, Listening
Instructions: Create and start the C4 server

Function: Wait(milliseconds of delay)

Alias, supports parameter overloading :Sleep

Note: Start delay, because the win32 command line without the use of powershell script, dealing with delayed execution is more troublesome

Function :Quiet(Bool)

Note: Quiet mode, default False

Function :SafeCheckTime(in milliseconds)

Note: Long cycle check time, default 45*1000

Function: PhysicsReconnectionDelayTime (floating point Numbers, the unit seconds)

Note: Time interval to retry the connection if the physical connection is lost after C4 is connected to the network. Default :5.0

Function: UpdateServiceInfoDelayTime (ms)

Description :DP schedules the update frequency of the server, the default value is 1000

Function: PhysicsServiceTimeout (in milliseconds)

Note: Connection timeout for physical server, default value 15*60*1000=15 minutes

Function: PhysicsTunnelTimeout (in milliseconds)

Description: Physical client connection timeout, default value 15*60*1000=15 minutes

Function: KillIDCFaultTimeout (in milliseconds)

IDC fault determination, disconnect time determination, reach this value to trigger IDC fault, disconnected client will be completely cleaned out The default value is h24*7=7 days

Function: Root (string)

Note: Set C4 working root directory, default value is.exe file directory, or linux execute prop file directory.

Function: Password (string)

Instructions: Set the password for C4 to access the network, the default value is DTC40@ZSERVER

UI function: Title (string)

Note: Only works with C4's Annotation UI template that sets UI window title

UI function: AppTitle (string)

Note: Only works with C4's annotation UI template that sets APP title

UI function: DisableUI (string)

Note: Only works with C4's annotated UI template, masking UI operations

UI function: Timer (in milliseconds)

Note: Only works with C4's annotated UI template, set the UI environment under the main loop in milliseconds period

C4 Help command

The Help command is C4's built-in server maintenance + development debugging command. Whether it's console or ui, help commands are built in, and these commands are universal.

Command :help

Instructions: Displays a list of available commands

Command :exit

Alias :close
Description: Close the server

Command :service(ip address, port)

Overload parameters: service(ip address)
Overloaded parameter: service()
Aliases :server,serv
Description: Server internal information report, including physical server information,p2pVM server, number of connections, traffic, server built-in startup parameters. If empty parameters will be easy to report.

Command :tunnel(ip address, port)

Overload parameters: tunnel(ip address)
Overloaded parameter: tunnel()
Aliases :client,cli
Note: Client internal information report. If the parameter is empty, it will report easily.

Command :reginfo()

Note: Output the registered c4 service, each service of c4 will have corresponding CS module, for example DP will have dp server +dp client.

Command :KillNet(ip address, port)

Overload parameter: KillNet (ip address)
Direct IDC fault mode to kill the corresponding c4 network service

Command :Quiet(Boolean)

Overload parameter: SetQuiet(Boolean)
Note: Switch quiet mode, in quiet mode, the server will not output the daily command execution status, but there are prompts for errors, such as command model execution exception

Command :Save_All_C4Service_Config()

Instructions:

Immediately save the current server parameters, this is a server expansion parameters, when c4 heap after too many server parameters,shell command line maximum limit is 8192, under normal circumstances, can not write too many start parameters in the shell command, so c4 provides a file form of parameters loading, in the default case, and no parameters file

By Save_All_C4Service_Config() can generate the suffix.conf server parameters file,.conf file is stored in the depnd subdirectory corresponding to the current server directory,.conf is a configuration file in ini format.

If you use.conf as a server parameter to start c4, the command line arguments will be overwritten.

Save_All_C4Service_Config() is mostly used to deploy startup arguments when running the server for the first time, mainly to reduce the size of the command line input. In real system integration, the command line reaches 200,300 characters, which is very difficult to read and modify. Therefore, the.conf startup parameter is an important part of deploying c4.

Command: Save_All_C4Client_Config()

Description: Immediately save all C4 client parameters of the current build completed, the effect is basically the same as Save_All_C4Service_Config(). In the system integration work, the client side parameters are very few, this is can be written directly into the shell command line, but if you want to beautify the command line, make it easy to read, then use the file parameters.

Command: HPC_Thread_Info()

Note: Immediately output all TCompute thread instances in the current process,Z threads are created and executed using TCompute, and each thread will have a thread_info string identifier, used to identify the role of this thread. In ZNet threads will be very diverse, there are HPC offloading thread,CompleteBuffer background decoder encoding thread,ZDB2 thread. If the RT library comes with the TThread directly, then HPC_Thread_Info() will not output the thread status.

This command is mostly used for server debugging, analyzing performance bottlenecks, and finding bugs

ZNet_Instance_Info()

Alias: ZNet_Info()

Note: Immediately output all the IO instances of ZNet, including physical connection,p2pVM connection. It is mostly used to diagnose the connection status and analyze the problems encountered when C4 is connected to the network.

Command: Service_CMD_Info()

Alias: Server_CMD_Info()
Description: Immediately output the cpu consumption statistics status of all command models in the server, these commands will be very many, hundreds. Mostly used in the analysis of performance bottleneck location. Service_CMD_Info() also contains a count of the number of commands sent, but not the cpu cost of sending the command.

Command: Client_CMD_Info()

Alias: Cli_CMD_Info()
Note: output all client immediately ordered model of CPU consumption statistics, and Service_CMD_Info () format is almost the same, because the C4 is an interactive network, the client statistics can reflect the server delay.

Command: Service_Statistics_Info()

Alias: Server_Statistics_Info()
Immediately output all the internal statistics of the server, including IO trigger frequency, encryption calculation frequency, the main cycle frequency, the amount of data sent and received and so on key information, the server will include physical server +p2pVM server

Command: Client_Statistics_Info()

Alias: Cli_Statistics_Info()
* * * Note: Output all client internal statistics at once, in almost the same format as Service_Statistics_Info()

Command: ZDB2_Info()

ZDB2 is a set of hierarchical architecture database system,ZDB2 has progressed to the third generation system, here ZDB2_Info () is also the output of the third generation of ZDB2 system, in the C4 framework integrated FS2,FS, such services, all C4 services made in 2021, are the first The second generation of ZDB2 system, can not be ZDB2_Info() unified output state. The amount of information and design of ZDB2_Info() output for the third generation system is very large, and can only be discussed here: it is a good way to use ZDB2_Info() to see the status of the database and background of the sixth generation monitoring.

Command: ZDB2_Flush()

Note: the ZDB2 write cache immediately into the physical device, the use of information and ZDB2_Info() are very large amount of information, here can only pass: in the sixth generation of monitoring data background debug array system hardware command, need to be combined with disk cache monitoring, memory monitoring, physical IO monitoring together to use. The function is to analyze the IO bottleneck of the array system.

ZNet Kernel technology - Lock multiplexing

Critical-Section is a hardware-based thread lock technique in operating systems

The more threads there are in a process, the more Critical-Section there will be. In the system monitor, the number of threads + the number of process handles can be seen. After the number of the two more, the whole system may be unstable, at least when the process is analyzed or the system crashes, the thread + handle of the target process are two very important indicators.

Generally speaking, each thread will correspond to at least 1 more Critical-Section handle, depending on the specific process.

Z kernel takes the multiplexing route for Critical-Section, ZNet server will see the handle peak in the monitoring when running, but this is not the real Critical-Section, you need to input hpc_thread_info through the command c4 console to see the real Critical-Section and thread status.

ZNet Kernel Technology -Soft Synchronize

Soft Synchronize technology is the main thread Synchronize of emulation rtl.

The design mechanism of ZNet relies heavily on the main Thread, so the Thread Synchronize architecture is used extensively. Thread Synchronize is also used to control the DIOCP/CrossSocket thread call in ZNet's asynchronous communication library. The use of more or WaitFor thread mutual exclusion wait, if the queue is not processed, send the exit command to the thread is easy to get stuck inside, this time the problem is often not correctly cleared by the outside program between the execution of the thread call, clean up the operation between the execution of the program, in fact, CheckSynchronize, this is a master Thread specific synchronization queue to execute the call. Whenever a thread Synchronize operation occurs, it can only respond through CheckSynchronize. In the vcl form system, this is automatically called by the application. If you bypass the application, you need to master the main loop technology.

For example, given in Thread.OnTerminate, if there is no CheckSynchronize outside, the event will not be fired.

Soft Synchronize solves the event passing mechanism between threads and replaces CheckSynchronize.

Kernel :Check_Soft_Thread_Synchronize

The emulation main thread Synchronize code is executed and the RTL system Synchronize code is not executed. Z is all use the approach to perform the Synchronize code, including DIOCP/Cross/ICS8 / ICS9 / Indy/Synapse. For example, when the cross/diocp asynchronous library uses the waitfor operation, this will make the Synchronize of the main thread perform the simulation work while the wait is in progress.

When the compile switch Core_Thread_Soft_Synchronize is turned off, the RTL system Synchronize mechanism is used.

This API mainly supports programs that need to run in a dual-main thread environment.

Kernel :Check_System_Thread_Synchronize

Executing the RTL system Synchronize code also executes the emulation main thread Synchronize code
The state is handled automatically, regardless of whether the compilation switch Core_Thread_Soft_Synchronize is turned on or off.
This API mainly supports running programs in the main thread environment.

ZNet Kernel Technology - Dual main thread

ZNet can open two main threads in a single process at the same time. When the dual main model is started, the following happens:

- The whole ZNet system and the various libraries included in ZNet all work on the secondary main thread.
- All RTL systems, including native lcl,vcl, and fmx, work on the original main thread.
- The secondary main thread and the original main thread will each maintain their own main loop. The main loop technique is omitted here
- The Synchronize technique is used to access the data between the primary and secondary threads
- Dual main thread technology can support win/android/ios and Linux programs built by fpc
- With UI programs, running the server will not have a sense of stagnation

- Putting ZNet in 1 dll/ocx is equivalent to opening 2 exe, in which exe and dll each walk a main thread
- The secondary thread can run http,c4,znet

The RTL primary thread is synchronized to the secondary main thread

After dual master mode, the data of the secondary main thread is accessed from the RTL main thread

```
TCompute.Sync(procedure
begin
  // Access ZNet data here, including c4,cross,diocp,ics these communication data
end);
```

The secondary main thread synchronously steps to the RTL primary thread

After dual master mode, it is from ZNet to access the data of RTL main thread, it is from ZNet to access VCL/FMX

```
TThread.Synchronize(TThread.CurrentThread, procedure
begin
  // Here access and modify vcl/fmx,UI in these data
end);
```

The main loop after the double main thread is opened

The ZNet secondary main thread API,Check_Soft_Thread_Synchronize, is located in the Z.Core.pas library

The RTL proto-master thread API,CheckSynchronize, is located in the vcl-System.Classes.pas/ld-classes.pas library

Guide to using the ZNet Performance Toolbox

CPS Toolbox =Caller Per second tool. All cps count cycles of 1 second. Located at Z.Coore.pas Library.

- CPS_Check_Soft_Thread: Secondary main loop performance counter.
- CPS_Check_System_Thread:RTL main loop performance counter.

Access method, CPS_Check_Soft_Thread.CPS, the value is the number of calls per second

All instances of ZNet have built-in CPS performance counters, which are used to calculate how often the server main loop is called per second as well as the cpu usage..

Performance Bottleneck Analysis

Start any C4 program and type hpc_thread_info from the command line.You will get the following feedback

RTL main-thread synchronize of per second:0.00,RTL Main loop calls per second

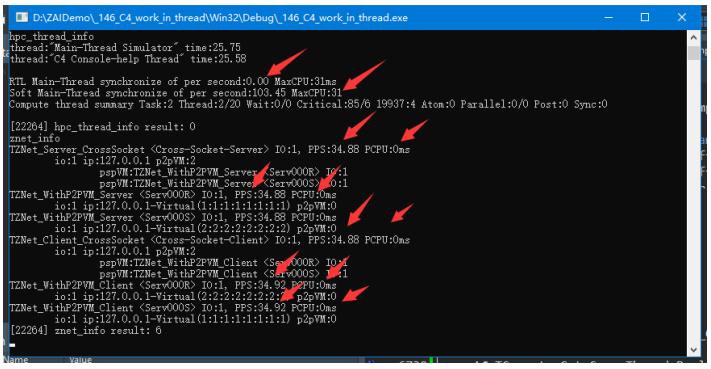
Corresponding to CPS_Check_System_Thread

Soft main-thread synchronize of per second:167.32, secondary Main loop calls per second

Corresponding to CPS_Check_Soft_Thread

Results: This C4 program, using the secondary main loop technology, the main loop occurs 167 calls per second, the higher the call frequency, the better the fluency, if the program stalls, you need to check the point of the stall: maybe a function occupied the conduction to the main loop, resulting in low CPS, especially the program with timer events.

Command line to type znet_info, get the following feedback



PPS: How often progress is called per second in ZNet

PCPU: Maximum cpu consumption of progress in ZNet

Result: Locate the instance in which the long process of the main loop is stuck, these instances can be the server or the client. After locating, we use Service_CMD_Info and Client_CMD_Info to find the bottleneck where the command is stuck.

If the program is not using C4, you can use the API, ZNet_Instance_Pool.Print_Status, to output the status directly

Exclude ZNet overlapping Progress

Firstly,progress has an automatic anti-loop mechanism, and the progress package progress will not loop endlessly.

C4 has optimized the progress overlap problem, each call to C40Progress can ensure that each ZNet instance will only trigger Progress once

In non-C4 frameworks, Progress can be triggered automatically by p2pVM DoubleTunnel. A single loop of the main thread may trigger 2-5 times more progress, which is a meaningless consumption. If the server is under heavy load, repeated overlapping progress will make the shard load can not be accurately estimated. This is a problem that must be solved when precise optimization is required.

Use the following program paradigm

- Server.Progress;
- Server.Disable_Progress; Disabling calls here will not trigger server.progress later
- other.progress;
- Server.Enabled_Progress;

The physical p2pVM tunnel instance will automatically iterate over all p2pVM virtual connections in it each progress.

When this is done, use ZNet_Instance_Pool.Print_Status to check the cps. If the cps value of the ZNet instance is similar to the cps value of the main loop, the progress is basically fine. Next, test the connection, process the command, and once all pass, then the resolution of the overlapping progress problem is complete.

Awe server main loop progress

Almost all server optimization efforts face the main loop problem, and there are a lot of solutions involved

- Threads: Threads are good if the main loop code is thread-safe, thread-safe is not lock-safe, but thread + main loop can't lock if the main loop is open in the thread.
- Sharding: The technique of sharding is to split up the computation so that only part of the main loop runs at a time
- State machine: A state machine allows the main loop to simply omit certain code under certain circumstances. This is especially true when it comes to flows like for and while
- Structure and algorithm optimization: The main loop will handle a lot of structures. Structure optimization, such as hash and biglist, can improve the efficiency of the main loop.
- CPU instruction level optimization can be ignored: such as sse,avx, this kind of optimization is difficult to say, it is difficult to improve several times, can not be compared with the algorithm level optimization, algorithm optimization generally starts with 10 times improvement.

The main loop is the lifeblood of the main thread,90% of the server scheduler is used to complete the main thread, child threads and coroutines are mostly used to share some special tasks. For example, in the pas circle, many servers like to use a ui and see the running status of the service request at any time. The ui of this vcl/fmx/lcl route runs under the main thread, which is deeply affected by the main loop.

The fully threaded server, such as erlang,go, will have a thread scheduling problem, here will use many complex program mechanisms to control the cooperation between the threads, and the fully threaded computing server will not make the project very smooth, let alone invincible, because the server bottom layer is affected by the hardware limit.

The main loop + child thread will still be the main model of the server in the future. Having no problem with the main loop can be equated to having solved 50% of the server performance bottleneck!!

ZNet kernel technology :TCompute thread model brief introduction

TCompute is not a thread technology, but a standard model of using thread programming.

TCompute has a very large downstream dependency system, and without TCompute, these downstream dependencies would break down

- Z-AI: GPU driver in AI system is thread binding,DNN-Thread technology is to open a thread in GPU and CPU each, and let it have a binding relationship, for example, a picture will be passed to the gpu during identification, which is to call the thread api, and then execute cuda copy in the thread, and finally caller gpu uses dnn library to do parallel calculation and get results, the whole IO process is threaded work, and the main thread goes to identify IO process mostly belongs to the demo demo api and mechanism. Regular gpu program threading technology is used in scale.
- AI toolchain: Since AI involves the field of big data, all data operations are unlikely to be completed instantaneouly, so they are all open threads, and their process is, lock UI, run thread processing,UI unlock. The most typical application is AI_Model_Builder, and there is often a granularity level data operation that takes several minutes.
- ZDB1: This is an old chained database system. In 2017, a major upgrade was made: the query process of ZDB1 was encapsulated into pipe, and then the main thread synchronize mechanism was used to schedule the query. As a reminder, when using zdb1, just checking Check_Soft_Thread_Synchronize (100) on the main thread will at least double the query speed. The secondary soft synchronize mechanism is more efficient than the rtl library CheckSynchronize. However, the problem of ZDB1 is also obvious: in 2017, the depth of the thread is not enough, resulting in the formation of a system solution, which can only be regarded as a flash in the blue, mostly used as a database partner, file packaging, installer and other small functions. Unable to go deep!
- ZDB2: can depth technology system, the future is three-dimensional form, the whole program as a data application model before and after three years in the design of perfect, just to build the foundation, has experienced three generations of appli cation system, in the current advanced generic structure + advanced thread technology, the development route can be basically clear: big data foundation support technology system, in ZDB2 system, a database, It can be run by 10 array disks, or 10 array systems together, and the technical solution used to drive these large arrays is threading, and each database file api works in an independent thread. ZDB2 in operation, the internal can vary from tens to hundreds of threads, even in the 10TB scale of small database IO threads can also eat >100GB memory +>20 core cpu, we do not use the array system thinking to look at ZDB2, data engine system and files can not be directly compared.
- Parallel program: Parallel program is a modern process must have a technical solution, but the parallel program also has a very obvious problem, the parallel support library used by fpc/d is not ideal: can not support the specified correlation core and hyper-thread, and even can not specify the number of parallel threads per start, and the biggest problem is the compatibility of the library and parallel granularity model, such as for and The line granularity can be block parallel or fold parallel, these places must be unified in order to heap large, otherwise the parallel program can only be used as a function point to solve the local acceleration problem, can not be used as part of the modern technology, because it can not heap large. Kernel parallel technology in space + time complexity + mechanism can be better than D +LCL system.
- The sixth generation monitoring system: the AI server side of the sixth generation monitoring, a quasi-GPU server + an AI server application, can take 80 4k videos, and the amount of data per second is calculated as :3840*2160*4*25*80= about the amount of data to be processed per second is 60G. This abnormal data scale requires a very deep grasp and practice of the whole architecture of topology + switch + network + thread +NUMA+CPU+GPU to make the process plan. And this process can not be done in one step. It is necessary to simulate and verify one by one from the local separate link, solve the bottleneck and optimize one by one, and then combine the process. The result of doing this is that the cost of computing power will directly drop by 5-10 times, which is from hell to heaven.
- Full server system: the main loop of the server and the thread are ininterlinked. Once the main loop encounters a process card with a amount of calculation for 30 seconds, it is common for people to keep the server running, observe the state, and then optimize the work. It can be said that the native TThread lack of stack and scheduling mechanism, and only the specification can meet the interadjustment between threads Mutual equality,TCompute is a thread model mined from the computer to mechanism, can really heap the thread system and unify the d/fpc specification.

The kernel technology of ZNet: a brief introduction to its architecture

The structure here is not the data structure in computer science, but the basic structure system of algorithm application.

Algorithm application is a kind of computational engineering, which will need to be unified, standardized, explorable, regular, and social. The structure system is the data source of computational engineering and the pre-design of the design algorithm. Because the algorithm program will need to pay the cost of human time, once the time reaches a certain degree, the algorithm scheme may not be ideal. Structure is a kind of preparation for solving. Without this preparation, the algorithm will become very difficult, and any idea will take time, even a lot of time.

Taking ZDB2 as an example, the core work of ZDB2 is only responsible for data IO, these IO are in a very complex schedule to work, the purpose of these works, is to provide the original data to the algorithm, the algorithm to convert the data into the data source, and then, is the calculation process. For example, loading 1 billion data from ZDB2, using the algorithm to do a speedup to 1 billion a certain key, the process to achieve this thing is to use the universal structure, such as TCritical_Big_Hash_Pair_Pool, because the IO of ZDB2 is all threaded, need a hash universal structure with lock, at this time, in the IO to The hash structure heap data pointer can be used to complete the simplest 1 billion millisecond query. As we go deeper, we can extend, delete, modify, count, and so on, all using processes to manipulate the structure. When this thing is solved, the special data engine will be completed, which is different from the general DB engine, the special engine is omnipowerful, as long as you want, you can use GPU to run sort+sum, and the calculation speed, performance, I think the general DB engine can only match.

Take ZNet as an example, in the ZNet framework, there are a large number of queue mechanisms, such as progress inside traversal TPeerIO is first copy all IO Pointers to a container, and then traversal the container, if the cpu time consumption is too large, exit, form a fragment processing mechanism, until the next progress will continue to process This IO container, when all the container processing is completed, then copy the container pointer again, and start the next fragmentation. ZNet's send, receive mechanism, on the other hand, also uses containers, instead of mindlessly copying into a stream. When the queue container structure is widely used, if the TList mechanism is used to deal with the queue, it is very useless. Every time the first queue is extracted, some copies have to be experienced. I believe many people in the pas circle have complained about TList, but TList has a coherent 1D data space, once the queue is extracted and deleted, the optimization can only change the pointer, otherwise it is a copy This efficiency is very painful. Finally, the length limit of TList is also basically unsolvable. The biggest advantage of TList is that it is easy to use. In small data scale projects such as UI, it is no problem to put it on the server. .. The data container used by ZNet is a chain structure. This chain structure is composed of small memory blocks connected by next Pointers, which is very suitable for queue extraction requirements. TOrderStruct to name, in the kernel library, sticky packet process,1 packet 1k,10M data is 10000 queue packets, in high traffic,TList computing power and TBigList can not be compared, this gap, using the test program will be 2000 times.

Take the method of Learn statistics as an example, the structure system can be regarded as an IO language, that is, the input is a structure, and the output is another structure. For example, classification algorithms, the principle of these algorithms can basically calculate junior high school students homework, first use some calculation methods, or random methods, generate some seed form of data, and then around the seed to search and match, all classification algorithms, basically this idea, change, perhaps it will be unconstrained, but the process is often very simple! And data access and output, this is the most troublesome calculation, 600585 seems to be the complete classification process of the project, pre-processing 45%, calculation 10%, post-processing 45%, to summarize: the proportion of the structural system in the statistical process will be greater than the algorithm. We usually see that calling an api and getting solve, the process is directly encapsulated, and it is also normal to package dozens of structures + classes. This is not a statistical algorithm, it is a solution, and the internal steps are 1,2,3,4,5 sequence to solve the problem. Face recognition, recommendation algorithm is this kind of model.

Taking thread as an example, thread is a big system, modern programs do not use threads, and threads and thread communication, the structure written out of thin air can not be stacked up to do projects, thread intermodulation,TThreadPost,TSoft_Synchronize_Tool, these universal structures can basically solve each other in 1-3 lines when used Intermodulation and so on, minimal use, otherwise how can heap large?

The most frequently used large struct in ZNet system: TBigList<>, TBig_Hash_Pair_Pool<>

Common small structs in ZNet: TOrderStruct<>, TAtom<>, TPair<>

The general structure of ZNet is commonly used Hash library Z.HashList.Templet

ZNet classic linked list library Z.ListEngine

ZNet kernel technology: a brief introduction to the structure of the combined fist

Taking SVM and K-Cluster as an example,SVM can be considered as a representative nonlinear algorithm, while KC is more linear. The calculation process of the two can be basically the same: first input, then input preprocessing, get operator data, and then calculate the final classification results. The derivation idea of SVM is dimension switching, and the dimension part is the core idea of the whole SVM, focusing on traversing the single-dimensional data equation, obtaining the single-dimensional points that can be divided by the optimal plane through traversing, and then combining several single dimensions is the hyperplane cut point, and then cutting the hyperplane from the maximum to the minimum according to the span, and then using the dimension to calculate the clustering, the idea process is to take the encoding and decoding route This step can have a bunch of optimization measures, if it is not optimized svm will be a very simple process idea, and svm can write thousands of lines, can also be dozens of lines to solve, because there is a dimension span, these spans can act as a memory condition, that is, training modeling, using the model this process,svm can also do very simple,pass system can also use dozens of lines To make svm automatic classification, the premise is that there must be structure support. The classical practice I refer to before is mainly from shogun(the origin project of c++ classical svm), if you are interested, you can find your own research. K-Mean derivation is a random number to generate the centroid classification, all the adjacent clustering, and then define the new centroid to go through the process again after completion, and iterate several times to complete the clustering. At the structure level, as long as the definition, input and output, the rest can be directly handed over to various open source computing libraries to do. We usually look for information on the Internet, a variety of complex formulas, which is a kind of expression language on the idea, the algorithm process, especially the main part, is not too complex. It is more difficult to understand the field of non line. If the self-created things are counted in this field, there can be hundreds of algorithms, and the core idea of non line is to do decoding and encoding preprocessing for data.

Taking high-speed range search as an example, this field has not yet had time to write a demo, its goal is to solve the rapid search within the range, for example, the amount of data to 1 billion, and the data at any time to add and delete, to search the time range and coordinate range, if the algorithm is not brute force traversal, perhaps a process will go for several minutes. The more effective way is to cut the time and coordinate range according to the measurement, such as disk array file coordinates, can be cut out every 1M area for hash, processing the range with 1M as a small span to memory, time segmentation is the same, can be cut, can also be cut on time, after the segmentation only need to be entered once can accurately locate. ZNet's approach is to cache Pointers,hash spans, reference library for Z.HashMinutes.Templet, implementation and combination mainly use TBig_Hash_Pair_Pool<>+TBigList<>. Among them, the time range acceleration algorithm is mainly used to search the monitoring fragments, basically all can be searched in seconds, and the coordinate range acceleration algorithm is mainly used to solve the emulation write cache, which is the function of emulation writing to the file and ensuring the consistency of reading and writing. It needs to be in the high-speed read and write environment, for example, to write 100 length coordinates at 1024 position, at this time, it is necessary to find a series of cache at 1024 position part buffer, so as to complete the file read and write data consistency.

Take the bidirectional matching algorithm pair as an example: bidirectional, each completion of a pairing, will traverse all targets, if optimization is not considered, matching 1000 to 1000, the amount of calculation is ten million, when each pairing is completed, eliminating the paired data, the amount of calculation will be much smaller, and then with thread, parallel these means, bidirectional pairing can be very fast. To solve the problem of deleting pairs,TList cannot be used,TList will reconstruct the array buffer, which is very cpu consuming, so TBigList<> must be used.

Take parallel sorting as an example: I have no idea of the fastest parallel algorithm on earth, my approach is to first store and then sort, for example 1-10, 11-20 in separate blocks, then use parallel sorting granularity blocks, and finally sort the whole block + build the output. The struct is TBigList<>, only TBigList can support large numbers like 1 billion, directly using memory Pointers would make sorting very complicated.

Review: Designing a generic TBigList<>

TBigList was not designed in one go. It's a bit of a historical review

1. The original predecessor of TBigList in 2015 was THashList located in Z.istengine library. This was the first Hash library written by the author, and it used TList extensively for conversion and saving functions internally

2. Around 2016 to 2017, there was a demand for sequence restoration of THashList, and it was impossible to recall what sequence was restored. In short, after adding 123, it should be possible to directly restore the order of 123 from THashList. This demand led to the transformation of the internal structure of THashList from single to chain.
3. TOrderStruct appeared in 2020, when it was decided to gradually transplant the whole code from the classical structure to the universal structure.
4. In 2020,TBigList was written. When TBigList was published, it was a very big fix because the test case was well written and the THashList stubborn century-old bug was fixed. After that, TPair<> and TBig_Hash_Pair_Pool<> were written in parallel
5. In 2021, TList will be deleted in all code of ZNet and replaced with TBigList<> universal structure.
6. One day in 2021, I was bored and wrote a small demo of BigList pk TList. TList used the optimal deletion, addition and modification methods to perform one-time pk with BigList. The result is that the processing capacity of BigList is almost 2000 times that of TList

The design idea of TBigList: First of all, we must solve the high-speed queue +Int64 level data volume + stacking programming that can be used in large-scale loop code: we must solve the loop demand for. Among them, the requirement of solving for loop is even higher than the requirement of performance. In simple words, for must be a very simple process model, and anonymous functions can not be used.
Finally,TBigList was designed into today's general general structure, on this basis, later,ZDB2, a variety of new algorithms, new structure systems, came into being, these structures and algorithms are too many,Z-AI system directly do not need to mention.
ZDB2 is written, in fact, it has declared that the independent technology has entered the era of big data. The next is to use time to iterate.

Review: Designing the scripting engine ZExpression

A long time ago, the compiler has been the author's regret, the original idea: the theory of a lot of, if you can't start to write once, the theory will be empty, this thing must have a personal experience, eat, drink and play a meaningless life (today's feeling is no money is meaningless).

- String parsing is the first problem in the face, the string parsing program is very complex, the beginning is to do word segmentation + word function, and then, began to try to do symbol parsing, inverse Polish, addition, subtraction, multiplication and division, the various characters into prototype structure. Finally, the lexical conversion was solved.
- The second step is to start trying to address lexical semantic structures, such as lexical legitimacy, where the lexical structure is a tree. Because the lexical structure of 1+(1-2) is equivalent to 1+a, and the structure of a is (1-2).
- The third step is to try to translate the lexical structure into the opcode code of the executable program. This mechanism is realized by emulating the opcode code, and the mechanism works in the same way as x86 decoding, the difference is that the code table is different.
- The fourth step, begin to greatly strengthen the Parsing support link, infinite approximation bison+yacc, technical details are omitted here, put in the next section.
- Step 5, start applying the ZExpression architecture :C4 startup,ZAI scripting, Generation 6 scripting,pascal code rewriting model, which are all induded in the ZExpession architecture.

Maternal transfer technology of ZNet: Z.Perkins

ZNet is a giant code project as a parent + carrier. These codes are actually generated by machine compilation technology analysis + reconstruction. We usually use prp to transplant ZS, or prp to upgrade ZNet, which is a data model + analysis and reconstruction technology at work.

The analysis + reconstruction technology in the compiler is based on the bison+flex/lex+yacc system.

Take a simple example: particles

if(1+1=2) kill;

if 1+1 = 2 do kill

These are two completely different lexical bodies. In the bison/yacc system, these are described by code expressions, which are scripting languages, and then encoded into a unified structured data that can express the flow.

In the Z.Sing system, there is a mechanism that can win without a trick, the probe technique

The probe technology is based on the pre-work of part-of-speech, the Parsing system will first, number, symbol, floating point, string, note,ASCII, and so on, they will be divided by part-of-speech, forming a part-of-speech chain structure.

For example if the kill (1 + 1 = 2), its part of speech for chain: ASCII, symbol, num, symbol, num, symbol... .

Probe technology, is to make an approximate judgment of the part of speech chain, and then enter the branching process.

Probe technology can distinguish between different combinations of part-of-speech structure + lexical body, and it will form conditional normal forms, which are the design idea of bison/yacc.

When the lexical program has the condition detection, it can use the ant to crawl and use the probe conditional normal form to open the branch program to deal with the random handwritten lexical body.

It is precisely because of the powerful probe mechanism that pascal rewriting model technology can correctly and completely p arse the pas code and reconstruct the object code it wants.

In the other direction, the internationalized note and string machine translation technology is also using the Z.Persing system, and the string translation is very simple and violent, that is to find the data structure translation of the string and the remark, and then reconstruct a different language. International project free you can go to https://github.com/PassByYou888/zTranslate

The probe of Z.Persing is flexible and free, it can hit where it points. In the ant program, it can make things that bison/ya cc cannot achieve. At this time, Z.Persing is the technical system that rules the cash flow: as long as the person using Z.Persing can write his own language correctly, and he can get the collective consistency to use his prophecy, he will become the godfather of software, games, an d programs, which will give him more power than any investor, general manager, or chairman Absolute technical voice. If the company is valued at $10 million, I wouldn't be surprised if the guy who used Z.Perkins to build the company's production system would be worth half of it. Because the company depends on the product, the product depends on the collective production, and the production core technology depends on the Z.Persing users.

On the other hand, Z.Persing system, no matter HTML,JS,Pas,C++,XML, can be done.

How to overturn a project using ZNet

If previously using ZS to go physical double channel items, overturn direct swap C4.

Overturn Direct Swap C4 if a non-C4 dual pass was previously used.

If previously using ics,indy,win socket non-Web class items, directly overturn for C4, full roll

If the project is already C4, you can change the registration name,RegisterC40('MY_Serv', TMY_Serv, TMY_Cli), and then start a new project based on C4, RegisterC40('MY_Serv2.0', TMY_Serv2, TMY_Cli2). Simply change the delta, add the version number to the cell name, and Reg the new service.

How to use ZNet to develop web projects

The data communication layer directly runs ZNet, and the UI layer uses webapi to access ZNet projects. For example, Post+Get can cover 90% of the webapi requirements, and ZNet comes with a webapi demo project.

ZNet with http and the web

ZNet= Large CS server

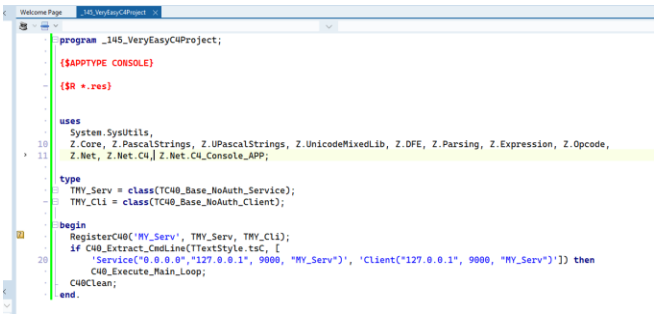
http= Communication protocol

web= Worldwide Wide Area Network, Internet

web contains ZNet, in the web environment with apache,nginx bridge communication module of large websites everywhere, perhaps readers can try to use a secondary domain name or domain server to do the bridge module and diversion, internal if it involves big data or

complex protocol, directly use ZNet as a communication layer package api for the web.

Text finally to a minimalist C4 CS demo



```
program _145_VeryEasyC4Project;

{$APPTYPE CONSOLE}

{$R *.res}


uses
  System.SysUtils,
  Z.Core, Z.PascalStrings, Z.UPascalStrings, Z.UnicodeMixedLib, Z.DFE, Z.Parsing, Z.Expression, Z.Opcode,
  Z.Net, Z.Net.C4, Z.Net.C4_Console_APP;

type
  TMY_Serv = class(TC40_Base_NoAuth_Service);
  TMY_Cli = class(TC40_Base_NoAuth_Client);

begin
  RegisterC40('MY_Serv', TMY_Serv, TMY_Cli);
  if C40_Extract_CmdLine(TTextStyle.tsC, [
      'Service("0.0.0.0","127.0.0.1", 9000, "MY_Serv")', 'Client("127.0.0.1", 9000, "MY_Serv")']) then
      C40_Execute_Main_Loop;
  C40Clean;
end.
```

Complete the text.

The 2023-10-6

by.qq600585