# Page 3:

首先，需要简单介绍一下TDX。TDX 技术是一个比较大的话题，如果要完整地介绍TDX，将非常困难。这里，我尝试用最简单的话来概括什么是TDX。

简单来说：

At first, a brief introduction of TDX is required. But TDX technology is a very large topic, it's very hard to introduce TDX detail in this meeting.

Here, I try to give a simplest summary about TDX.

In short:

什么是TDX?

TDX的目标，是为了提供一个硬件隔离的，机密的虚拟机。

What is TDX?

Intel TDX is to provide a hardware-isolated, confidential Virtuanl Machines. Which called Trust Domain.

TDX 基于2个技术：

1. VMX extensions。 VMX 是一个现成的技术，它是虚拟机软件所依赖的底层技术。

TDX，是在原来VMX的基础上，增加了新的指令集，以实现新的TDX需要的功能。

2. MKTME: 全内存加密技术

TDX mainly base on two technologies:

1. VMX extensions： Here, VMX is traditional hardware virtualization technology. It is the foundation of all virtualization software. For example, qemu / virtual box

2. MKTME：This technology is to encrypts the platform’s entire memory with multiple encryption keys. TME, when it is enabled in BIOS configuration, it will ensures that all memory accessed from the Intel processor is encrypted.

# Page 4:

现在再次简单地介绍 glibc-bench 的SW stack.

Now briefly introduce the SW stack of glibc-bench again.

什么是Glibc

1. Glibc 是 Linux 系统的核心库
2. 它提供了基本的Api：比如 open / read / write / malloc / pthread\_craete
3. 它是连接 用户空间和内核空间的桥梁，通过系统调用进入内核空间。

What is “glibc”

1. Core library of Linux system
2. Foundational facilities APIs: open, read, write, malloc, printf, pthread\_create…
3. Which is a bridge of user space and kernel space , user space enter kernel space via system call.

What is “glibc-bench”

1. Firstly, the source code of “glibc-bench” is in package “glibc”. That is to say, there is no specific package called “glibc-bench”.
2. “glibc-bench” is used to measure the performance of a part of the glibc APIs
   1. math, string operation, memory allocate, thread create . But, for other APIs, such as: socket, file operation, and the other. Glibc-bench didn’t provide corresponding test cases.
   2. Glibc-bench will call an api repeatedly, and then record the consumed tsc to measure how much time it takes to call an api once.
3. For some APIs, glibc support multiple instruction sets. For example, for Intel X86 architecture, the same Api, memcpy, glibc support SSE, AVX2, AVX512 at the same time.

Therefore, glibc-bench provides a comparison for the performance of different instruction sets.

# Page 5

This page will introduce the KPI definition

For glibc-bench, the main KPI, is the average run time

average run time is equal of Time stamp Counter required to call api once.

There is a sample as below:

Left part is …… This is a math function

Here

Duration is 2.20683e+09 , which mean “2.2 multiply by ten to the power of nine”

Iterations: 1.29888e+08, which mean “1.29 multiply by ten to the power of eight

So, the average run time is 16.9 TSC

# Page 6

From this page, start to introduce how to define the test case.

After an initial analysis of the test data, at least I found two pain points of how to define the test case.

1. is the test result data is massive

Please see below table:

For workload “glibc-becnh”

There are 6 categories of test case: malloc / math / pthread / string / stdlib / stdio ..

There are 188 binary files, every binary file is a test case.

There are 80 output files, and more than 290000 test results.

It seems not necessary to choose all data as the final result.

2. is the data consistency is not good.

Please see below table: these are two samples about data consistency

First, I run the full test 10 times, mark by Round 1 to 10

Then, Select two groups of result to analysis.

1. Sample 1 is a good sample,

**standard deviation**

Sample 2 is a bad sample,

# Page 7

Here, I introduce a new variable, coefficient of variation. short call is CV

For the definition of CV, please see right picture.

下面的图片是一个具体的例子，当测试结果的一致性比较好的时候，CV的值非常小，反之，当CV的值很大的时候，说明数据的一致性很差

The following picture is a example. When the consistency of the test results is good, the value of CV is very small. On the contrary, when the value of CV is large, the consistency of the data is very poor.

# Page 8

Finally, a summary is given. When to define the test case set of “glibc-bench”, the following principles need to be followed:

1. Test coverage needs to be comprehensive. So I select test cases from every category.
2. No need to select all test cases for one category, So I only select some typical cases for one category.
3. The data consistency is very important, therefore, I choose that test cases with better data consistency as much as possible

The final test cases are as right table.

# Page 9

This page introduce the **resource usage** for workload “glibc-bench”

Because this workload mainly runs in TDVM, most of the system resource requirements come from the TDVM environment itself , but not workload 'glibc-bench'.

'glibc-bench' itself is very simple, most of the test cases only run on a single thread, and at the same time, the memory requirement is very low.

Below table list the major resource requirement:

# Page 10

This page introduce the POD design and provisioning

For “glibc-bench”, single k8s node is enough.

# Page 11:

This page introduce the “Run stage”

1. For setup stage:

No need to add extra setup stage.

1. For warm up stage:

Glibc-bench implement self warm up code.

# Page 12:

Finally, the next steps:

I think in next steps, we can try the following optimization plan:

1. Try an

2. Try icc,

Since TDX must run on the SPR platform regardless of compatibility, it is reasonable if we use better compile optimizations.

3.To