**adding zstd comp/decomp tools on sel4 - Test Design**

1. **Change History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Remarks** |
| v1.0 | 10/24/2024 | dale.qi | First Version |

2. **Overview**

Zstd 是一种快速的无损压缩算法，是针对 zlib 级别的实时压缩方案，压缩后产生 .zst 格式的文件。

经过开发研究，zstd现代压缩算法比现有的传统文件压缩gzip性能更好

zstd常用的命令如下

* -o：用于指定输出文件的名称，如果没有指定，则发送到标准输出（以.zst结尾的文件）
* -c:指定标准输出流
* -k或--keep：保留输入文件（包括普通文件和压缩文件）
* --rm：在压缩或者解压后移除输入文件
* -#：其中的#是一个介于1到19的数字，用于指定压缩级别，数字越大，压缩率越高，但压缩和解压速度越慢，默认为3
* -d或--decompress：解压缩模式（也可以用unzstd）
* -f或--force：强制覆盖已存在的文件而不再提示
* -l 或--list:显示已压缩文件的元数据，如压缩大小、原始大小、压缩级别等
* -v或--verbose：显示详细压缩过程
* -h：显示简洁用法
* -H或--help：显示所有用法
* -V或--version：显示当前版本

3. **Requirement**

|  |  |
| --- | --- |
| **Dut Availability check** | **JIRA Tikcket** |
| adding zstd comp/decomp tools on sel4 - Test Design | [NVOS-77946](https://jira.nioint.com/browse/NVOS-77946) |
| adding zstd comp/decomp tools on sel4 - Test Implement | [NVOS-77949](https://jira.nioint.com/browse/NVOS-77949) |

4. **Test Case**

设置两个测试文件，名称分别为a.cpp、b.txt、c.txt，路径在/tmp下,取c.txt大小为250MB

|  |  |  |  |
| --- | --- | --- | --- |
| test data name | description | file name | original size |
| DI | a disk image | ext4part.bin | 250MB |
| FS | a root FS | rootfs.tar | 598MB |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test ID** | **NVOSQ** | **Test Name** | **category** | **Objective** | **Precedures** | **Expected Results for Each Steps** |  |
| 1 |  | test\_zstd\_version | 功能测试 （单参数） | 查看当前zstd版本 | 1. 在sel4或Linux中运行 2. zstd -V | 执行命令成功，  输出当前zstd版本 | AUTO |
| 2 |  | test\_zstd\_help | 查看zstd的在线帮助 | 1. 在sel4或Linux中运行 2. zstd -h | 执行命令成功，  显示ztsd常用参数  重要字段存在，输出符合预期 | AUTO |
| 3 |  | test\_zstd\_compress\_single\_file | 压缩当前文件 | 1. 在sel4或Linux中运行 2. zstd a.cpp | 执行命令成功，  在当前文件夹下出现a.cpp.zst，且a.cpp保留 | SKIP 与4一起实现 |
| 4 |  | test\_zstd\_compress\_v | 带有-v压缩当前文件 | 1. 在sel4或Linux中运行 2. zstd -v a.cpp | 执行命令成功，在当前文件夹下出现a.cpp.zst，且显示更详细压缩过程a.cpp保留 | AUTO |
| 5 |  | test\_zstd\_compress\_multi\_file | 同时压缩当前文件夹下多个文件 | 1. 在sel4或Linux中运行 2. zstd a.cpp b.txt | 执行命令成功，  在当前文件夹下出现a.cpp.zst和b.txt.zst，且a.cpp和b.txt保留 | AUTO |
| 6 |  | test\_zstd\_compress\_to\_stdout | 测试将压缩文件输出到指定文件名 | 1. 在sel4或Linux中运行 2. zstd a.cpp -o a.cpp.zst | 执行命令成功，  在同目录下生成a.cpp.zst，且a.cpp保留 | AUTO |
| 7 |  | test\_zstd\_compress\_c | 测试将压缩文件输出到标准输出 | 1. 在sel4或Linux中运行 2. zstd -c a.cpp > a.cpp.zst | 执行命令成功，  在同目录下生成a.cpp.zst，且a.cpp保留 | AUTO |
| 8 |  | test\_zstd\_compress\_l | 测试查看压缩文件的属性 | 1. 在sel4或Linux中运行 2. zstd a.cpp 3. Zstd -l a.cpp.zst | 执行命令成功，  在同目录下生成a.cpp.zst，且a.cpp保留  且显示a.cpp.zst的详细属性 | AUTO |
| 9 |  | test\_zstd\_comperss\_force | 测试在已经存在.zst压缩包的情况下再次强行压缩 | 1. 在sel4或Linux中运行 2. zstd a.cpp 3. 修改a.cpp内容 4. zstd -f a.cpp 5. 对比两次压缩包的md5值 | 执行命令成功，  两次生成的a.cpp.zst的md5值不同 | AUTO |
| 10 |  | test\_zstd\_compress\_T0 | 测试-T0参数，使用所有cup核心数进行压缩 | 1. 在sel4或Linux中运行 2. zstd -T0 a.cpp | 执行命令成功，  在同目录下生成a.cpp.zst | AUTO |
| 11 |  | test\_zstd\_compress\_rm | 测试在压缩后删除原文件 | 1. 在sel4或Linux中运行 2. Zstd --rm a.cpp | 执行命令成功  在同目录下生成a.cpp.zst且a.cpp被删除 | AUTO |
| 12 |  | test\_zstd\_decompress\_sigle\_file | 测试解压单个文件 | 1. 在sel4或Linux中运行 2. zstd --rm a.cpp 3. 检查a.cpp.zst是否存在 4. Zstd -d a.cpp.zst 5. 比较解压前后的a.cpp的md5值 | 执行命令成功  在同目录下生成a.cpp.zst且解压前后的a.cpp的md5值相同 | AUTO |
| 13 |  | test\_zstd\_decompress\_multi\_file | 测试解压多个文件 | 1. 在sel4或Linux中运行 2. zstd --rm a.cpp b.txt 3. 检查a.cpp.zst和b.txt.zst是否存在 4. Zstd -d a.cpp.zst b.txt.zst 5. 比较解压前后的a.cpp和b.txt的md5值 | 执行命令成功  在同目录下生成a.cpp.zst和b.txt.zst且解压前后的a.cpp和b.txt的md5值相同 | AUTO |
| 14 |  | test\_zstd\_dc | 功能测试 （多参数） | 测试解压到标准输出流 | 1. 在sel4或Linux中运行 2. zstd --rm a.cpp 3. 检查a.cpp.zst是否存在 4. zstd -dc a.cpp.zst > a.cpp 5. 比较解压前后的a.cpp的md5值 | 执行命令成功  在同目录下生成a.cpp.zst且解压前后的a.cpp的md5值相同 | SKIP 与12一起实现 |
| 15 |  | test\_zstd\_df | 测试强制解压 | 1. 在sel4或Linux中运行 2. zstd a.cpp 3. 检查a.cpp.zst是否存在 4. zstd -df a.cpp.zst 5. 比较解压前后的a.cpp的md5值 | 执行命令成功  在同目录下生成a.cpp.zst且解压前后的a.cpp的md5值相同 | SKIP 与16一起实现 |
| 16 |  | test\_zstd\_dvf | 测试强制解压且同时显示详细信息 | 1. 在sel4或Linux中运行 2. zstd a.cpp 3. 检查a.cpp.zst是否存在 4. zstd -dvf a.cpp.zst 5. 比较解压前后的a.cpp的md5值 | 执行命令成功  在同目录下生成a.cpp.zst且解压前后的a.cpp的md5值相同 | AUTO |
| 17 |  | test\_zstd\_rm\_d | 测试在解压的同时删除原压缩文件 | 1. 在sel4或Linux中运行 2. zstd --rm a.cpp 3. 检查a.cpp.zst是否存在 4. zstd -d --rm a.cpp.zst 5. 比较解压前后的a.cpp的md5值 | 执行命令成功  在同目录下生成a.cpp.zst且解压前后的a.cpp的md5值相同 | AUTO |
| 18 |  | test\_zstd\_compress\_with\_tar | 功能测试 （文件tar打包） | 测试在使用tar之后再进行压缩和解压缩 | 1. 在sel4或Linux中运行 2. tar -I "zstd -T0 --ultra -22" -cvf a.cpp.tar.zst a.cpp 3. 检查a.cpp.tar.zst是否存在 4. tar -I "zstd -T0" -xvf a.cpp.tar.zst 5. 检查a.cpp的md5值与原始是否相同 | 执行命令成功，在同目录下生成a.cpp.tar.zst  且a.cpp的md5值不变 | SKIP 与25一起实现 |
| 19 |  | test\_zstd\_cpmpress\_compressed\_file | 异常测试 | 测试压缩以.zst结尾的文件 | 1. 在sel4或Linux中运行 2. zstd a.cpp 3. 检查a.cpp.zst是否存在 4. zstd a.cpp.zst 5. 检查a.cpp.zst.zst是否存在 | 执行命令成功  在同目录下生成a.cpp.zst  和a.cpp.zst.zst | AUTO |
| 20 |  | test\_zstd\_uncompress\_special\_file | 测试以.zst结尾但是并非是压缩文件的文件 | 1. 在sel4或Linux中运行 2. 手动创建c.zst 3. Zstd -d c.zst | 报错，显示无法解压当前类型文件 | AUTO |
| 21 |  | test\_zstd\_compress\_stdout\_unzst | 测试将压缩的文件输入到一个不是以.zst结尾的文件 | 1. 在sel4或Linux中运行 2. zstd -c a.cpp > d.py | 执行命令成功，成功在同文件夹下生成d.py | AUTO |
| 22 |  | test\_zstd\_compress\_default\_time | 性能测试 | 测试不修改压缩级别的情况下的压缩c.txt的时间 | 1. 在sel4或Linux中运行 2. Time zstd c.txt | 执行命令成功，输出压缩文件的时间 | AUTO |
| 23 |  | test\_zstd\_compress\_level\_time | 测试压缩级别分别为6、12、19的情况下的压缩c.txt | 1. 在sel4或Linux中运行 2. Time zstd -6 c.txt 3. Time zstd -12 c.txt 4. Time zstd -19 c.txt | 执行命令成功，分别输出三种不同压缩级别下的压缩时间 | AUTO |
| 24 |  | test\_zstd\_compress\_high\_level | 测试--ultra -22最大压缩参数 | 1. 在sel4或Linux中运行 2. zstd --ultra -22 c.txt | 执行命令成功，压缩产生c.txt.zst 如果内存不够，则会报错 |  |
| 25 |  | test\_zstd\_with\_tar\_time | 测试在使用tar之后再进行压缩和解压缩一共花费的时间 | 1. 在sel4或Linux中运行 2. Time tar -I "zstd -T0 --ultra -22" -cvf a.cpp.tar.zst a.cpp 3. 检查a.cpp.tar.zst是否存在 4. Time tar -I "zstd -T0" -xvf a.cpp.tar.zst 5. 检查a.cpp的md5值与原始是否相同 | 执行命令成功，然后分别输出打包压缩和解压归档所用时间 | AUTO |

5. **Reference**

[Compression Algorithms: Choosing Compression Algorithms Wisely for FOTA/DCL/DLT -- 压缩算法对FOTA和大量数据传输的影响](https://nio.feishu.cn/wiki/N0RkwTavIimAFZkpWxtczjb3nbd?from=from_copylink)  
[Test Case: Compressed Based Tests on VDF FOTA Image Data](https://nio.feishu.cn/wiki/L0tgwdzdWilSDPkemj4cqtGynYb?from=from_copylink)  
[Compression and Encryption for FOTA/DCL/DLT: Choose the Order Wisely -- 压缩和加密在FOTA和大量数据传输中调用顺序的重要性](https://nio.feishu.cn/wiki/MhHnwmp1di2cyMkp4lOc1sf5nSd?from=from_copylink)