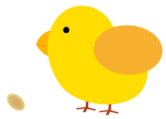
从PCRE到Hyperscan

原创 DPDK开源社区 2017-04-21

作者 昌昊



点击蓝字关注DPDK开源社区

Hyperscan作为高性能的正则表达式匹配库，使用者通常乐意将其与传统的正则表达式匹配库进行比较。“为什么要用Hyperscan？”“使用Hyperscan对我有怎样的好处？”本文以一款广泛使用的传统的正则表达式匹配库PCRE为例，说明Hyperscan与PCRE的差异与优势。

PCRE简介

PCRE是Perl Compatible Regular Expressions的简称，是一款十分流行的用C语言编写的正则表达式匹配库，其灵感来源于Perl语言中的正则表达式功能，其语法比POSIX和其他许多正则表达式库更强大更灵活。

PCRE与Hyperscan的功能对比

PCRE只支持块模式编译和匹配，而Hyperscan不仅支持块模式，还支持流模式。流模式在真实的网络应用场景中更加实用和灵活。

PCRE只支持单条正则表达式的编译和匹配，而Hyperscan可支持多条正则表达式的编译和匹配。在实际应用中经常会遇到多条正则规则的场景，对此Hyperscan可以做到一次编译和一次匹配即完成所有任务，非常高效。

从PCRE到Hyperscan的接口替换

PCRE与Hyperscan的常用接口类似，也分为编译期和运行期。

编译期接口替换：

在使用PCRE的产品中，通常用下面的接口对每一条正则规则分别进行编译:

```
#include <pcre.h>
pcre *pcre_compile2(const char *pattern, int options, int *errorcodeptr, const char **errp
int *erroffset, const unsigned char *tableptr);
```

若要替换为Hyperscan的编译接口十分方便：

```
#include <hs_compile.h>
hs_error_t hs_compile(const char *expression, unsigned int flags,
                      unsigned int mode, const hs_platform_info_t *platform,
                      hs_database_t **db, hs_compile_error_t **error);
```

参数说明：
expression – 单条正则规则。对应pcre的pattern。
flags – 正则规则的标志。对应pcre的options。
mode – 模式选择，与pcre对应的是块模式。
platform – 平台信息。
db – 返回编译生成的Hyperscan数据库。对应pcre编译的返回值。
error – 返回编译错误信息。
返回值为成功或错误码。

另外Hyperscan为多模式匹配提供了接口：

```
hs_error_t hs_compile_multi(const char *const *expressions,
                            const unsigned int *flags, const unsigned int *ids,
                            unsigned int elements, unsigned int mode,
                            const hs_platform_info_t *platform,
                            hs_database_t **db, hs_compile_error_t **error);
```

该接口支持多条正则规则同时编译生成一个Hyperscan数据库，其中参数与hs_compile()的区别为：
expressions – 多条正则规则的数组。
flags – 多条正则规则的标志数组。
Ids – 多条正则规则的id数组。
elements – 正则规则的数量。
其余部分保持不变。

运行期接口替换：

得到编译好的pcre数据库后，通常使用下面的接口对输入语料进行扫描：

```
#include <pcre.h>
int pcre_exec(const pcre *code, const pcre_extra *extra, const char *subject, int length
*ovector, int ovecsize);
```

替换为Hyperscan的运行接口也很方便：

```
hs_error_t hs_scan(const hs_database_t *db, const char *data,
                   unsigned int length, unsigned int flags,
                   hs_scratch_t *scratch, match_event_handler onEvent,
                   void *context);
```

参数说明：
db – Hypersan数据库。对应pcre的code。
data – 输入语料。对应pcre的subject。
length – 输入语料长度。对应pcre的length。
flags – 保留选项。
scratch – 运行中保存临时状态的空间，由hs_alloc_scratch()分配。
onEvent – 匹配命中时的回调函数，由用户自定义。
context – 回调函数参数，用户自定义。
返回值为成功或错误码。

更多API信息请参考：http://01org.github.io/hyperscan/dev-reference/api_constants.html

PCRE与Hyperscan的性能对比

本次对比演示选取了15条正则表达式，既有纯字符串，又有各种正则规则:

id	Signature
1	Twain
2	(?i)Twain
3	[a-z]shing
4	Huck[a-zA-Z]+|Saw[a-zA-Z]+
5	\b\w+nn\b
6	[a-q][^u-z]{13}x
7	Tom|Sawyer|Huckleberry|Finn
8	(?i)Tom|Sawyer|Huckleberry|Finn
9	.{0,2}(Tom|Sawyer|Huckleberry|Finn)
10	.{2,4}(Tom|Sawyer|Huckleberry|Finn)
11	Tom.{10,25}river|river.{10,25}Tom
12	[a-zA-Z]+ing
13	\s[a-zA-Z]{0,12}ing\s
14	([A-Za-z]awyer|[A-Za-z]inn)\s
15	["'][^"']{0,30}[?!\.]["']

测试运行于Intel Xeon E5-2699 @ 2.30GHz的单核上，选取本地随机语料作输入（random-1500b，大小750000Byte）循环2000次。PCRE与Hyperscan完成测试的用时以及吞吐量如下表：

Corpus: random-1500b

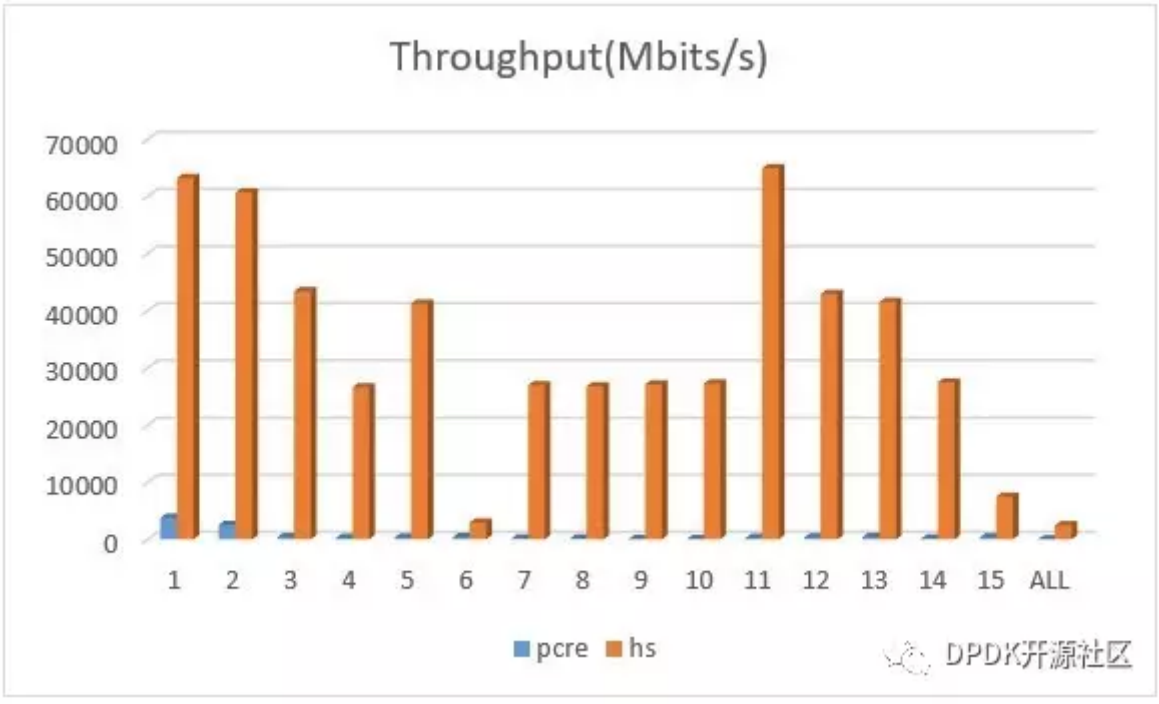
Total data: 750000Byte * 2000

	time(s)		throughput(Mbit/s)	
id	pcre	hs	pcre	hs
1	3.175	0.19	3779.98	63233.97
2	4.77	0.198	2515.79	60691.96
3	32.778	0.276	366.09	43416.93
4	53.027	0.451	226.3	26605.94
5	43.099	0.291	278.43	41247.26
6	30.922	3.936	388.08	3048.78
7	92.815	0.445	129.29	26949.86
8	96.017	0.448	124.98	26796.57
9	317.573	0.443	37.79	27108.72
10	340.121	0.44	35.28	27271.4
11	53.122	0.185	225.9	64948.74
12	40.22	0.279	298.36	42938.99
13	30.907	0.289	388.27	41547.02
14	99.669	0.437	135.34	27454.99
15	38.623	1.614	310.7	7436.96
ALL	1276.84	4.755	9.39822	2523.66

从结果可见，对以上每一个规则，Hyperscan对比PCRE都有巨大性能优势，吞吐量最少提升7.54倍，最多提升多达773倍。

Hyperscan的另一个优势也在结果中体现，那就是当需要对多条正则规则进行匹配时。Hyperscan可以将所有规则编译进一个数据库，运行时只需以此数据库对输入语料运行一遍，完成全部工作用时4.879秒，少于15次用时的累加10.086秒。而PCRE只能对单条规则做编译和运行，对于全部15条规则，每条规则都需要单独编译并对输入语料运行一次，完成全部工作用时1276.84秒，为15次用时的累加。

以下是根据吞吐量生成的柱状图：



PCRE与Hyperscan的差距如此之大？事实上PCRE也有快速选项pcre_jit，在编译时花费更多的时间和资源来加速运行时的扫描速度，我们也将Hyperscan与pcre_jit进行了对比试验。

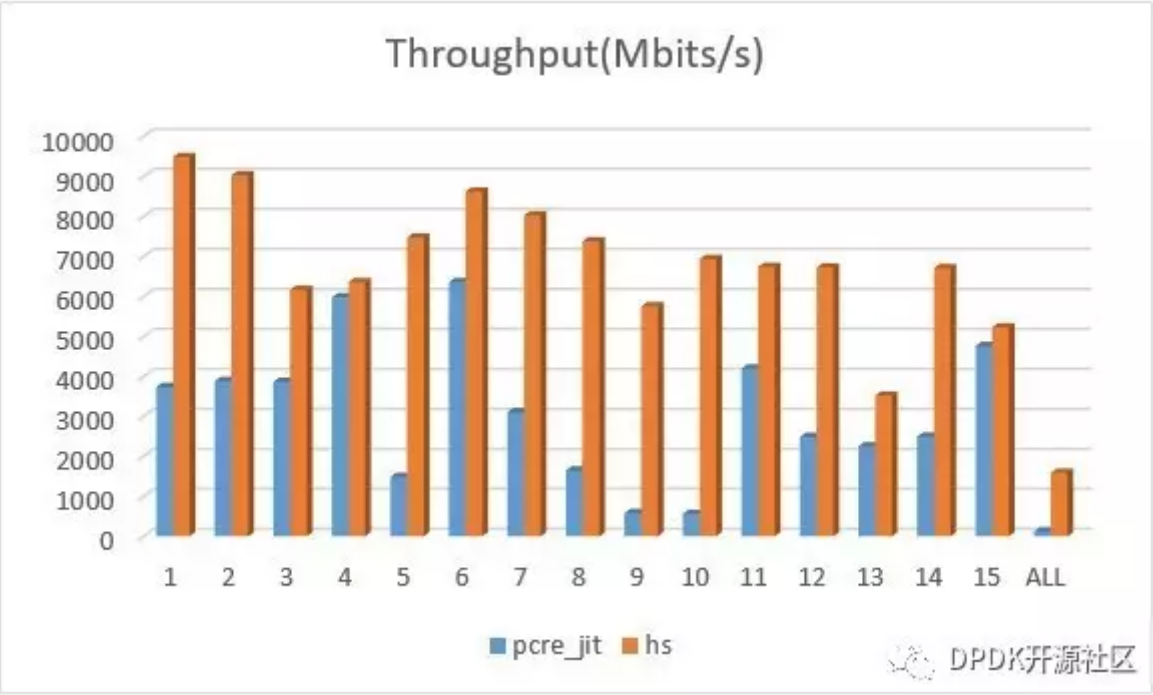
随机语料作为输入，其中匹配数量较少，那么对于匹配数量较多的情形呢？为此我们选取一本20M左右（18905427Bytes）的合适的电子书（http://www.gutenberg.org/files/3200/old/mtent12.zip）作为输入语料，在相同测试环境下循环20次。pcre_jit与Hyperscan完成测试的用时如下表：

Corpus: mtent.txt

Total data: 18905427Byte * 20

	time(s)		throughput(Mbits/s)	
id	pcre_jit	hs	pcre_jit	hs
1	0.811	0.319	3729.8	9482.35
2	0.779	0.335	3883.01	9029.46
3	0.783	0.491	3863.18	6160.63
4	0.506	0.476	5978	6354.77
5	2.036	0.405	1485.69	7468.81
6	0.476	0.351	6354.77	8617.86
7	0.975	0.377	3102.43	8023.52
8	1.836	0.41	1647.53	7377.73
9	5.188	0.526	583.05	5750.7
10	5.407	0.436	559.44	6937.77
11	0.721	0.449	4195.38	6736.9
12	1.217	0.45	2485.51	6721.93
13	1.345	0.859	2248.97	3521.38
14	1.214	0.451	2491.65	6707.03
15	0.636	0.579	4756.08	5224.3
ALL	**23.93**	**1.901**	**126.4**	**1591.2**

可见pcre_jit大幅度提升了pcre的扫描速度，但在每单条正则规则的表现上还是Hyperscan占优；而对于全部15条规则的多模式匹配，Hyperscan相对pcre_jit的优势更加明显。根据吞吐量生成的柱状图为：



除了性能上的巨大优势，Hyperscan对流模式的支持更是功能优势，因为快和实用，Hyperscan适合在越来越多的使用场景中替换原有的正则表达式匹配引擎，前景大好。

作者简介

昌昊：英特尔软件工程师，负责Hyperscan算法开发和性能调优相关工作。主要研究领域包括自动机与正则表达式匹配等。

- 基于virtio-user的新exception path方案
- DPDK Release 17.02
- Hyperscan Release 4.4.0
- DPDK Release 16.11
- 无锁队列详细分解——Lock与Cache，到底有没有锁？
- 从计算机架构师的角度看DPDK性能
- 欢迎搭乘Hyperscan号极速列车~
- 无锁队列详细分解 — 顶层设计
- VMware Player 搭建DPDK实验平台
- Qemu/Kvm 搭建DPDK实验平台

DPDK开源社区

最权威的DPDK社区

长按二维码关注

投诉