深入理解 irqbalance

2015/11/23
renyl

1 介绍

1) irqbalance 服务用于自动优化硬件中断分配。它通过对定期采集(每隔 10 秒采集一次)的系统数据（CPU softirq 和 irq 负载）进行分析，根据分析的结果来修改文件（/proc/irq/N/smp_affinity）的值（即改变中断号的亲属性），从而达到优化硬件中断分配的目的。

2) irqbalance 分为两种模式：Performance mode（默认模式）和 Power-save mode。
   a) Performance mode 时，irqbalance 会将中断尽可能均匀地分配给每个 CPU，从而充分利用 CPU 多核特性来提升性能。
   b) Power-save mode 时，irqbalance 会将中断集中分配给某个 CPU，从而保证其它空闲 CPU 的睡眠时间，降低系统能耗。

注 1：本文在如下平台进行 irqbalance 研究。

-	描述
os	RHEL7.0_x86_64
kernel	kernel-3.10.0-110.el7.x86_64
cpu	Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
glibc	glibc-2.17-52.el7.x86_64。
irqbalance	v1.0.9 commit_id:8922ff13704dd0e069c63d46a7bdad89df5f151c

注 2：irqbalance 代码托管地址：https://github.com/Irqbalance/irqbalance

2 背景知识

在介绍 irqbalance 的实现原理之前，先了解下硬件中断的相关背景知识。系统的硬件中断主要有两大来源：网卡中断和 iSCSI 中断，其中网卡中断又最为常见。

2.1 硬件中断的角色

以网卡的接收数据为例，来看下硬件中断在其中扮演的角色，其过程如下：

1) 硬件接受：网卡通过物理层接收到数据帧，然后将其存储至网卡的缓冲队列中。
2) 硬件中断：网卡向 CPU 发送硬件中断告诉 CPU 网卡里有数据快来取。CPU 响应该中断并调用中断处理程序进行处理（主要就是将网卡中的数据拷贝到内存中）。
3) 软中断：CPU 把内核中的原始数据交给网络协议栈进行处理。
4) 应用接受：应用层程序通过调用相关系统调用（如 recvfrom()）从网络协议栈中获取数据。

注：系统把硬件中断和软中断进行分开处理的原因：

a) 由于硬件中断仅仅只是响应一下网卡，速度非常快，同时其必须要快，因为硬件中断不容许内核抢占，如果速度不快的话，稍微有点网络负载的话，用户空间的程序将有巨大延迟，这是用户无法接受的。
b) 同时，软中断也不一定就是在硬件中断发生后就立即发生，而是会选择一个适当的时间点开始执行。

2.2 硬件中断的优化

如上所述，网卡的接受数据大致可以分为上述四个过程，那么针对网卡的优化就可以对上述四个过程分别进行优化：（irqbalance 服务就是针对硬件中断进行优化）

1) 硬件接受：
   a) 网卡的接受/发送队列的 buffer 是可以进行调节的，可以通过命令“ethtool -g XXX”进行调节。
   b) 同时 ethtool 还可以调节 rx-usecs 来控制何时向 CPU 发送硬件中断。

2) 硬件中断：
   a) 在网卡单队列时，一个网卡同一时刻只能由一个 CPU 进行响应硬件中断。（需要注意的是，由于中断处理程序速度很快，所以一般硬件中断不会导致性能瓶颈。）
   b) 为了提高性能，现代网卡都是多队列的（即一个网卡可以有多个中断号），那么一个网卡可以同时有多个中断响应（即可由多个 CPU 同时进行处理）。

3）软件中断：
   a) 在网卡单队列时，软件中断将和硬件中断采用相同的 CPU，于是软件中断很有可能成为性能瓶颈。Tom Herbert 为了解决这个问题，搞了个 RPS（Receive Packet Steering，相关设置在/sys/class/net/eth0/queues/目录下），可以把软件中断分散到多个CPU上，避免单个CPU负载过大导致性能瓶颈，从而实现达到负载均衡。

b) 网卡多队列时，软件中断也是和硬件中断采用相关的 CPU，硬件中断可以配置文件 smp_affinty_list 来设置不同的 CPU 亲属性，这样软中断也就均匀分配到不同的 CPU 上，同样达到负载均衡的效果。

4）应用程序接受：

a) 不管是网卡单队列还是多队列，都会出现软中断处理的 CPU 和应用程序处理的 CPU 不是同一个 CPU 的情况，这样 CPU 之间就会发生 IPI(处理器间中断），CPU Cache 的利用率就会变低，从而影响程序性能。

b) 为了解决 CPU Cache 利用率低的问题，Tom Herbert 又搞了个 RFS（Receive Flow Streering），尽可能的让软中断和应用程序使用相同的 CPU 进行处理，从而提高 CPU 的 Cache 利用率，达到性能优化的目的。

注：

a) 在 2.6.21 版本内核之前，linux 仅支持单队列的网卡，这样一块网卡只能有一个中断号，不过一个中断号可以绑定到多个 CPU 上,即网卡中断可以由不同的 CPU 响应,但某一个时刻只有一个 CPU 在响应。

b) 在 2.6.21 版本内核之后，linux 开始支持多队列的网卡了，这样一块网卡可以有多个中断号，那么多个 CPU 就可以同时响应一个网卡上的不同中断号。

c) 网卡的中断分配是通过四元组（源 IP，源 Port，目的 IP，目的 Port）进行相关计算被分配到某一个中断号上，即相同的四元组将始终触发相同的网卡中断号。

d) Message Signaled Interrupts(MSI)是 PCI 规范的一个实现,可以突破 CPU 256 条 interrupt 的限制，使每个设备具有多个中断线变成可能，多队列网卡驱动给每个 queue 申请了 MSI。

3 实现原理

通过研究 irqbalance 源代码，可以知道其处理过程，大致如下图所示：

图 3-1：irqbalance 的处理流程

由上图可知 irqbalance 服务的完整处理流程，接下来，将针对各个流程的具体实现方法进行详细说明。

3.1 构建 CPU 树和 irq_db

1) CPU 树构建是通过读取两个目录（/sys/devices/system/node/ 和 sys/devices/system/cpu/）下的文件来完成的。
2) irq_db 的构建是通过读取目录（/sys/bus/pci/devices）下的文件，以及 /proc/interrupts 文件来完成的。

构建完成后的 CPU 树和 irq_db 结构，如下图所示：

图 3-2 CPU 树结构

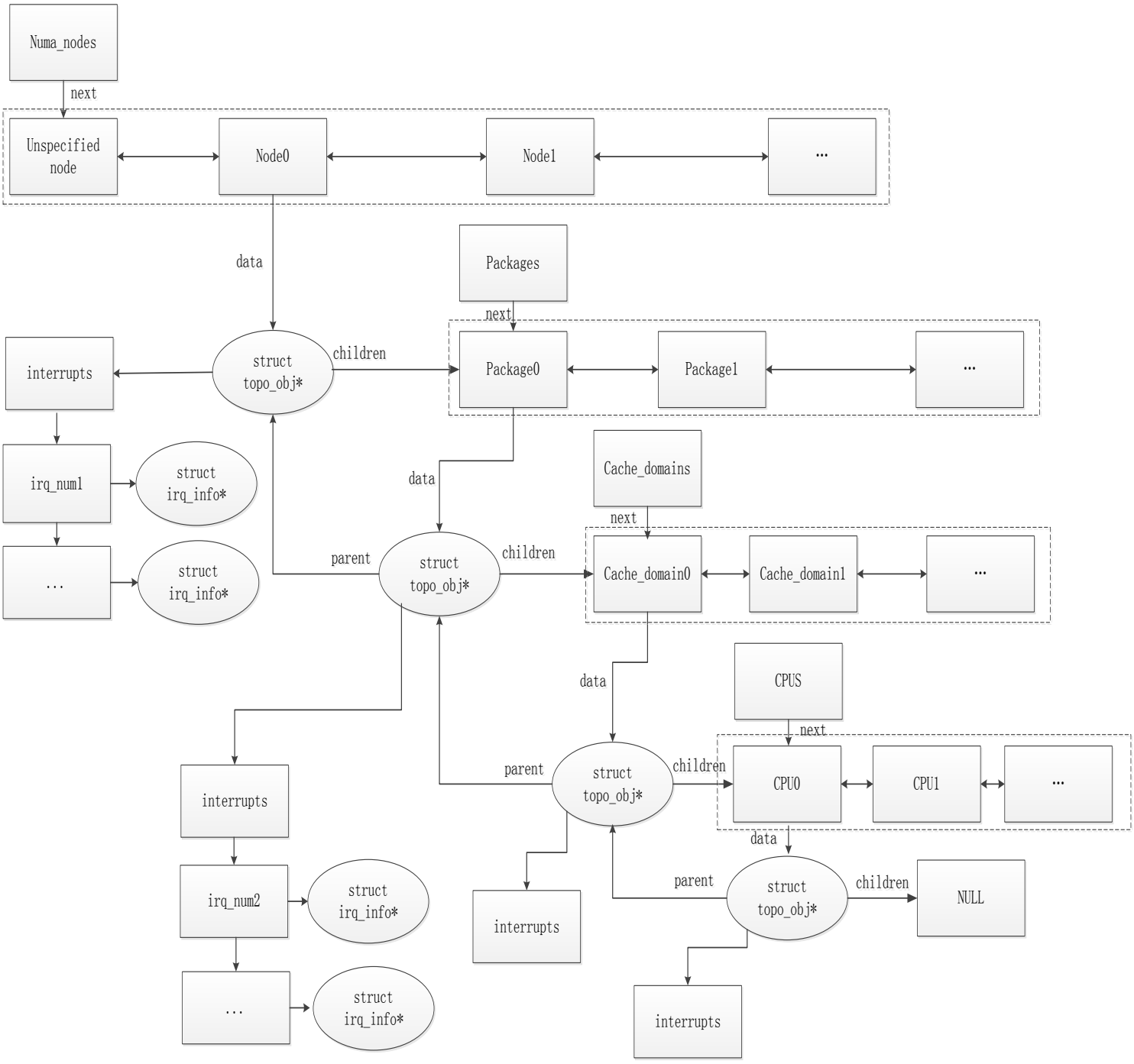
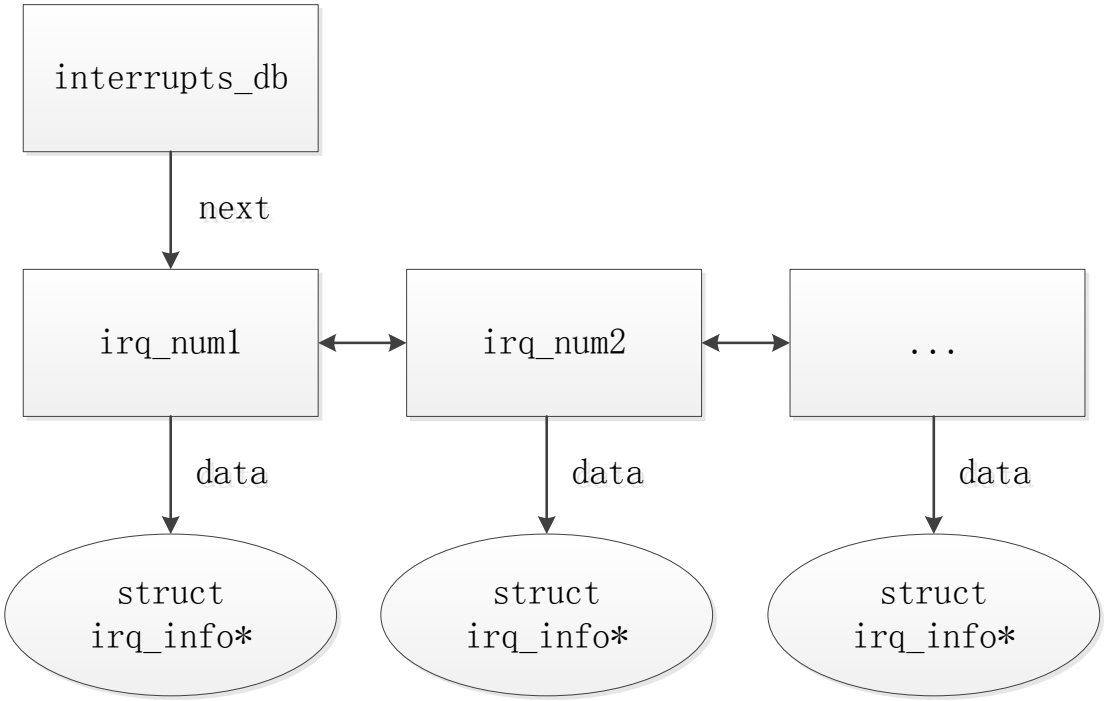


图 3-3 irq_db 结构



说明：

方括号表示的都是一个 Glist 结构体，椭圆号表示的是 CPU 和 irq 相关的信息，具体结构定义如下：

-	-	-

struct_GList {
    void *data;
    GList *next;
    GList *prev;
};
	struct topo_obj {
	    uint64_t load;
	    uint64_t last_load;
	    uint64_t irq_count;
	    enum obj_type_e obj_type;
	    int number;
	    int powersave_mode;
	    cpumask_t mask;
	    GList *interrupts;
	    struct topo_obj *parent;
	    GList *children;
	    GList **obj_type_list;
		struct irq_info {
		    int irq;
		    int class;
		    int type;
		    int level;
		    int flags;
		    struct topo_obj *numa_node;
		    cpumask_t cpumask;
		    cpumask_t affinity_hint;
		    int hint_policy;
		    uint64_t irq_count;
		    uint64_t last_irq_count;
		    uint64_t load;
		    int moved;
		    struct topo_obj *assigned_obj;
		    unsigned int warned;
		    char *name;
		};

3.2 解析/proc/interrupts 文件

1) 读取/proc/interrupts 文件来统计各个 irq 的中断数。
2) 修改 CPU 树中各个 irq 的 irq_info 结构体中的 irq_count 和 last_irq_count 参数，为后面找出需要移植的 irq 做准备。

3.3 解析/proc/stat 文件

1) 读取/proc/stat 文件来统计各个 CPU 的负载，
2) 修改 CPU 树中各个 CPU 的 topo_obj 结构体中的 last 和 last_load 参数，为后面找出需要移植的 irq 做准备。

3.4 找出需要移植的 irq

在 CPU 树中找出所有需要移植的 irq，这个过程是从 CPU 树的底层向顶层逐层进行寻找，即从 CPU->Cache_domain->Package->Node 这样一层一层向上的方法进行寻找，整个寻找过程较为复杂，详细如下：（以 CPU 层次为例）

1) 先找出CPU层次中，所有irq中load(irq_info结构体的参数load)的最小值min_load。

2) 针对 CPU 层次中的每个 CPU 进行如下判断：
   a) 只有在，当前 CPU 的 load（topo_obj 的参数 load）大于 min_load，且该 CPU 连接的 irq 数大于 1 时，才继续进行下一步，否则换 CPU 层次中的下一个 CPU 继续。

   b) 针对当前 CPU 下连接的每个 irq 进行如下判断操作：

```
if (cpu.load  -  min_load > 2 * irq.load )
{
    cpu.load=cpu.load  -  irq.load
    min_load=min_load + irq.load
}
else
    return;
migrate_irq();
```

3) 通过对 CPU 层次中的所有 CPU 进行扫描判断，最后可以得出 CPU 层次中所有需要进行移植的 irq。

通过同样的方法，再对 Cache_domain、Package 和 Node 层进行寻找，最终将找到整个 CPU 树中所有需要进行移植的 irq（即 rebalance_irq_list 链表）。

3.5 在 CPU 树中移植 irq

在 CPU 树中找出需要移植的 irq（即 rebalance_irq_list）是通过从底层向顶层一层一层寻找的，而在 CPU 树中移植 irq 则是通过顶层向底层一层一层移植的，具体过程如下：

1) 扫描 rebalance_irq_list 中的每个 irq，根据 irq 所对应的 node 号（由 irq_info 结构中的 numa_node 参数决定），把该 irq 连接到 CPU 数中对应的 node 下。

2) 接下来，就是从 Node->Package->cache_domain 这样一层一层向下移植 irq，具体移植过程如下：（以 Package 层为例针对 Packages 层次中的每个 Package 所连接的 irq 进行如下判断）
   a) 判断当前 irq 是否需要进行移植（检查 irq_info 结构的 moved 标志），如果不需要移植的话，换下一个 irq 进行判断。
   b) 判断当前 irq 的移植平衡范围是否为 BALANCE_PACKAGE，如果是的话，表明不需要移动到下一层（即 cache_domain 层），换下一个 irq 从 a）开始。
   c) 在当前 Package 层次中所连接的 cache_domain 层次中，寻找一个负载最小的 cache_domain，移植该 irq 到负载最小的 cache_domain 上。

通过同样的方法，再对 Cache_domain 进行移植，最终将移植完所有的 irq 到 CPU 树中。

3.6 修改移植 irq 的 CPU 亲属性

修改移植 irq 的 CPU 亲属性，其过程如下：

1) 针对 CPU 树中的每个 irq 进行判断,是否是移植过的 irq,如果是的话继续下一步操作。

2) 针对当前 irq 的 hint_policy 不同进行不同处理:
   a) 如果是 HINT_POLICY_EXACT，那么该 irq 的 CPU 亲属性就是 irq_info 结构的 affinity_hint 参数值。
   b) 如果是 HINT_POLICY_SUBSET,那么 irq 的亲属性就是 irq 所连接的 CPU 子码 cpu.mask 和 irq 的 irq_info 结构 affinity_hint 参数之间的“与”值(即 irq 的亲属性 = cpu.mask & irq.affinity_hint)。
   c) 如果不是上面两种情况，表明 hint_policy 为 HINT_POLICY_IGNORE,此时 irq 的新属性将为 irq 所连接的 CPU 子码 cpu.mask。

3) 最后,通过修改/proc/irq/N/ smp_affinity 文件中的值来完成改变 irq 的 CPU 亲属性。

4 使用方法

关于 irqbalance 的使用方法，可以通过命令行指定相关参数，也可以通过修改 irqbalance 的配置文件（/etc/sysconfig/irqbalance）来完成，主要参数介绍如下：

序号	参数	说明
1	--oneshot	irqbalance 只运行一次就退出。
2	--debug	debug 模式，可以向终端打印出 irqbalance 运行过程中 irq 的终端数和负载等信息。
3	--foreground	前台模式运行。
4	--hintpolicy= [exact | subset | ignore]	设置 irq 的 affinity hinting 如何被处理，默认值为 ignore。
5	-i	设置禁止被 irqbalance 改变亲属性的 irq 号。
6	--deepestcache	这个参数将会影响到 CPU 树的构建架构，默认情况下该参数为 2，表明 cache_domain 层管理的是系统的逻辑 CPU（非超线程开启的 CPU）。
7	--interval=<time>	设置 irqbalance 多久针对 irq 进行一次 balance，默认值为 10 秒。
8	IRQBALANCE_ONESHOT	同参数 --oneshot 一样。
9	IRQBALANCE_DEBUG	同参数 --debug 一样。
10	IRQBALANCE_BANNED_CPUS	同参数 --i 一样。

5 参考资料

https://github.com/Irqbalance/irqbalance