Network Heartbeat in Linux

Francesco Piermaria

francesco.piermaria@gmail.com

Linux Kernel Hacking Free Course IV Edition

Rome, 14 May 2008

Outline

Goal

- Synchronize all cluster nodes ticks
    - To implement a base component of Cluster Advanced Operating System (CAOS)
        - To improve HPC application performances

Cluster Advanced Operating System

Cluster Advanced Operating System (CAOS):

- Born from a System Programming Research Group's idea
- A forthcoming *distributed operating system*

What's new:

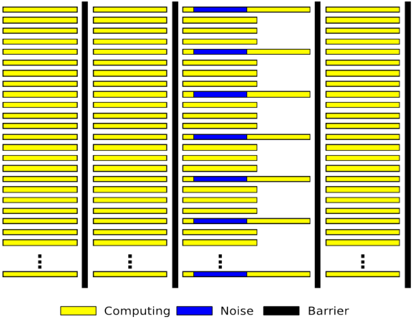
- Synchronize ticks and tick's related activities *(timekeeping, process accounting, profiling, etc.)*
- Globally schedule: forcing all nodes in the cluster to perform in the same moment
  - Software timers related activities *(flush cache, etc.)*
  - System asynchronous activities *(page frame reclaiming, time sharing, etc.)*
- Offer additional features like *distributed data check-pointing, distributed debugging, process migration* etc.

Here we focus on the *Network Heartbeat* solution, which deals with the first point

Tick Synchronization

Improve HPC application performance

A possible solution is to force all nodes to perform system activities in a *synchronous* way, as shown in figure



(a) co-scheduled OS noise

It's not easy to achieve *global synchronization* of system activities for the whole cluster, in such a way to be sure that *all* nodes will execute the activities *exactly* in the same moment

Tick Synchronization (2)

Why is not easy to achieve global synchronization of system activities?

- Each node uses his *own* timer device to *measure* time
- Even timer devices of the same type oscillate at slightly different frequencies
- Tick period is slightly different from node to node

Machines have different perception of the flow of time

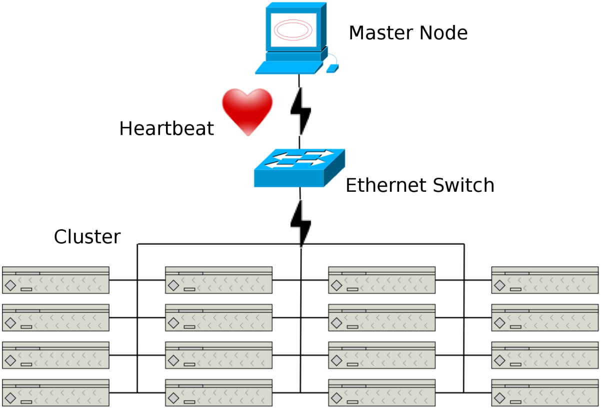
Tick Synchronization (3)

Goal: To synchronize cluster's nodes system activities

- We have to synchronize the flow of time among nodes
  - Network Time Protocol (NTP) is NOT a solution! It synchronizes only the *Wall Clock Time*...
  - We have to synchronize *system ticks*
- We need a general mechanism to schedule overall cluster's system activities at the same future time

As a CAOS's first implementation step, we introduce the *Network Heartbeat*, which allows tick synchronization.

Network Heartbeat Schema



(b) network heartbeat

Timekeeping Kernel's Related Data Structures

Linux kernel offers specific abstract data structures to represent time related hardware devices:

- `struct clocksource`: provides a time value to the kernel
- `struct clock_event_device`: notifies the kernel that a *well defined amount* of time is elapsed
- `struct tick_device`: represents the best *clock event device* available to the kernel

Clocksource Device

The Linux kernel keeps track of the best available *clocksource* in the `clock` global variable

```
struct clocksource {
    char *name;
    int rating;
    cycle_t (*read)(void);
    u32 mult, shift;
    unsigned long flags;
    cycle_t cycle_last;
    /* ... */
};
```

The *clocksource* data type is used to represent hardware like *TSC, PIT, HPET, ACPI_PM, TIMEBASE, etc.*

I.e. the `update_wall_time()` function updates *wall clock time* using the `clock` variable

```
offset = clock->read() - clock->cycle_last;
xtime.tv_nsec += offset * clock->mult;
```

Clock Event Device

Clock Event Devices are used to raise hardware timer interrupts at specified time

```
struct clock_event_device {
    const char *name;
    unsigned long max_delta_ns,
        min_delta_ns;
    unsigned long mult, features;
    int shift, rating, irq;
    cpumask_t cpumask;
    int (*set_next_event)(unsigned
        long evt);
    void (*set_mode)(enum
        clock_event_mode mode);
    void (*event_handler)();
    ktime_t next_event;
    /* ... */
};
```

The *clock event* data type is used to represent hardware like *LAPIC, HPET, DECREMENTER, etc.*

The function `set_next_event(msec)` is used to program the time of the *next time event*

Tick Device

The Linux kernel keeps track of the best available *clock event device* in the `tick_cpu_device` per-CPU variable (*tick device* for short)

If *High Resolution Timers* are enabled:

- The *tick device* raises an hardware interrupt at the time of the next time event (not necessarily a tick time event)

```
struct tick_device {
    struct clock_event_device
        *evtdev;
    enum tick_device_mode
        mode;
};
```

The *mode* field can be

- *periodic*: periodic tick mode
- *oneshot*: dynamic tick mode

Network Heartbeat Main Topics

The Network Heartbeat:

- **Deals with the problem of the overall cluster's nodes *tick synchronization***
- Patch for the Linux 2.6.24 kernel
- Designed for Symmetric-Multi-Processing (SMP) systems
- Developed for the *Intel IA32*, *Intel64* and *PowerPC64* architectures
- Based on the *Ethernet* communication channel
- Permits to dynamically change node's tick frequency

Network Heartbeat Implementation

The *Network Heartbeat Implementation* is built on two software components:

- **Network Event Device:** a *virtual* per-CPU timer event device, which replaces *local metronomes*
- **Nettick:** an interrupt *emulation* module, which translates the event *"new Ethernet frame"* in a *timer interrupt* event

Network Heartbeat Implementation (2)

The *nettick* module registers a new network protocol. It accomplishes the following:

- Recognize a new *Ethernet frame* type, introduced to notify the *tick* event type
- Send an *Inter Processor Interrupt* to all the online CPUs, thus "simulating" a local timer interrupt

The *Network Event Device* `net_event` :

- Handles the IPI sent by the *nettick* module and instructs the CPU on which is registered to execute tick and time management related functions
- Exports an *user interface* which allows users to choose the kernel operating mode (*global metronome* or *local metronomes*)

Network Event Device Implementation

The *virtual* Network Event Device implements the `set_mode()` and `set_next_event()` methods

```
static struct clock_event_device
    net_event = {
    .name           = "net_event",
    .features = CLOCK_EVT_FEAT_ONESHOT,
    .set_mode       = net_timer_setup,
    .set_next_event = net_next_event,
    .set_new_rate   = set_new_rating,
    .event_handler  = net_handle_noop,
    .rating         = 0,
    .irq            = -1,
    .nr_events      = 0,
};
DEFINE_PER_CPU(struct clock_event_device
    , net_events);
```

The `(*event_handler)()` will be re-associated to the right handler either when device is registered or when rating changes.

The timekeeping framework's interface was extended to make possible to change the rating value of a *clock event* device

Nettick Implementation

The *nettick* module registers a new network protocol. As in example, on the *Intel x86* architecture:

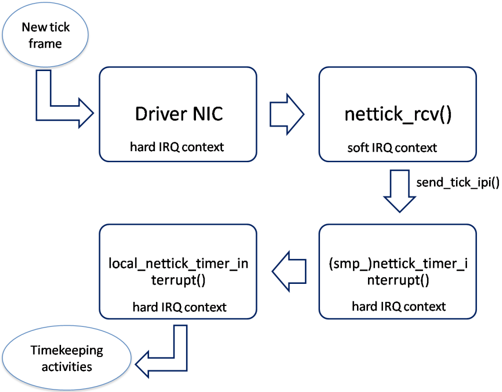
```
static struct packet_type
  nettick_packet_type = {
  /* ETH_P_NETTICK 0x88CB */
  .type = htons(ETH_P_NETTICK),
  .func = nettick_rcv,
};
dev_add_pack(&nettick_packet_type);
                ...
nettick_rcv(){
  send_ipi_mask(cpu_online_mask,
            NETTICK_TIMER_VECTOR);
}
```

The *nettick* protocol handler sends an Inter Processor Interrupt to all *online* CPUs.

N.B. the `smp_call_function()` function **cannot** be used because the `nettick_rcv()` handler is executed in *softirq* context!

Timer Interrupt Emulation

The *new* network protocol is required to make the implementation *independent* of the particular Network Interface Card's driver



(c) From Ethernet Frame to Timer Interrupt Emulation

- *nettick* handler is executed in *soft irq* context
- *tick* related activities must be executed in *hard irq* context
- IPI is the only way to interrupt other CPUs

Timer Interrupt Emulation (2)

A new IPI message has to be registered by calling the `set_intr_gate()`.

- The IPI *network tick* message handler cleans the interrupt channel and then call the following function

```
local_nettick_timer_interrupt(){
  struct clock_event_device *dev =
      &__get_cpu_var(net_events);
  dev->nr_events++;
  dev->event_handler(dev);
}
```

1. *Gets* the ref. to the per-CPU *network event device*
2. *Increments* the device's event counter
3. Enters the *timekeeping* system

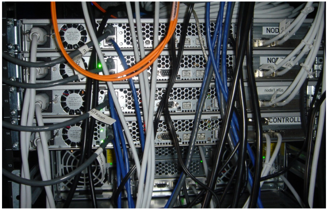
- The chain of events is identical as the chain originated by a local timer interrupt!

Test Environment

Tests was performed on a 24-nodes *Apple Xserve* cluster, generously offered by *Italian Defence's General Stuff*

- Each node is equipped of 2 *dual core* Intel Xeon 5150 processors (freq. 2.66GHz) overall *96 cores*, 4GByte of RAM and 2 Gigabit Ethernet Network Interface Cards
- One of the 24 nodes was employed as *master node*

(d) Cluster Front View

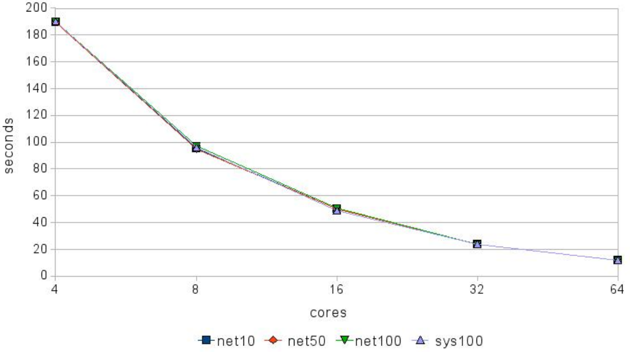


(e) Cluster Back View

Network Heartbeat Test

Tests goal: ***overhead*** and *scalability*

- NAS Parallel Benchmark - Embarrassing Parallel

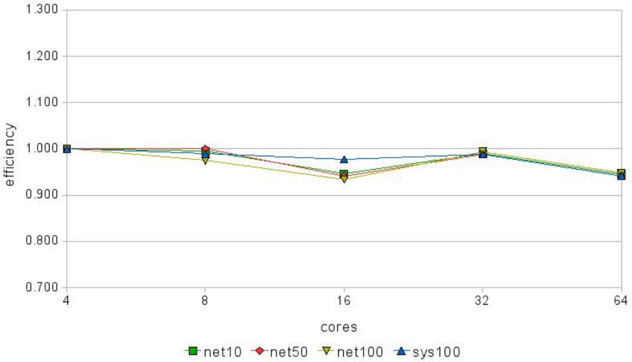


(f) Benchmark EP Results

Network Heartbeat Test (2)

Tests goal: *overhead* and ***scalability***

- NAS Parallel Benchmark - Embarrassing Parallel



(g) Benchmark EP Efficiency

Conclusions and Future Works

The Network Heartbeat allows to synchronize cluster's nodes ticks by means a *global metronome*

Tests highlight that proposed approach:

- *does not* introduce *overhead*
- *scales* adequately when cluster's nodes increase in number

Future works:

- Extend *nettick* protocol to allow master node to schedule activities in a *centralized* way

Bibliography

- D.P. Bovet, M. Cesati, *Understanding the Linux Kernel, 2nd Edition*, O'Reilly
- R. Gioiosa and F. Petrini and K. Davis and F. Lebaillif-Delamare *Analysis of System Overhead on Parallel Computers*, The 4th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2004)
- D. Tsafrir and Y. Etsion and D.G. Feitelson and S. Kirkpatrick *System noise, OS clock ticks, and fine-grained parallel applications*, ICS '05: Proceedings of the 19th annual international conference on Supercomputing, Cambridge, Massachusetts, 303–312